Architecture pour le filtrage dans les bases de données relationnelles

Michel Scholl

▶ **To cite this version:**

Michel Scholl. Architecture pour le filtrage dans les bases de données relationnelles. Modélisation et simulation. Institut National Polytechnique de Grenoble - INPG, 1985. tel-00316167

HAL Id: tel-00316167
https://theses.hal.science/tel-00316167

Submitted on 3 Sep 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre:

THÈSE

présentée à

L' INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR D'ÉTAT ÈS SCIENCES

par

Michel SCHOLL

Spécialité : **Informatique**

Sujet de la thèse :

ARCHITECTURE POUR LE FILTRAGE DANS LES BASES DE DONNÉES RELATIONNELLES

Soutenue le 19 juin 1985 devant la Commission composée de :

MM.	**G.**	**VEILLON**	Président
	M.	**ADIBA**	Examinateurs
	F.	**BANCILHON**	
	C.	**DELOBEL**	
	B.	**LORHO**	
	A.	**PIROTTE**	
	J.	**ROHMER**	

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président: Daniel BLOCH

Vice-Présidents: René CARRE
Hervé CHERADAME
Jean-Pierre LONGEQUEUE

Année universitaire 1983-1984

Professeur des Universités

ANCEAU	François	E.N.S.I.M.A.G
BARIBAUD	Michel	E.N.S.E.R.G
BARRAUD	Alain	E.N.S.I.E.G
BAUDELET	Bernard	E.N.S.I.E.G
BESSON	Jean	E.N.S.E.E.G
BLIMAN	Samuel	E.N.S.E.R.G
BLOCH	Daniel	E.N.S.I.E.G
BOIS	Philippe	E.N.S.H.G
BONNETAIN	Lucien	E.N.S.E.E.G
BONNIER	Etienne	E.N.S.E.E.G
BOUVARD	Maurice	E.N.S.H.G
BRISSONNEAU	Pierre	E.N.S.I.E.G
BUYLE BODIN	Maurice	E.N.S.E.R.G
CAVAIGNAC	Jean-François	E.N.S.I.E.G
CHARTIER	Germain	E.N.S.I.E.G
CHENEVIER	Pierre	E.N.S.E.R.G
CHERADAME	Hervé	U.E.R.M.C.P.P
CHERUY	Arlette	E.N.S.I.E.G
CHIAVERINA	Jean	U.E.R.M.C.P.P
COHEN	Joseph	E.N.S.E.R.G
COUMES	André	E.N.S.E.R.G
DURAND	Francis	E.N.S.E.E.G
DURAND	Jean-louis	E.N.S.I.E.G
FELICI	Noël	E.N.S.I.E.G
FONLUPT	Jean	E.N.S.I.M.A.G
FOULARD	Claude	E.N.S.I.E.G
GANDINI	Alessandro	U.E.R.M.C.P.P
GAUBERT	Claude	E.N.S.I.E.G
GENTIL	Pierre	E.N.S.E.R.G
GUERIN	Bernard	E.N.S.E.R.G
GUYOT	Pierre	E.N.S.E.E.G
IVANES	Marcel	E.N.S.I.E.G
JALINIER	Jean-Michel	E.N.S.I.E.G
JAUSSAUD	Pierre	E.N.S.I.E.G
JOUBERT	Jean-Claude	E.N.S.I.E.G
JOURDAIN	Geneviève	E.N.S.I.E.G
LACOUME	Jean-Louis	E.N.S.I.E.G
LATOMBE	Jean-Claude	E.N.S.I.M.A.G
LESIEUR	Marcel	E.N.S.H.G
LESPINARD	Georges	E.N.S.H.G
LONGEQUEUE	Jean-Pierre	E.N.S.I.E.G
LOUCHET	François	E.N.S.E.E.G
MASSELOT	Christian	E.N.S.I.E.G
MAZARE	Guy	E.N.S.I.M.A.G
MOREAU	René	E.N.S.H.G
MORET	Roger	E.N.S.I.E.G
MOSSIERE	Jacques	E.N.S.I.M.A.G
PARIAUD	Jean-Charles	E.N.S.E.E.G
PAUTHENET	René	E.N.S.I.E.G
PERRET	René	E.N.S.I.E.G
PERRET	Robert	E.N.S.I.E.G
PIAU	Jean-Michel	E.N.S.H.G
POLOUJADOFF	Michel	E.N.S.I.E.G
POUPOT	Christian	E.N.S.E.R.G
RAMEAU	Jean-Jacques	E.N.S.E.E.G
RENAUD	Maurice	U.E.R.M.C.P.P
ROBERT	André	U.E.R.M.C.P.P
ROBERT	François	E.N.S.I.M.A.G
SABONNADIERE	Jean-Claude	E.N.S.I.E.G
SAUCIER	Gabrielle	E.N.S.I.M.A.G
SCHLENKER	Claire	E.N.S.I.E.G
SCHLENKER	Michel	E.N.S.I.E.G
SERMET	Pierre	E.N.S.E.R.G
SILVY	Jacques	U.E.R.M.C.P.P
SOHM	Jean-Claude	E.N.S.E.E.G
SOUQUET	Jean-Louis	E.N.S.E.E.G
VEILLON	Gérard	E.N.S.I.M.A.G
ZADWORNY	François	E.N.S.E.R.G

Professeurs Associés

BLACKWELDER	Ronald	E.N.S.H.G
HAYASHI	Hirashi	E.N.S.I.E.G
PURDY	Gary	E.N.S.E.E.G

Professeurs Université des Sciences Sociales (Grenoble II)

BOLLIET	Louis
CHATELIN	Françoise

Chercheurs du C.N.R.S

FRUCHART	Robert	Directeur de recherche
JORRAND	Philippe	Directeur de recherche
VACHAUD	Georges	Directeur de recherche
ALLIBERT	Michel	Maître de recherche
ANSARA	Ibrahim	Maître de recherche
ARMAND	Michel	Maître de recherche
BINDER	Gilbert	Maître de recherche
BORNARD	Guy	Maître de recherche
CARRE	René	Maître de recherche
DAVID	René	Maître de recherche
DEPORTES	Jacques	Maître de recherche
DRIOLE	Jean	Maître de recherche
GIGNOUX	Damien	Maître de recherche
GIVORD	Dominique	Maître de recherche
GUELIN	Pierre	Maître de recherche
HOPFINGER	Emil	Maître de recherche
JOUD	Jean-Charles	Maître de recherche
KAMARINOS	Georges	Maître de recherche
KLEITZ	Michel	Maître de recherche
LANDAU	Ioan-Dore	Maître de recherche
LASJAUNIAS	Jean-Claude	Maître de recherche
MERMET	Jean	Maître de recherche
MUNIER	Jacques	Maître de recherche
PIAU	Monique	Maître de recherche
PORTESEIL	Jean-Louis	Maître de recherche
THOLENCE	Jean-Louis	Maître de recherche
VERDILLON	André	Maître de recherche
SUERY	Michel	Maître de recherche

PAPIER RECUPERE ET RECYCLE

Personnalités habilitées à diriger des travaux de recherche
(Décision du Conseil Scientifique)

ALLIBERT	Colette
BERNARD	Claude
BONNET	Roland
CAILLET	Marcel
CHATILLON	Catherine
CHATILLON	Christian
COULON	Michel

E.N.S.E.E.G.

DIARD	Jean Paul
EUSTATHOPOULOS	Nicolas
FOSTER	Panayotis
GALERIE	Alain
HAMMOU	Abdelkader
MALMEJAC	Yves (CENG)
MARTIN GARIN	Régina
NGUYEN TRUONG	Bernadette
RAVAINE	Denis
SAINFORT	(CENG)
SARRAZIN	Pierre
SIMON	Jean Paul
TOUZAIN	Philippe
URBAIN	Georges (Laboratoire des ultra-réfractaires ODEILLO).

E.N.S.E.R.G.

BARIBAUD	Michel
BOREL	Joseph
CHOVET	Alain
CHEHIKIAN	Alain
DOLMAZON	Jean Marc
HERAULT	Jeanny
MONLLOR	Christian

E.N.S.I.E.G.

BORNARD	Guy
DESCHIZEAUX	Pierre
GLANGEAUD	François
KOFMAN	Walter
LEJEUNE	Gérard
MAZUER	Jean
PERARD	Jacques
REINISCH	Raymond

E.N.S.H.G.

ALEMANY	Antoine
BOIS	Daniel
DARVE	Félix
MICHEL	Jean Marie
OBLED	Charles
ROWE	Alain
VAUCLIN	Michel
WACK	Bernard

E.N.S.I.M.A.G.

BERT	Didier
CALMET	Jacques
COURTIN	Jacques
COURTOIS	Bernard
DELLA DORA	Jean
FONLUPT	Jean
SIFAKIS	Joseph

U.E.R.M.C.P.P.

CHARUEL	Robert

C.E.N.G.

CADET	Jean
COEURE	Philippe (LETI)
DELHAYE	Jean Marc (STT)
DUPUY	Michel (LETI)
JOUVE	Hubert (LETI)
NICOLAU	Yvan (LETI)
NIFENECKER	Hervé
PERROUD	Paul
PEUZIN	Jean Claude (LETI)
TAIEB	Maurice
VINCENDON	Marc

Laboratoires extérieurs :

C.N.E.T.

DEMOULIN	Eric
DEVINE	R.A.B.
GERBER	Roland
MERCKEL	Gérard
PAULEAU	Yves

I.N.S.A. Lyon

GAUBERT	C.

****

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur : **Monsieur M. MERMET**
Directeur des Etudes et de la formation : **Monsieur J. LEVASSEUR**
Directeur des recherches : **Monsieur J. LEVY**
Secrétaire Général : **Mademoiselle M. CLERGUE**

Professeurs de 1ère Catégorie

COINDE	Alexandre	Gestion
GOUX	Claude	Métallurgie
LEVY	Jacques	Métallurgie
LOWYS	Jean-Pierre	Physique
MATHON	Albert	Gestion
RIEU	Jean	Mécanique - Résistance des matériaux
SOUSTELLE	Michel	Chimie
FORMERY	Philippe	Mathématiques Appliquées

Professeurs de 2ème catégorie

HABIB	Michel	Informatique
PERRIN	Michel	Géologie
VERCHERY	Georges	Matériaux
TOUCHARD	Bernard	Physique Industrielle

Directeur de recherche

LESBATS	Pierre	Métallurgie

Maîtres de recherche

BISCONDI	Michel	Métallurgie
DAVOINE	Philippe	Géologie
FOURDEUX	Angeline	Métallurgie
KOBYLANSKI	André	Métallurgie
LALAUZE	René	Chimie
LANCELOT	Francis	Chimie
LE COZE	Jean	Métallurgie
THEVENOT	François	Chimie
TRAN MINH	Canh	Chimie

Personnalités habilitées à diriger des travaux de recherche

DRIVER	Julian	Métallurgie
GUILHOT	Bernard	Chimie
THOMAS	Gérard	Chimie

Professeur à l'UER de Sciences de Saint-Etienne

VERGNAUD	Jean-Maurice	Chimie des Matériaux & chimie industrielle

****

"Les Pessimistes ont généralement raison, mais ils ne font rien. Les optimistes ont tort la plupart du temps. Seulement eux, ils font quelque chose"

René Dubos : "Courtisons la Terre"

Je remercie G. Veillon, Directeur de l'INPG, de m'avoir fait l'honneur de présider ce jury.

C. Delobel a étudié en détail ce travail. Je le remercie pour le soin qu'il a mis à lire ce mémoire. Ses conseils et critiques ont largement contribué à l'améliorer. Son objectivité m'a été précieuse. Je lui exprime ma profonde reconnaissance.

Je remercie A. Pirotte et J. Rohmer dont les rapports très constructifs m'ont permis d'améliorer le contenu et la mise en forme définitive du mémoire.

Je remercie M. Adiba très amicalement de la confiance qu'il m'a témoignée en participant au jury de cette thèse.

B. Lorho me presse depuis des années à présenter une thèse. Je lui en suis très reconnaissant et je le remercie de participer à ce jury.

Sans F. Bancilhon également, cette thèse n'aurait pas vu le jour. J'exprime ma profonde gratitude à un ami avec qui j'ai travaillé pendant cinq ans. Son dynamisme communicatif a souvent eu raison de mon pessimisme viscéral. J'ai parfois apprécié son sens de la formule et de l'humour-vérité.

C. Decalf ne manque pas d'humour non plus. C'est elle qui s'est occupée de la réalisation matérielle impeccable de ce mémoire. Je profite de cette occasion pour lui dire sincèrement mon amitié.

P. Richard, mon compagnon de travail depuis deux ans m'a soutenu par sa patience et sa gentillesse. Je suis heureux de lui exprimer mon amitié sincère.

Je tiens à remercier A. Guebo qui m'a toujours donné, de bonne grâce, des renseignements sur le matériel des modules d'échange HACKERS.

Je tiens également à remercier mon frère P.C. Scholl qui m'a épargné nombre de démarches administratives à l'Université de Grenoble.

Enfin j'exprime ma reconnaissance à tous mes collègues du projet VERSO ainsi qu'aux membres de l'équipe SCD de l'INRIA et de la société HACKERS.

AVANT - PROPOS

Le travail présenté ici est intimement lié à la réalisation dans le cadre du projet VERSO de l'INRIA d'un prototype de Machine Bases de Données.

L'étude a été démarrée il y a cinq ans par F. Bancilhon et l'auteur. Je tiens à rendre hommage à F. Bancilhon, l'âme de cette équipe VERSO, sans qui le prototype n'aurait pas vu le jour.

La réalisation de VERSO a eu pour retombées un certain nombre d'études plus théoriques, notamment celle des V-relations. Celles-ci introduites par F. Bancilhon ont d'abord été étudiées avec P. Richard et A. Verroust. C'est cette formalisation qui est utilisée dans cette thèse. Une étude plus poussée a depuis été entreprise par S. Abiteboul et N. Bidoit et a abouti notamment à la formalisation d'une Algèbre V-relationnelle.

Le travail décrit ici s'appuie sur la réalisation de la couche physique de VERSO, c'est-à-dire le filtrage des données. Il en existe deux implantations :

La première, matérielle a été réalisée par D. Tusera et D. Fortin (projet INRIA "SCD"). Cette équipe avait déjà réalisé le processeur de filtrage TREFLE limité à la consultation de données. Cette expérience leur a permis de réaliser un processeur aux fonctionnalités plus complètes implanté sous forme de 7 cartes et en cours de microprogrammation. S. Gamerman a évalué cette architecture.

L'autre implantation, logicielle a été réalisée par P. Richard et l'auteur sur un module d'échange disque utilisant un processeur 8086 et commercialisé par HACKERS SARL (gérant D. Lefort).

Les personnes suivantes ont participé à la conception et la réalisation des couches de plus haut niveau de VERSO :

S. Abiteboul, F. Bancilhon, N. Bidoit, C. Cabart, S. Gamerman, J.M. Laubin, T. Mostardi, D. Plateau, J. Samuel et A. Verroust.

TABLE DES MATIERES

1. - INTRODUCTION 1

1.1. La Charrue avant les boeufs 3
1.2. Filtrage par automate fini 5
1.3. VERSO : MBD ou SGBD logiciel 8

2. - FILTRAGE DES DONNEES PAR AUTOMATE FINI 11

2.1. Disque Intelligent 13
2.2. Filtrage par automate fini 14
2.3. Structure de fichiers 16
2.4. Format de compaction 18
2.5. Intérêt d'une telle structure 26
2.6. Restriction/Projection 26
2.7. Structure du filtre 27
2.8. Complexité du filtrage 31
2.9. Requêtes traitées en une seule passe 31
2.10. Une CNS pour qu'un requête soit traitée en une passe 39
2.11. Conclusion 41

3. - VERSO (Filtrage Matériel) 43

3.1. Introduction 45
3.2. Les V-relations 46
3.3. VERSO : Le logiciel 57
3.4. Architecture matérielle de VERSO 66

4. - ARCHITECTURES DE MBD : Machines à Filtre 75

4.1. Architectures MIMD vs Disques Intelligents 78
4.2. Disques Intelligents : Machines à filtre 81
4.3. Architectures multiprocesseurs faiblement couplées 99
4.4. Les Architectures du futur rapproché 102

5. - Filtrage Logiciel 111

5.1. Architecture 114
5.2. La Sélection/Projection 118
5.3. Principe du Filtrage 119
5.4. Format des Données 119
5.5. Matrice de transition 120
5.6. Automate de comparaison de chaînes de caractères 121
5.7. Fonctions 127
5.8. Algorithme de filtrage 128
5.9. Taille des Automates. Comparaison avec le Filtrage matériel 134
5.10. Conclusion 143

6. - COMPARAISON ENTRE FILTRAGES LOGICIEL ET MATERIEL 145

6.1. Estimation de temps de cycle en Filtrage logiciel et de la vitesse relative du filtrage par rapport aux E/S. 147
6.2. Partage de la mémoire entre le processeur et le disque 157
6.3. Algorithme pour la Sélection/Projection 163
6.4. Evaluation du nombre de tampons Source et Cible 166
6.5. Temps de réponse (Filtrage Logiciel) 168
6.6. Conclusion sur le nombre de tampons 175
6.7. Comparaison des performances des 3 architectures pour la Sélection/Projection 176
6.8. Conclusion 183

7. - CONCLUSION 187

REFERENCES BIBLIOGRAPHIQUES 195

ANNEXE 201

LEXIQUE 203

CHAPITRE 1

INTRODUCTION

1. - INTRODUCTION

1.1. La Charrue avant les boeufs

Deux démarches initalement disjointes sont en train de converger dans le domaine des Bases de Données.

D'une part, les premiers Systèmes de Gestion de Bases de Données (SGBD) relationnels voient le jour à partir de 1975. Ils correspondent à une demande de meilleure qualité de la part des utilisateurs.

Parallèlement, on s'interroge sur la complexité et la médiocrité des performances des SGBD qui doivent faire face à une demande accrue des besoins usagers en matière de traitement d'information (un plus grand volume de données et de meilleures performances). Et si ces médiocres performances étaient dûes au fait que les SGBD sont implantés sur des architectures conventionnelles ? Les premières idées d'architectures adaptées au traitement des données remontent à 1970. Slotnick propose déjà de décentraliser des fonctionnalités du processeur central vers le périphérique [49].

On assiste alors pendant une dizaine d'années à deux phénomènes : d'une part, les systèmes relationnels se développent, et avec le développement de la technologie et de la micro-informatique des constructeurs commencent à offrir des produits commerciaux qui ne nécessitent plus une grosse configuration. Toutefois même si un certain nombre de principes pour une "bonne" conception ont pu être dégagés, aucune méthodologie concluante ne s'est imposée qui permettrait d'évaluer et d'améliorer les performances des systèmes relationnels. D'autre part, une activité de recherche intense est menée dans la direction des "Machines" Bases de Données (MBD) depuis 1975 et on commence à voir sur le marché quelques produits commerciaux utilisant des processeurs matériels spécialisés.

Qui l'emportera dans cette lutte entre deux types de produits : logiciels relationnels portés sur petits systèmes sous UNIX notamment ou produits commerciaux matériels ?

En fait, la frontière entre les deux lignes de produits est actuellement ténue.

Les produits logiciels sont développés plus rapidement, n'ont pas toujours toutes les fonctionnalités requises et n'ont pas les performances attendues.

Les produits dits matériels sont simples de conception (comparés aux prototypes développés dans les laboratoires de recherche), ont un seul processeur "ad hoc" destiné à optimiser la fonction d'accès aux données. Dans certains produits, le processeur est optionnel : parfois il est annoncé mais n'est pas disponible commercialement.

Pour schématiser le débat entre les deux approches, on peut considérer que les concepteurs se rangent dans deux catégories :

- ceux qui croient que la matérialisation d'une ou plusieurs fonctions SGBD augmentera les performances d'un ordre de grandeur sans augmenter la complexité du système,

- ceux qui croient que toute architecture SGBD même spécialisée doit être bâtie avec des éléments banalisés ("off the shelf").

Ces derniers pensent

(1) que les architectures à base de processeurs "ad hoc" sont beaucoup trop complexes pour être viables,

(2) que le gain obtenu par un processeur matériel "ad hoc" décroit très vite avec le temps : pendant qu'on développe et microprogramme un tel processeur les progrès de la

technologie sont tels qu'un nouveau processeur banalisé aura des performances multipliées par un facteur de 2, voire de 5 sans le même effort de développement,

(3) enfin et surtout qu'il ne sert de rien d'optimiser telle ou telle fonction d'un SGBD par une implantation matérielle quand aucune méthodologie ne permet actuellement de décider où sont les goulets d'étranglement.

Ce dernier argument nous semble important : "On a mis la charrue avant les boeufs". Les efforts de recherche sanctionnés par une première génération de maquettes de MBD ont légèrement précédé la maîtrise des SGBD relationnels.

Une grande partie du travail présenté ici est au coeur de cette problématique.

1.2. Filtrage par automate fini

Par filtrage des données, on entend habituellement, la sélection d'un sous-ensemble de données, satisfaisant un critère précis. De manière plus générale on peut envisager que le résultat du filtrage est un bloc de données de taille égale ou supérieure à celle du bloc à filtrer. On peut en effet vouloir modifier les données sources, voire insérer de nouvelles données.

On appellera filtre un processeur matériel ou logiciel qui inspecte séquentiellement un bloc de données, par exemple le contenu d'une piste du disque chargé dans un tampon source et produit un bloc de données résultat dans un tampon cible. On élimine de ce fait les algorithmes qui d'une manière ou d'une autre font de l'accès direct à l'intérieur du bloc, c'est-à-dire se servent d'une structure de données ou d'un adressage à l'intérieur du bloc : les données à filtrer sont balayées caractère par caractère. Cependant l'algorithme peut nécessiter plusieurs passes, c'est-à-dire des retours en arrière sur la chaîne de caractères source.

S'il s'agit d'une sélection d'un sous-ensemble de données à garder en fonction d'un critère, chaque fois qu'un caractère est lu dans le tampon source on doit décider si on l'écrit dans le tampon cible. S'il s'agit d'une insertion ou d'une modification contenue dans un deuxième tampon source, on doit décider quand écrire dans le tampon cible le contenu de ce deuxième tampon source. Pour simplifier nous nous placerons seulement dans le premier cas, celui d'une sélection.

Un filtrage par automate fini (AF) est un mécanisme de reconnaissance des données lues qui à chaque couple (état courant de l'automate, caractère lu dans le tampon source) associe un état suivant (transition) et un certain nombre d'actions élémentaires permettant de copier les "bonnes" données dans le tampon cible. Un tel mécanisme sera étudié en détail dans le chapitre 2 pour le traitement des requêtes relationnelles. En simplifiant, on peut l'illustrer de la façon suivante.

Soit à construire un automate permettant d'éliminer tous les caractères "blanc" sauf un dans une chaîne de "blancs". L'automate qui permet ce traitement simple peut être représenté par son diagramme de transitions. Celui-ci est un graphe orienté où un arc du noeud (i) vers le noeud (j) étiqueté par le caractère c, indique que la lecture du caractère c lorsque l'automate est dans l'état i force l'état à j. La deuxième étiquette (1 ou 2) sur un arc indique quelle action élémentaire doit être effectuée lors de la transition : 1 on garde le caractère ; 2 on l'efface

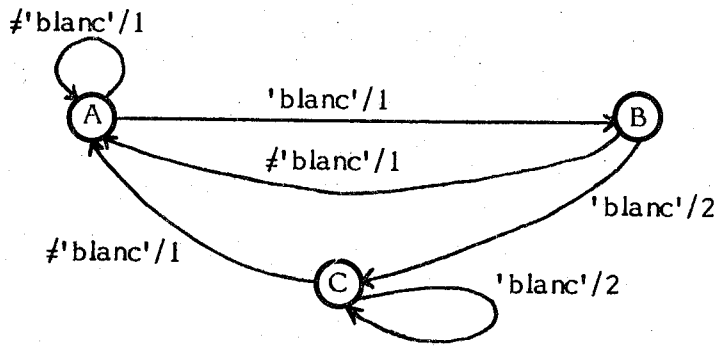


Figure 1.1. Automate d'élimination de blancs

Le mécanisme suivant (Fig. 1.2) permet le filtrage par automate fini :

Les caractères lus en séquences dans le tampon source sont écrits dans le tampon cible et analysés par l'automate (i.e. ils sont comparés au caractère 'blanc'). En fonction du résultat, l'automate change d'état. La seule action effectuée lors d'une transition consiste à incrémenter ou non l'adresse dans le tampon cible où écrire le caractère suivant. Si le caractère est à garder l'adresse est incrémentée. Sinon, le caractère suivant écrasera le caractère précédemment écrit.

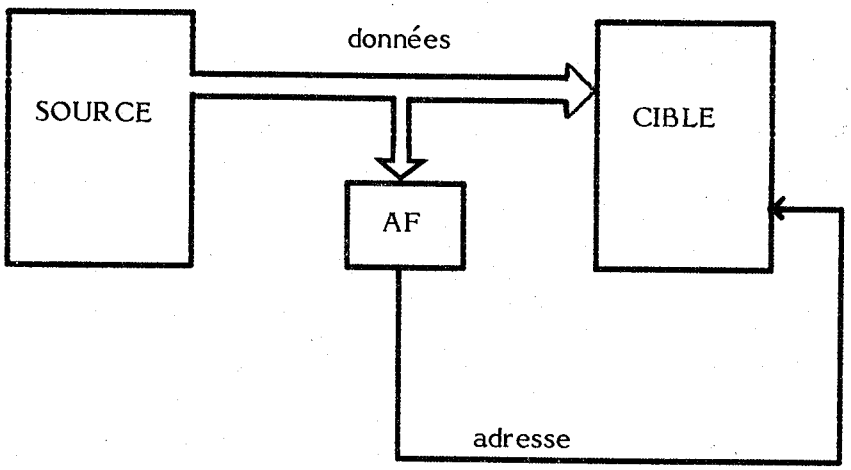


Figure 1.2. Mécanisme de filtrage par AF

Dans sa version la plus simple, le filtre est donc un AF ayant un jeu de fonctions de sortie correspondant aux actions à prendre lors des transitions de l'automate. Ce mécanisme est utilisé dans nombre d'applications parfois au niveau du matériel (automates câblés).

Par opposition, on dira que le filtre est programmable, si étant donné un jeu de fonctions de sortie (instructions élémentaires), il accepte tout automate fini. Le filtre est alors une machine dont

les données (code exécutable) sont un ensemble de transitions et de fonctions de sortie. C'est ce type de filtrage qui sera utilisé dans le chapitre 2 pour le traitement des requêtes relationnelles : un compilateur prend comme données d'entrées une requête et une structure de fichier à filtrer et fournit en sortie un code exécutable par le filtre.

1.3. VERSO : MBD ou SGBD logiciel utilisant le Filtrage par Automate

Le premier objectif de cette thèse est de mettre en valeur la notion de filtrage des données par automate fini décentralisé près de la mémoire de masse comme moyen efficace de traiter les opérations relationnelles élémentaires c'est-à-dire la sélection, la projection et la jointure (Chapitre 2) :

en décentralisant le filtrage des données vers un processeur près de la mémoire de masse, on soulage l'Unité Centrale qui exécute toutes les autres fonctions SGBD et les ressources communes (bus et mémoire à accès direct) où ne transitent plus que les données utiles.

Le Chapitre 2 étudie un mécanisme de filtrage général pour traiter efficacement les opérations relationnelles élémentaires. Il rassemble et complète deux études publiées dans [7, 11]. Si notre contribution personnelle dans ce chapitre est la conception et la caractérisation du mécanisme de filtrage, il s'appuiera sur un travail fait en collaboration avec F. Bancilhon et P. Richard en ce qui concerne la caractérisation des objects à filtrer (V-relations) et des requêtes sur ces objets. Les V-relations introduites d'abord par F. Bancilhon [9] sont une extension de la notion de relations. Une étude plus poussée a été depuis entreprise par S. Abiteboul et N. Bidoit [1, 13] qui a abouti à la formalisation d'une Algèbre V-relationnelle. Une V-relation est représentée par un fichier de structure compacte qui généralise la représentation classique des relations en tables. Cette structure d'abord introduite par

d'abord introduite par F. Bancilhon et l'auteur [SIGMOD] est caractérisée par un format de compaction. P. Richard a établi le lien entre un ensemble de contraintes d'intégrité respectées par la V-relation et le fichier le plus compct pouvant représenter cette V-relation [VLDB].

Le deuxième objectif de cette thèse est de tirer les enseignements de la conception et la réalisation d'une maquette de système relationnel appelé VERSO (pour "Back-End") à la lueur de la problématique ci-dessus.

VERSO qui utilise un filtrage par automate est développé autour d'une configuration SM 90 en Pascal sous UNIX. Le système de gestion de fichiers UNIX est court-circuité et remplacé par un Filtre implanté "près" de la mémoire de masse (disques à tête mobile). Ce Filtre se charge de l'accès aux données et du traitement des opérations relationnelles élémentaires.

VERSO a été initialement conçu comme une MBD où le filtre est implanté sous forme de processeurs matériels spécialisés (7 cartes). Ce filtre matériel réalisé par l'équipe SCD de l'INRIA (D. Tusera et D. Fortin) et connecté au bus de la SM est en cours de microprogrammation (D. Tusera). La description de VERSO en partant des couches de haut niveau jusqu'au filtre matériel, est faite dans le Chapitre 3.

Le chapitre 4 présente les différentes architectures de MBD que l'on trouve depuis 10 ans dans les laboratoires de recherche ou sur le marché. Plutôt que d'être un nouvel état de l'art en la matière (il en existe déjà plusieurs), cette présentation permet de mieux resituer VERSO par rapport aux différents choix architecturaux.

Parallèlement à l'implantation matérielle du filtrage, nous avons conçu une implantation logicielle du filtrage qui a été développée sur un Module d'échange standard utilisant un processeur

processeur INTEL 8086 (Chapitre 5) par P. Richard et l'auteur. La version de l'algorithme de filtrage pour l'opération de Sélection/Projection effectivement implantée est due à C. Cabart [15]. Elle correspond à une optimisation importante et s'écarte très sensiblement de la conception initiale présentée dans ce chapitre. Cette implantation fait de VERSO un SGBD relationnel entièrement logiciel meilleur marché et plus facilement portable que la version matérielle (le filtre matériel étant à l'état de maquette, sa duplication coûte cher en argent et en temps de développement).

Les deux architectures sont comparées dans le Chapitre 6 et une des conclusions de l'étude est que bien que le processeur de filtrage matériel soit considérablement plus rapide que le processeur de filtrage logiciel, sa puissance est mal utilisée et il est préférable dans l'application Bases de Données d'implanter le filtrage sur un microprocesseur du commerce.

CHAPITRE 2

FILTRAGE DES DONNEES PAR AUTOMATE FINI

2. - FILTRAGE DES DONNEES PAR AUTOMATE FINI(*)

Le concept de filtrage dans le traitement de l'information non numérique en général et dans les Bases de Données en particulier est synonyme de traitement séquentiel et est lié à la décentralisation de la sélection des données de l'unité centrale (UC) vers la mémoire de masse :

On optimise une fonction primordiale des SGBD : l'accès aux données satisfaisant un critère de sélection donné. La mémoire de masse (Disque) étant un support séquentiel, on "filtre" au fur et à mesure que l'on lit les données du Disque, les "bonnes" données des mauvaises. Qu'est-ce qu'on a gagné ?

- Parallélisme : l'Unité Centrale est déchargée d'un certain travail ;

- Filtrage au vol : on profite du temps de transfert des données pour les filtrer ;

- Occupation ressources : Le volume des données gardées après filtrage étant inférieur au volume avant filtrage, on décharge le bus d'entrée/sortie ainsi que la mémoire centrale.

2.1. Disque Intelligent

Slotnick [49] est le premier à avoir l'idée d'attacher à une piste de Disque de la logique cablée permettant la comparaison de chaines de caractères et l'évaluation booléenne. Il est suivi dans cette direction par les premiers projets de Machines tels que CASSM [17] ou RAP [41] qui utilisent des dispositifs dits "cellular logic devices".

(*) Par abus de langage mais dans un souci de brièveté, on écrira Automate fini pour Automate d'Etats Finis dans tout ce qui suit.

Depuis, les architectures utilisant un "Disque intelligent" ont considérablement évolué. La logique de filtrage n'est plus implantée "sur" le disque mais au moyen d'un processeur standard ou spécialisé qui traite les données venant du disque au vol ou non. Les fonctionnalités du filtre lui-même se sont considérablement étoffées. Toute cette évolution est décrite dans le Chapitre 4 qui reprend les principales architectures à filtrage des dernières années.

2.2. Filtrage par automate fini

Certains produits comme SARI [32] utilisent déjà des automates câblés pour la comparaison d'un champ à une clé donnée. TREFLE [22] est la première machine à automate fini programmable. Initialement conçue à des fins documentaires, elle est bien adaptée à la reconnaissance de fichiers de structure régulière et à la consultation automatique de tels fichiers. L'idée originale d'un tel filtrage est d'associer à une requête de consultation donnée et à une structure de fichiers donnée un automate fini. Celui-ci est un programme destiné à être exécuté par une machine spécialisée - le filtre - qui balaie séquentiellement le fichier à filtrer. Une instruction de ce programme correspond à un couple état de l'automate, caractère lu dans le fichier et a pour résultat d'exécuter un certain nombre de fonctions de sortie et de brancher à un état suivant(*).

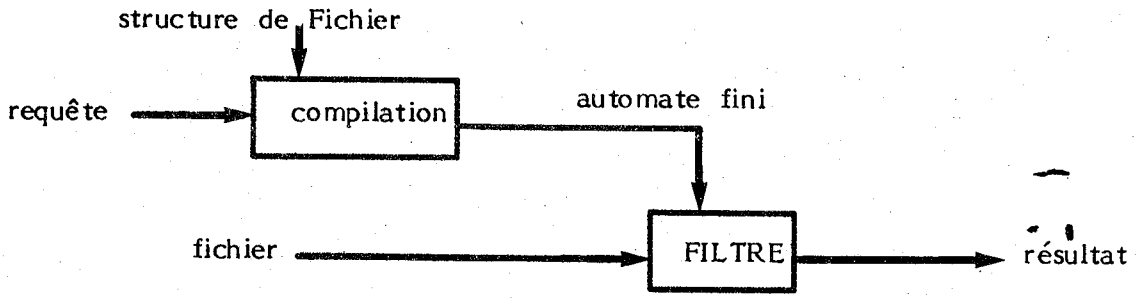


Figure 2.1. Filtre à Automate fini programmable

(*) On utilise tout au long de la thèse, le terme d'automate pour représenter le filtre, de façon quelque peu impropre. Un automate est un mécanisme de reconnaissance. Comme on lui ajoute un mécanisme de traduction de la chaîne d'entrée, il aurait été plus approprié de parler de transducteur.

Cette idée a été reprise dans SCHUSS [27] dont le processeur est microprogrammable pour s'adapter à différentes applications et dans VERSO [5] à des fins Bases de Données pour optimiser les fonctions d'accès et de manipulations de données. On trouvera également dans [44] une étude détaillée du filtrage par automate.

Le but de ce chapitre est de montrer que le filtrage par automate fini est bien adapté à l'implantation des opérations de l'algèbre relationnelle.(*)

On définira d'abord une structure de fichiers (et on essaiera de justifier le choix d'une telle structure) ensuite une architecture de machine minimale pour implanter le filtrage. On montrera ensuite par construction qu'à une requête de Sélection/Projection et un fichier satisfaisant la structure de fichiers correspond un automate fini qui, exécuté par le Filtre, permet le filtrage en une seule passe sur le fichier (en temps linéaire avec la longueur du fichier).

On montrera ensuite que toutes les requêtes de Sélection/Projection ne sont pas satisfaites en une seule passe. On exhibera une classe de fichiers dits "Simple-Droits" pour lesquels toute requête est satisfaite en une seule passe. Pour les fichiers non "simple-droits", on donnera un algorithme permettant de décider si une requête peut être effectuée en une seule passe.

(*) Seule la Sélection/Projection sera étudiée en détail dans cette thèse. Pour plus de détails concernant les autres opérations, le lecteur est reporté à [5], [6] qui montrent comment implanter l'insertion, la suppression de n-uplets et les opérations binaires et à [1], [7] qui définissent précisément quelles sont les opérations à implanter. Le traitement de ces opérations est analogue à celui de la Sélection/Projection à quelques détails près qui seront indiqués quand nécessaires. Retenons pour l'instant que pour les opérations binaires (e.g. la fusion), le filtre balaie 2 fichiers source et qu'une condition nécessaire pour un tel filtrage est que les 2 fichiers source soient triés. Cependant si ceux-ci ne sont pas triés, on peut utiliser le filtrage pour effectuer certaines semi-jointures en plusieurs passes (voir par exemple [44]).

2.3. Structure de Fichiers

Supposons pour l'instant que les données constituent au niveau logique un ensemble de relations de la forme $R(A_1:D(A_1), A_2:D(A_2),...,A_n:D(A_n))$ où R est le nom de la relation, A_i est un nom d'attribut $\in U$, $D(A_i)$ le domaine associé à l'attribut A_i. Une instance de relation est un ensemble de n-uplets de la forme $(x_1,x_2,...,x_n)$ tels que $x_i \in D(A_i)$.

Au lieu de représenter sur le disque les relations par des tables, on les compacte de façon hiérarchique. Par exemple, si on considère la relation

R(COURS,ETUDIANT,NOTE,PROFESSEUR)

où pour abréger on a supprimé la description des domaines, on peut choisir la décomposition hiérarchique suivante :

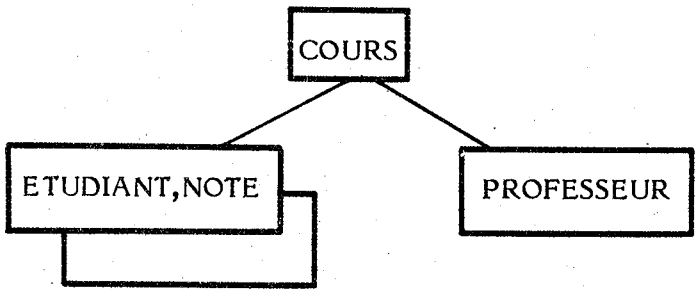


Figure 2.2. Format hiérarchique

D'après ce format, le fichier sera composé d'un ensemble d'enregistrements, chacun étant la concaténation d'un COURS suivi d'une liste de couples (ETUDIANT,NOTE), suivi d'un PROFESSEUR.

Cette représentation compactée peut être décrite par l'expression régulière :

(COURS(ETUDIANT,NOTE)*PROFESSEUR)*

ou en abrégeant les noms d'attribut :

(C(EN)*P)*

Remarques

1) On utilise la même représentation compactée de l'information que dans le modèle hiérarchique, mais toute ressemblance avec le modèle hiérarchique s'arrête là.

2) Il y a plusieurs façons de compacter la même relation. Par exemple on aurait pu utiliser (P(C(EN)*)*)* ou (CP(EN)*)*.

3) Une relation est représentée par un fichier séquentiel. Les valeurs d'attributs peuvent être soit de longueur fixe, elles sont alors représentées par un identificateur d'attribut (ia), suivi de la valeur, soit de longueur variable, elles sont alors représentées par l'ia suivi de la valeur suivi d'un identificateur de fin de champ (if).

On décrit maintenant un peu plus formellement cette représentation compacte en définissant un Format de Compaction. Plus précisément, on s'intéresse dans le paragraphe suivant à la représentation la plus compactée possible d'une relation. La présence d'une '*' dans un FC tel que (CP(EN)*)* indique qu'il est possible que le fichier soit vide, ou que dans un enregistrement, il n'y ait qu'un cours et un professeur, la liste d'étudiants-note étant vide. Dans le paragraphe qui suit, nous supposons par contre qu'il ne peut y avoir de telles "valeurs nulles". Les '*' dans le FC sont donc remplacées par des '+'.

2.4. Format de Compaction [8]

Dans ce paragraphe, on suppose qu'un fichier séquentiel sans identificateur d'attribut ni identificateur de fin de champ représente une relation.

Définition 1.

Un fichier F sur l'ensemble d'attributs U est une chaine sur D^+ où D^+ désigne la fermeture stricte de $D = \bigcup_{A \in U} D(A)$.

Par exemple si $U = \{C,E,N\}$ alors

F_o = MATH TOTO 15 LULU 10 PHYSIQUE TOTO 12 est un fichier sur U.

Définition 2.

Un format de compaction (FC) sur U est défini récursivement comme suit :

i) A et A^+ sont des FC sur A, où A^+ désigne l'ensemble des chaînes formées du caractère A de taille quelconque ≥ 1.

ii) Si C est un FC sur $X \subset U$ alors $(C)^+$ est un FC sur X.

iii) Si C_1 et C_2 sont des FC sur X_1 et $X_2 \subset U$ et $X_1 \cap X_2 = \emptyset$ alors $C_1.C_2$ est un FC sur $X_1 \cup X_2$, où l'opérateur '.' désigne la concaténation.

Par exemple $(CEN)^+$ et $(C(EN)^+)^+$ sont des FC sur $U = \{C,E,N\}$. Le premier correspond à la représentation classique en table de la relation R(C,E,N). Avec le 2e FC, la relation est

compactée : dans un enregistrement on associe à un cours sa liste d'étudiants et leur note.

Une telle définition produit un sous-ensemble particulier des expressions régulières.

Définition 3.

Le langage $\mathscr{L}(C)$ associé à C est défini par :

i) $\mathscr{L}(A) = D(A) \quad \forall A \in U$

ii) $\mathscr{L}(C_1.C_2) = \mathscr{L}(C_1).\mathscr{L}(C_2)$ (*)

iii) $\mathscr{L}(C^+) = (\mathscr{L}(C))^+$

On dira que le fichier F satisfait le FC C si $F \in \mathscr{L}(C)$.
Par exemple,

F_1 : MATHS TOTO 15 MATHS LULU 10 PHYSIQUE TOTO 10

satisfait C_1 : $(CEN)^+$,
tandis que

F_2 : MATHS TOTO 15 LULU 10 PHYSIQUE TOTO 10

satisfait C_2 : $(C(EN)^+)^+$

Définition 4.

Enregistrement - Soit C le format d'un fichier E. Si C peut se mettre sous la forme C : $(E)^+$, le fichier est constitué d'une séquence de fichiers de format E appelés enregistrements de F. Sinon F est constitué d'un seul enregistrement de format C.

(*) Par la suite le '.' pour désigner la concaténation sera omis. La concaténation de A et B s'écrira AB.

Définition 5.

Fichier caractéristique ou Fichier plat - La représentation classique en "table" d'une valeur de relation, c'est-à-dire en une séquence de n-uplets de format $(A_{i1} \ldots A_{in})^+$ est appelée fichier caractéristique ou fichier plat, où $\{A_{i1} \ldots A_{in}\}$ est une permutation des n attributs $A_1,\ldots,A_n$ de la relation.

Compaction d'un Fichier - Etant donnée une relation, on s'intéresse à la représentation la plus compacte. Dans l'exemple ci-dessus le fichier F2 est plus compact que le fichier plat F1. Remarquons que le fichier F1 qui satisfait $(CEN)^+$ satisfait également $(C(EN^+)^+)^+$, $(CE^+N^+)^+$ et $(C(EN)^+)^+$. Quant au fichier F2 il satisfait $(C(EN^+)^+)^+$ et $(C(EN)^+)^+$. Indubitablement, c'est le format $(CE^+N^+)^+$ qui devrait compacter au maximum l'information. Mais observons que le fichier

F3 : MATH TOTO LULU I5 10 PHYSIQUE TOTO 10

qui satisfait $(CE^+N^+)^+$ ne représente pas la même information que les fichiers F1 et F2. On a - dans F3 - perdu l'information qui attache à un étudiant la note qu'il a eu dans ce cours.

Etant donnée une relation, après avoir choisi dans quel ordre représenter les attributs (cet ordre étant évidemment invariable d'un enregistrement à l'autre), le problème se pose du choix du format de compaction.

On peut toujours choisir une représentation en fichier plat peu compacte. Quels sont les autres formats possibles ? - Nous avons vu dans l'exemple ci-dessus que la contrainte "à un étudiant dans un cours est associée une note" interdit derechef le choix de $(CE^+N^+)^+$ comme format de compaction.

Parmi les formats possibles, existe-t-il un ordre partiel ou total nous assurant - étant donné un ensemble de contraintes sur la relation - que tel format est plus compact qu'un autre.

Le résultat qui suit nous permet de répondre partiellement à cette question : La connaissance d'un certain nombre de contraintes d'intégrité sur la relation permet de choisir un format garantissant la compaction maximale.

Fichier compacté maximal (FCM)

Pour définir un FCM, on utilise la représentation suivante d'un FC :

$$C : (A_1 \ldots A_p C_1 \ldots C_n)^+$$

où $A_i \in X \subseteq U$, $i \in [1,p]$ et $C_1,\ldots,C_n$, $n \geq 0$ sont des formats de compaction. C_i est dit sous-format de C. Pour abréger, on représente C par :

$$C : (XC_1 \ldots C_n)^+$$

X est appelée racine du format.

Soit maintenant un fichier F satisfaisant C. F est constitué d'un certain nombre q d'enregistrements, chaque enregistrement satisfaisant le format : $XC_1C_2\ldots C_n$. F est constitué d'un sous-fichier racine (obtenu en projetant F sur la racine X de C) et de $nq \geq 0$ sous-fichiers "queues" F_{ij}, $i \leq n$, $j \leq q$, (le sous-fichier F_{ij} satisfaisant le sous-format C_i).

Définition 6. [8]

F est dit compacté maximal par rapport à C si et seulement si :

(1) La projection de F sur la racine de C ne contient pas de dupliqué,

(2) Pour chaque enregistrement de F, chaque sous-fichier queue est compacté maximal.

Par exemple,

a) le fichier F_4 :

$$a_1b_1c_1c_2a_1b_2c_1c_2a_2b_1b_2c_1c_2$$

qui satisfait $(AB^+C^+)^+$ n'est pas compacté maximal car sa projection sur la racine $a_1a_1a_2$ contient des dupliqués.

b) le fichier F_5 :

$$a_1b_1b_2b_1c_1a_2b_1b_2c_1c_2$$

qui satisfait $(AB^+C^+)^+$ n'est pas compacté maximal car le sous-fichier $b_1b_2b_1$ contient un dupliqué

c) par contre le fichier F_6 :

$$a_1b_1b_2c_1c_2a_2b_1b_2c_1c_2$$

est compacté maximal.

Une définition plus formelle d'un FCM est donnée dans [8].

Contraintes d'intégrité associées à un FC

Etant donné un FC C sur U, on s'intéresse aux relations R(U) qui ont une représentation compactée maximale avec C.

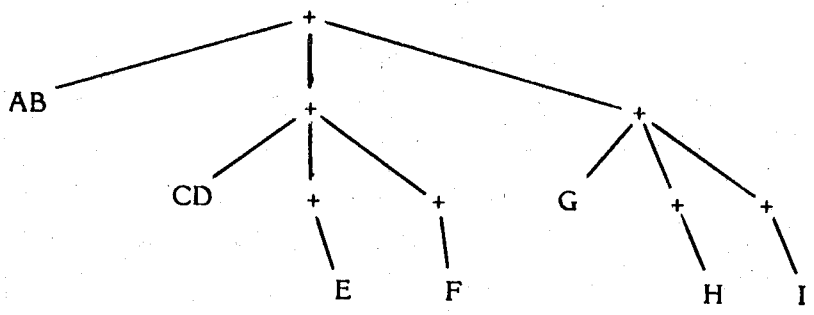
Afin de caractériser ces relations, on introduit un ensemble particulier de contraintes, appelé ensemble des contraintes d'intégrité associées à un format de compaction C, et dénoté CI(C). C'est en fait un ensemble particulier de dépendances multivaluées à savoir les dépendances hiérarchiques comme définies dans [18].

Pour définir informellement CI(C), on procède en deux étapes :

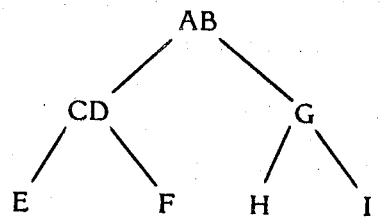
i) on associe à C son arbre syntaxique : Par exemple soit le FC

$$C : (AB(CDE^{+}F^{+})^{+}(GH^{+}I^{+})^{+})^{+}$$

à C on associe l'arbre suivant :



ii) CI(C) est obtenu de l'arbre syntaxique en enlevant les '+' :



De cette décomposition hiérarchique, on obtient l'ensemble suivant de dépendances multivaluées :

$$CI(C) : \begin{cases} AB \twoheadrightarrow CDEF/GHI \\ ABCD \twoheadrightarrow E/F \\ ABG \twoheadrightarrow H/I \end{cases}$$

Théorème :

Soit C un FC sur U. Soit CI(C) l'ensemble des contraintes d'intégrité associées à C. Une représentation de R(U) satisfaisant C est compactée maximale si et seulement si R(U) satisfait CI(C).

Démonstration : [8]

Le théorème précédent nous donne un moyen de choisir un FC à partir d'un ensemble d'attributs, en vue d'obtenir un FCM comme représentation de relations. On vérifie s'il existe une décomposition hiérarchique sur les relations qu'on désire stocker et on crée un FC C tel que l'ensemble des contraintes d'intégrité est CI(C). On peut alors compacter les relations sans danger puisque cette compaction est sans perte d'information.

Exemples :

i) Ainsi dans l'exemple précédent, la connaissance de l'ensemble des contraintes sur la relation :

(1) $AB \twoheadrightarrow CDEF/GHI$

(2) $ABCD \twoheadrightarrow E/F$

(3) $ABG \twoheadrightarrow H/I$

nous assure que la représentation la plus compacte se fera suivant le FC :

$$(AB(CDE^{+}F^{+})^{+}(GH^{+}I^{+})^{+})^{+}$$

sans perte d'information.

Si la contrainte (2) n'existait pas, avec une représentation suivant C, on perd l'information

$$ABCD \rightarrow EF$$

qui détermine la valeur du sous-nuplet EF, connaissant la valeur du sous-nuplet ABCD.

La représentation la plus compacte serait alors :

$$(AB(CDEF)^{+}(GH^{+}I^{+})^{+})^{+}$$

ii) Dans l'exemple de la relation R(Cours,Heure,Salle,Etudiant, Note) satisfaisant la contrainte (1) $C \twoheadrightarrow HS/EN$, le FC : $(C(HS)^{+}(EN)^{+})^{+}$ donne un fichier plus compact que le FC : $(C(HS(EN)^{+})^{+})^{+}$.

Si au lieu de la contrainte (1), on a les contraintes

$$\begin{cases} C \twoheadrightarrow HSEN \\ CHS \twoheadrightarrow EN \\ CEN \twoheadrightarrow HS \end{cases}$$

on ne peut choisir le FC $(C(HS)^{+}(EN)^{+})^{+}$.

Les représentations suivant les formats

$(C(HS(EN)^{+})^{+})^{+}$ ou $(C(EN(HS)^{+})^{+})^{+}$

sont autorisées et plus compactes que la représentation en fichier plat, mais rien ne nous dit laquelle parmi les deux est la plus compacte.

Enfin notons que si aucune contrainte n'existe, seul le fichier plat est autorisé sans perte d'information, ce qui correspond bien évidemment à la seule représentation d'une relation normalisée.

2.5. Intérêt d'une telle structure

Une telle représentation compactée des informations est intéressante à plusieurs titres. D'une part, elle économise de l'espace et ceci est primordial car nous verrons en dessous que le temps de filtrage est proportionnel à la taille du fichier à filtrer. Par ailleurs cette compaction est naturelle - nous avons déjà vu qu'elle correspond à l'existence d'un certain nombre de contraintes d'intégrité - et ceci sera plus clair dans le chapitre suivant lorsqu'on introduira la notion de V-relation : en particulier elle permet (i) la représentation de relations sous forme non normalisée sans les anomalies de mise à jours et (ii) les valeurs incomplètes.

2.6. Restriction/Projection

La Projection R(X), $X \subseteq U$ d'une relation R(U) est la table obtenue en supprimant de la table R(U) les colonnes de U-X, et en éliminant les n-uplets dupliqués.

Soit C une condition booléenne. La restriction R | C est constituée de tous les n-uplets de R satisfaisant C.

C est de la forme :

$$C = \cup C_i$$

avec

$$C_i = \cap C_{ij}$$

$$C_{ij} = A_k \langle comp \rangle \langle valeur \rangle$$

où $A_k \in U$, <comp> $\in \{=, \neq, >, \geq, <, \leq\}$ et <valeur> $\in D(A_k)$

Le modèle de données ainsi que l'opération à effectuer sur le fichier étant définies, il nous reste à définir une architecture de machine pour implanter le filtrage.

2.7. Structure du Filtre

Le filtre lit et traite un bloc de données (e.g. le contenu d'une piste du disque) chargé dans un tampon source.

La figure 2.3 décrit la structure du filtre :

On n'étudie pas dans ce chapitre l'environnement du filtre, c'est-à-dire comment le filtre reçoit des commandes, comment le tampon Source est chargé du Disque, etc..
Les chapitres 3 et 5 décrivent en détail deux implantations du filtre.

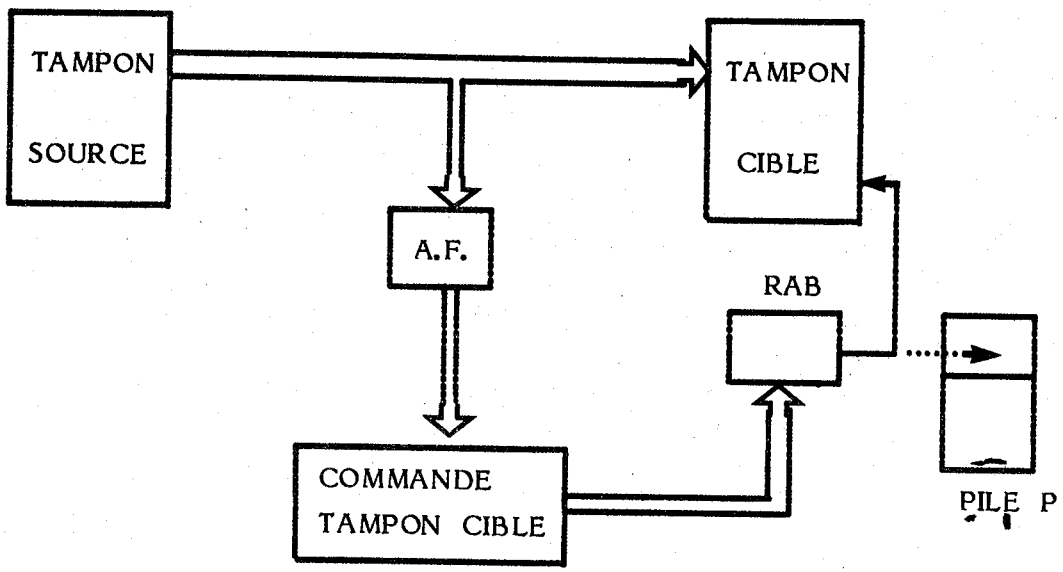


Figure 2.3. Structure du filtre

Le tampon Source est balayé séquentiellement. Chaque caractère lu, est écrit dans le tampon Cible à l'adresse contenue dans le Registre Adresse Buffer RAB, et en même temps ce caractère est analysé par l'automate fini (AF).

Les fonctions de sortie associées à un couple (état automate, caractère lu) consistent à agir sur l'adresse cible où on écrit le caractère lu (i.e. à modifier le contenu de RAB).

Nous avons déjà dit que le filtre est programmable : à chaque requête correspond un ensemble précis de transitions et de fonctions de sortie, ensemble qui constitue le code exécuté par le filtre.

L'AF fait à la fois l'analyse lexicale (reconnaissance de la valeur d'un attribut) et l'analyse syntaxique (reconnaissance de la structure du fichier).

L'analyse lexicale se fait grâce à un sous-automate de reconnaissance de chaine de caractères. Par souci de clarté nous ignorons cette phase et supposons que l'automate traite des mots de longueur un caractère dont il reconnait le type (identificateur d'attribut).

Les fonctions de sortie ou actions exécutées par l'automate sont les suivantes :

①	RAB ← RAB + 1	Le mot en séquence est transféré dans le tampon cible à l'adresse suivante.
②	EMPILER RAB dans P (MARQUER)	Le contenu du registre RAB est empilé dans la pile P : on mémorise l'adresse courante dans le tampon cible.

③	RAB ← DEPILER(P) (EFFACER)	Le registre RAB reçoit le sommet de pile et P est dépilée ; Le mot suivant ne sera pas écrit en séquence mais à la dernière adresse mémorisée : on écrase tous les mots écrits depuis la dernière mémorisation.
④	DEPILER(P) (GARDER)	P est dépilée : on efface la dernière marque, c'est-à-dire qu'on garde (valide) tous les mots écrits depuis la dernière adresse mémorisée.

Le mécanisme de filtrage est illustré par l'exemple suivant : Soit la relation R(COURS,HEURE,SALLE,ETUDIANT,NOTE, PROFESSEUR) représentée par le fichier ayant pour FC $(C(HS)^*(EN)^*P)^*$.
Soit la requête sur ce fichier :

"Trouver les classes et étudiants dont la note est B".

La figure 2.4 donne le diagramme de transition de l'automate correspondant à cette requête. Les arcs sont étiquetés par le mot lu et la (les) fonction(s) de sorties effectuées. Les noeuds représentent les états de l'automate. On n'a pas représenté les arcs correspondant à un échec ou une erreur (e.g. après un cours on ne trouve pas une heure mais une salle ; après un étudiant on ne trouve pas de note). On suppose le pointeur RAB initialisé à 0.

Lorsqu'on analyse un enregistrement de ce fichier, c'est seulement après avoir lu tous les couples (étudiant, note) qu'on saura si au moins un étudiant a eu la note B et dans ce cas qu'on pourra garder le champ "COURS".

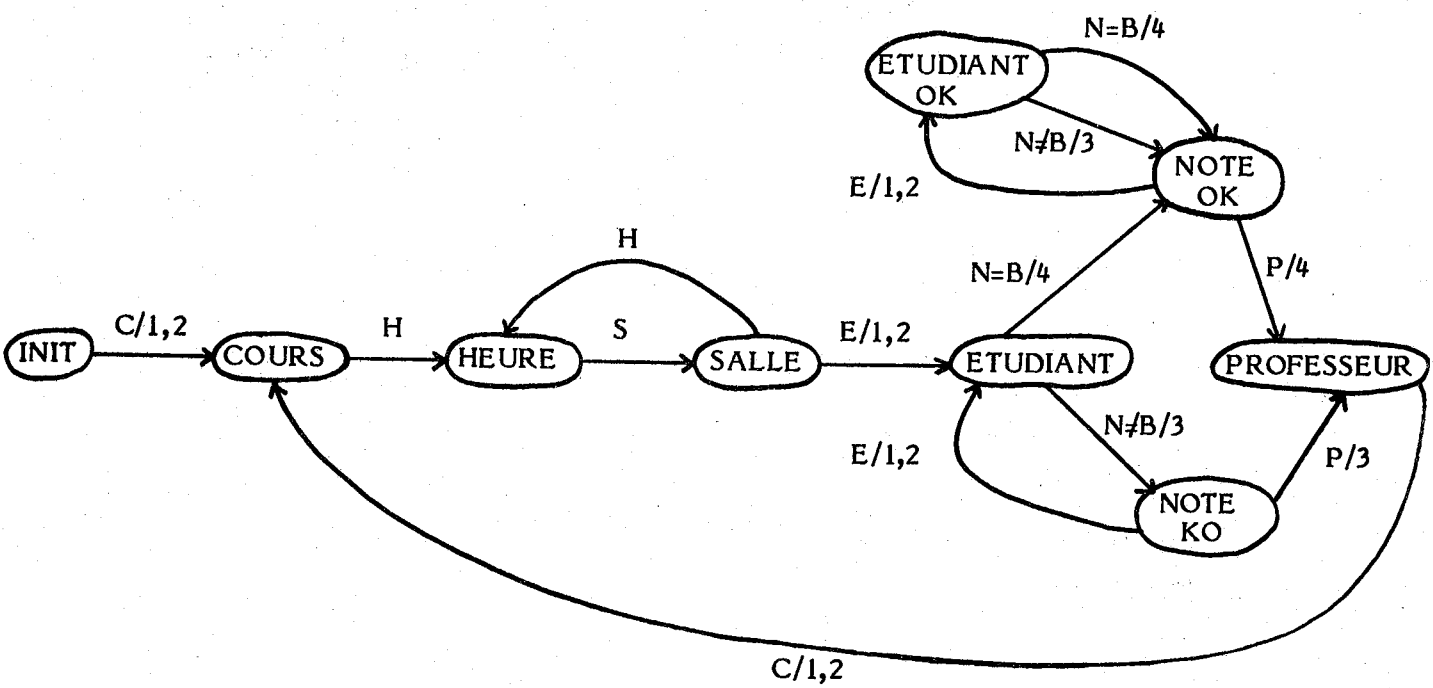


Figure 2.4. Un Automate de Restriction/Projection

C'est pourquoi l'adresse du mot "COURS" dans le tampon cible est mémorisée (action (2)). S'il y a au moins un couple (E,B), l'adresse mémorisée est effacée (action (4)) : on garde le COURS. Sinon le COURS est effacé : action (3) .

Pour éliminer les mots qui ne sont pas à projeter (H,S,N,P) il suffit de ne pas incrémenter RAB : le mot est écrit mais sera immédiatement écrasé par le mot suivant.

Lorsque le tampon source a été balayé, les données filtrées sont disponibles dans le tampon cible. On peut charger le tampon Source avec de nouvelles données à filtrer et recommencer le filtrage. Celui-ci est interrompu chaque fois que le tampon cible est plein, le temps de vider ce dernier.

2.8. Complexité du filtrage

Remarquons que le temps de filtrage est indépendant de la complexité de la requête. A une requête complexe, correspond un automate de taille plus grande, mais le temps de passage d'un état au suivant ne dépend pas de la requête. Dans la mesure où la mémoire du filtre est assez grande pour contenir le code correspondant à l'automate, le temps de filtrage, quelle que soit la requête, est proportionnel à la taille du fichier balayé. Plus exactement, si τ est le temps de cycle (temps pour lire un caractère, l'écrire dans le tampon cible, effectuer les fonctions de sortie et passer à l'état suivant) supposé constant et $|F|$ le nombre de caractères du fichier, le temps de filtrage est :

$$T = |F| . \tau$$

Le temps de génération de l'automate (traduction de la requête en un code exécutable par le filtre) ainsi que sa taille sont par contre fonction de la complexité de la requête.

Une analyse de la taille de l'automate peut être trouvée dans [42].

En résumé, nous avons donc montré - du moins, sur un exemple - qu'il existait un mécanisme de filtrage par AF qui permettait de filtrer en une seule passe un fichier, c'est-à-dire sans retour en arrière sur la chaine de caractères source.

Malheureusement toute restriction/projection ne peut pas être filtrée en une seule passe. Le paragraphe suivant étudie ce problème.

2.9. Requêtes traitées en une seule passe

Il est facile de trouver des contrexemples où le mécanisme décrit plus haut ne fonctionne pas, i.e. la requête ne peut être traitée en une passe.

Soit le fichier : $(CP(EN)^*(HS)^*)^*$

et la requête sur ce fichier : "Projeter cours et étudiants pour les étudiants ayant eu la note B ou ayant suivi un cours en salle 1108", requête que nous notons : (R/N=B ou S=1108)[C,E].

Après avoir balayé les étudiants d'un COURS, ou bien on a éliminé ceux qui n'ont pas eu B et on peut découvrir plus tard que la salle du cours est 1108 et donc qu'il aurait fallu garder tous les étudiants, ou bien on garde tous les étudiants et on découvre que la salle n'est pas 1108 et dans ce cas il faut faire une deuxième passe pour éliminer les étudiants dont la note n'est pas B.

Les résultats qui suivent tentent de répondre à la question :

"Etant donné un FC, un fichier satisfaisant ce FC et une requête de restriction/projection, peut-on générer un AF permettant de filtrer le fichier en une passe ?"

2.9.1. Formats de Compaction SIMPLE-DROITS ou Factorisations linéaires.

Parmi tous les FC possibles pour une relation nous allons nous intéresser à un sous-ensemble, celui des factorisations linéaires.

Définition.

Un FC est simple-droit si dans l'arbre syntaxique associé toutes les '*' sont sur une seule branche et si à un niveau donné, l' '*' est toujours à droite.

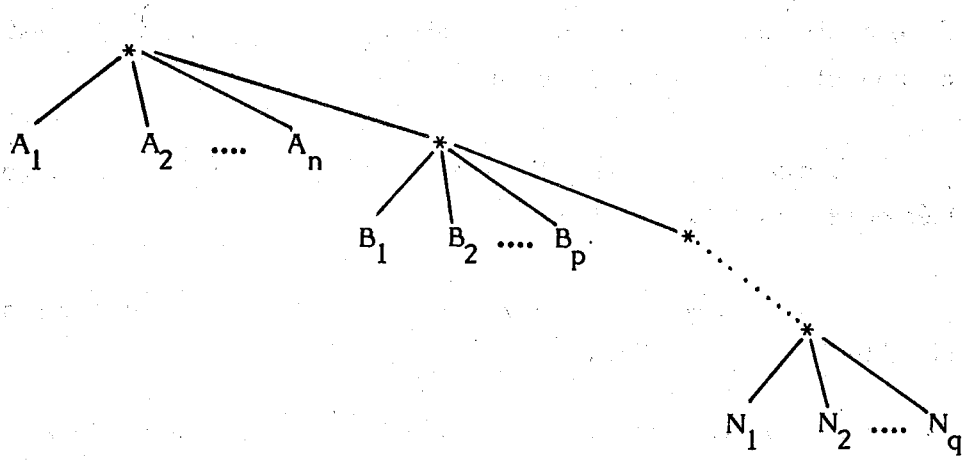
Par exemple le FC : $(CP(EN)^*)^*$ est simple-droit. Par contre $(CP^*(EN)^*)^*$ ne l'est pas.

Théorème 2.9.1.

Soit F un fichier satisfaisant un FC simple-droit. Alors toute restriction/projection peut être effectuée en une passe sur F.

Démonstration

L'arbre syntaxique d'un CF simple-droit est dans le cas général de la forme :

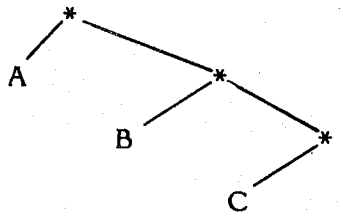


Une condition booléenne de restriction générale est du type

$$\mathscr{C} = C_1 \vee C_2 \vee \ldots \vee C_n$$

où chaque C_i doit être évaluée pour un ou plusieurs attributs de U (en tant que ET de conditions élémentaires sur un attribut).

Dans un souci de clarté nous restreignons au cas où l'arbre a trois niveaux avec un attribut par niveau, où la condition n'a que deux termes : $\mathscr{C} = C_\alpha \vee C_\beta$ et où l'on projette tous les attributs.



Pour démontrer le théorème, on choisit une preuve constructive, i.e. on donne un AF qui permet de traiter en une passe la restriction.

A chaque instant, l'automate balaie une valeur d'attribut et il peut décider si la valeur satisfait C_α ou C_β, i.e. si C_α est vrai ou non et si C_β est vrai ou non.

L'espace d'états de l'automate est choisi de façon à mémoriser les faits suivants :

1) Le niveau de l'arbre où l'automate se situe : on a une variable L prenant les valeurs 0, 1, 2 ou 3.

2) A chaque niveau de l'arbre, on mémorise l'évaluation courante de C_α et C_β : on a besoin de 6 booléens

$$E_{i\alpha}, E_{i\beta}, \quad i=1,2 \text{ ou } 3$$

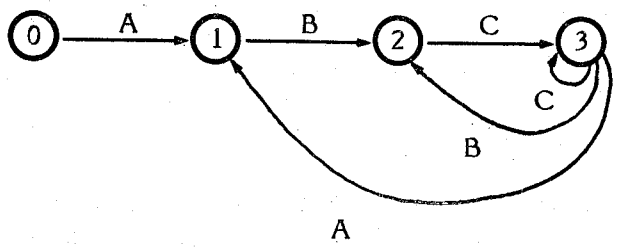
où $E_{i\alpha}(E_{i\beta})$ est vrai si C_α (C_β) = vrai (a été satisfaite) au niveau i.

3) A Chaque niveau, on mémorise si le préfixe de l'enregistrement courant(*) a déjà été validé ou pas : on a besoin de 3 booléens H_1, H_2, H_3. H_i, i=1,2 ou 3 est vrai si le préfixe de niveau i est validé.

(*) Rappelons qu'un enregistrement d'un fichier F de FC : $(XF_1...F_n)^*$ est un fichier de format $XF_1...F_n$. Le fichier F est une concaténation d'enregistrements.
On appelle préfixe de niveau i d'un enregistrement d'un fichier F de FC simple droit $(A_1(A_2(...(A_i(...A_n^*)^*...)^*$ la chaîne de caractères commençant au début de l'enregistrement et s'arrêtant au niveau i. C'est un enregistrement satisfaisant le FC : $(A_1(A_2(...A_i^*)^*...)^*$.

Rappelons que les fonctions de sortie de l'automate sont, soit MARQUER (EMPILER RAB sur P), soit EFFACER (DEPILER(P)), soit ECRASER (RAB ← DEPILER(P)) c'est à dire ramener RAB à la valeur du dernier marqueur.

L est mis à jour suivant le diagramme suivant :



Le reste de la fonction de transition et des fonctions de sortie est donné par la table suivante :

Niveau	Symbole Courant	Transition	Sortie
0	A	$H_1=H_2=H_3=$Faux $E_{1\alpha}=C_\alpha(A)$ $E_{1\beta}=C_\beta(A)$	MARQUER
1	B	$E_{2\alpha}=E_{1\alpha} \wedge C_\alpha(B)$ $E_{2\beta}=E_{1\beta} \wedge C_\beta(B)$	MARQUER
2 ou 3	C	$E_{3\alpha}=E_{2\alpha} \wedge C_\alpha(C)$ $E_{3\beta}=E_{2\beta} \wedge C_\beta(C)$ $H_3=H_3 \vee E_{3\alpha} \vee E_{3\beta}$ $H_2=H_2 \vee H_3$ $H_1=H_1 \vee H_3$ $H_3=$Faux	MARQUER si $H_3=$vrai EFFACER si $H_3=$faux ECRASER
3	B	$H_2=$Faux	si $H_2=$vrai EFFACER si $H_2=$faux ECRASER
3	A	$H_1=$Faux	si $H_1=$vrai EFFACER si $H_2=$faux ECRASER

La première colonne indique à quel niveau l'on se trouve. La deuxième colonne précise quel est l'attribut qui a été balayé (on suppose que l'automate traite des mots de longueur 1 caractère : en fait il existe un sous-automate pour la reconnaissance de chaque attribut). La fonction de la colonne 3 complète la transition : Un état de l'automate est représenté par une valeur de niveau et un ensemble de valeurs des booléens définis plus haut.

Enfin la dernière colonne indique les fonctions de sortie à exécuter suivant l'état.

La preuve se généralise aisément aux cas de L niveaux (L > 3) et de plus de deux conditions séparées par un OU dans la condition de Sélection : Pour représenter l'espace d'états on a besoin d'un booléen par condition et par niveau et de un booléen par niveau.

Théorème 2.9.2.

Quel que soit le FC et un fichier satisfaisant ce FC, toute restriction/projection telle que la condition de restriction ne contient pas de 'ou' peut être traitée en une passe.

Démonstration

Soit un FC : $(XF_1F_2...F_n)^*$, $n \geq 0$, $X \subseteq U$

Pour simplifier, restreignons-nous au cas où le FC est du type $(AF_1F_2)^*$ où F_1 et F_2 sont deux FC sur X_1 et $X_2 \subset U$ tels que $X_1 \cap X_2 = \emptyset$, et où la racine est réduite à un attribut.

La condition de restriction peut toujours s'écrire :

$$\mathscr{C} = C_A \wedge C_{F_1} \wedge C_{F_2}$$

où C_A est une condition élémentaire sur l'attribut A (une tautologie si la condition $\mathscr{C}$ ne porte pas sur A), C_{F_1} est un ET de conditions sur X_1, C_{F_2} est un ET de conditions sur X_2.

La preuve est récursive.

Supposons qu'il existe un sous-automate A1 capable de filtrer en une passe un fichier de format F1. Dans l'enregistrement courant (de format AF1F2), après avoir parcouru le préfixe de format F1, A1 a évalué C_{F_1} et gardé les attributs à projeter si C_{F_1}=vrai.

Supposons qu'il existe un sous-automate A2 capable de filtrer en une passe un fichier de format F_2. A la fin de l'enregistrement courant (de format AF1F2) le suffixe de format F2 a été analysé par A2 et les attributs à projeter ont été gardés si C_{F_2}=vrai.

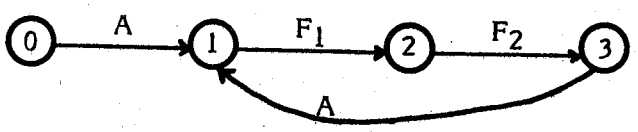
La preuve est en deux étapes :

a) On montre que si A_1 et A_2 existent, il existe alors un automate $\mathcal{A}$ capable de filtrer en une passe un fichier satisfaisant $(AF1F2)^*$.

b) on montre que A_1 et A_2 existent.

a) Pour montrer que $\mathcal{A}$ existe, on prend la même démarche constructive que dans le théorème précédent.

On a besoin seulement d'une variable booléenne E et des 3 variables booléennes C_A, C_{F_1} et C_{F_2}. Les diagrammes d'états et la fonction de transitions/sorties sont donnés ci-dessous



Etat	Symbole	Transition	Sortie
0	A	$E=C_A$	MARQUER
1	F_1	$E=E \wedge C_{F_1}$	
2	F_2	$E=E \wedge C_{F_2}$	
3	A		si E=Vrai EFFACER si E=Faux ECRASER
		$E=C_A$	MARQUER

b) A_1 existe. En effet,

- ou bien F_1 est simple-droit et le résultat suit par le théorème 2.8.1.

- ou bien F_1 peut être décomposé sous la forme BG1G2 (sans perte de généralité) où G_1 et G_2 sont des FC sur Y_1, $Y_2 \subset X_1$ tels que $Y_1 \cap Y_2 = \emptyset$. Le résultat suit du fait que au plus tard après N-1 décompositions où N est la profondeur de l'arbre syntaxique associé à F_1, on rencontrera un CF simple-droit.
  La démonstration tient évidemment pour A_2.
  On généralise facilement la démonstration au cas où le FC est de type $(XF_1F_2...F_n)^*$ où n > 2 et la racine contient plus d'un attribut.

CQFD

Théorème 2.9.3.

Quel que soit le FC et un fichier satisfaisant ce FC, toute restriction/projection telle que la condition de restriction ne contient des 'ou' qu'entre conditions élémentaires portant sur le même attribut peut être traitée en une passe.

Démonstration

Supposons que la condition booléenne de restriction soit du type

$$\mathscr{C} = C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

où C_i est soit une tautologie,

soit de la forme $A_i comp_1 valeur_1 \vee A_i comp_2 valeur_2 \vee \ldots \vee A_i comp_m valeur_m$

alors il existe un sous-automate permettant de comparer l'attribut A_i à la liste de valeurs {valeur i} i=1,m et d'évaluer la condition C_i. Pour s'en convaincre, le lecteur est reporté à [42] qui analyse la taille de tels automates dont le diagramme est un arbre lexicographique et dont les transitions finales sont au nombre de 2m+1. On peut alors appliquer le théorème précédent.

Les résultats qui précèdent sont cependant restrictifs. Existe-t-il des fichiers non simple-droits tels que toute restriction-projection soit effectuée en une passe ? L'auteur a été incapable d'exhiber une classe de tels fichiers.

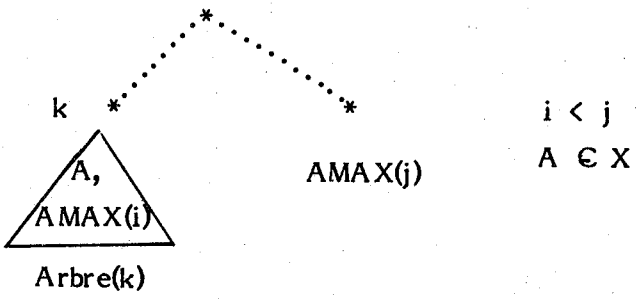
Cependant on donne plus bas un algorithme qui, étant donné un FC non simple-droit et une opération contenant des 'ou' décide si l'opération peut être effectuée en une passe ou non. Dans le dernier cas, l'opération doit être décomposée en plusieurs opérations (chacune étant effectuée en une passe).

2.10. Une CNS pour qu'une requête soit traitée en une passe

Définition

Le rang de l'attribut A, dénoté rang(A) est le nombre de noms d'attributs comptés avant A lorsqu'on balaie le FC de gauche à droite.

La figure ci-dessous illustre le cas où **P** n'est pas vérifiée :

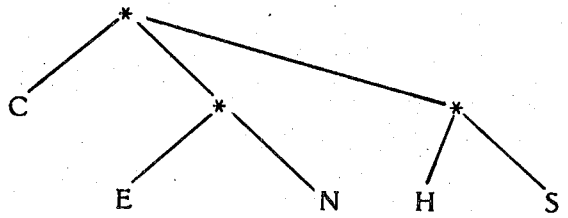


Démonstration : voir [8]

Par exemple, considérons le FC :

$$C_F : (C(EN)^*(HS)^*)^*$$

d'arbre syntaxique :



La requête q1 : $(R \mid N=B \vee S=1108)[C,E,H]$ ne peut être effectuée en une passe, puisque dans le sous-arbre (E,N) :

1) il y a à la fois un attribut à projeter (E) et un attribut de la condition C_1 (N=B),

2) S qui appartient à C_2 (S=1108)

   - apparaît à droite de N dans le FC
   - et n'appartient pas au sous-arbre (E,N).

Par contre, sur le même fichier la requête

$$q_2 : (R \mid N=B \vee S=1108)[C,H]$$

peut être traitée en une passe.

Remarque.

On vérifie aisément que les requêtes où les seuls "ou" sont entre conditions élémentaires sur le même attribut qui peuvent donc être traitées en une passe (Théorême 2.9.3) satisfont toujours la propriété **P.**

2.11. Conclusion

Nous avons montré dans ce chapitre comment implanter l'opération relationnelle de restriction/projection par un filtrage à AF programmable en choisissant une représentation des relations compactées hiérarchiquement. Nous avons ensuite caractérisé les types de fichiers et les types de requêtes qui permettent un filtrage par balayage séquentiel en une seule passe.

Les deux intérêts essentiels d'un tel mécanisme de filtrage sont que

1) le temps de filtrage est indépendant de la complexité de la requête,

2) il s'adapte bien à d'autres opérations de l'algèbre relationnelle.

Ce mécanisme s'applique évidemment à des fichiers dits "plats" c'est-à-dire ceux où les relations ne sont pas compactées mais représentées sous forme tabulaire.

Une implantation matérielle d'un tel mécanisme est détaillée dans le chapitre suivant.

CHAPITRE 3 _

VERSO
(FILTRAGE MATERIEL)

3. - VERSO (FILTRAGE MATERIEL)

3.1. Introduction

VERSO est un SGBD relationnel en cours de développement à l'INRIA. Sa principale caractéristique est l'utilisation d'un mécanisme de filtrage par AF près de la mémoire de masse (disque à tête mobile). Ce filtrage permet non seulement d'exécuter les opérations unaires de l'Algèbre relationnelle (sélection, projection) mais encore l'insertion, la suppression et les opérations binaires sur des relations au préalable triées (union, intersection, différence, division et surtout jointure).

Il existe deux implantations en cours, du mécanisme de filtrage :

- un Filtre matériel que nous étudions en détail dans ce chapitre,

- un Filtre logiciel décrit dans le chapitre 5.

Les performances des deux implantations sont étudiées dans le chapitre 6.

Initialement, seul le filtrage matériel était prévu :

Le premier objectif du développement de VERSO était de montrer qu'on peut construire un SGBD relationnel complet performant autour d'un tel filtre.

Comparé aux autres MBD prototypes ou produits industriels, VERSO se situe dans la catégorie des "Disques Intelligents" plutôt que dans celle des systèmes multiprocesseurs, à savoir qu'il n'y a pas de parallélisme (excepté celui entre le filtre et l'Unité Centrale), pas de processeurs de tri ni de jointure, etc..

Le chapitre suivant permet de restituer plus précisément VERSO par rapport aux autres approches architecturales.

Pourquoi le choix d'une seule fonction à optimiser (l'accès aux données) et le choix de l'implanter sans parallélisme ?

La raison en est la simplicité de conception et de réalisation :

On verra plus bas que déjà la réalisation d'un seul processeur de filtrage est bien plus complexe que ne le laissait prévoir la simplicité du principe.

Le deuxième objectif de VERSO était d'étoffer l'interface relationnelle classique par de nouvelles fonctionnalités. Bien que l'interface relationnelle subsiste dans VERSO, l'usager "averti" a à sa disposition une interface dite V-relationnelle. Le concept de V-relation est une généralisation de la notion de relation qui permet d'introduire certaines valeurs nulles (information incomplète), les dépendances d'inclusion et de traiter facilement des relations non-normalisées.

On commence par définir les V-relations (paragraphe 3.2). On décrira ensuite brièvement les différentes couches logicielles du système VERSO (paragraphe 3.3). L'architecture de la machine et la structure détaillée du filtre sont présentées dans le dernier paragraphe.

3.2. Les V-relations [1,7,10,13,54]

Le modèle V-relationnel brièvement présenté ici n'est pas neuf bien que sa formalisation soit récente. Le vieux modèle hiérarchique - par pragmatisme et sans rigueur peut être - utilise déjà un certain nombre de notions analogues qu'on rencontre aussi

chez d'autres auteurs [52]. On trouve actuellement dans la littérature un certain nombre d'approches semblables [23,29,45].

3.2.1. Motivation

Dans les systèmes où chaque relation est implantée sous forme d'un fichier plat, on a le choix entre

a) Eviter les anomalies(*) et donc créer de multiples tables, chacune représentant une Relation normale.

Soit l'Univers d'attributs U = {Cours, Professeur, Etudiant, Note, Heure, Salle}. La sémantique qui sous-tend cet univers est qu'on a des cours, des étudiants qui ont des notes dans ces cours, lesquels sont enseignés par des professeurs et ont lieu à une heure déterminée dans une salle précise.

On peut représenter cette sémantique par l'ensemble des Relations normalisées :

$r_1(C)$, $r_2(CEN)$, $r_3(CP)$, $r_4(CHS)$

De cette manière, l'utilisateur gère lui-même ses mises-à-jour. Il peut détruire un étudiant et sa note sans détruire un cours, son professeur et son horaire. Il peut insérer un nouveau cours sans information sur le professeur ou l'heure ou la salle. Toutefois des problèmes subsistent : s'il insère un professeur (insérer n-uplet {cours, professeur} dans r_3), il faut vérifier la dépendance d'inclusion "le cours existe déjà dans r_1".

Dans cette approche de multiples relations donc de multiples fichiers, les jointures entre relations coûtent cher et les performances se dégradent.

(*) le lecteur non familiarisé avec la théorie relationnelle de la décomposition est reporté à [53].

b) Eviter trop de jointures en regroupant les Relations :

A l'extrême, dans l'exemple précédent, on aurait la relation universelle

R(C P E N H S)

Il n'y a plus de jointures mais que des restrictions. Les mises-à-jour cependant deviennent problématiques à cause des anomalies dues à la forme non normale de la relation.

Le concept de V-relation a été introduit pour éviter les inconvénients de ces deux approches.

3.2.2. Description informelle d'une V-relation

Une V-relation a un ensemble d'attributs U, e.g.

$U = \{C, E, N, P, H, S\}$

et un ensemble d'atomes ou unités de mise à jour, e.g.

C, CEN, CP, CHS

Ceci veut dire que l'usager peut modifier les sous-nuplets définis sur ces atomes : par exemple, on peut insérer un nouveau professeur (atome CP) dans un cours sans se soucier ni des Etudiants, ni des horaires de ce cours.

La valeur de la V-relation est l'ensemble des valeurs des relations ordinaires $r_1(C)$, $r_2(CEN)$, $r_3(CP)$ et $r_4(CHS)$. Ces quatre relations satisfont les dépendances implicites d'inclusion :

$$r_2(C) \subseteq r_1(C),\ r_3(C) \subseteq r_1(C),\ r_4(C) \subseteq r_1(C).$$

Autrement dit, toute valeur de l'attribut 'cours' dans les relations r_2, r_3 et r_4 existe dans la relation r_1.

Définition

Une V-relation est décrite par un schéma $R\{U,A\}$ où R est le nom de la V-relation, U est l'univers d'attributs et $A \subseteq \mathcal{P}(U)$, l'ensemble des parties de U, est un ensemble d'atomes couvrant U, tel que :

(1) A est fermé sous intersection :

$\forall\ X_1, X_2 \in A,\quad X_1 \cap X_2 \in A$ (et $\emptyset \in A$)
i.e. l'intersection de deux atomes est un atome

(2) $\forall\ X \in A,\ E_X = \{Y \cap X \mid Y \in A\}$ est totalement ordonné par inclusion, i.e.

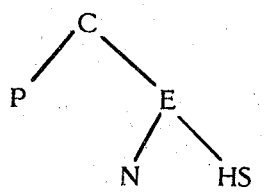
$\forall\ Y_1, Y_2 \in E_X$ on a soit $Y_1 \subseteq Y_2$
soit $Y_2 \subseteq Y_1$

Dans l'exemple précédent, si $X \equiv CEN$, alors $E_X = \{C,CEN\}$ et l'atome CEN contient bien l'atome C.

Intuitivement, on veut pouvoir construire une arborescence. Ainsi avec le schéma

$U = \{C,P,E,N,H,S\}$
$A = \{C,CP,CE,CEN,CEHS\}$

on construit l'arborescence



qu'on interprète de la façon suivante :

Condition 1 : Tout chemin de la racine à une feuille est un atome, i.e. CP,CEN,CHS sont des atomes ; toutes les racines communes sont également des atomes, i.e. C et CE sont également des atomes.

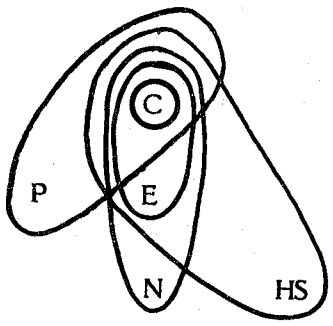
Condition 2 : Les racines communes à un atome X et à tous les atomes de la V-relation sont tous les atomes sur le chemin constituant X. Par exemple E_{CEN} = {C,CE,CEN} où C est commun à CP et CEN, CE est commun à CEHS et CEN, et CEN est bien évidemment l'intersection de CEN avec lui-même. Sur un même chemin, il existe bien entendu une relation d'ordre des sous-chemins en partant de la racine.

Remarque : La représentation par arborescence est ambigue. Le schéma A = {CP,CEN,CEHS} se représente par la même arborescence que ci-dessus. Et pourtant ce n'est pas une V-relation car C et CE ne sont pas des atomes (condition 1 non satisfaite).

C'est pourquoi dans les exemples qui suivent, pour vérifier qu'on a à faire à une V-relation, on préfère l'illustration ensembliste suivante : à chaque atome correspond une "patate". Ainsi, la V-relation

A = {C,CP,CE,CEN,CEHS}

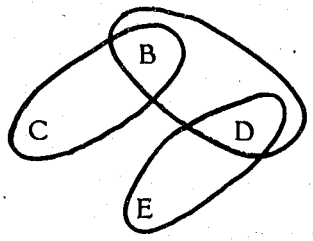
est représentée par le croquis suivant :



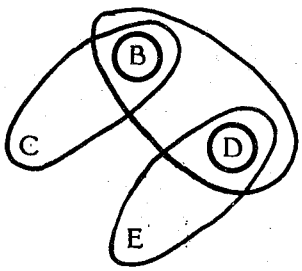
Exemples

Les schémas suivants ne satisfont pas la définition de V-relation sur U = {B,C,D,E}.

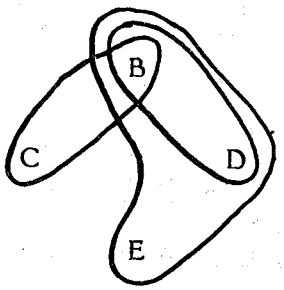
a) A = {BC,DE,BD} : La propriété 1) n'est pas satisfaite car B et D ne sont pas des atomes



b) A = {B,D,BC,DE,BD} : La propriété 1) est satisfaite, mais la propriété 2) ne l'est pas : E_{BD} = {B,D} et on n'a ni B ⊆ D, ni D ⊆ B. En effet B n'est pas une "racine" commune à BC, BD et BE : idem pour D.



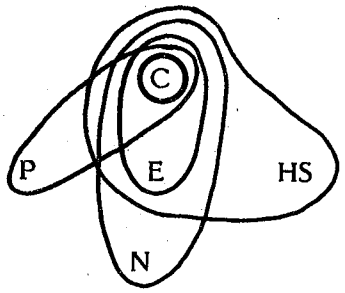
c) A = {BC,BD,BDE} : La propriété 1) n'est pas satisfaite : B n'est pas atome



Par contre, si on rajoute l'atome B, on a une V-relation.

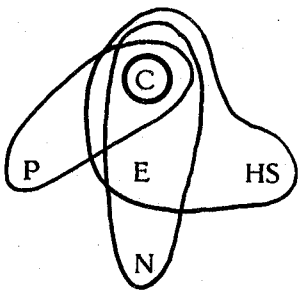
Remarquons que sur U = {C,P,E,A,N}

$\mathscr{A}$ = {C,CP,CE,CEN,CEHS} est une V-relation



d) Par contre $\mathscr{B}$ = {C,CP,CEN,CEHS} n'est pas une V-relation :

CEN CEHS = CE $\notin \mathscr{B}$



Remarques

1) Une V-relation sera implantée sous forme de fichier compacté. Par exemple R(U,A), où U = {C,P,E,N}, et A = {C,CP,CEN} est une V-relation.
R sera implantée sous forme d'un fichier F de FC : $(CP^*(EN)^*)^*$.
On a la dépendance : C $\twoheadrightarrow$ P | EN.

La V-relation S(U,A) où U = C,P,E,N,H,S
et A = {C,CP,CE,CEN,CEHS}
sera représentée par le Fichier F de FC :
$(CP^*(EN^*(HS)^*)^*)^*$

2) On économise ainsi les jointures, e.g. avec la V-relation R(U,A) ci-dessus, la requête "Professeurs de l'Etudiant 'TOTO' est traitée par une restriction/projection sur le fichier F.

3) On peut traiter un certain nombre d'informations incomplètes. Ainsi, dans l'exemple précédent, on peut insérer un nouveau cours sans connaître, ni ses professeurs, ni ses étudiants. On peut insérer un nouveau professeur dans un cours sans connaître les étudiants et leur note.

3.2.4. V-Opérations

Une algèbre d'opérations sur V-relations est étudiée dans [13]. Nous ne décrivons ici que la V-union et la V-intersection par un exemple ainsi que la V-sélection de façon un peu plus formelle. On trouve dans la littérature des notions équivalentes à la V-Union et la V-interprétation. Lacroix et Pirotte [31], par exemple ont introduit la notion de jointure généralisée.

a) **V-union** (jointure externe)

Soient R et S 2 V-relations définies respectivement sur U_1 = {A,B} et U_2 = {A,C} telles que $\mathcal{A}_1$ = {A,AB} et $\mathcal{A}_2$ = {A,AC} et de valeurs

a) pour R les relations ordinaires $r_1(A)$, $r_2(AB)$
b) pour S les relations ordinaires $s_1(A)$, $s_2(AC)$

La V-union de R et S notée 'U', est la V-relation T=R U S définie sur U = {A,B,C} avec pour ensemble d'atomes $\mathcal{A}$ = {A,AB,AC} ayant pour valeur les relations ordinaires (*) :

(*) On peut vérifier que le résultat satisfait les dépendances implicites d'inclusion de l'ensemble $\mathcal{A}$.

$$t_1(A) = r_1(A) \cup s_1(A)$$
$$t_2(AB) = r_2(AB)$$
$$t_3(AC) = s_2(AC)$$

b) **V-intersection** (jointure interne)

Soient R et S les V-relations ci-dessus. La V-intersection notée '$\cap$' de R et S est la V-relation T=R $\cap$ S définie sur U = {A,B,C} avec pour ensemble d'atomes $\mathscr{A}$ = {A,AB,AC} ayant pour valeur les relations ordinaires :

$$t_1(A) = r_1(A) \cap s_1(A)$$

$$t_2(AB) = (r_2(AB) *_A s_2(AC))[AB] \quad \text{(équi-jointure et projection sur AB)}$$

$$t_3(AC) = (r_2(AB) *_A s_2(AC))[AC] \quad \text{(équi-jointure et projection sur AC)}$$

Remarque

Dans les 2 cas (V-union et V-intersection) si R et S sont implantées suivant les formats maximal compactés : $(AB^*)^*$ et $(AC^*)^*$, T peut être implantée suivant le format maximal compacté $(AB^*C^*)^*$.

c) **V-sélection**

Soit C un FC défini de façon récursive :

$$C = X(C_1)^* \ldots (C_p)^* \quad p \geq 0, \quad X \subseteq U \text{ racine}$$

où $C_1, \ldots, C_p$ sont des formats.

C décrit un enregistrement E du fichier F.

La Sélection S(F) du fichier F est une fonction booléenne définie de la façon suivante :

Soit S(E) la fonction booléenne qui associe à un enregistrement E la valeur 'VRAI' si et seulement si E satisfait la condition de restriction $\mathscr{C}$.

$\mathscr{C}$ peut s'exprimer sous la forme :

$$\mathscr{C} = \mathscr{C}_1 \wedge \mathscr{C}_2 \wedge \ldots \wedge \mathscr{C}_m$$

où chaque $\mathscr{C}_i$ est une condition sur un attribut $A_i \in U$ de la forme

$$\mathscr{C}_i = A_i \; op_1 \; valeur_1 \vee \ldots \vee A_i op_\ell valeur_\ell$$

où op_j, $j \leq \ell$ est un comparateur arithmétique et $valeur_j$, $j \leq \ell$ $\in D(A_i)$.

$S(A_i)$ est une fonction booléenne qui associe à la valeur d'un attribut $A_i \in U$ dans un enregistrement E, la valeur 'VRAI' si et seulement si :

ou bien A_i n'apparaît pas dans la condition $\mathscr{C}$

ou bien A_i apparaît et $\mathscr{C}_i$ est satisfaite par la valeur de A_i dans l'enregistrement.

On peut alors définir S(F) :

$$S(F) = \underset{*}{U} S(E)$$

où l'union est prise sur tous les enregistrements du fichier F. On dénotera S(F) par $S^*(E)$.

S(E) est définie de façon récursive :

$$S(E) = S(X) \underset{i=1,p}{\cap} \sigma(C_i)$$

où $S(X) = \underset{A \in X}{\cap} S(A)$ (la racine satisfait la condition si tous ses attributs la satisfont)

et où :

σ est soit l'identité : $\sigma(C_i)$ est vrai si et seulement si $S^*(C_i)$ est vrai

soit la négation : $\sigma(C_i)$ est vrai si et seulement si $S^*(C_i)$ est faux

soit la fonction constante : $\sigma(C_i)$ est toujours vrai

Exemple : Soient le FC $(COURS(ETUDIANT(NOTE)^*)^*)^*$
et la requête :
"Pour chaque cours donner le cours et la liste des étudiants qui n'ont pas de note inférieure à 10"
On peut exprimer la V-sélection de la manière suivante : COURS!(ETUDIANT!(¬ existe NOTE < 10))
où le '!' indique qu'il faut projeter l'attribut et où la condition de restriction est $\mathscr{C}$: NOTE < 10.
On a :

S(NOTE) est vrai ssi NOTE < 10
S(ETUDIANT) est vrai ssi S^*(NOTE) est faux (σ : négation)
S(COURS) est vrai ssi S^*(ETUDIANT) est vrai (σ : identité)

Donc satisfont à la définition toutes les requêtes qui sont des conjonctions de requêtes élémentaires sur un attribut (celles-ci pouvant être des disjonctions).

Il est clair que la définition de V-sélection satisfait au théorème 2.8.3.

Il existe donc un automate fini permettant de filtrer tout fichier pour de telles requêtes en une passe.

Ceci termine la description des V-relations. On passe maintenant à la description des couches logicielles de VERSO.

3.3. VERSO : Le logiciel

Comme tout SGBD relationnel VERSO a les fonctionnalités suivantes :

- Définition, Recherche et Manipulation des données,
- Gestion des Transactions,
- Contrôle de la concurrence et reprise sur pannes.

Nous ne décrivons ici que la première fonctionnalité. Le lecteur intéressé par les points 2 et 3 est reporté à [10].

On peut décomposer le logiciel en 4 couches (Figure 3.1). Chaque couche est définie par un ensemble d'objets et un ensemble d'opérations sur ces objets. Ces objets sont soit locaux soit définis au moyen d'objets de la couche immédiatement inférieure. De même les opérations sont soit intrinsèques soit définies par des algorithmes utilisant des opérations de la couche inférieure.

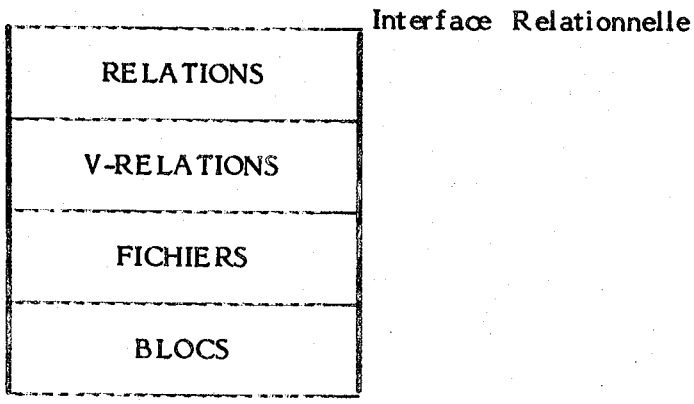


Figure 3.1. Décomposition en couches de VERSO

Au niveau le plus élevé, on met à jour des relations et on affecte des valeurs à une relation au moyen d'une expression de l'algèbre relationnelle (usager naïf). Les requêtes relationnelles sont

traduites en requête V-relationnelles. Mais l'usager 'expert' peut manipuler directement des V-relations sans passer par l'interface relationnelle.

Au niveau juste inférieur on manipule des V-relations au moyen de l'algèbre V-relationnelle et on fait des changements de format. Au niveau d'en dessous, on manipule des fichiers, c'est-à-dire un ensemble de blocs ayant une localisation physique.

Enfin, au niveau le plus bas on manipule un bloc (ou deux blocs) dont on connaît l'adresse et sur lequel on effectue les opérations unaires et binaires de filtrage.

On s'aidera d'exemples pour décrire les 4 couches.

3.3.1. Relations

On ne s'étendra pas sur cette couche. L'usager voit des relations de trois types : entrée, sortie ou base. Il peut créer, détruire, manipuler des relations, leur affecter une valeur, etc.

Par exemple en reprenant les notations usuelles, l'usager fait la requête

$$Q[\text{Revenu}] \leftarrow [(R_1/\text{Cours}=\text{Math}) *_{\text{Prof}=\text{Prof}} R_2]\,[\text{Revenu}]$$

où R_1 et R_2 sont les relations base

R_1(Cours,Etudiant,Note,Prof)
R_2(Prof,Adresse,Revenu)

La relation de sortie Q aura pour valeur "les revenus des Professeurs de Math".

3.3.2 V-Relations

A ce niveau on manipule i) des V-relations qui sont en fait caractérisées par un type (entrée, sortie, base et temporaire), un nom et un format de compaction et ii) le Catalogue des V-relations.

C'est la responsabilité du concepteur de la base de définir des V-relations en fonction de la sémantique de l'usager.

Les opérations à ce niveau sont la création, la destruction, la manipulation et les opérations de la V-algèbre.

On y trouve également

1) des opérations de changement de format
2) l'accès et la mise à jour du catalogue des V-relations

Par exemple, dans l'exemple ci-dessus on a 2 V-relations VR1 et VR2 représentées schématiquement par leurs formats ; VR1 : $(P(C(EN)^{*})^{*})^{*}$ et VR2 : $(A(PR)^{*})^{*}$. La requête de la couche "Relations" est décomposée en 6 opérations (en omettant l'accès au catalogue) :

1) CREER R'_1 TEMPORAIRE de format P^{*}
2) $R'_1 \leftarrow (R_1 \mid C = MATH)[P]$ (V-sélection)
3) CREER R'_3 TEMPORAIRE de format $(PR^{*}A^{*})^{*}$
4) $R'_3 \leftarrow$ CHANGT-FORMAT (R_2)
5) CREER Q_2 SORTIE de format R^{*}
6) $Q_2 \leftarrow (R'_1 \underset{P=P}{*} R'_3)[R]$ (V-intersection + projection)

Pour effectuer la V-intersection 6) par filtrage, il est nécessaire que les formats aient la même racine P. D'où l'opération 4) de changement de format de la V-relation base R_2.

3.3.3. Fichiers

L'objet FICHIER est caractérisé par un nom et une liste d'adresses de blocs (où est rangé le fichier représentant la V-relation). On voit également à ce niveau l'Index Maître qui permet la "localisation" des données.

Dans VERSO tous les fichiers sont triés par ordre lexicographique croissant sur tous les champs. Par exemple, dans le fichier $(C(EN)^*(HS)^*P^*)^*$ tous les enregistrements sont triés sur le nom du cours et dans un enregistrement, les couples (étudiant,note) sont triés sur le nom d'Etudiant.

L'Index maître est un Index non dense unique (pour toutes les bases et toutes les relations) dont la structure est décrite dans le paragraphe suivant.

Les opérations de cette couche sont :

i) les opérations de consultation et de mise à jour de l'index,
ii) les opérations de sélection/projection, insertion, suppression dans un fichier,
iii) les opérations binaires sur deux fichiers (e.g. correspondant à la V-union et à la V-intersection),
iv) les opérations de tri d'un fichier (correspondant à un changement de format).

3.3.4. Implantation de l'INDEX par un arbre lexicographique élagué [42]

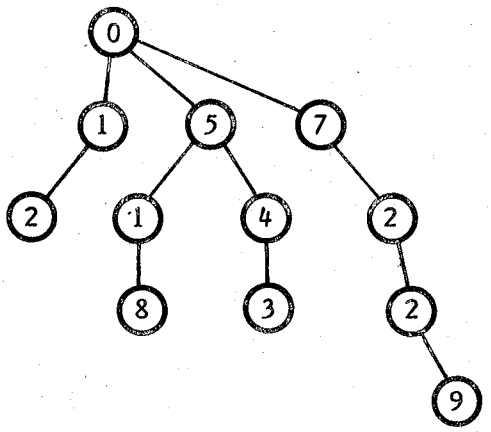
L'Index maître range pour toutes les bases gérées par le système, pour toutes les V-relations de chaque base et pour tous les blocs de chaque V-relation, le plus "petit" enregistrement de chaque bloc qui est représenté en entier (et non pas par une clé) suivi de l'adresse de bloc. Cependant l'information stockée est compactée

grâce à une structure d'arbre lexicographique qu'on va maintenant décrire. L'index est lui-même trié et découpé en pages et il en existe deux implantations dont on donnera les avantages et les inconvénients.

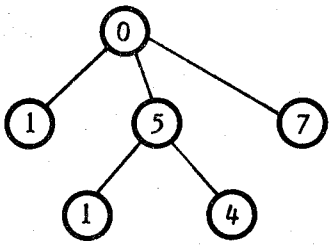
L'intérêt principal d'une telle structure de données pour l'Index est qu'elle permet une représentation suffisamment compacte pour que l'Index réside intégralement en mémoire RAM.

3.3.4.1. - Arbre lexicographique élagué

Un Arbre lexicographique élagué (ALE) est obtenu à partir d'un Arbre lexicographique (AL) en tronquant (élaguant) les sous-arbres filiformes. Un sous-arbre est filiforme si tous ses noeuds n'ont qu'un fils (il ne représente qu'une chaine de caractères). On représente un sous-arbre filiforme par l'étiquette de sa racine. Soit l'AL défini sur le vocabulaire $\Sigma = \{0, 1, \ldots, 9\}$



L'ALE associé à cet AL est le suivant :



L'ALE peut être représenté linéairement en parcourant l'arbre de façon symétrique. S'il n'y a qu'un fils, on l'écrit. Sinon on ouvre une parenthèse qu'on referme quand on quitte le sous-arbre.

La représentation linéaire de l'ALE précédent est :

(0(15(14)7))

3.3.4.2. - ALE linéaire comme représentation de l'Index

Les chaines de caractères représentées par l'AL correspondent aux valeurs du plus petit enregistrement de chaque bloc précédé du nom de base et du nom de relation.

On compacte donc l'information pour obtenir un ALE et à chaque feuille on rajoute l'adresse de bloc.

Considérons par exemple la partie d'un tel index pour la V-relation R_o de format (Adresse,Nom)*, de la base B_o (Figure 3.2). La partie a) de la figure donne la liste des plus petits enregistrements de chaque bloc, la partie b) donne la représentation par un ALE linéaire. Le caractère ■ est l'indicateur d'adresse de bloc.

[42] évalue la taille de l'ALE et applique ces résultats à l'évaluation de la taille de l'Index Maître. Sous un certain nombre d'hypothèses, on estime que pour une base de taille 300 Moctets, l'index devrait être de l'ordre de 100 Koctets.

B_oR_o ALLOS PIERRE ■ α

B_oR_o AUBENAS FRANCOIS ■ β

B_oR_o PARIS AMANTA ■ γ

B_oR_o PARIS ANNE ■ δ

B_oR_o PARIS MICHEL ■ ϵ

B_oR_1 ... ■ ζ

⋮

B_1 ... ■ η

⋮

a) liste de "plus petits" enregistrements suivis de l'adresse de bloc

$(B_o(R_o(A(L$ ■ α U ■ β)PARIS(A(M ■ γ N ■ δ)M ■ ϵ)))R_1... ■ ζ)B_1 ...

b) ALE linéaire

Figure 3.2. Représentation de l'Index par un ALE linéaire
Base B_o, V-relation R_o, $(Adresse,Nom)^*$

3.3.4.3. - ALE linéarisé ou non

L'index a été implanté sous forme d'ALE linéaire dans VERSO. Outre sa forme extrêmement compacte, l'avantage lié à la linéarisation de l'arbre est qu'il existe un automate permettant la consultation de l'Index [42]. Cependant les opérations de mise à jour sont très compliquées, en particulier à cause de la pagination de l'Index.

C'est pourquoi, bien que légèrement moins compacte [42], une structure d'ALE arborescente a été préférée. C'est elle qui est actuellement implantée dans VERSO [38], les opérations de consultation et de mise à jour s'effectuent par des méthodes classiques (et non pas par filtrage séquentiel).

3.3.5. Bloc

Le seul objet de cette couche est le bloc caractérisé par son adresse. Avant de décrire cette couche, il est nécessaire de donner quelques indications sur la représentation physique d'une V-relation par un fichier.

Rappelons tout d'abord que tous les fichiers sont triés par ordre lexicograqphique croissant. Un fichier est une séquence d'enregistrements répartis sur plusieurs blocs dont les adresses ont été au préalable allouées. Bien qu'il soit prévu à terme qu'un enregistrement puisse être de taille supérieure à un bloc(*), nous considérons pour l'instant qu'un bloc est composé de :

i) une en-tête contenant notamment le nom de la base, le nom de la relation,

---

(*) Dans ce cas on le décompose en plusieurs parties ayant même racine, e.g. si le fichier a pour format $(AB^*)^*$ et que l'enregistrement $a_0b_0...b_n$ dépasse la taille d'un bloc, on le décompose en deux parties : $a_0b_0...b_p$ et $a_0b_{p+1}...b_n$. Et l'index contient 2 feuilles : 'a_0b_0' et 'a_0b_{p+1}'.

ii) une séquence de $n \geq 1$ enregistrements se terminant par un caractère de fin de bloc ';'

EN-TETE	$enregistrement_1$ ..... $enregistrement_n$;

Le format d'un enregistrement est le format de compactage. Quant au format d'un champ, i.e. de la représentation d'un attribut, c'est le suivant :

$$< t_R > < \text{valeur} > < t_{fin} >$$

où t_A est un caractère identifiant l'attribut, < valeur > est une chaîne de caractères de longueur variable représentant la valeur de l'attribut prise dans son domaine de définition et où t_{fin} est un caractère représentant l'étiquette de fin de champ.

Les opérations sur ce bloc sont de deux types :

a) **filtrage** : il s'agit des opérations de sélection/projection, d'insertion, de suppression et des opérations binaires (union, intersection notamment).
Elles sont effectuées par le filtre (dont l'implantation est décrite plus bas) après compilation (génération de l'automate) et chargement dans la mémoire du filtre du code exécutable par le filtre (automate).

b) tri : celui-ci, nécessaire notamment pour effectuer un changement de format, n'étant pas de complexité linéaire n'est pas effectué par le filtre(*).

(*) on peut cependant imaginer un algorithme de tri interne d'un bloc par filtrage, si la mémoire du filtre est assez grande [44].

La traduction d'une opération sur fichier en opération sur bloc, consiste 1) à générer l'automate s'il s'agit de filtrage, 2) enchainer des opérations sur blocs entrecoupées de mouvements de blocs (entre mémoire de masse et mémoire RAM).

On trouvera dans [24,25] les algorithmes pour enchainer les opérations sur bloc, tant pour la Sélection/projection que pour les opérations binaires.

Les opérations binaires peuvent s'effectuer par filtrage parce que les fichiers sont triés : il suffit d'ajouter un deuxième tampon source et un comparateur de caractères ; les fichiers étant triés, l'opération est linéaire par rapport à la taille des fichiers (voir Section 3.4.3).

Les changements de format se font par tri-fusion : alors que le tri interne d'un bloc se fait par logiciel, la partie fusion sur blocs triés se fait par filtrage (V-Union).

3.4. Architecture matérielle de VERSO

3.4.1. Description générale

Le prototype VERSO en cours de réalisation est constitué d'un ordinateur banalisé SM 90 auquel on rajoute un Module d'échange (ME) spécialisé conçu et réalisé à l'INRIA qui s'interconnecte entre la mémoire de masse (un ou plusieurs disques à tête mobile) et le bus de la SM 90.

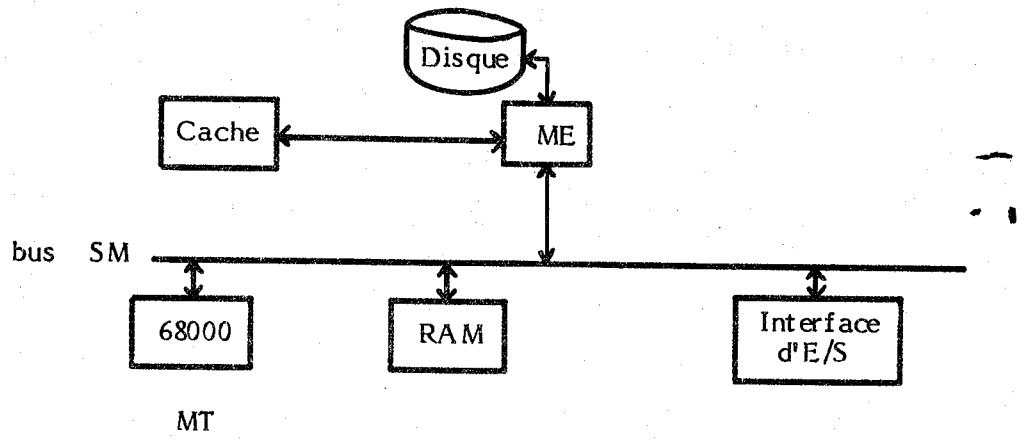


Figure 3.3. Architecture de VERSO

Comme le montre la figure 3.3., VERSO comprend les éléments suivants :

a) **partie SM 90**

On trouve le Bus SM, le module de traitement (MT) comprenant i) un processeur Motorola 68000, ii) sa mémoire locale et iii) son unité gestion mémoire (UGM), de la mémoire RAM et une interface de communication avec l'extérieur (communication avec une machine hôte, ou un réseau local, ou des terminaux).

L'UGM du MT lui permet d'accéder à toute mémoire sur le bus : mémoire commune, mémoire du ME, mémoire de l'interface, mémoire locale au MT ou à d'autres MT.

Il est possible de brancher sur le bus SM plusieurs MT et ainsi d'introduire un certain pipelinage des fonctions SGBD. Pour l'instant, l'implantation du SGBD de haut niveau n'utilise qu'un seul MT, le logiciel étant développé en Pascal et C sous UNIX.

Le MT a la responsabilité des tâches suivantes

i) dialogue avec l'extérieur ;
ii) dialogue avec le ME (commande des E/S disque, commande du filtre, lecture des résultats) ;
iii) couches relation, V-relation et fichiers. Il est prévu de déporter la couche fichiers sur le ME ;
iv) gestion des transactions, contrôle de la concurrence et reprise sur pannes ;
v) compilation (génération des automates) et tri interne d'un bloc.

b) **partie "Disque Intelligent" : le ME**

Le ME a essentiellement pour tâche la couche bloc, c'est-à-dire le filtrage d'un bloc.

Une extension mémoire de 512Koctets, improprement appelée CACHE est connectée au ME et n'est pas visible du MT. Outre son interface suivant la norme du bus SM, le ME comprend un contrôleur

(C), un filtre (F), un tampon Source (S) et un tampon Cible (Ci), chacun de 32 Koctets (deux fois la taille d'une piste) et une mémoire RAM(MF) pour stocker les automates (Figure 3.4)

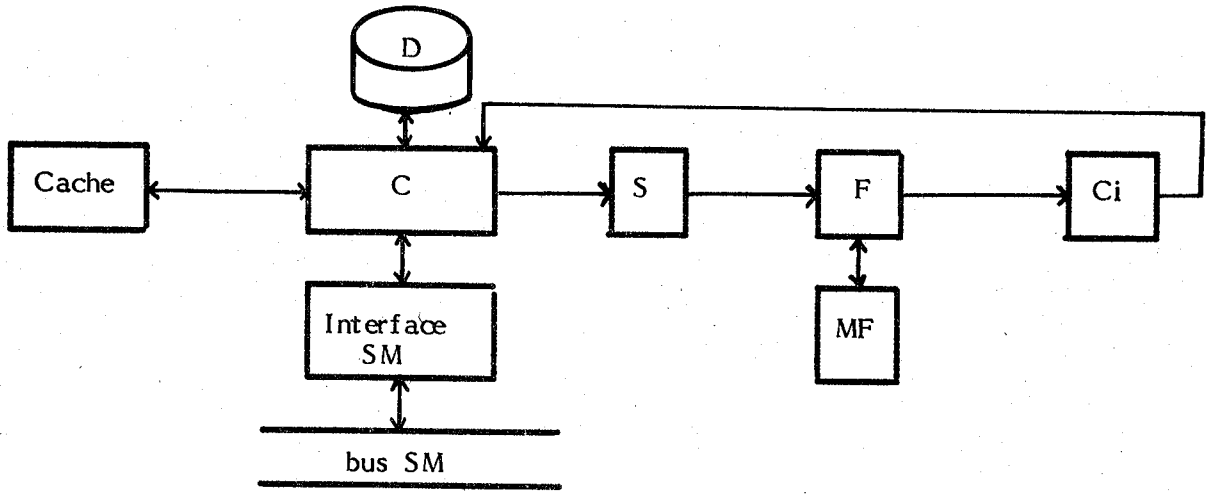


Figure 3.4. Description du ME

Le contrôleur réalisé à partir de micoprocesseurs bipolaires en tranches AMD xxxx est microprogrammable. Il est responsable des E/S disque, des transferts de données entre le Disque, le Cache, les tampons S et Ci, du chargement de l'automate dans MF et du contrôle du Filtre.

Il pourra éventuellement accueillir la couche "fichier" si sa mémoire de microprogrammes est de taille suffisante.

Le Filtre F est aussi réalisé avec des microprocesseurs AMD xxxx et de la logique cablée. Il balaie les données à filtrer dans S, écrit le résultat dans Ci. S est chargé soit du disque ou du Cache, soit d'une mémoire de la SM 90. Ce chargement peut se faire en parallèle avec le filtrage (lecture de S par F). C_i est vidé soit sur le Disque (résultats permanents) ou dans le cache (relations temporaires) soit vers la SM 90 (relation de type Sortie). On décrit maintenant successivement l'architecture du Filtre et celle du contrôleur.

3.4.2. Le Filtre

Une fois l'automate chargé, le contrôleur lance simultanément le chargement d'un bloc dans S et le Filtre. Un ou plusieurs blocs de 16K octets sont successivement chargés dans S. Le filtre n'a pas besoin d'attendre que S soit complètement chargé. Lorsqu'une centaine d'octets sont chargés, le Filtre commence à balayer S. Un mécanisme matériel bloque le filtre s'il va plus vite que le processus de chargement.

Trois autres événements arrêtent le filtre :

a) la fin de fichier est rencontrée (fin de filtrage),
b) le tampon Source a été entièrement lu : une interruption déclenche le chargement du bloc suivant,
c) le tampon Cible est plein : une interruption déclenche le vidage de Ci.

S peut être décomposé en deux tampons logiques S_1 et S_2 de 16 Koctets chacun, un pour chaque source d'une opération binaire (voir Figure 3.5). Le filtre analyse un caractère lu dans S_1 ou S_2 et écrit éventuellement le résultat dans Ci.

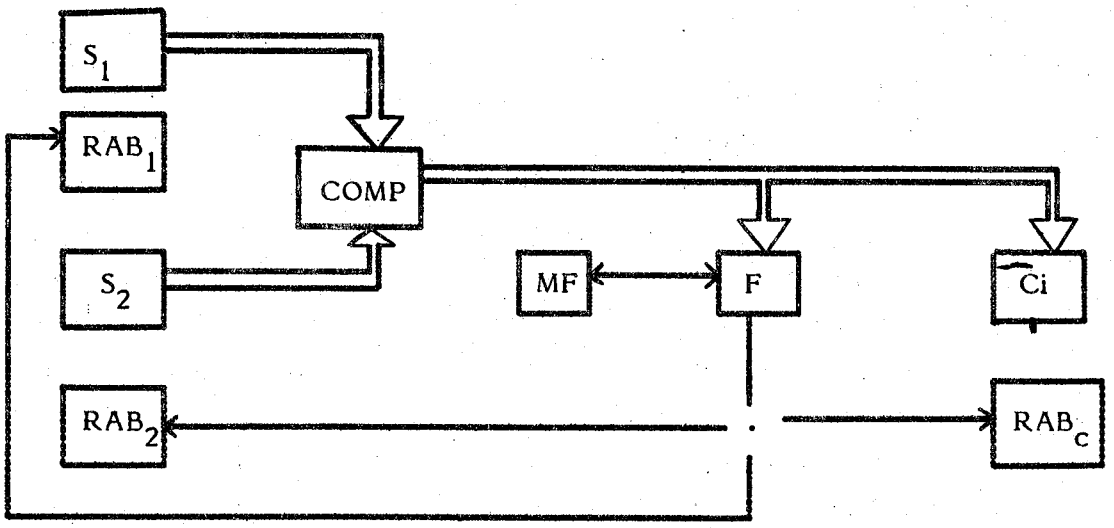


Figure 3.5. Filtrage pour les opérations binaires

Un comparateur (COMP) permet au filtre de lire le plus petit des caractères lus dans S_1 et S_2. Les fonctions de sortie de l'automate consistent schématiquement en la mise à jour des registres RAB_1, RAB_2 (où lire les caractères source suivants) et RAB_c (où écrire le caractère suivant).

La mémoire MF du filtre est une matrice de 256 lignes (états) et 256 colonnes (caractères). Chaque couple (état, caractère) adresse un mot de 32 bits dans MF qui contient l'état suivant (n° de ligne) et l'adresse d'une instruction (fonctions de sortie) dans une mémoire de microcommandes MAPMACRO. Cette instruction est interprétée et exécutée avant de passer à l'état suivant (Figure 3.6)

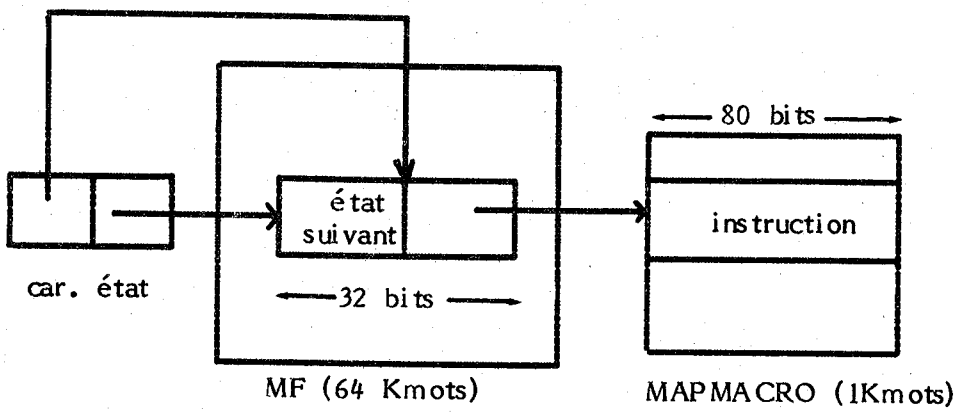


Figure 3.6. Le Filtre

La mémoire MAPMACRO ne change pas d'une requête à l'autre. Elle a été microprogrammée pour l'application.

Un cycle du filtre commence quand, dans un état donné, un caractère est lu et se termine juste avant la lecture du caractère suivant. En général la fonction de sortie est constituée d'une seule microinstruction de MAPMACRO (durée maximale d'interprétation et d'exécution 550 ns). Il peut arriver qu'une fonction de sortie complexe nécessite le déroulement de plusieurs microinstructions. Cependant, le temps moyen de cycle est estimé à 400 ns. Chaque microinstruction est divisée en quatre champs, interprétés et exécutés en parallèle. Un microprocesseur AMD 2910 est utilisé pour le séquencement.

3.4.3. **Opérations binaires** (voir [6] pour plus de détails)

Pour convaincre le lecteur qu'avec quelques fonctions supplémentaires, il existe un mécanisme par automate fini capable d'effectuer les opérations binaires, on prend l'exemple de la V-union de deux fichiers de format $(NOM)^*$.

On a besoin de deux piles supplémentaires S1 et S2 d'adresses dans les tampons Source S1 et S2 et des ordres suivants :

(11) Marquer Pointeur Source 1 : EMPILER RAB1 dans S1

(12) Marquer Pointeur Source 2 : EMPILER RAB2 dans S2

(2) Incrémenter les Pointeurs Source et Cible : cette instruction compare les 2 caractères pointés par RAB1 et RAB2 dans les deux tampons source, écrit le plus petit en séquence dans le tampon cible et incrémente le pointeur Cible et le pointeur Source correspondant au plus petit caractère.

(31) Effacer Marque : DEPILER S1

(32) Effacer Marque : DEPILER S2

(41) Restaurer RAB1 : RAB1 ← SOMMET de S1

(42) Restaurer RAB2 : RAB2 ← SOMMET de S2

On suppose pour simplifier l'exposé que les valeurs d'attributs sont des mots de longueur 1 caractère.

L'automate dont le diagramme de transition et les ordres associés à chaque transition sont donnés ci-dessous permet la fusion des deux fichiers de format $(NOM)^*$.

Bien qu'il ne s'agisse pas à proprement parler d'un automate fini (il existe deux têtes de lecture), on se ramène au cas d'un automate fini en considérant que dans la tête de lecture unique, il y

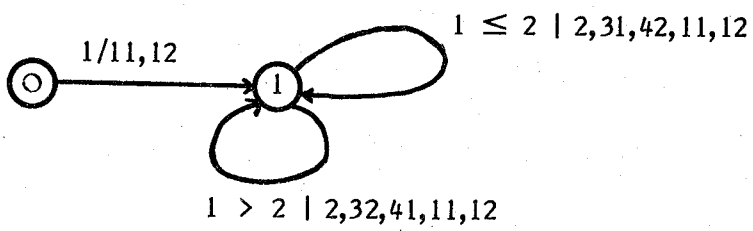
a soit un caractère lu dans l'une des deux sources, soit le résultat de la comparaison des caractères lus dans les deux sources.

Dans cet exemple simple, l'automate n'a qu'un état (à part l'état initial) et 2 transitions possibles :

$1 \leq 2$: le caractère le plus petit a été lu dans la source 1

$1 > 2$: le caractère le plus petit a été lu dans la source 2.

A chaque transition, une séquence d'ordres est associée dont la liste a été donnée plus haut



Prenons par exemple le cas $1 \leq 2$: après écriture du plus petit des caractères (Source 1, ordre (2)), on efface la marque dans la Source 1 (ordre (31)), on restaure le pointeur dans la Source 2 (ordre (42)) puisqu'il faudra comparer ce caractère au caractère suivant dans la Source 1 et on empile les pointeurs (ordres (11) et (12)).

Dans le cas général (fichiers de format quelconque), un certain nombre d'ordres supplémentaires est nécessaire. Le lecteur est reporté à [6] qui décrit le mécanisme par automate pour la V-union et la V-intersection dans le cas général.

3.4.4. Le Contrôleur (voir lexique)

Le Contrôleur comprend :

i) un contrôleur proprement dit microprogrammable composé d'une ALU (AMD 29116), d'un séquenceur (AMD 2910) et d'un contrôleur d'interruptions (AMD 2914)

ii) 2 DMA, un pour le Disque et un pour le Cache

iii) un formatteur

a) Le contrôleur

Il lance le filtre, prépare les transferts en initialisant les DMA et contrôle le formatteur, le filtre et les E/S disque grâce aux interruptions.

b) Les DMA

La figure 3.7 illustre schématiquement le parallélisme entre transferts de données et filtrage que permet l'architecture du ME :

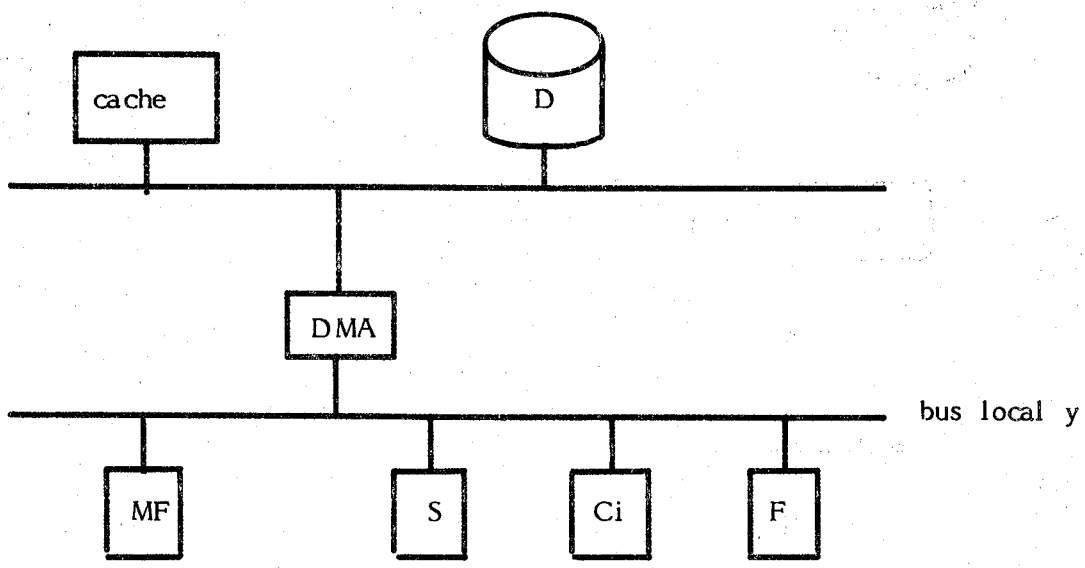


Figure 3.7. Parallélisme entre transferts et filtrage

Pendant que le filtre F travaille, on peut effectuer un transfert entre le Cache et D. En fait ce parallélisme est mal utilisé.

Le DMA (chargement de S et éventuellement vidage de Ci) et le Filtre se partagent les ressources bus local y et mémoires.

L'occupation du bus y par le DMA disque est minimisée grâce à l'utilisation de deux files FIFO d'une centaine d'octets (Figure 3.8). En effet le Disque étant lent, le formatteur travaille continuellement tandis que le DMA écrit ou lit dans la mémoire des trains de données à une vitesse plus grande.

Comme tout DMA, le DMA cache est initialisé par le contrôleur qui spécifie une adresse CACHE et un nombre d'octets à transférer. Pendant le transfert entre le CACHE et le formatteur le DMA génère une adresse CACHE. Le Cache est vu par le formatteur comme une mémoire séquentielle similaire au Disque.

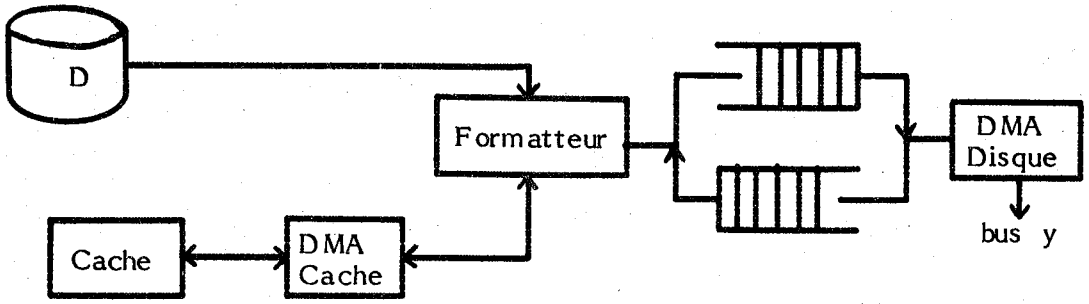


Figure 3.8. Transfert des données

c) Le formatteur

Il fait la conversion série/parallèle, il synchronise les FIFO avec le Disque, fait la correction d'erreurs ainsi que le formattage du Disque.

Pour terminer, notons que le ME (+ le Cache) est implanté sous forme de 7 cartes testées et en cours de microprogrammation.

On étudiera dans le chaptire 5 une implantation logicielle du mécanisme de filtrage : le ME tel que nous venons de le décrire est remplacé par un ME au standard SM 90 sur lequel on implante la couche fichier et le filtrage des blocs.

CHAPITRE 4

ARCHITECTURES DE MBD :

Machines à filtre

4. - ARCHITECTURES DE MBD : MACHINES A FILTRE

La complexité et la médiocrité des performances des logiciels SGBD ont été à l'origine d'une intense activité de recherche et de développement d'architectures spécialisées à la gestion des bases de données.

Cette activité qui marque actuellement le pas a commencé un peu avant 1975 et a culminé vers 1980-81. Nous reviendrons sur les raisons d'un déclin qui n'est que temporaire malgré l'affirmation de Boral et Dewitt [23] : "les Machines Bases de Données : une idée révolue ?".

Il ne s'agit pas ici de faire un panorama exhaustif de l'activité dans ce domaine. Un état de l'art - complet à l'époque et qui retrace bien la première partie de cette activité - est celui de Maryanski [35]. Une excellente introduction au problème est celle de Bancilhon [9]. A notre connaissance, l'article de synthèse le plus récent sur le sujet est celui de Schweppe, Stiege [46].

Ce Chapitre a pour objectif de décrire les différentes approches de Disques Intelligents (Machines à filtre) afin de mieux mettre en perspective les choix architecturaux de VERSO. On reprendra brièvement la taxonomie de Bancilhon [9], on survolera les autres approches que le Disque Intelligent ainsi que la préhistoire du filtrage représentée par les dispositifs appelés "cellular logic devices", sujets bien décrits dans les ouvrages déjà cités. Ensuite on étudiera en détail les différentes approches de Filtrage en faisant le tour à la fois des projets de recherche et des produits commerciaux. Enfin on discutera des architectures qui ont - à notre sens - une chance de devenir réalité dans un avenir très proche.

4.1. Architectures MIMD vs Disques Intelligents

Il est communément admis qu'il existe deux grandes classes d'architectures de MBD.

La première que nous appellerons Architectures MIMD regroupent des systèmes multiprocesseurs qui veulent tirer parti "à tout crin" du parallélisme, pour apporter une solution au problème des performances. Un certain nombre de processeurs indépendants effectuent des tâches différentes (M.I) sur des données différentes (M.D).

La seconde regroupe les systèmes qui rapprochent plus ou moins d'Intelligence auprès de la mémoire de masse, d'où la vocable de Disque Intelligent. Ces systèmes bien qu'ils ambitionnent également de tirer profit d'un certain parallélisme à ce niveau sont plus modestes : leur objectif est l'amélioration des performances en optimisant uniquement la fonction d'accès aux données ; on profite du transfert du Disque vers la mémoire pour effectuer un certain traitement : telle était l'idée initiale du filtrage au vol qui a donné naissance aux premiers prototypes de MBD : CASSM, RAP ou RARES, appelés "cellular logic devices".

Enfin, citons pour mémoire l'approche "processeurs associatifs" maintenant abandonnée. Le prototype STARAN [12] en est un exemple. On veut bénéficier de la puissance de nombreux processeurs associatifs pour traiter un grand volume de données. En vue d'utiliser à fond la puissance de tels calculateurs, il faut alimenter un élément de traitement élémentaire à partir de la mémoire de masse à un débit impressionnant. On est limité par les E/S et l'utilisation des processeurs associatifs est mauvaise. D'où l'abandon d'une telle approche.

Il faut également mentionner parmi les précurseurs des MBD, le projet MAGE [40].

Dans les deux paragraphes suivants, on survolera l'approche MIMD ainsi que la préhistoire des Disques Intelligents. Pour plus de détails, le lecteur doit consulter [9, 35, 46].

4.1.1. Architectures MIMD

Suivant [9], les prototypes qui ont suivi cette approche se distinguent suivant trois types de critères :

a) **le degré du parallélisme :** Y a-t-il seulement entrelacement de transactions, ou également parallélisme entre étapes (consultation ou mise à jour) voire entre sous-tâches d'une même étape, enfin en sous-traitant une sous-tâche à plusieurs processeurs ?

b) **le degré de spécialisation :** Les processeurs peuvent être banalisés (e.g. Direct) ou spécialisés à une fonction (e.g. SABRE).

c) **le degré d'homogénéité :** Les processeurs spécialisés à une fonction peuvent être des processeurs standards (DBMAC,SABRE) ou conçus à cet effet (RDBM).

Il n'existe actuellement que deux produits commerciaux qui ont annoncé avoir suivi cette approche très ambitieuse et relativement complexe à mettre en oeuvre. Il s'agit de DORSAL développé par la Société française COPERNIQUE [2] et du système DBC/1012 commercialisé en Avril 84 par la société californienne TERADATA [48].

Les principaux prototypes de recherche réalisés ou en cours de réalisation sont les suivants :

RDBM est une machine développée à l'Université de Braunschweig [3,47]. On retrouve autour d'un bus sophistiqué un disque intelligent (avec plusieurs processeurs de sélection), un processeur de mémoire, un processeur de tri et un processeur de jointure. Tous ces processeurs spécialement conçus et réalisés à cet effet, sont en cours de réalisation.

DBMAC en cours de conception au CNR Rome est une machine multiprocesseurs banalisés connectés à deux bus et pouvant supporter plusieurs stratégies de parallélisme [36,37].

SABRE, réalisé à l'INRIA est une machine monoprocesseur qui devrait donner naissance à une version multiprocesseurs où chaque processeur serait spécialisé fonctionnellement. A la différence de RDBM, chaque fonction serait implantée au moyen d'un processeur standard [26].

DBC [11] est un projet de l'Université d'Ohio qui n'a pas dépassé le stade "papier".

Enfin DIRECT [20] est le seul prototype opérationnel (1981) : m processeurs banalisés (LSI11) accèdent à n modules mémoires (CCD) au moyen d'une matrice d'interconnection. Les modules CCD sont directement chargés des mémoires secondaires. Un contrôleur communique avec l'ensemble des processeurs au moyen d'un bus unique où sont multiplexés les résultats d'une requête. Ce contrôleur dialogue avec un frontal où sont implantées les couches de haut niveau de Ingres sous Unix.

Une autre tendance est à suivre, celle des Machines à flots de données. Plusieurs prototypes ayant une approche "Flot de Données" sont en cours de réalisation, au Japon (e.g. [51]).

4.1.2. Disques Intelligents : la préhistoire

Dès le début des années 1970 [17,49], on expérimente les premières mémoires associatives : on associe à chaque élément de mémoire rotative (e.g. une piste de disque), un tampon et une unité de traitement (filtrage). Ces unités de traitement peuvent faire des opérations simples (reconnaître des champs, évaluer un critère de sélection) éventuellement marquer d'un bit le n-uplet satisfaisant le critère : on lira lors d'une deuxième passe les n-uplets marqués.

CASSM est le premier prototype [17]. Conçue en 1970, donc avant l'existence du modèle relationnel, la machine a un processeur par piste de disque et traite des fichiers hiérarchiques. L'ensemble

des processeurs communiquent via un bus avec un ordinateur frontal. L'évaluation d'une condition peut nécessiter plusieurs tours de piste. Des bits de marquage sont utilisés à cet effet dans les données.

RAP [41] un peu plus récent a été développé à l'Université de Toronto. Contrairement à CASSM, les cellules de mémoire rotative jouent le rôle d'anté-mémoire chargées directement de la mémoire de masse (il semble que cette connexion n'ait jamais été réalisée). Chaque processeur a une tête de lecture et une tête d'écriture. Le n-uplet lu et stocké dans le tampon est analysé par le processeur (après autant de tours de piste q'il y a de conditions élémentaires dans le critère de sélection). Grâce à la tête d'écriture, le n-uplet est éventuellement réécrit après modification.

Enfin il faut mentionner de cette époque RARES [34].

4.2. Disques Intelligents : Machines à filtre

A partir de 1975, on voit apparaître des Disques intelligents plus sophistiqués e.g. SURE [33], TREFLE [22], certains utilisant le filtrage par automate fini (Trèfle ou Machine CIA [43]).

Ces processeurs de filtrage sont ensuite utilisés dans des projets de Machines Relationnelles complètes non sans avoir subi des modifications. SURE donne naissance à RDBM déjà cité, Trèfle a été remanié pour VERSO ou SCHUSS [27].

Enfin certains produits commerciaux voient le jour qui tous ont prévu un élément matériel spécialisé conçu pour optimiser les performances ; pour certains il n'est pas tout à fait clair que le matériel spécialisé soit opérationnel faisant de ces produits des "machines" logicielles.

Parmi les produits commerciaux, citons CAFS [4,16] développé par ICL, IDM commercialisé par Britton-Lee et deux produits français SARI [32] et DORSAL 32 [2] respectivement commercialisés par SINTRA et COPERNIQUE.

4.2.1. Taxonomie des Machines à Filtre

Quatre critères nous ont paru essentiels, chacun correspondant à un choix architectural parmi deux ou plusieurs solutions. Il s'agit

i) du mode de connection du filtre à la mémoire de masse (filtrage au vol ou pas),

ii) du mode de traitement (balayage séquentiel ou accès "direct"),

iii) comme pour les machines MIMD du degré d'homogénéité du ou des processeurs et enfin

iv) du type de parallélisme utilisé.

a) Mode de connection du filtre à la mémoire de masse

Comme CASSM, RAP les premiers systèmes (TREFLE,SURE,CAFS) ont fait du filtrage au vol :

Le processeur de filtrage est synchronisé avec le transfert disque. Le filtre de vitesse au moins égale à celle du Disque scrute les caractères au fur et à mesure qu'ils sont lus. Dans Trèfle, par exemple, (Fig. 4.1), le Filtre ne peut à tout instant avoir que quelques caractères de retard.

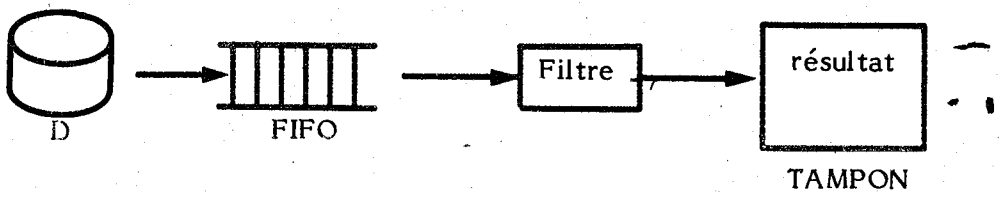


Figure 4.1. Filtrage au vol : l'exemple de TREFLE

Le filtrage au vol a été depuis abandonné. Même si le Filtre est plus rapide que le transfert Disque, on désynchronise le filtrage du transfert en interconnectant un ou plusieurs tampons entre le Disque et le Filtre. La taille minimale du tampon correspond à l'unité d'adressage et de balayage. S'il n'y a qu'un tampon, le filtre doit attendre que le tampon soit chargé avant de commencer son filtrage.

L'alternative est soit d'implanter un mécanisme coûteux tel que celui de VERSO où le filtre peut lire les données du tampon, parallèlement à son chargement (voir Chapitre 3) soit d'utiliser plusieurs tampons : tandis que les données sont chargées du disque dans un tampon, le filtre traite un autre tampon (Fig. 4.2).

Les avantages d'une telle approche sont les suivants :

i) le processeur de filtrage peut être temporairement plus lent que le transfert disque,

ii) on délie le filtre de l'obligation de scruter les données en séquence. Il peut, en particulier, revenir sur un caractère déjà lu,

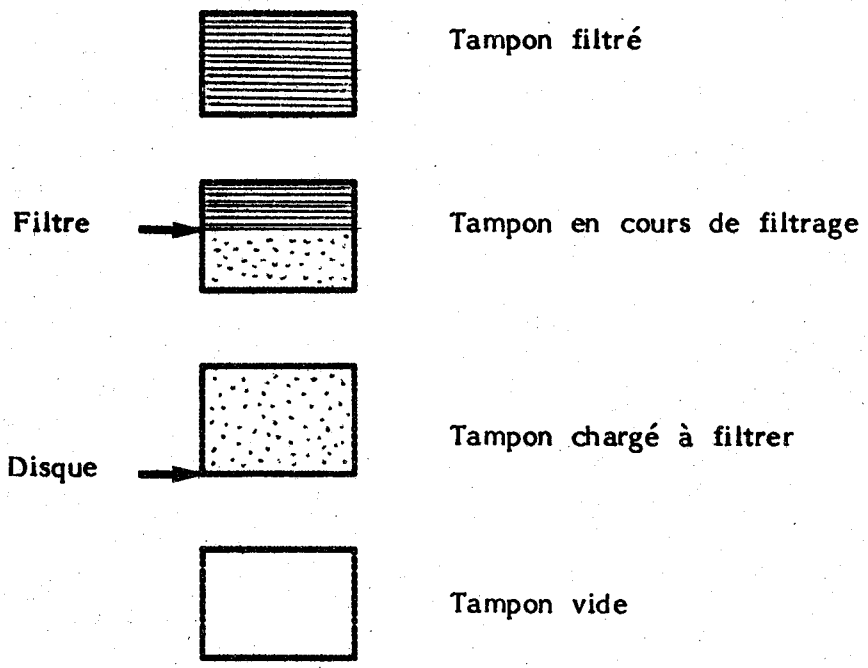


Figure 4.2. Tamponnage : l'Exemple du Filtrage logiciel dans VERSO (chapitre 5)

b) Mode de traitement

Le filtrage "au vol" est forcément un balayage séquentiel. Si le tamponnage existe, toute méthode de traitement est autorisée aussi classique soit-elle. Par balayage séquentiel, on distingue entre 1) le filtrage par automate fini programmable (TREFLE,SCHUSS,VERSO) et 2) les traitements "ad hoc" : e.g. SARI a un automate cablé de reconnaissance de chaînes de caractères, SURE nécessite la connaissance préalable des frontières de champ.

La souplesse est la principale qualité du filtrage par automate fini programmable. Il permet un format variable des données. Il ne nécessite pas une structure figée des fichiers et une connaissance préétablie de cette structure. Celle-ci est programmable (à chaque fichier correspond un automate de reconnaissance).

c) degré d'homogénéité des processeurs

Comme pour les machines MIMD, on trouve des conceptions où le filtre est implanté sur un processeur banalisé (Verso a une version du Filtre en logiciel, IDM a d'abord été commercialisée sans l'accélérateur matériel annoncé) mais la plupart des systèmes ont conçu un processeur ad-hoc ne seraient ce que les processeurs de filtrage au vol obligés de "tenir" la cadence imposée par le transfert Disque. Toutefois on peut se demander si la tendance n'est pas d'abandonner des processeurs matériels spécialisés coûteux à développer et d'utiliser au contraire des processeurs standards moins puissants, mais dont les performances s'améliorent d'année en année. Un autre argument en faveur des processeurs standards est que la puissance du processeur de filtrage est extrêmement mal utilisée (voir [25] et le chapitre 6) quand il est plus rapide que le transfert Disque. C'est qu'on est limité par les E/S (I/O bound) non seulement pour les opérations binaires mais également pour la Sélection/Projection.

d) degré de parallélisme

Certaines machines ne traitent qu'une requête à la fois sur toutes les données (SISD). C'est le cas de TREFLE, VERSO, SCHUSS, IDM. Elles utilisent toutes comme mémoire de masse la technologie Disques à tête mobile et un seul filtre. Elles appartiennent à la catégorie dite "Processeurs par Disque" (PPD) [21].

Les architectures MISD sont celles où un même flot de données est traité par plusieurs filtres, chacun effectuant une opération différente. C'est le cas de CAFS et RDBM. Ces deux machines utilisent une organisation PPD. Tandis que CAFS traite plusieurs requêtes avec un seul filtre, RDBM par contre utilise une connexion mémoire par tampons gérés dynamiquement et attaqués par plusieurs filtres matériels.

Les architectures SIMD effectuent la même opération mais sur plusieurs pistes en parallèle. Pour ce faire, on peut envisager une organisation "Processeur par Piste" (PPP) ou une organisation "Processeur par Tête (PPT), toujours suivant le terminologie de [21]. L'organisation PPP consiste à filtrer en parallèle toutes les pistes d'un disque à têtes fixes. En une révolution, le disque entier est balayé. Cette idée extrêmement complexe et coûteuse à mettre en oeuvre a été abandonnée.

L'organisation PPT par contre est utilisée dans SARI et SURE. Elle consiste à lire en parallèle toutes les pistes d'un même cylindre de disques à tête mobile : en une révolution, on a balayé un cylindre.

Cette idée correspond à un bon compromis coût/performances mais présente certaines difficultés de réalisation, notamment en ce qui concerne la synchronisation entre les têtes de lecture.

On peut réaliser une architecture SIMD en répartissant les données sur plusieurs disques et en utilisant plusieurs filtres (de type VERSO ou SCHUSS par exemple), chaque filtre "attaquant" un disque à tête mobile (organisation PPD).

On trouvera dans [21] une comparaison des 3 organisations (PPD, PPP et PPT).

Le paragraphe suivant étudie avec plus de détails les plus caractéristiques parmi les Machines à filtre (Disques Intelligents) citées plus haut.

4.2.2. Exemples de Machines à Filtre

4.2.2.1. TREFLE [5]

Machine développée à l'INRIA (75-78), TREFLE a été conçue pour faire de la consultation rapide de gros volumes de données. C'est un processeur de filtrage au vol. Les données lues du disque transitent à travers une mémoire FIFO que le filtre à automate fini lit caractère par caractère. Les caractères gardés sont écrits dans un tampon cible (Fig. 4.1). Le programme du filtre (automate) est une séquence d'instructions de la forme :

CAR	f_1	E_1	f_2	E_2	f_3	E_3

dont la sémantique est la suivante :

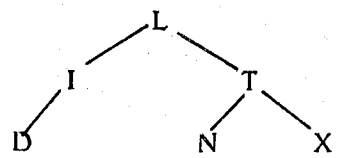
> Si le caractère dans la tête de lecture est égal à CAR, alors l'instruction suivante est E_1 et la fonction de sortie à effectuer est f_1.
>
> Si le caractère lu est inférieur à CAR, exécuter f_2 et aller à E_2.
>
> Si le caractère lu est supérieur à CAR, exécuter f_3 et aller à E_3.

Par exemple, la séquence d'instructions suivante permet de reconnaître MAD $\vee$ MAI $\vee$ MAL $\vee$ MAN $\vee$ MAT $\vee$ MAX.

On a éliminé les champs f_1, f_2, f_3 (fonctions de sortie, e.g. écrire le caractère reconnu dans le tampon cible).

Etat	N° instruction courant	caractère à reconnaître	N° instruction si =	N° instruction si <	N° instruction si >
1	1	M	2	10	10
2	2	A	3	10	10
3	3	L	9	4	6
	4	I	9	5	10
	5	D	9	10	10
	6	T	9	7	8
	7	N	9	10	10
	8	X	9	10	10
4	9	succès			
5	10	échec			

Dans cet exemple, la première transition (état 1 ⟶ état 2 ou état 5) prend un cycle élémentaire de la machine où le cycle est la durée d'exécution d'une intruction). Il en est de même pour la deuxième transition (état 2 ⟶ état 3 ou état 5). Par contre la troisième transition (état 3 ⟶ état 4 ou état 5) dure de 1 à 3 cycles. En effet elle correspond à la recherche dichotomique suivante :



Dans le pire des cas, une transition dure $Log_2\ \alpha$ cycles, où α est la taille de l'alphabet.

Donc avec un alphabet de 26 lettres et 10 chiffres, on aura au maximum 6 instructions à dérouler pour une transition de l'automate : en fait on n'aura que rarement plus de 3 instructions. Sachant que le débit du disque est de 10 Mbits/seconde (800ns/caractère), il suffit que le temps de cycle soit de l'ordre de 200 à 300 ns. Ce sont effectivement les performances estimées de la

logique cablée de Trèfle. La FIFO (Fig. 4.1) garantit qu'en cas de transition lente, aucun caractère lu du disque n'est perdu.

Comparée à une représentation de l'automate par Matrice de transition (e.g. VERSO, voir Chapitre 3) la représentation de l'automate dans TREFLE est plus compacte mais donne des transitions plus lentes (une transition dans VERSO dure toujours un seul cycle, soit 400ns environ).(*)

Les automates étant plus compacts, le temps de chargement des automates est plus court. Cette structure-plus complexe- est bien adaptée à des applications où les automates sont pseudo-linéaires (le nombre de transitions à partir d'un état est rarement supérieur à 1).

L'expérience de TREFLE est à l'origine de la conception de VERSO et de SCHUSS que nous décrivons maintenant.

4.2.2.2. SCHUSS [27]

Développé au Centre de Recherche de Bull, SCHUSS est un processeur de filtrage destiné à l'origine à être utilisé par le buroviseur KAYAK [39] pour la recherche de documents. Mais l'un des soucis des concepteurs est de rendre leur processeur utilisable par plusieurs applications dont les SGBD.

De conception analogue à VERSO, SCHUSS est connecté à un disque à tête mobile au moyen de deux tampons en bascule : Le filtre à automate fini analyse séquentiellement les données d'un tampon (1KOctets) tandis que l'autre est chargé du disque. Les données filtrées partagent la même mémoire RAM (64 Koctets) que le programme de filtrage (automates).

Sur le buroviseur, tout le système (filtre, tampons, mémoire) est connecté à un port d'E/S parallèle du contrôleur ISBC 215 lequel est responsable des transferts entre le DISQUE et un MULTIBUS où est branché l'ordinateur hôte.

(*) Voir également [44] pour une étude du filtrage par automate et de la représentation des automates.

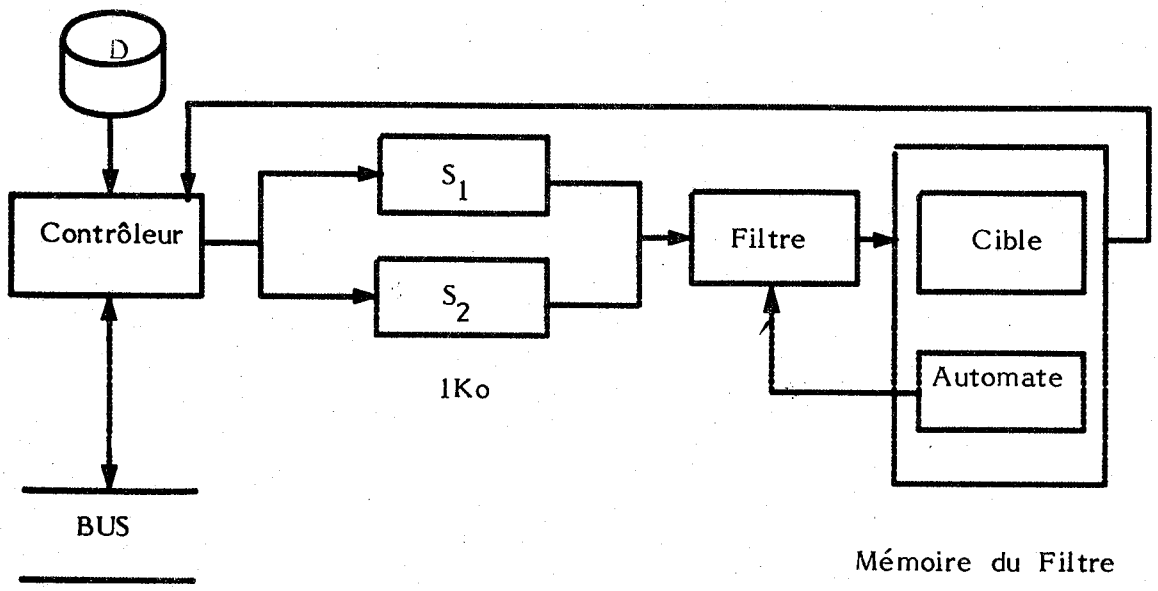


Figure 4.3. SCHUSS

Le fait d'utiliser un port parallèle d'un contrôleur standard a permis aux concepteurs de se concentrer sur la réalisation rapide du filtre sans souci des performances médiocres de ce contrôleur. Pour pallier à cette lenteur du contrôleur, un nouveau contrôleur a dû être développé, comme dans VERSO.

Une instruction élémentaire du filtre a, comme dans VERSO plusieurs champs (elle a 112 bits de large) et est constituée de plusieurs actions déroulées en parallèle.

A la différence de VERSO qui n'a qu'un algorithme de filtrage "cablé" performant, SCHUSS a un filtre microprogrammable implanté au moyen de 2 ALU AMD 2903 et 2900. On peut donc adapter à l'application l'algorithme de filtrage : Par exemple on peut choisir entre une matrice de transition de type VERSO, des instructions à branchement dichotomique de type TREFLE, ou tout autre représentation pour l'automate.

De plus l'automate de filtrage étant stocké dans la mémoire RAM du filtre, il peut être modifié dynamiquement, ce qui peut être utile pour les opérations binaires.

4.2.2.3. SURE [33,47]

Réalisé à l'Université de Braunschweig (74-78) SURE est un prototype opérationnel utilisant une organisation PPT.

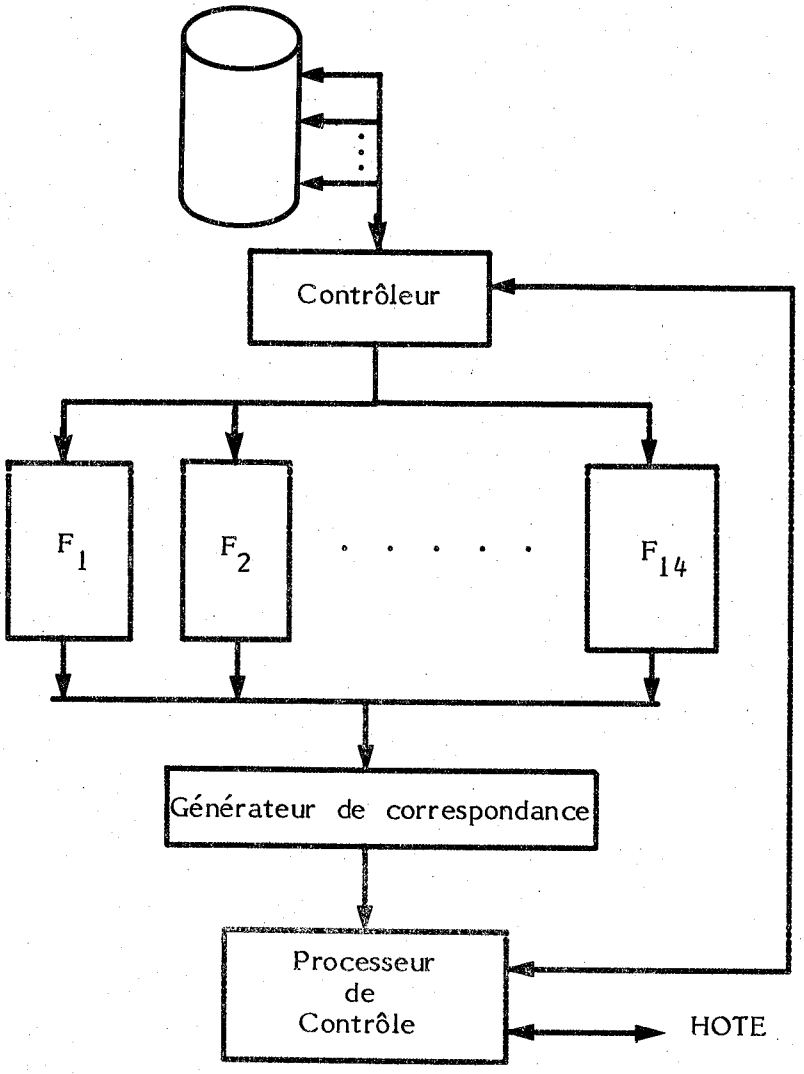


Figure 4.4 SURE

Un disque SIEMENS de 72 Moctets a été modifié pour permettre la lecture simultanée des 9 pistes d'un cylindre. Les flots de données sont multiplexés par le contrôleur permettant à la sortie un débit de 7,3 Moctets/seconde. Ce flot est analysé au vol simultanément par 14 filtres (chacune des 9 pistes est analysée indépendamment). Les filtres ont 137 ns pour reconnaître une chaine de caractères (une valeur d'attribut). Le générateur de correspondance en cas de succès délivre au processeur de contrôle, l'adresse du n-uplet ainsi qu'un vecteur de 14 bits (un par processeur). Les n-uplets satisfaisant la requête doivent être retrouvés par un 2e accès au disque. Le processeur de contrôle sert d'interface avec l'ordinateur hote. C'est lui qui charge les filtres avec les critères de sélection, lance la sélection, effectue la projection (accès séparé au disque).

4.2.2.4. IDM [9]

L'IDM 500 est une MBD relationnelle commercialisée par la compagnie américaine Britton & Lee.

La machine est configurée autour d'un bus rapide où sont connectés un Z8000 qui exécute le SGBD, de la mémoire, un contrôleur chargé des E/S avec un ensemble de disques a têtes mobiles, une interface avec l'extérieur et enfin un accélérateur [Fig. 4.5.].

Ce processeur spécialisé optionnel rapide joue le rôle de filtre.

Le logiciel SGBD est tiré de INGRES (il a été conçu par des membres de l'équipe INGRES de Berkeley). Il n'est pas clair si l'accélérateur annoncé est disponible sur le marché.

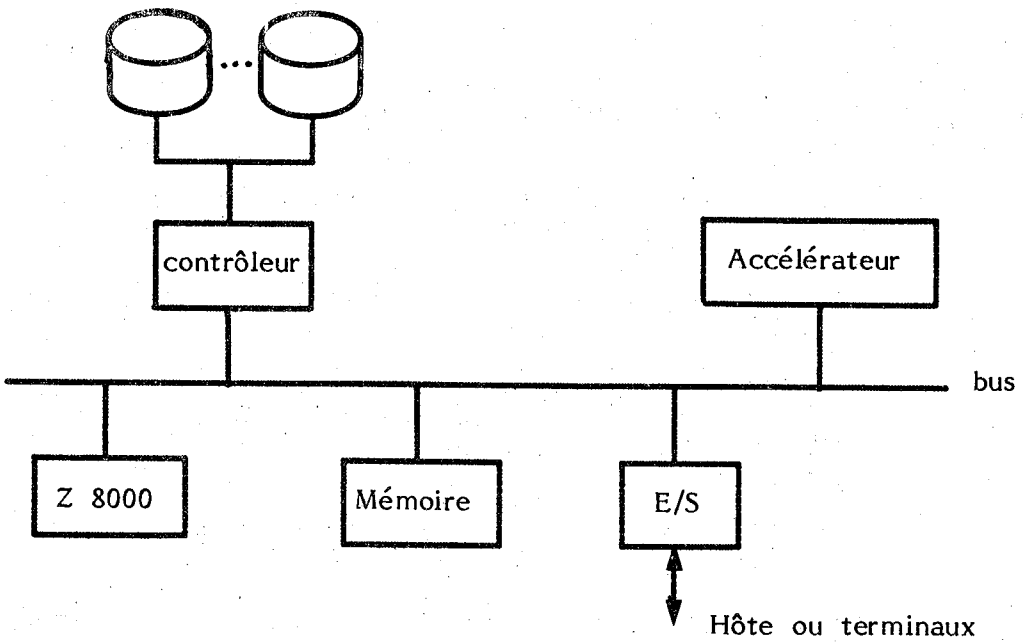


Figure 4.5. IDM 500

4.2.2.5. SARI [32]

Conçu par la société SINTRA pour la consultation rapide de fichiers {noms,adresse,téléphone} (bottin électronique) SARI est un filtre PPT. Il utilise

i) un disque AMPEX PPT de 300 Moctets dont il lit 4 pistes en parallèle (débit 1,2 Moctets/s par tête de lecture, 19 pistes/cylindre).

ii) un Filtre permettant les opérations suivantes "au vol" :

- recherche et extraction d'informations répondant à une fonction "logique" de la clé de recherche,

- effacement par marquage d'informations répondant à une fonction logique de la clé de recherche.

La clé de recherche est un concaténation de un ou plusieurs champs et a pour taille maxima 256 caractères.

Le processeur traite les fonctions suivantes :

ET, OU, SAUF, JOCKER, FIN DE MOT et SAUT.

Les fonctions ET OU et SAUF portent sur plusieurs clés.
Un OU ne porte que sur deux critères (idem pour le SAUF).
Le JOCKER placé au milieu ou en fin de clé remplace un ou plusieurs caractères indifférents.
FIN DE MOT est le séparateur de mot. Placé dans la clé, il indique que le caractère qui précède doit être le dernier du mot.
Enfin SAUT lie 2 mots de la clé qui doivent être obligatoirement adjacents.

Le Principe de filtrage est le suivant :

l'automate cablé fait du "pattern matching" sur la clé. La logique de contrôle et la logique de fonction permettent l'évaluation de la requête.

Le temps de cycle (temps de traitement d'un caractère) varie de 300 ns à 700 ns (cas défavorables du OU et du SAUF).

4.2.2.6. DORSAL 32 [2]

La Société COPERNIQUE vient de commercialiser une MBD navigationnelle et prévoit très prochainement d'en sortir une version relationnelle. Le DORSAL 32 est prévu pour être une machine multiprocesseurs à base de Processeurs Motorola 68000 appelée la machine de Gestion Logique (MGL) qui emploie pour ses E/S et une recherche rapide, une autre machine multiprocesseurs appelée la Machine de Gestion Physique (MGP), déjà commercialisée sous le nom de DIRAM 32.

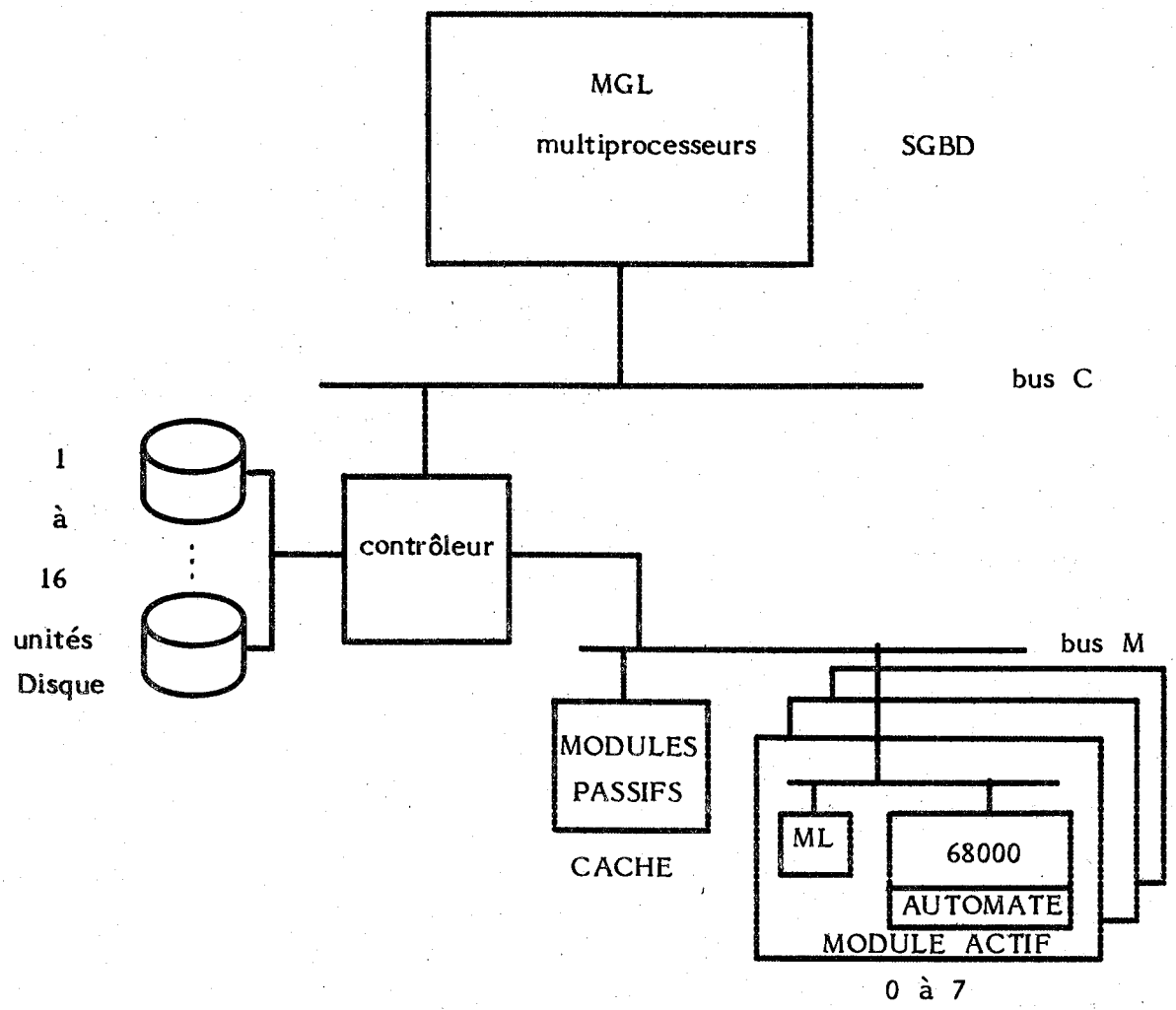


Figure 4.6. DIRAM 32 : La MGP de DORSAL 32

On n'étudie ici que la MGP qui est constituée [Fig. 4.6]

- d'un contrôleur intelligent
- de modules passifs pouvant aller jusqu'à 16 Moctets.
- de $n \leq 7$ modules actifs.

Le contrôleur sophistiqué permet à la fois le formattage du disque, les E/S vers la MGL et la gestion du Cache.

Les Modules passifs sont une mémoire de 1 à 16 Moctets effectivement utilisés en tant que CACHE avec politique de remplacement LRU, soit pour les E/S sans recherche associative, soit comme "Module passif" pour la recherche associative.

Chacun des 7 Modules actifs est constitué d'un processeur Motorola 68000, d'un automate cablé et de 256 Koctets de mémoire locale (ML).

L'automate cablé est utilisé, comme dans SARI, pour la reconnaissance rapide de chaines de caractères. Le 68000 par contre est programmé pour effectuer tout type de filtrage, depuis l'évaluation logique (ET, OU, SAUF) des requêtes (les champs étant reconnus par l'automate cablé) jusqu'aux fonctions statistiques, etc...

La requête est chargée dans la ML du module actif, ainsi que les données sources (lues du Cache ou du Disque) et les résultats du filtrage.

4.2.3. Discussion

Derrière le vocable de Machines à Filtre, on a - peut être arbitrairement - regroupé des architectures relativement conventionnelles auxquelles on rajoute un processeur, banalisé ou spécialisé, décentralisé près du Disque et chargé du filtrage rapide des données, bien que des architectures plus ambitieuses dites Multiprocesseurs peuvent également utiliser des Filtres (e.g. RDBM ou DORSAL 32).

Le Tableau 4.7 résume les caractéristiques des principaux Filtres étudiés.

La tendance actuelle va vers la simplification : au lieu de faire du filtrage au vol, on désynchronise les transferts par l'utilisation de tampons intermédiaires. Après avoir essayé d'augmenter le débit du disque (organisation PPT) on est revenu à une utilisation classique des disques à tête mobile (organisation PPD). Les techniques de filtrage par automate fini programmable - défendues ici - n'ont pas encore été utilisées dans des produits commerciaux.

En ce qui concerne l'implantation des Filtres, on peut se demander si elle nécessite des processeurs spécialisés coûteux à développer plutôt que des processeurs banalisés surtout lorsqu'une prédiction de l'amélioration globale sur le système apportée par telle ou telle implantation de la fonction de filtrage n'est pas disponible ?

Ce sont des interrogations de cet ordre qui ont conduit à la conception et la réalisation du filtrage en logiciel pour VERSO sur un Module d'échange de la SM 90 ayant comme processeur un INTEL 8086. (Chapitre 5). Le Chapitre 6 en comparant les Filtres matériel et logiciel de VERSO apportera quelques éléments de réponse à la question ci-dessus.

MACHINES	Filtrage au vol ?	Filtrage par automate ?	Homogénéité	Parallélisme
TREFLE	oui	automate fini programmable	Filtre cablé	SISD PPD
SCHUSS	non	automate fini programmable	ALU "en tranches" micro-programmées	SISD PPD
SURE	oui	non	Filtre cablé	SIMD PPT
IDM 500	non	non	Machine logicielle + accélérateur matériel	SISD PPD
SARI	oui	automate cablé (reconnaissance de clé)	Filtre cablé	SIMD PPT
DIRAM 32	non	automate cablé (reconnaissance de clé)	Multi processeurs banalisés + automate cablé	MIMD PPD
CAFS	oui	non	Processeurs de recherche et jointure cablés	SISD PPD
VERSO	non	automate fini programmable	Filtre cablé et contrôleur "en tranches" micro-programmé	SISD PPD

Figure 4.7. Caractéristiques des principaux filtres

4.3. Architectures multiprocesseurs faiblement couplées

4.3.1. Machines Bases de Données : Une Idée révolue ?

Les architectures multiprocesseurs mentionnées jusqu'à présent étaient bâties autour d'une répartition "fortement couplée" du traitement par parallélisme ou pipelinage.

L'utilisation de plusieurs filtres pour accélérer l'accès aux données et/ou de plusieurs processeurs pour traiter plusieurs requêtes en parallèle devraient considérablement augmenter les performances au prix d'une conception parfois extrêmement complexe et d'une mise en oeuvre très difficile.

C'est peut-être cette complexité qui explique le peu de succès actuel de ce type d'architecture qu'on ne retrouve pas dans les produits commercialisés ou annoncés.

Cependant le problème des performances reste crucial et le sera encore plus avec les nouvelles applications (e.g. systèmes de gestion de connaissances, systèmes de gestion d'imges et de textes).

C'est pourquoi si les recherches en "Machines Bases de Données" ont marqué le pas ces derniers temps, il faut s'attendre à un regain d'activité dans cette direction, malgré l'affirmation de Boral et Dewitt [14] énoncée certes dans un contexte plus précis et sur laquelle nous reviendrons dans le pragraphe 4.4.

L'effort entrepris par les organismes japonais ICOT avec la machine DELTA [30] et américain MCC ainsi que le succès industriel de la machine DBC/1012 de TERADATA dont nous allons parler maintenant sont le témoignage le plus frappant de ce regain d'activité.

4.3.2. Architectures distribuées

L'alternative moins ambitieuse aux architectures multi-processeurs fortement couplées consiste en une distribution du traitement entre plusieurs processeurs identiques communiquant au moyen d'un réseau. Par exemple, MICRONET [50] prévoit que le(s) hôte(s) communique(nt) avec un nombre variable de processeurs bases de données. Dans ce type d'architecture, on utilise des processeurs banalisés et l'effort se porte plutôt sur la conception du réseau et du système opératoire dans un souci de diminuer la complexité dûe à la communication inter-processeurs.

Cette approche semble promise à un certain avenir. Le DBC/1012 commercialisé en Avril 1984 par la Société Californienne TERADATA (25 systèmes devaient être installés fin 1984) suit exactement cette voie (Figure 4.6).

Le découpage fonctionnel est simple. Comme dans une machine à Filtre (e.g. VERSO) pour traiter une requête, on a besoin de deux processeurs, un processeur appelé IFP chargé des couches logiques et un processeur appelé AMP chargé des couches physiques (Filtrage) connecté à un disque à têtes mobiles WINCHESTER de capacité 474 Moctets (organisation PPD).

En rajoutant plusieurs disques (et donc plusieurs AMP) on augmente le parallélisme d'accès aux données (on peut pour une même requête balayer plusieurs disques simultanément). En rajoutant plusieurs processeurs IFP, on peut traiter en parallèle plusieurs requêtes. Tous les processeurs sont banalisés (INTEL 8086 et 8087, ce dernier pour les opérations numériques).

Le souci des concepteurs était triple :

1 - augmenter considérablement les performances du système en ajoutant des processeurs en fonction de la charge du système,

2 - Réduire au maximum les problèmes d'interconnection : l'ajoût d'un processeur doit être rapide et ne pas remettre en question le système ni nécessiter un effort de développement logiciel. Une des conséquences de ceci est que l'implantation du logiciel sur chaque processeur est la même que s'il était seul sur le système.

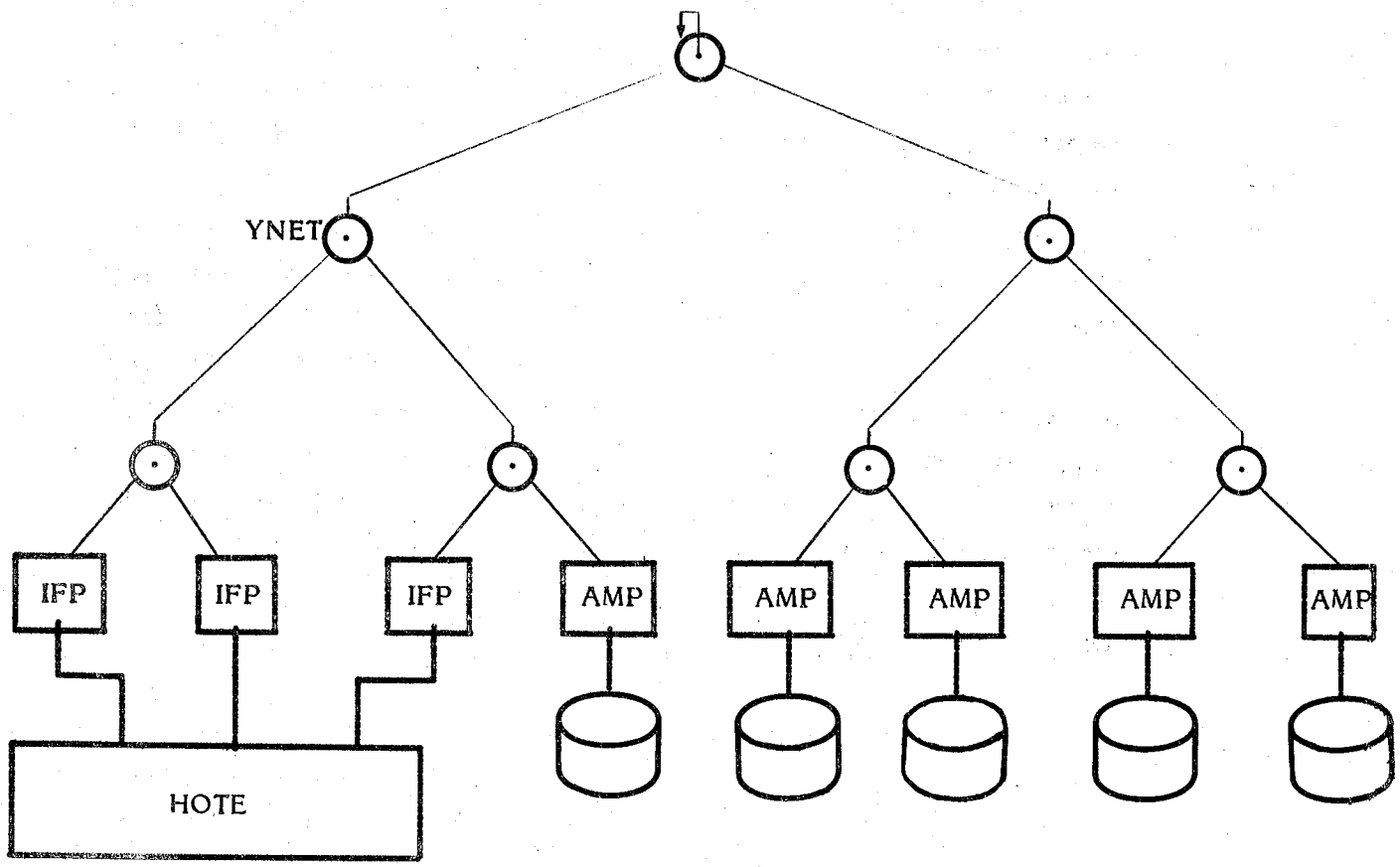


Figure 4.8. Architecture multiprocesseurs faiblement couplée : Le DBC/1012 de TERADATA

3 - Veiller aux problèmes de communication entre processeurs : il ne faut pas que le réseau de communication entre les processeurs devienne le goulot d'étranglement d'une part, il faut que le réseau s'adapte à un nombre variable de processeurs d'autre part. D'où la conception et la réalisation d'un réseau spécialisé et breveté sous le nom de Y-NET.

Ce réseau est une arborescence dont les feuilles sont les processeurs qui s'envoient les uns aux autres des paquets et dont les noeuds sont des processeurs matériels de tri comparant les paquets caractère par caractère. Tant que les caractères sont les mêmes ils sont transmis à l'étage supérieur. Dès qu'il y a une différence, c'est le paquet avec le caractère le plus petit qui gagne, i.e. qui est envoyé en entier vers l'étage supérieur, l'autre paquet attendant son tour. Donc si n est le nombre de processeurs (IFP + AMP), il font au minimum $Log_2 n$ cycles élémentaires pour un paquet pour atteindre la racine de l'arbre, où le cycle mesure le temps de parcours d'un étage pour un paquet de taille fixe. Une fois arrivé à la racine, le paquet est diffusé en un cycle à tous les processeurs qui reconnaissent à l'en tête du paquet s'il lui est destiné ou non.

4.4. Les Architectures du futur rapproché

Il est difficiel de prédire quelle sera l'architecture des SGBD des années 1990, d'une part à cause de l'évolution technologique future, d'autre part à cause de l'évolution des applications dont les besoins en gestion de données sont encore mal spécifiés.

Cependant on peut essayer de parier sur les tendances actuelles (à 3 ou 5 ans). Celles-ci s'appuient sur une thèse qui est celle de l'utilisation de processeurs du commerce plutôt que de processeurs spécialisés microprogrammés et/ou utilisant des circuits VLSI. Pour étayer cette thèse, nous développerons les arguments de Boral et Dewitt [14] en caractérisant d'une part les opérations "Bases de Données", d'autre part les tendances technologiques actuelles.

4.4.1. Caractérisation des opérations Bases de Données

Il faut distinguer entre les fonctions de haut niveau ou couches logiques (gestion des vues, décomposition et optimisation d'une requête, gestion des transactions et chemin d'accès) et les opérations élémentaires de bas niveau ou filtrage (opérations relationnelles sur un ou plusieurs blocs).

Le filtrage consiste en

i) l'exécution proprement dite des opérations relationnelles que nous caractérisons plus bas,

ii) les mouvements de données entre différentes mémoires (RAM ou disque),

iii) le contrôle de ces mouvements et du filtrage : il s'agit d'un ensemble mal défini de tâches que nous appellerons comme les anglosaxons "overhead" système. Si l'effort de conception porte sur l'exécution rapide du filtrage, on oublie souvent l'impact de l'overhead. Celui-ci comporte suivant les choix architecturaux des tâches aussi variées que le mouvement de bras du disque, la gestion dynamique de buffers, la communication interprocesseurs, etc... Il ne sert à rien de concevoir un filtrage efficace sur quelques blocs si le goulot d'étranglement est en fin de compte le temps passé en communications.

Les opérations relationnelles quant à elles sont simples et nécessitent un accès important à la mémoire de masse

a) les opérations relationnelles sont simples. Ce sont essentiellement des opérations de manipulation de caractères. Le calcul d'aggrégats (pourcentage, somme, etc...) ne nécessite pas une puissance considérable de calcul. En conséquence, le pourcentage de temps passé

en accès mémoire RAM par rapport au temps total de traitement est considérable.

b) Elles nécessitent un accès important à la mémoire de masse. Actuellement, quelle que soit l'opération, les données sont lues la plupart du temps du disque. Le résultat du filtrage est souvent écrit sur le disque. Plus le volume de données est important, plus cette affirmation est vraie. Il faut toutefois tempérer ce "credo" : Le nombre de transferts de blocs entre le disque et la mémoire RAM lors d'une consultation dépend de l'indexation, de la destination et de la taille du résultat.

La création d'une relation comme jointure de deux relations existantes nécessite effectivement le balayage de deux relations sources lues du Disque et l'écriture sur disque de la relation cible quels que soient l'architecture, l'algorithme de jointure. Ceci n'est pas le cas d'une semi-jointure étape intermédiaire d'une requête, dont le résultat et/ou les sources peuvent, i) être de taille limitée, ii) résider en mémoire RAM ou être des relations d'entrée/sortie.

4.4.2. Tendances technologiques

Nous reprenons ici les arguments de Boral et Dewitt [14] qu'on peut schématiser de la façon suivante. Etant donnée l'évolution technologique et les caractéristiques des opérations de filtrage, il faut pour implanter le filtrage,

1) d'une part abandonner les organisations PPT et PPP non viables et se restreindre à un processeur par disque (PPD),

2) d'autre part utiliser des processeurs standards et notamment éviter à la fois de concevoir de nouveaux circuits VLSI ou des processeurs spécialisés que l'on microprogramme.

Si le premier point est acquis, le deuxième demande des éclaircissements :

a) Mémoires

Deux évolutions sont dignes d'intérêt.

Tout d'abord, la croissance du débit des disques à tête mobile est limitée, avec une valeur asymptotique de l'ordre de 3 Millions d'octets par seconde [14]. Bien que les contrôleurs actuels ne suivent pas cette cadence, on peut s'attendre à ce que les constructeurs mettent sur le marché des contrôleurs très rapides.

La croissance des performances et notamment de la vitesse des microprocesseurs étant loin d'être limitée, on peut escompter d'ici quelques années un filtrage extrêmement rapide sur des processeurs standards alors qu'actuellement seuls des processeurs spécialisés (e.g. VERSO ou SCHUSS) sont capables de suivre la cadence des disques.

Ensuite et surtout, le coût de la mémoire RAM diminue de façon exponentielle. De grands constructeurs étudient actuellement des antémémoires de taille considérable (100 Millions d'octets !).

Ceci peut constituer une véritable révolution : des organismes comme MCC étudient d'ores et déjà pour leur système de gestion des connaissances des algorithmes de traitement des données stockées en mémoire RAM.

b) Processeurs

Nous avons déjà dit que d'ici quelques années les microprocesseurs du commerce seront extrêmement rapides. On peut alors s'interroger sur l'intérêt de développer des processeurs spécialisés qui s'ils sont pour l'instant plus rapides demandent un

grand effort de micro-programmation. Le développement de microcode est une tâche lente et les outils en la matière ne progressent que peu. Une fois le processeur microprogrammé, on risque de trouver sur le marché des microprocesseurs aux performances avoisinantes.

D'autres auteurs préconisent l'utilisation de VLSI spécialisés aux tâches de filtrage, jointure et tri. Boral et Dewitt [14] sont opposés à une telle démarche pour les raisons suivantes : (i) les microprocesseurs standards ont un jeu d'instruction de plus en plus riche (en particulier tous auront d'ici un an des instructions de traitement de chaînes de caractères) et ils gèrent des types de données de plus en plus variés ; (ii) le gain apporté par une VLSI est maigre car les opérations de manipulation de chaînes de caractères qui constituent l'essentiel des opérations de bas niveau en gestion de bases de données nécessitent un fort pourcentage d'accès extérieurs à la puce.

4.4.3. L'avenir proche

Notre thèse a été dans ce chapitre de défendre deux approches architecturales comme actuellement "raisonnables" (en attendant les révolutions futures, notamment en ce qui concerne les mémoires RAM). Par raisonnable, on entend qu'elles procurent un gain considérable en performances tout en étant faciles à mettre en oeuvre donc à être industrialisées dans un futur proche. Ces deux approches sont soit les Machines à Filtre, soit les Architectures Multiprocesseurs faiblement couplées. Pour étayer cette thèse, nous reprenons la taxonomie du départ (paragraphe 4.1.4).

a) Degré de parallélisme

Rappelons qu'une transaction est une séquence de requêtes ou étapes. Une requête est par exemple une consultation ou une insertion. Le système le plus simple accepte l'entrelacement de transactions mais traite séquentiellement les requêtes. C'est le cas

des Machines à Filtre dans leur conception actuelle, d'où le vocable de SISD ou SIMD. En se restreigant aux organisations PPD où un Filtre est connecté à un disque à tête mobile, la machine à Filtre la plus simple (e.g. VERSO, IDM 500) a un seul filtre et traite une seule requête à la fois (Machine SISD). On peut envisager des Architectures multi-Filtres SIMD traitant une seule requête mais où les données sont réparties sur plusieurs unités à disque et où le filtrage se fait en parallèle grâce à plusieurs filtres.

Enfin on peut envisager des architectures MIMD type DBC/1012 de TERADATA, où plusieurs requêtes peuvent être traitées en parallèle.

Rappelons qu'il n'existe actuellement aucune méthodologie permettant des évaluations quantitatives sérieuses de ces diverses solutions architecturales.

b) Degré de spécialisation : Un processeur banalisé ou n processeurs spécialisés à une fonction pour traiter une requête.

Nous avons également défendu l'idée de simplicité suivante en ce qui concerne le découpage fonctionnel du traitement d'une requête : celui-ci est réparti entre deux processeurs l'un chargé des couches logiques (PCL), l'autre du filtrage (F), ces deux briques servant de base à toute architecture. Si l'on en prend un de chaque catégorie, on a une MACHINE A FILTRE SISD, si on en prend plusieurs de chaque catégorie, on peut arriver à toutes les combinaisons SIMD, MIMD.

Pourquoi un découpage en deux et non en trois ou plusieurs processeurs ? Pour des raisons de simplicité de conception, de développement :

Plutôt que de rajouter un processeur chargé du tri pipeliné avec le PCL, il est préférable d'évaluer si le TRI doit être à la charge du Filtre ou du PCL, quitte à augmenter le nombre de PCL (ou de Filtres). Plutôt que de pipeliner plusieurs requêtes sur un même PCL il est préférable d'augmenter le nombre de PCL chacun traitant les requêtes séquentiellement.

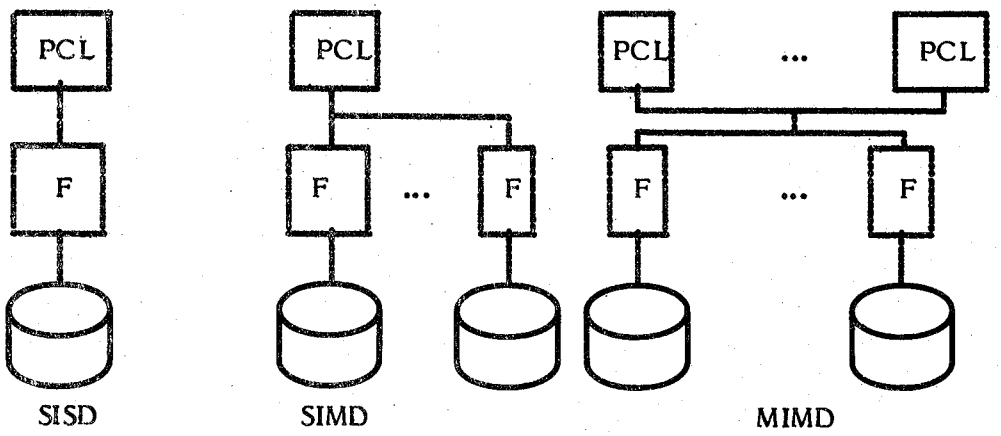


Figure 4.9. Découpage fonctionnel du traitement d'une requête

Ce choix s'appuie sur les hypothèses suivantes :

i) le traitement "logique" d'une requête est raisonnablement rapide,

ii) le filtrage est un ensemble de fonctions qu'on sait traiter rapidement et dont la complexité est une fonction linéaire en la taille des données à traiter. Par conséquent si on veut accélérer le filtrage, on répartit les données entre plusieurs disques qu'on filtre en parallèle.

iii) Enfin rappelons que le découpage entre PCL et Filtre (historiquement la décentralisation près du disque de la fonction de sélection) a au moins une justification qualitative, c'est de ne faire transiter d'un processeur à l'autre que les données utiles (résultats, insertions). On

optimise ainsi le partage de la ressource bus ou réseau de communication.

c) **Degré d'homogénéité**

Faut-il utiliser des processeurs standards ou bien des processeurs spécialement conçus à cet effet ?

Le PCL est bien évidemment implanté par un processeur standard. Dans la discussion qui précède nous avons implicitement éliminé toute idée de réalisation matérielle "ad hoc" de processeurs de jointure ou de tri.

Il nous reste à discuter du choix du processeur de filtrage.

Faut-il utiliser un filtre tel que SCHUSS ou VERSO (Chapitre 3) extrêment performant ou implanter le filtrage sur un processeur standard ?

L'un des objectifs essentiels de cette thèse est de répondre à cette question. Un filtre logiciel a été conçu pour VERSO (Chapitre 5) et ses performances sont comparées à celles du filtre matériel dans le chapitre 6. L'intérêt du Filtre matériel est qu'il permet un filtrage "au vol", c'est-à-dire qu'il se fait au fur et à mesure qu'on lit les données du Disque. Le temps de filtrage est donc limité par le temps de transfert disque, le processeur spécialisé étant plus rapide que le Disque.

Cependant épousant la thèse de Boral et Dewitt (paragraphe 4.4.2), on peut d'ores et déjà se demander si l'effort en vaut la peine. D'une part, développer un matériel spécialisé est une opération coûteuse en temps. D'autre part, utiliser plusieurs filtres plus lents en parallèle est une solution envisageable sans compter que les performances des processeurs banalisés augmentent considérablement avec le temps. Enfin, aucun résultat d'évaluation ne permet actuellement de décider (i) si le goulot d'étranglement est

situé au niveau du filtrage et de l'accès aux données, (ii) si le pourcentage d'opérations de filtrage qui nécessitent le balayage d'un grand nombre de blocs est non négligeable (si quelques blocs seulement sont balayés pour la plupart des requêtes, il n'est plus fondamental de gagner un ordre de grandeur sur le temps de filtrage.

Pour terminer, signalons que le même problème d'homogénéité et de portabilité existe au niveau du logiciel. Il faudra utiliser un logiciel de bas niveau standard (e.g. UNIX multiprocesseur) quand de tels outils seront répandus, plutôt qu'un logiciel spécialisé. Les systèmes opératoires distribués étant encore peu répandus, des compagnies comme TERADATA ont fourni un effort conisdérable en développant en assembleur 8086, le logiciel de bas niveau implanté sur chaque processeur de la machine DBC/1012.

CHAPITRE 5 _

FILTRAGE LOGICIEL

5. - FILTRAGE LOGICIEL

On étudie dans ce chapitre une alternative à l'architecture matérielle étudiée dans le Chapitre 3 pour l'implantation du Filtrage : il s'agit de réaliser le Filtrage d'une Relation Physique au moyen d'un logiciel sur un Module d'échange (ME) du SM 90 connecté d'une part au bus SM, d'autre part au disque où sont rangées les données (Fig. 5.1). Ce module possède une mémoire partagée MP (128 Koctets) accessible à la fois par le module de traitement (MT) 68000 d'une part et par le microprocesseur (Intel 8086) et le DMA (Intel 8089) d'autre part. Dans la MP sont rangés à la fois le logiciel de Filtrage, les données (tampons source et cible) et la Matrice de Transition (automate de Filtrage).

Une première version de Filtrage logiciel est en cours d'implantation sur un Module d'échange (ME) à interface SASI. Bien que la réalisation concerne à la fois les opérations unaires et les opérations binaires de l'Algèbre Relationnel, nous n'étudierons dans ce chapitre que la Sélection/Projection.

Un deuxième ME plus puissant est en cours de réalisation [28]. Celui-ci permettra une interface SMD et comportera un DMA cablé. Le processeur 8086 sera remplacé par un 80186 à 8MHz et la mémoire partagée atteindra 512 Koctets. Ces deux architectures sont comparées dans le Chapitre suivant.

Après quelques généralités sur l'implantation du Filtrage logiciel sur le ME SASI, nous étudions en détail le principe de Filtrage pour la Sélection/Projection et nous donnons une estimation de la taille des automates. Contrairement au Filtrage matériel, nous montrons que les automates sont plus condensés et que la phase de compilation d'une requête est beaucoup plus courte, l'essentiel de l'automate étant précompilé (automate fichier).

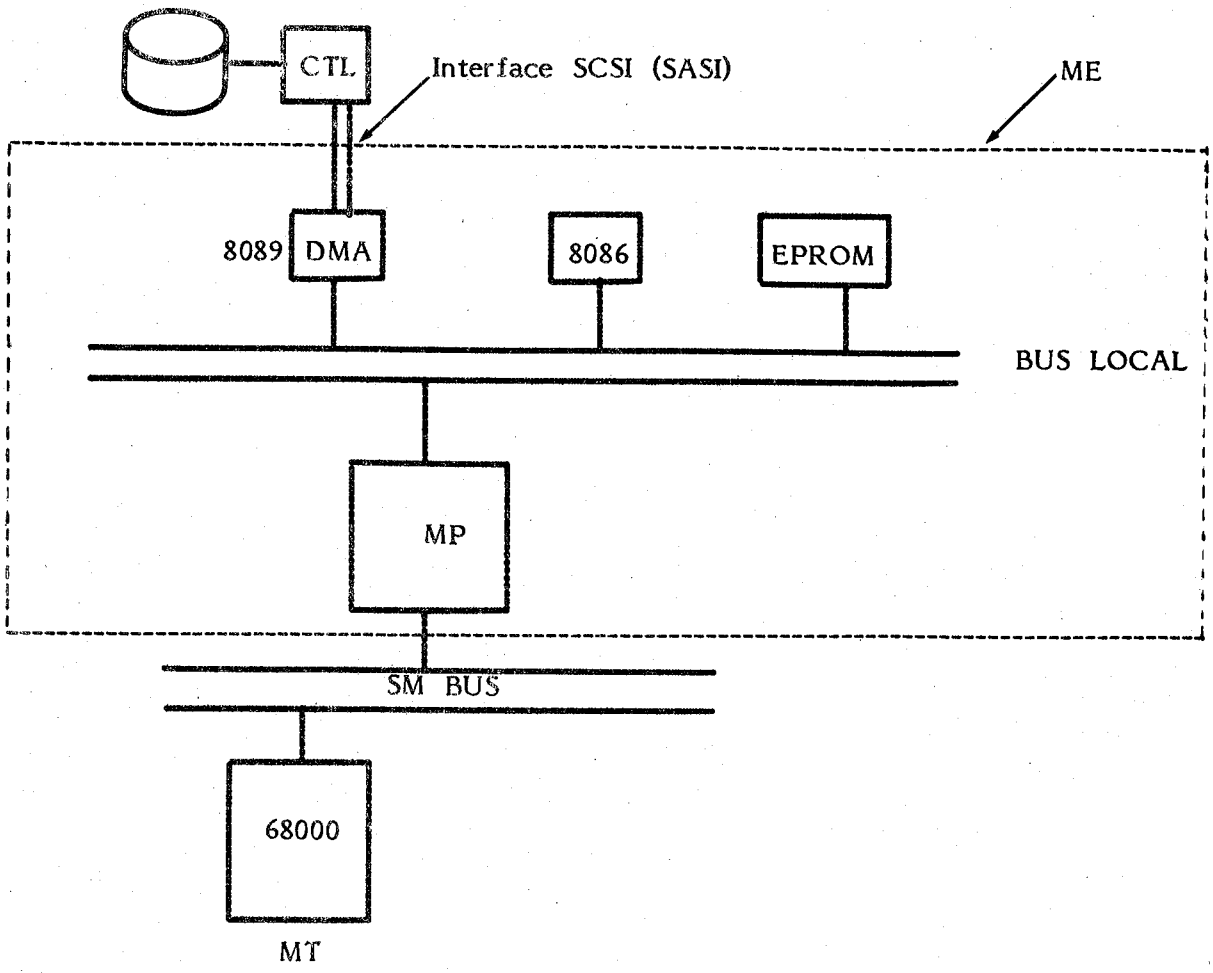


Figure 5.1. Filtrage logiciel : Architecture matérielle

5.1. Architecture

Comme auparavant (Filtrage matériel), le 68000 est chargé des couches de haut niveau du SGBD (niveau logique, chemin d'accès) de la compilation des requêtes et du tri. Tout ce qui a été décrit dans le Chapitre 3 concernant les couches de haut niveau reste valable dans ce chapitre.

Le MT envoie au ME des ordres de transfert de données et de Filtrage. Le MT charge directement l'automate de Filtrage dans la mémoire MP à une adresse indiquée par le ME. De la même façon il y écrit les n-uplets à insérer et y lit les résultats (Relations d'Entrée/Sortie).

Le ME possède un moniteur multi-tâches et 2 tâches système (dialogue avec le MT, tâches d'E/S disque), le tout implanté sur une mémoire EPROM.

Le 8086 est l'Unité Centrale qui exécute, sur une liste de blocs, le Filtrage logiciel. Celui-ci est essentiellement constitué des tâches suivantes qui partagent deux ensembles ou "Pools" de tampons et deux Files d'adresses de tampons. Lors d'une requête de Filtrage, après chargement de l'automate, 3 tâches sont lancées qui se synchronisent grâce à l'état des Pools et des Files (Figure 5.2).

5.1.1. Le Chargeur

Il charge l'ensemble des blocs de la Relation à filtrer du Disque dans les tampons Source. Pour chaque bloc (une piste de 16 Koctets), il demande un tampon source à l' ALLOCATEUR qui lui alloue une adresse de tampon si le Pool des tampons Source n'est pas vide. Dans le cas contraire le chargeur attend la libération d'un tampon. Lorsqu'un bloc est chargé, le chargeur enfile son adresse dans la file F1 des blocs chargés (i.e. à filtrer). Le chargeur s'arrête si le Pool Source est vide ou si tous les blocs de la relation Source ont été chargés.

5.1.2. Le Filtreur

Il demande à l'allocateur un tampon source (de la file FIFO F1 des chargés) à Filtrer et un tampon cible (du Pool des tampons cible). Il lance alors la tache INTERPRETE qui exécute l'automate rangé dans la Matrice de transition MT et rappelle le Filtreur dans deux cas : Bloc Source Filtré ou Bloc Cible plein.

Dans le premier cas, le Filtreur demande un autre bloc Source à Filtrer et relance l'INTERPRETE.

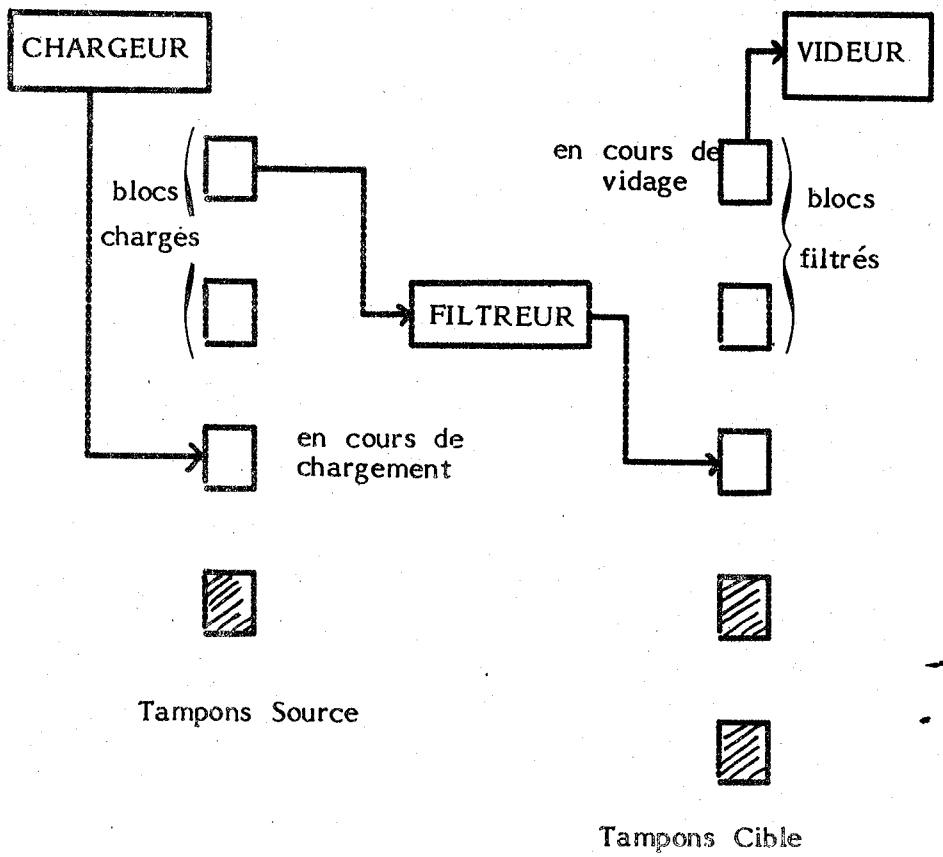
Dans le deuxième cas, le Filtreur signale l'adresse du bloc résultat, en enfilant son adresse dans la file FIFO F2 des blocs à vider et demande un autre tampon à l'Allocateur pour écrire la suite de la relation Cible.

Le Filtreur attend

a) quand la file F1 est vide
b) quand le Pool cible est vide
c) quand l'INTERPRETE est en train de filtrer
d) quand le Filtrage est terminé.

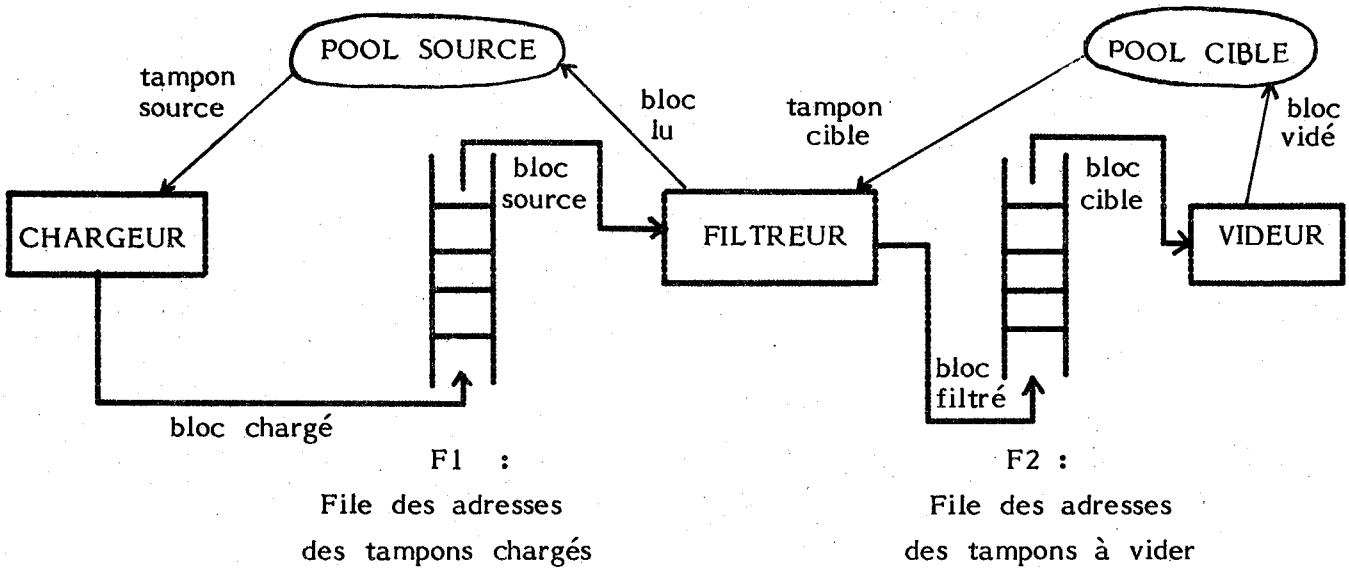
5.1.3. Le Videur

Il attend que la File F2 des blocs à vider contienne au moins une adresse de bloc à vider, le vide sur le disque ou demande au MT de la lire. Une fois le bloc vidé, il le signale à l'ALLOCATEUR qui libère ce bloc, i.e. le remet dans le Pool cible.



a) Principe du Filtrage : tâches CHARGEUR, FILTREUR, VIDEUR

: bloc libre



b) Partage des tampons entre tâches

Figure 5.2. Principe du Filtrage et partage des ressources

5.2. La Sélection/Projection

On décrit plus bas en détail un algorithme de filtrage logiciel pour la Sélection/Projection. L'algorithme proposé n'a pas été implanté pour le Filtre logiciel VERSO, tel qu'il est décrit plus bas. On lui a préféré une variante optimisée décrite dans [15] qui en suit les grandes lignes mais s'en écarte considérablement dans les détails d'implantation.

Avant de décrire le principe du filtrage, ses différences avec le Filtre VERSO matériel, on rappelle brièvement la définition de l'opération de Sélection (ou Restriction) telle qu'elle a été donnée au paragraphe 3.2.4.

La sélection S(f) d'un fichier f de format $F=X(F_1)^*...(F_p)^*$ où $X \subseteq U$ et $F_1,..,F_p$ sont des formats

est la fonction booléenne $S(f) = \overset{*}{\underset{}{U}} S(E)$, c'est-à-dire l'union sur tous les enregistrements de f, notée également $S^*(E)$:

la sélection d'un enregistrement de f est $S(E)=S(X) \cap \sigma(F_1) \cap ... \cap \sigma(F_p)$ où σ est l'identité ou la négation ou la fonction constante avec

$S(X) = \underset{A \in X}{\cap} S(A)$

et $S(A) = C_1 \cup C_2 \cup ... \cup C_m$

où $C_i = A\ op_i\ a_i$:

op_i est un opérateur arithmétique et $a_i \in D(A)$

Exemple : $F = COURS(ETUDIANT(NOTE)^*)^*$

Requête 1 : "COURS,ETUDIANTS qui ont une note > B et la liste de ces notes"

S(NOTE) est vrai si note > B

S(ETUDIANT) est vrai si S^*(NOTE) est vrai (σ : identité)

S(COURS) est vrai si S^*(ETUDIANT) est vrai (σ : identité)

Requête 2 : "COURS où il n'y a ni TOTO ni LULU"

S(ETUDIANT) est vrai si (ETUDIANT = TOTO) $\vee$ (ETUDIANT = LULU) est vrai

S(COURS) est vrai si S^*(ETUDIANT) est faux (σ : négation)

5.3. Principe du filtrage

On montre plus bas comment construire un automate fini qui filtre en une passe tout fichier pour de telles requêtes, en projetant les attributs à garder.

Informellement, on peut décrire l'algorithme de la façon suivante : récursivement quel que soit l'enregistrement qu'on est en train de scruter, on évalue d'abord la racine X, on projette les attributs à garder de cette racine après marquage (empilement de l'adresse cible du 1er caractère projeté). Ensuite, pour chaque enregistrement du sous-fichier de format F_i scruté, on évalue $S(F_i)$, on marque et on projette. En sortant du sous-fichier, $(S^*(F_i))$ est évalué. Si cette fonction a la valeur vraie, la marque est effacée. Sinon le pointeur cible est restauré à la dernière marque (on écrasera les attributs projetés). Dans tous les cas S(F) est mis à jour et on continue en scrutant le sous-fichier suivant.

5.4. Format des Données

Un attribut est représenté par son identificateur (ia) suivi d'une longueur de champ (lgr) , suivi de 'lgr' octets représentant la valeur d'attribut :

<AH> := <ia><lgr><valeur>

<lgr> := <m>/255<n_1><n_2>

La longueur est donc représentée par un octet <m> si elle est inférieure à 255 octets (m=0,1...,254). Sinon (m=255) elle est représentée par 3 octets : lgr = n_1 x 256 + n_2 , (n_1,n_2 = 0,1,...,255). Le codage de la longueur est adapté à des champs ayant une taille $\leq$ 64 Koctets. Mais on pourrait représenter des champs de longueur > 64 Koctets, en utilisant un autre codage.

5.5. Matrice de transition (figure 5.3)

Comme pour le filtrage matériel, l'automate est représenté par une matrice de transition (MT) dont les lignes sont les états et les colonnes, les caractères de la tête de lecture. Seul le format d'un noeud MT(e,c) diffère. Il sera donné plus bas. Le noeud contient l'état suivant et les fonctions à exécuter au cours de la transition (alors que dans le filtrage matériel, le noeud contient un pointeur vers une mémoire de commandes MAPMACRO, voir Figure 3.6).

Le filtrage proprement dit est réalisé par la tâche INTERPRETE. Un cycle élémentaire du filtre consiste à lire le noeud d'adresse contenue dans les registres ETAT, CAR, à interpréter séquentiellement de gauche à droite les fonctions à exécuter et à mettre à jour les registres ETAT,CAR. Le premier est mis à jour par la valeur du champ "Etat Suivant" du noeud de la matrice. Le second contient le caractère suivant lu dans le tampon source.

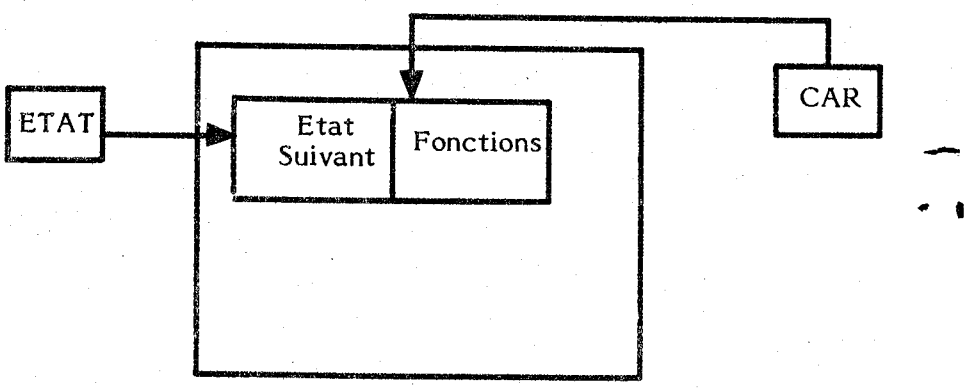


Figure 5.3. Matrice de transition

Un cycle élémentaire du filtre ne correspond plus à la lecture d'un seul caractère du tampon Source et à sa réécriture éventuelle dans le tampon Cible. Sa durée ainsi que le nombre de caractères lus et écrits sont variables.

On peut distinguer essentiellement 3 types de cycles :

a) cycle "ia" : lecture de l'identificateur d'attribut (ia) et de la longueur de l'attribut stockée dans un registre LGR ;

b) cycle "saut" : l'attribut courant est sauté (incrémentation du pointeur Source) ;

c) cycle "comparaison" : la valeur de l'attribut courant est comparée à une ou plusieurs chaines de caractères : cas d'un attribut intervenant dans la requête de Sélection. Cette comparaison est étudiée en détail dans le paragraphe suivant (Instruction COMP)

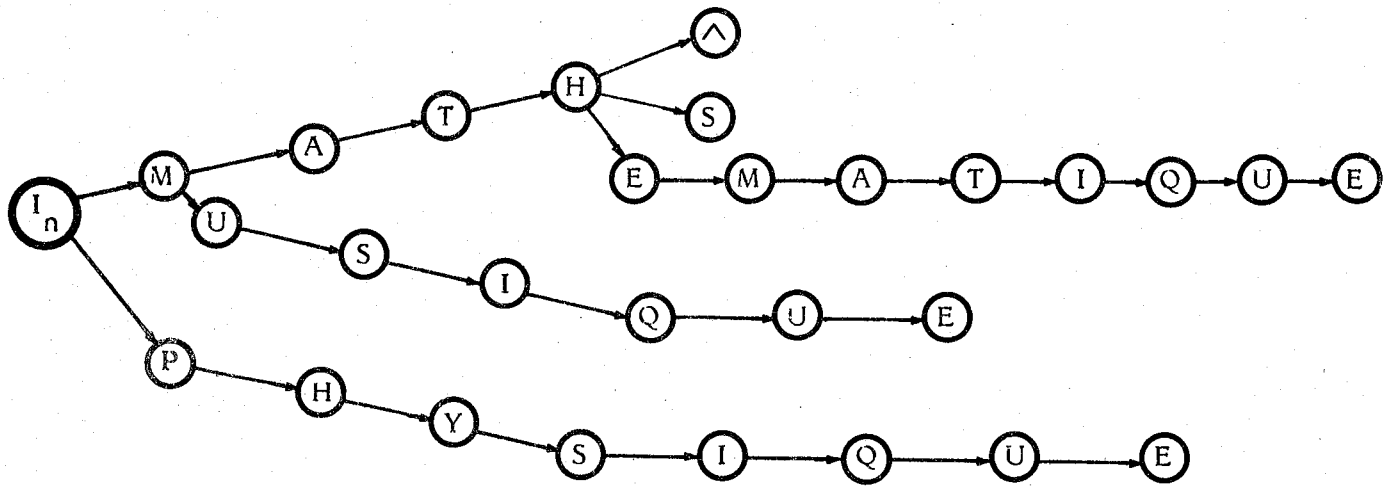
5.6. Automate de comparaison de chaines de caractères

Une condition sur un attribut A est une disjonction de conditions élémentaires A op a où a $\in$ D(A). Soit à résoudre par exemple la requête élémentaire COURS = MATH $\vee$ MATHS $\vee$ MATHEMATIQUE $\vee$ MUSIQUE $\vee$ PHYSIQUE.

A une condition correspond un automate fini de reconnaissance dont le diagramme de transitions peut être représenté de la façon suivante :

A chaque état, correspond un noeud étiqueté par le caractère présent dans la tête de lecture. L'arc (J) $\longrightarrow$ (K) correspond à la transition état J $\longrightarrow$ état K qui advient lorsque dans l'état J le caractère suivant est K.

A la requête ci-dessus, correspond l'automate suivant (est l'état initial).



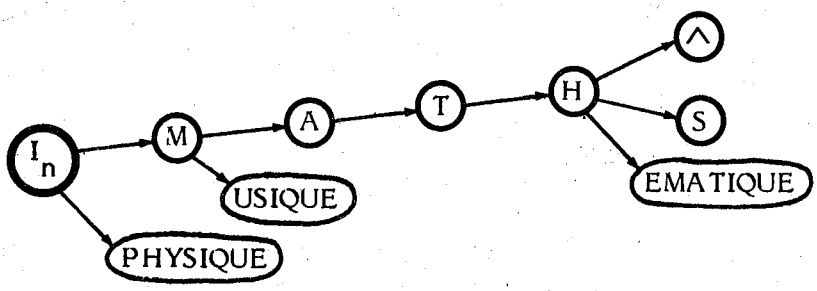
Dans ce diagramme, nous n'avons présenté que les transitions correspondant à un succès. Dans le filtrage matériel, la compilation d'une requête ayant ce sous-critère (sur l'attribut A) génère effectivement autant d'états qu'il y a de caractères dans l'arbre lexicographique (AL) ci-dessus.

Observons toutefois que la plupart des chaines sont linéaires, i.e. la plupart des noeuds internes de l'AL n'ont qu'un fils. Dans un alphabet de taille ≥ 26, ceci sera le cas pour la plupart des requêtes sauf dans des applications balayant des dictionnaires.

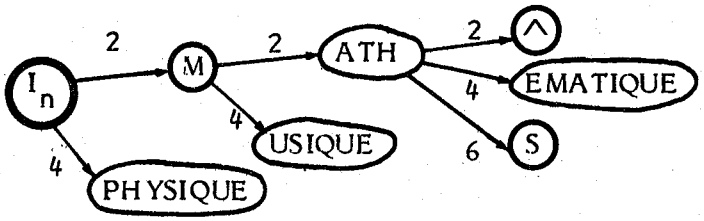
Il devient alors intéressant de contracter tous les sous-chemins linéaires de la racine de l'AL jusqu'à chacune des feuilles : en effet un échec sur "U" ou "Q" lors de la comparaison de la source à MUSIQUE donnera lieu à la même transition. Un succès nécessite l'égalité de deux chaines de caractères sur $\ell \geq 1$ caractères.

Rappelons que l'idée de contraction est utilisée dans la structure de l'Index non dense de VERSO (chapitre 3) où tous les plus grands n-uplets de chaque bloc sont représentés sous forme d'un

AL élagué (ALE) dont on a contracté (élagué) les fins de chaine linéaires. L'ALE correspondant à l'AL précédent aurait la forme suivante :



Nous appellerons AL contracté ALC, l'AL où on a contracté tous les sous-chemins linéaires y compris ceux qui ne se terminent pas par une feuille. L'ALC correspondant à l'AL ci-dessus est :



Dans le filtre logiciel ce sont effectivement les noeuds internes de l'ALC qui sont représentés comme états de la matrice de transition. Soit p le degré d'un noeud interne n (nombre d'arcs sortant). Le nombre de transitions possibles lorsqu'on est dans l'état n est alors 2p+1. Par exemple si on est dans l'état "M", c'est que le premier caractère du champ lu dans la SOURCE est 'M'; A l'issue de la comparaison de la fin de champ, on a 5 cas, dont 3 correspondent à un échec et 2 à un succès. On donne ci-dessous 5 exemples de valeurs d'attributs lues dans la SOURCE, chacune correspondant à une des 5 transitions

1 :	MATEMATIQUE	échec	(ATE < ATH)
2 :	MATHS	succès	(ATH = ATH)
3 :	MEDECINE	échec	(ATH > E, E < USIQUE)
4 :	MUSIQUE	succès	(USIQUE = USIQUE)
5 :	MUSIQUES	échec	(USIQUES > USIQUE)

L'instruction élémentaire de comparaison 'COMP' retourne un entier $\leq 2p+1$ appelé retour-calculé qui servira à adresser la MT dans l'état suivant ("caractère" suivant : registre CAR). L'ALC ci-dessus est étiqueté avec les valeurs du retour-calculé.

5.6.1. Colonnes de la matrice

Rappelons qu'il y a trois types de cycles : au cours d'un cycle ia, un identificateur d'attribut est lu ; au cours d'un cycle saut, on incrémente le pointeur source de la longueur de l'attribut et aucun caractère n'est lu dans la source ; enfin au cours d'un cycle de comparaison, aucun caractère n'est lu mais l'instruction COMP de comparaison renvoie un "retour-calculé" qui tient lieu de caractère pour adresser la Matrice de transition. Ainsi dans l'exemple précédent, la comparaison du champ lu dans Source à l'une des 5 chaînes prend un à 3 cycles (une à 3 comparaisons) au cours desquels le registre CAR est alimenté de une à trois fois par le retour-calculé de l'instruction COMP.

En résumé, le registre CAR (adressage des colonnes de la matrice de transition) est soit alimenté par un numéro d'attribut (identificateur d'attribut en début de champ) soit par un "retour-calculé" en cas de comparaison.

Le fait que l'automate ne voit que les ia ou les retours-calculés permet de réduire considérablement le nombre de colonnes de la matrice et donc sa taille et par conséquent le temps de chargement. On verra en fin de chapitre que le nombre d'états est lui aussi considérablement réduit.

5.6.2. Exemple d'automate de comparaison

Le sous-automate correspondant à la requête précédente est représenté dans la Matrice de la façon suivante :

retour-calculés

états		1	2	3	4	5	6	7	
	0	1 COMP							Comparaison à 'M' et 'Physique'
(I_n)	1	5	2 COMP	5	4	5	5	5	Comparaison à 'ATH' et à 'USIQUE'
(M)	2	5	3 COMP	5	4	5	5	5	Comparaison à ' ', 'S' et "EMATIQUE"
(ATH)	3	5	4	5	4	5	4	5	
(succès)	4								
(échec)	5								

Chaque noeud de la Matrice est schématiquement représenté par un champ indiquant l'Etat suivant (n° de ligne) et un champ indiquant s'il y a une comparaison à effectuer (laquelle comparaison est indiquée en marge). L'état initial est le couple (0,1) : on force l'état à 0 et le caractère à 1. On compare la valeur de l'attribut à 'M' et 'Physique' et l'état suivant est 1 (noeud (I_n) du diagramme de transition). Il existe alors trois transitions possibles à partir de l'état 1 qui dépendent du résultat de la comparaison à 'M' et 'Physique'

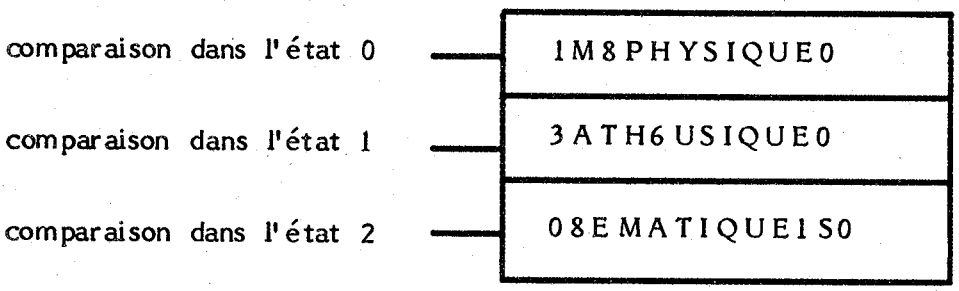
- Soit 'M' a été reconnu (le 1er caractère de l'attribut est M). Le retour calculé de l'instruction COMP est 2. On passe alors à l'état 2 (noeud (M)) et on compare l'attribut à partir du 2e caractère à 'ATH' et à 'USIQUE', etc.

- Soit 'PHYSIQUE' a été reconnu (la valeur de l'attribut est exactement 'PHYSIQUE'). Le retour calculé de COMP est 4. La reconnaissance est terminée. On passe à l'état 4 (succès).

- Soit ni 'M' ni 'PHYSIQUE' n' a été reconnu (retour-calculé ≠ 2,4). La reconnaissance est terminée : on passe à l'état 5 (échec).

Outre l'automate, le compilateur génère une séquence de listes de sous-chaines de caractères (et leurs longueurs). A chaque état où a lieu une comparaison, le fichier source doit être comparé sur une certaine longueur à une ou plusieurs sous-chaines de caractères. Les adresses de ces sous-chaines sont rangées dans le noeud interne correspondant.

Pour la requête précédente, le compilateur génère la séquence suivante :



Ainsi dans l'état 0, on compare la chaîne SOurce à M sur 1 caractère. Si la comparaison échoue, on essaie alors la comparaison sur 8 caractères à PHYSIQUE.

Cette comparaison entre une chaîne lue dans source et une sous-chaîne se fait en utilisant les instructions spécialisées de comparaison de chaînes de caractères du processeur 8086(*).

(*) Dans le cas du filtrage matériel, on peut considérer qu'on a une hiérarchie d'automates. En effet la reconnaissance de chaînes de caractères est elle-même faite par un sous-automate. A la sortie de l'automate, la transition (retour-calulé) dépend du résultat de la comparaison aux chaînes de caractères.

5.7. Fonctions

On décrit maintenant le format d'un noeud de la matrice MT. A chaque champ correspond une intruction élémentaire du Filtre (fonction exécutée au cours de la transition, Figure 5.4).

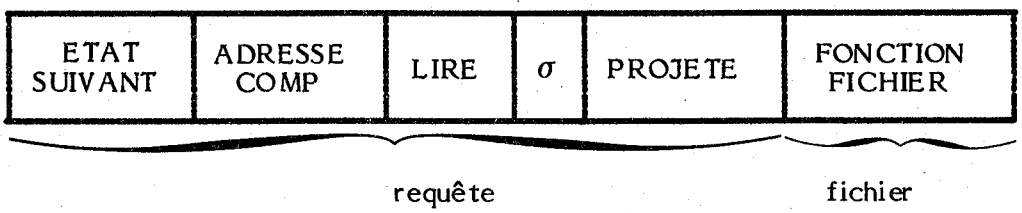


Figure 5.4. Un noeud de la Matrice de transition

Le 1er champ indique l'état suivant.

Les 4 champs suivants ont des valeurs qui dépendent de la requête, alors que le dernier champ ne dépend que du format du fichier.

Le 2e champ est l'adresse de la liste de chaines de caractères en cas de comparaison.

Le booléen PROJETE indique si l'attribut est à garder (copier dans le tampon CIBLE).

Le 3e champ (LIRE) indique dans quel type de cycle on est et quel type d'opération est à effectuer sur l'attribut lu dans le tampon SOURCE :

"LIRE-IA" : lit l'ia et la longueur de l'attribut. Celle-ci est rangée dans le registre LGR. On pointe alors sur le premier caractère du champ. L'ia et la longueur sont copiées dans le tampon cible si l'attribut est à garder.

"SAUT" : le pointeur SOURCE RAB_S est incrémenté du contenu du registre LGR. On pointe alors sur l'attribut suivant. La valeur de l'attribut est copiée dans la Cible si l'attribut est à garder.

"COMP" : une comparaison est effectuée entre la SOURCE et la liste de chaines d'adresses, le contenu du 2e champ de MT. Si p est le nombre de chaines, après comparaison, le registre CAR a une valeur comprise entre 1 et 2p+1 (retour calculé). Les caractères lus dans SOURCE sont copiés dans la Cible si l'attribut est à garder.

Le champ σ indique quelle fonction (identité, négation, constante) appliquer sur le sous-fichier qui commence avec ce noeud.

Le champ "Fonction Fichier" qui ne dépend pas de la requête est décrit plus bas.

5.8. Algorithme de filtrage

Outre les instructions LIRE et PROJETE ci-dessus qui modifient l'endroit où lire (RAB_S) et l'endroit où écrire (RAB_C), on a besoin des ordres suivants reconnaissant le fichier.

Un fichier est une phrase de un ou plusieurs mots de format F. Un mot (ou enregistrement) du fichier a une racine (un ou plusieurs attributs) suivie éventuellement de une ou plusieurs sous-phrases (fichiers) de format $F_1,\ldots,F_p$.

A chaque niveau de l'arbre syntaxique associé au format, correspond une sous-phrase dont on est en train de balayer un mot.

Les ordres suivants (Fonctions Fichiers) sont des ordres mots et sous-phrases :

"DEB-ARBRE" : exécuté lorsqu'on balaie le premier mot d'une sous phrase ;

"FIN-ARBRE" : correspond à la fin de sous-phrase (après le dernier mot de cette sous-phrase) ;

"DEB-INSTANCE" : exécuté à chaque début de mot ;

"FIN-INSTANCE" : exécuté à chaque fin de mot ;

"ARBRE-VIDE" : cet ordre est prévu pour le cas de valeur nulle : une sous-phrase est absente du mot en cours.

Le champ "Fonction Fichier" d'un noeud de la matrice (Fig. 5.4) indique le(s) quel(s) parmi ces ordres doi(ven)t être exécuté(s).

5.8.1. Automate de Sélection/Projection

Comme pour le filtre VERSO matériel, à chaque requête et chaque fichier correspond un automate chargé dans MT avant filtrage. Un cycle du filtrage consiste à interpréter en séquence les instructions élémentaires rangées dans le noeud de MT, instructions dont nous venons de donner la description.

Contrairement au filtre VERSO matériel, la compilation (génération de l'automate) est plus courte :

A chaque fichier correspond un automate fichier précompilé dont la description est donnée au paragraphe suivant. Il est représenté par une matrice de transition où seuls les champs 'Etat Suivant' et 'Fonction Fichier' sont remplis.

La génération de l'automate de Sélection/Projection consiste alors à (Fig. 5.5)

i) générer les sous-automates de comparaison des attributs intervenant dans un critère de sélection, et la séquence de listes de chaines de caractères (voir paragraphe 5.6).

ii) fusionner ces sous-automates avec l'automate fichier pour obtenir l'automate de Sélection/Projection. On modifie alors les champs 2 à 5 des noeuds de la matrice en fonction de la requête.

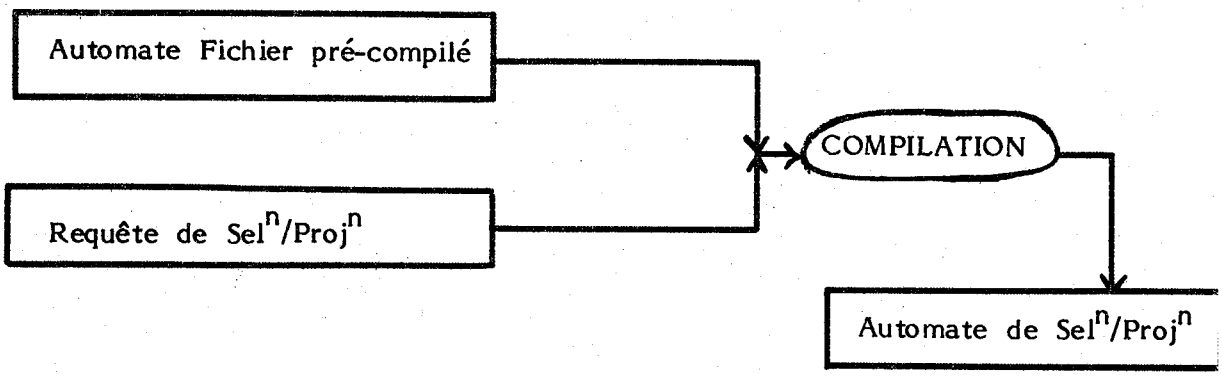


Figure 5.5. Génération de l'automate de Sélection/Projection

5.8.2. Automate fichier

A chaque format du fichier est associé un automate fini qu'on décrit maintenant au moyen d'un exemple. Soit à reconnaître le fichier de format $(A(BD^+)^+C^*)^+$. Dans ce format les '+' indiquent qu'il n'existe pas d'information incomplète : Chaque sous-phrase a au moins un mot. La figure 5.6 donne le diagramme de transitions. Un noeud du diagramme correspond à un état de l'automate. Le label d'une branche sortante correspond à une transition et indique le caractère lu dans la Source en minuscules (ia de l'attribut) ainsi que les instructions élémentaires (fonctions fichiers) que nous avons déjà énumérées et dont nous donnons l'interprétation dans le paragraphe suivant.

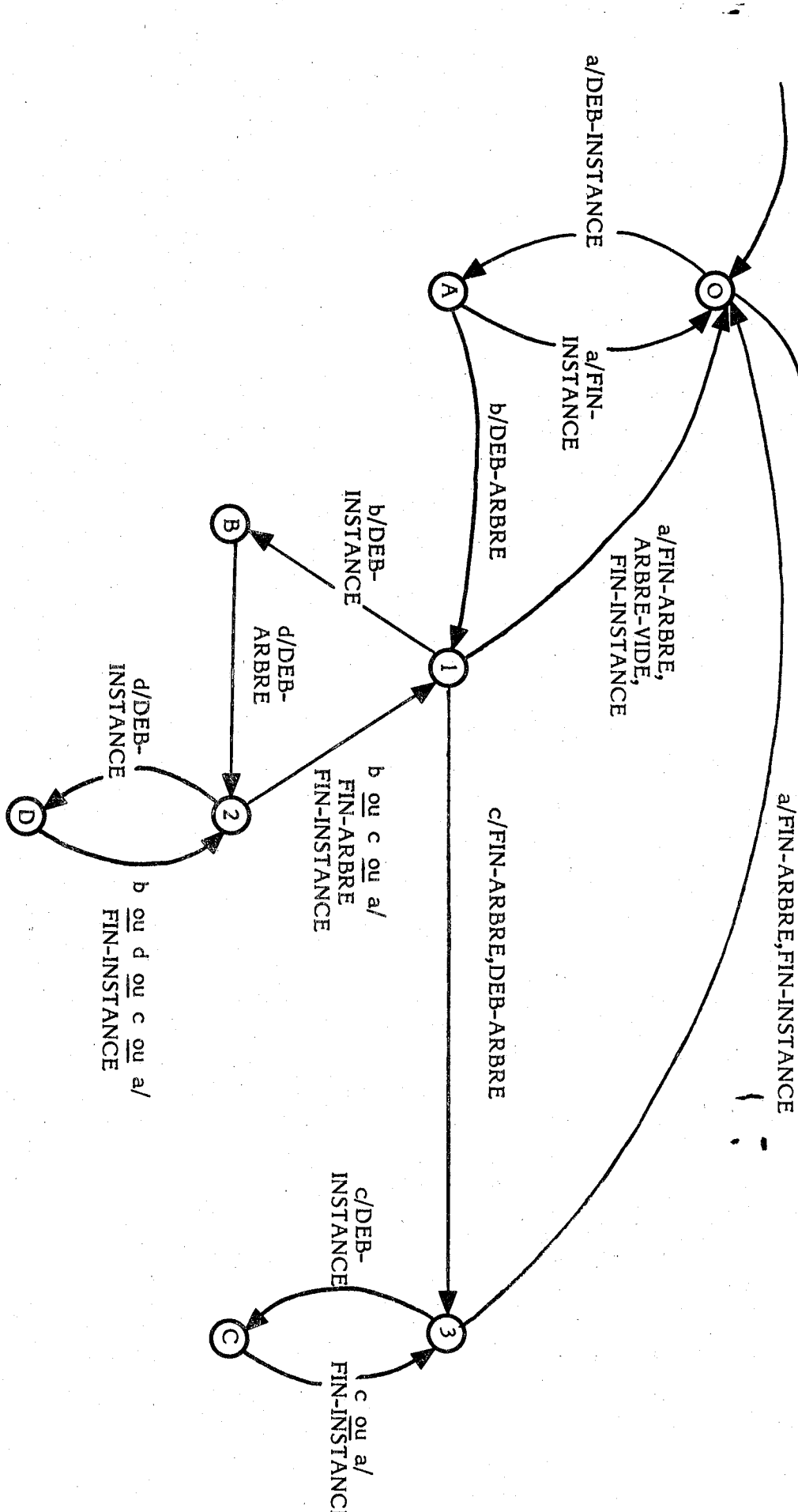


Figure 5.6. Automate de Reconnaissance d'un fichier de format $(A(BD^+)^+ C^*)^+$

5.8.3. Interprétation des fonctions fichier

Un cycle élémentaire du filtre consiste à lire un noeud de la matrice de transition (MT), à interpréter les différents ordres qui y sont rangés et à mettre à jour le registre ETAT avec la valeur 'ETAT SUIVANT' lue dans le noeud.

Pour que l'algorithme de filtrage soit complet, il nous faut donner l'interprétation des ordres DEB-ARBRE, FIN ARBRE, DEB-INSTANCE, FIN-INSTANCE et ARBRE-VIDE (champ 'Fonction Fichier' du noeud de MT).

On utilise comme pour le filtre VERSO matériel une pile P pour mémoriser l'adresse RAB_C où on écrit dans la Cible.

EMPILER RAB_C dans P — correspond à une mémorisation (marquage)

DEPILER P — a lieu lorsque les données écrites satisfont la requête : on efface la dernière mémorisation.

RESTAURER RAB_C — au contraire restaure le pointeur RAB_C à la dernière valeur mémorisée : on écrasera les données ne satisfaisant pas la requête. La Sémantique de RESTAURER est :

$$\begin{cases} RAB_C \leftarrow \text{SOMMET DE PILE P} \\ \text{DEPILER P} \end{cases}$$

Outre la pile P, on a besoin de 4 bascules binaires et de 4 piles supplémentaires :

Bascules :

H_{-1} et H initialisées à 'VRAI' indiquent si le mot courant (H) et le mot de niveau supérieur (H_{-1}) satisfont à la requête (VRAI).

E indique si la sous phrase contient au moins un mot satisfaisant la requête.

Enfin, PROJ indique si le champ courant est à projeter.

Les Piles :

P_E, P_H, P , P_{PROJ} permettent de mémoriser pour chaque niveau l'état des bascules E, H_{-1}, σ , PROJ où rappelons que σ est associé à une sous-phrase et est l'une des 3 fonctions (identité, négation, constante).

A chaque scrutation d'un attribut racine intervenant dans le critère de sélection de la requête, on modifie éventuellement H : H ← Faux, si le critère n'est pas satisfait.

Aussitôt que H est faux, on cesse les comparaisons et copies : on ne fait que des SAUTS jusqu'à la fin du mot.

On peut maintenant donner la Sémantique des ordres fichiers :

DEB-ARBRE :

```
E ← Faux ;
EMPILER H-1 ;
H-1 ← H ;
EMPILER σ ;
{σ et PROJ reçoivent les valeurs lues dans le noeud de MT} ;
```

DEB-INSTANCE :

```
EMPILER E ;
EMPILER PROJ ;
SI H-1=VRAI alors  DEBUT
                   SI Proj=VRAI alors EMPILER RAB_C ; (marquer)
                   H ← VRAI ;
                   FIN ;
Sinon H ← FAUX ;
```

FIN-INSTANCE :

$E \leftarrow$ DEPILE(P_E)
$PROJ \leftarrow$ DEPILE(P_{PROJ})
Si H_{-1}=VRAI alors DEBUT
si Proj=VRAI
Alors Si H=VRAI alors DEPILER RAB_c {effacer}
Sinon RESTAURER RAB_c ; {écraser}
$E \leftarrow E \cup H$;
FIN ;

FIN-ARBRE :

$\sigma \leftarrow$ DEPILE(P_σ) ;
$H_{-1} \leftarrow H_{-1} \cap \sigma(E)$;
$H \leftarrow H_{-1}$
$H_{-1} \leftarrow$ DEPILE(P_H) ;

ARBRE-VIDE :

$H \leftarrow H \cap \sigma$ (Faux) {σ est lu dans le noeud de MT}

Ceci complète l'algorithme de filtrage logiciel.

Dans le paragraphe suivant, on estime la taille des automates de Sélection/Projection et on la compare à celle des automates du Filtre VERSO matériel.

5.9. Taille des Automates. Comparaison avec le Filtrage matériel

On commence par évaluer le nombre de lignes de la matrice de transition. On évaluera ensuite le nombre de colonnes.

5.9.1. Nombre d'Etats

Soient E le nombre d'états d'un automate de sélection, F le nombre d'états de l'automate fichier et S le nombre d'états des sous-automates de comparaison.

On a :

$$E = F+S \qquad (1)$$

Commençons par évaluer F. On a :

$$F = (1+a)N \qquad (2)$$

où N est le nombre d'attributs et a < 1 mesure le degré de compaction du fichier* : a est le rapport entre le nombre de noeuds internes ('*') et le nombre de feuilles dans l'arbre syntaxique associé au format du fichier.

En effet le nombre d'états de l'automate fichier est exactement le nombre de noeuds (internes + feuilles) de l'arbre syntaxique associé au format du fichier (voir paragraphe 5.8.2 et Figure 5.6). Le nombre de feuilles est égal à N (nombre d'attributs), tandis que le nombre de noeuds internes est le nombre de '*' de l'arbre syntaxique associé au format.

Dans le cas d'un fichier plat (de format $(A_1 A_2 \ldots A_n)^*$), on a

$$F = 1+N$$

Dans le cas d'un fichier de format simple-droit,

$$(A_1(A_2(\ldots A_N^*)^*\ldots)^*$$

---

* Un fichier est plus ou moins compacté suivant son format. La représentation classique d'une relation en fichier "plat" n'est pas compactée (a=0), tandis qu'un fichier de format simple-droit peut être très compact (a=1), voir chapitre 2.

on a $F = 2N$

On montre facilement que dans le cas où le fichier n'est pas simple-droit on a également $F \leq 2N$.

Evaluons maintenant le nombre S d'états dus à la comparaison. Soit p le nombre d'attributs intervenant dans la condition de la requête. Soit $m_i \geq 1$ le nombre de chaînes sur lesquelles porte la comparaison pour l'attribut A_i : la condition sur l'attribut A_i est de la forme,

$$C_i : A_i \; op_1 val_1 \; \underline{ou} \; A_i \; op_2 val_2 \ldots \underline{ou} \; A_i \; op_{m_i} val_{m_i} \qquad i=1,\ldots,p$$

Le nombre d'états S_i du sous-automate de comparaison correspondant à A_i est égal au nombre de noeuds <u>internes</u> de l'arbre lexicographique contracté (ALC, voir paragraphe 5.6).

Les tailles des Arbres lexicographiques (AL) élagués ou non ont été étudiés dans [42] sous un certain nombre d'hypothèses.

Le nombre de noeuds internes de l'ALC est le nombre de noeuds d'un AL n'ayant <u>qu'un seul</u> fils. Nous nous servons de l'approximation asymptotique suivante de cette quantité donnée dans [42].

Hypothèses :

1) chaque chaîne a une longueur fixe ℓ telle que $\alpha^\ell >> m_i$ où α est la taille de l'alphabet Σ,

2) chaque chaîne prend ses valeurs uniformément dans Σ^ℓ indépendamment des autres chaînes.

3) on fait l'approximation $\ell \to \infty$.

On a alors :

$$Si \simeq 1 + \frac{m_i}{\log \alpha} \frac{\alpha-1}{\alpha} + 0(m_i^{2/3}) \qquad (3)$$

et

$$S = \sum_{i=1}^{p} S_i \qquad (4)$$

Considérons l'exemple du paragraphe 5.6 sur le fichier $(A(BD^*)^*C^*)^*$

C = MATH $\vee$ MATHS $\vee$ MATHEMATIQUE $\vee$ MUSIQUE $\vee$ PHYSIQUE.

On a F = 8
S = 3
E = 11

Remarquons que l'approximation (3) donne S = 1,7 (m_i=5, α=32).

Dans ce qui suit, on considère que bN $\leq$ N attributs interviennent dans la condition de sélection, chacun générant un sous-automate de taille S_i=2.

On a donc S = 2bN et de (1) et (2), on obtient l'approximation pour le nombre d'états de l'automate :

$$E = (1+a+2b)N \qquad (5)$$
$$a,b < 1$$

Filtrage Matériel

L'automate de Sélection dans le cas du Filtrage matériel a été étudié dans [42]. On y distingue les états "logiques" (qui correspondent aux états "fichier" du filtrage logiciel) et les états des sous-automates de comparaison de chaînes de caractère.

Bien que la taille des automates fichier (F) du filtrage logiciel soit bien inférieure au nombre d'états logiques (Filtrage Matériel) il est difficile de faire une comparaison quantitative par manque de résultats pour le filtrage matériel. On ne trouve dans [42] qu'une borne supérieure du nombre d'états logiques E_1, E_2 respectivement pour les formats $(A_1 \ldots A_N)^*$ et $(A_1(\ldots(A_N)^*)^*\ldots)^*$ sous certaines hypothèses restrictives :

$$E_1 = 1 + N \times 2^p$$

$$E_2 = 1 + \sum_{k=1}^{N} (k+1)^p$$

où p est le nombre de disjonctions de la condition de sélection :

$$C = C_1 \cup C_2 \cup \ldots \cup C_p$$

La taille des sous-automates de comparaison, par contre, a été évaluée [42]. Elle est donnée par le nombre de noeuds internes d'un AL de m chaînes, chacune ayant une longueur :

$$T(m,\ell) = 1 + m\ell - \frac{m}{\log \alpha} [\log m + \gamma - 1 - \log \alpha] + 0(m^{1/2})$$

Une approximation grossière est la suivante :

$$T(m,\ell) = m\ell \qquad (6)$$

La comparaison des équations (3) et (6) confirme que la contraction de l'AL réduit considérablement le nombre de noeuds donc le nombre d'états des sous-automates de comparaison.

En résumé, sans être capable de donner une estimation quantitative précise, on peut néanmoins affirmer que le nombre d'états est bien moins grand dans le cas du Filtrage logiciel décrit plus haut que dans le cas du Filtre VERSO matériel.

On étudie maintenant le nombre de colonnes de la matrice de transition.

5.9.2. Nombre de Caractères

Dans le cas du filtrage matériel, ce nombre est la taille α de l'alphabet Σ.

Par contre, le nombre de caractères pour le filtrage logiciel est donné par

$$NC = MAX(N, 2m_{max} + 1) \qquad (7)$$

où m_{max} est le nombre maximum de chaînes à comparer pour un attribut intervenant dans la requête. En effet NC est le maximum entre le nombre d'identificateur d'attribut (ia) différents (N) et le nombre maximum de transitions possibles à l'issue d'une comparaison (valeur maximum du retour calculé, voir paragraphe 5.6).

Le gain en nombre de colonnes est encore plus important que le gain en nombre de lignes par rapport au Filtrage matériel.

Notons toutefois que pour réduire le nombre de colonnes dans le cas du Filtrage matériel, on peut ne garder que les caractères utiles, c'est-à-dire les ia et 2q+1 caractères où q est le nombre total de caractères différents intervenant dans la requête. Ceci nécessite une matrice de transcodage des caractères lus dans la Source, cette matrice étant chargée en même temps que l'automate.

5.9.3. Taille de la matrice

5.9.3.1. Exemple précédent (Fichier : $(A(BD^*)^*C^*)^*$; requête : C = MATH U ... U PHYSIQUE)

a) Filtrage logiciel : E=11 (nombre de lignes)
NC=7 (nombre de colonnes)
NN=E × NC=77 (nombre de noeuds)

En supposant qu'un noeud peut être représenté par 5 octets (voir Figure 5.4), la taille de la matrice est :

TM = 385 octets

b) Filtrage matériel : Le nombre d'états du sous-automate (nombre de noeuds de l'AL) est 29 ; le nombre d'états dits "logiques" est estimé à 10. On a :

E = 39

En supposant l'existence d'une matrice de transcodage, le nombre de caractères différents est q=11. Par conséquent on a :

NC = 2q + 1 = 23
NN = E × NC = 897

La taille d'un noeud est effectivement de 4 octets. Par conséquent

TM = 3588 octets.

5.9.3.2. Filtrage logiciel, cas général

Dans l'application numérique qui suit, on suppose que $2m_{max}+1 \leq N$, i.e. que NC = N (8).

Des équations (5) et (8), on tire l'estimation suivante du nombre de noeuds :

$$NN = (1 + a + 2b)\ N^2 \qquad (9)$$

Les figures 5.7 et 5.9 donnent une idée de la taille de la matrice TM en fonction du nombre d'attributs N, du degré de compaction a et du nombre d'attributs bN de la requête. On suppose que la taille d'un noeud est de 5 octets : TM = 5 NN où NN est donné par l'équation (9).

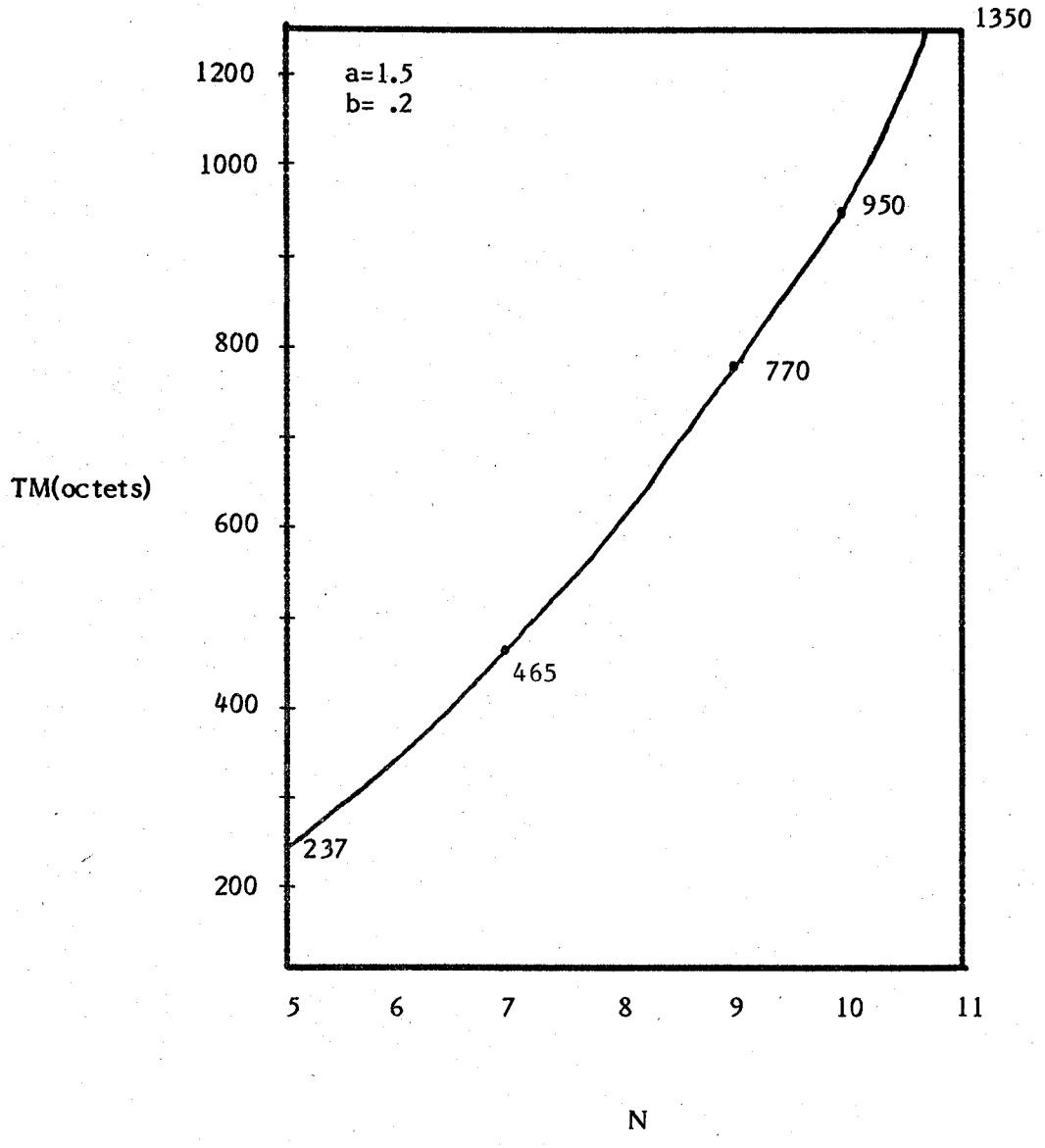


Figure 5.7. Taille de la Matrice en fonction du nombre d'Attributs

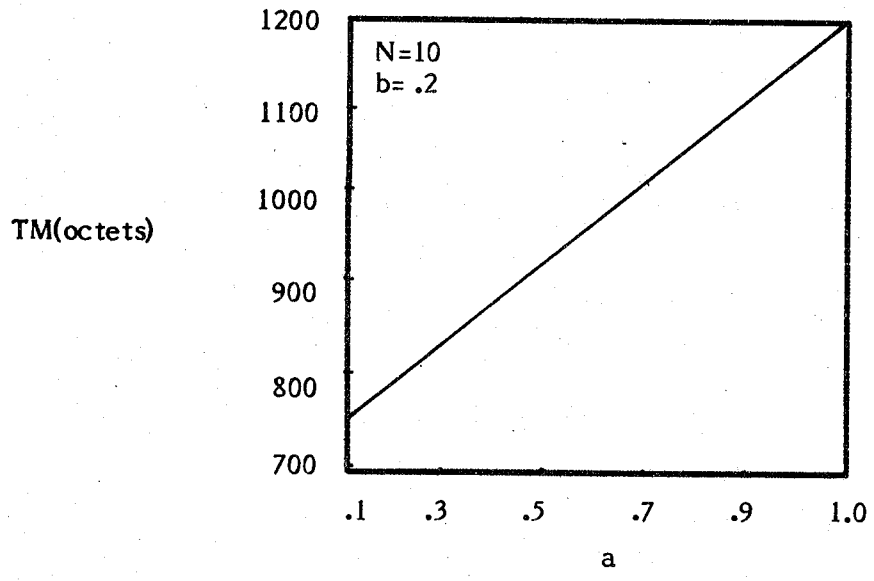


Figure 5.8. Taille de la Matrice en fonction du degré de compaction

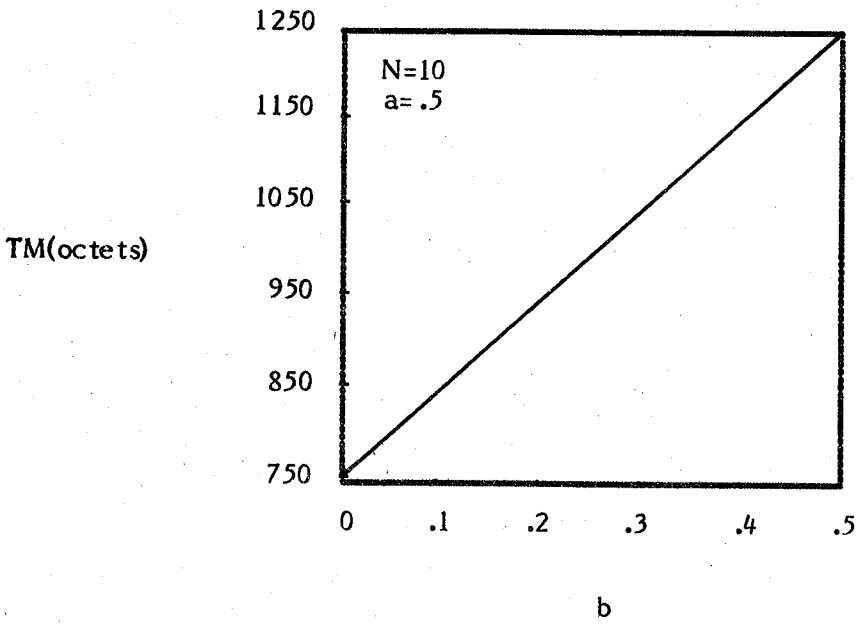


Figure 5.9. Taille de la Matrice en fonction du nombre d'attributs dans la requête (bN).

5.10. Conclusion

Une implantation logicielle du filtrage par automate fini a été étudiée en détail dans ce chapitre. Une variante optimisée de cette implantation [15] a été réalisée pour le système VERSO, et représente l'alternative au filtrage matériel décrit dans le Chapitre 3.

Bien que seul le cas de la Sélection/Projection ait été traité dans ce chapitre, l'implantation des opérations binaires se traite de manière analogue et a été réalisée pour le système VERSO.

La souplesse d'une implantation logicielle a permis d'améliorer l'algorithme de filtrage (ce que ne permet pas une architecture matérielle figée) et de pallier à la lenteur d'un processeur banalisé en améliorant le balayage :

En particulier, alors que le filtre matériel balaie les données caractère par caractère, le filtre logiciel quant à lui ne "voit" que des mots et ne scrute caractère par caractère que les mots nécessaires.

Une des conséquences de ceci est que la taille des automates est considérablement réduite. Ce n'est pas tant le gain en place mémoire qui est intéressant que le gain en temps de préparation du filtrage (compilation et surtout chargement de l'automate).

Toutefois, on ne peut pas s'attendre à des temps de filtrage aussi courts avec cette solution logicielle qu'avec une architecture spécialement conçue à cet effet.

L'intérêt d'une architecture logicielle, c'est-à-dire essentiellement la modicité de son coût, n'a-t-il pas pour contrepartie, un ralentissement considérable du filtrage ?

Le Chapitre suivant tente de répondre à cette question en comparant les temps de réponse à une requête de Sélection/projection pour les deux architectures.

CHAPITRE 6

COMPARAISON ENTRE FILTRAGES LOGICIEL ET MATERIEL

6. - COMPARAISON ENTRE FILTRAGES LOGICIEL ET MATERIEL

Le filtrage logiciel est implanté sur deux types de Modules d'échanges (ME). On estimera dans ce chapitre les temps de réponse à une requête de sélection/projection avec chacun des Modules d'échange et on comparera les temps obtenus à celui que procure le Filtre matériel décrit dans le Chapitre 3.

En vue d'évaluer le temps de réponse avec les architectures logicielles, on estimera tout d'abord le temps de cycle c'est-à-dire le temps moyen de traitement d'un caractère.

On étudiera ensuite en détail l'architecture des Modules d'échange. Ceci nous permettra d'évaluer le ralentissement du processeur banalisé lorsque, simultanément au filtrage, a lieu le chargement d'un bloc à filtrer du Disque vers la mémoire ou un vidage vers le Disque d'un bloc filtré.

L'utilisation de plusieurs tampons en mémoire permet un certain parallélisme entre le filtrage d'une part et le chargement ou le vidage d'un bloc d'autre part. Même si le filtrage est ralenti par une E/S simultanée, intuitivement on s'attend à ce que plus le nombre de tampons est grand, meilleur est le temps de réponse. On évaluera le nombre optimal de tampons source et cible, c'est-à-dire celui qui minimise le temps de réponse.

6.1. Estimation du temps de cycle en Filtrage logiciel et de la vitesse relative du filtrage par rapport aux E/S

Rappelons qu'un cycle commence avec l'accès à un noeud de la matrice de transition identifié par un couple (ETAT,CAR) et se termine juste avant l'accès au noeud suivant après interprétation des ordres contenus dans la matrice.

Dans le filtre matériel, à l'issue d'un cycle, la tête de lecture de l'automate contient le caractère suivant : le pointeur a avancé d'un caractère dans le tampon Source. Le temps moyen de cycle τ est égal à 400 ns.

Dans le Filtre logiciel par contre, entre 2 accès successifs à la Matrice de transition, le nombre de caractères lus dans le tampon Source est variable : la reconnaissance d'attribut, i.e. la lecture de l'identificateur d'attribut et de la longueur du champ fait avancer le pointeur de 2 ou 4 caractères ; un saut ou une copie fait avancer le pointeur de la longueur de l'attribut ; lors d'une comparaison, une partie de l'attribut est parcourue.

Nous appellerons par extension "cycle" en filtrage logiciel, le traitement d'un caractère. Soit θ l'espérance mathématique du temps de cycle (calculé en nombre de périodes élémentaires du processeur). Le but de ce paragraphe est de donner une estimation grossière de la valeur de θ. Un modèle plus fin permettrait de valider les résultats ci-dessous.

θ dépend de trois types de paramètres :

a) langage d'écriture du filtre logiciel et vitesse du processeur :
Soit τ le temps moyen en μs de filtrage d'un caractère :

$$\tau = \theta \times t_c \qquad (1)$$

où t_c désigne la période élémentaire du processeur. Avec le ME rapide (horloge à 8MHz) $t_c = 0{,}125\ \mu s$, tandis que le ME lent (horloge à 5MHz) a une période $t_c = 0{,}2\ \mu s$.
La version actuelle du filtre logiciel est réalisée en Pascal mais l'estimation ci-dessous correspond à une écriture en assembleur 8086.

b) la V-relation : son format, son nombre d'attributs, la taille des attributs.

c) la Requête : le nombre d'attributs sur lequel porte une comparaison et/ou qui sont à projeter.

6.1.1. Hypothèses et Notations

1) V-relation : on supposera une V-relation en format plat. Le nombre d'attributs est noté A. On supposera que tous les attributs ont la même longueur de a caractères.

2) Requête : Soient S le nombre d'attributs intervenant dans le critère de Sélection et P le nombre d'attributs à projeter

3) Durée de cycle : on suppose qu'il existe trois types d'attributs seulement :

3.a) **Attributs sautés** : Les attributs qui ne sont ni à comparer ni à projeter sont sautés. Le filtrage de tels attributs consiste à lire la matrice, lire la longueur du champ, incrémenter le pointeur Source, lire le caractère suivant et mettre à jour le registre d'état courant. Le temps moyen de cycle pour de tels attributs est estimé à :

$$\theta_s = \frac{N_s}{a}$$

où N_s est le nombre de périodes élémentaires nécessaires pour sauter un attribut.

3.b) **Attributs à projeter** : Il s'agit des attributs à recopier du tampon Source dans le tampon Cible sans comparaison. On utilise à cet effet les instructions spécialisées de copie de chaines de caractères. La projection d'un attribut consiste à lire un noeud de la matrice, à copier l'identificateur d'attribut dans la

Cible, à lire la longueur du champ et à la copier dans Cible et enfin à recopier le champ lui-même. Cette copie est proportionnelle à la longueur a du champ. Soit N_p le nombre de périodes élémentaires nécessaires pour lire un caractère et le copier dans la Cible. Le temps moyen de cycle est alors égal au temps de copie plus le temps de saut :

$$\theta_p = N_p + \theta_s$$

3.c) **Attributs de Sélection** : Il s'agit des attributs à comparer et éventuellement à projeter. Le temps de comparaison d'un attribut dépend bien évidemment du résultat (échec ou succès) et s'il y a eu échec quand celui-ci s'est produit. La comparaison est plus longue s'il y a en même temps projection de l'attribut et le nombre d'accès à la matrice dépend du nombre m de chaines de caractères à comparer. Le temps de comparaison à une chaine est proportionnel à la taille des chaines. Des instructions spécialisées de comparaison de chaines de caractères sont utilisées. Nous supposerons pour simplifier que :

(i) il y a toujours succès,

(ii) la comparaison d'un attribut consiste en 2 accès à la matrice de transition plus la comparaison et la copie d'une chaine de taille a. Soit N_c le nombre de périodes élémentaires nécessaires pour lire un caractère dans le tampon Source, le comparer à un caractère donné et le copier dans le tampon Cible. Le temps d'accès à la matrice est supposé comme pour la projection être celui d'un Saut et durer N_s périodes élémentaires. On a alors :

$$\theta_c = 2\theta_s + N_c$$

6.1.2. Temps moyen de cycle

De ce qui précède, on en déduit l'estimation suivante de θ :

$$\theta = \frac{A-S-P}{A}\,\theta_S + \frac{P}{A}\,\theta_p + \frac{S}{A}\,\theta_c$$

$$= \left(1 + \frac{S}{A}\right)\frac{N_s}{a} + \frac{P}{A}\,N_p + \frac{S}{A}\,N_c \qquad (2)$$

Remarques :

1) L'estimation précédente suppose donc que le filtrage d'un attribut consiste en

(i) un "overhead" en temps constant quel que soit le type d'attribut traité (N_s)

(ii) éventuellement d'un temps de comparaison/copie proportionnel à la longueur de l'attribut.

L'"overhead" comptera d'autant moins dans le temps total que l'attribut est plus long.

2) L'estimation ci-dessus est très pessimiste car elle ne tient pas compte des optimisations consistant à abandonner sélection et projection dans une phrase (ou au moins une sous-phrase) dès qu'il y a échec et à n'effectuer alors que des SAUTS.

Dans l'application numérique ci-dessous, on a choisi $\frac{S}{A} = \frac{P}{A} = \frac{\rho}{2}$.

$$\theta = \frac{N_s}{a} + \frac{\rho}{2}\left[\frac{N_s}{a} + N_p + N_c\right] \qquad (3)$$

θ est tracé en fonction de ρ dans la Figure 6.1 pour 3 tailles d'attributs. Les valeurs de N_p et N_c correspondant aux notices Intel, la valeur de N_s a été estimée à 120 périodes. (voir Annexe).

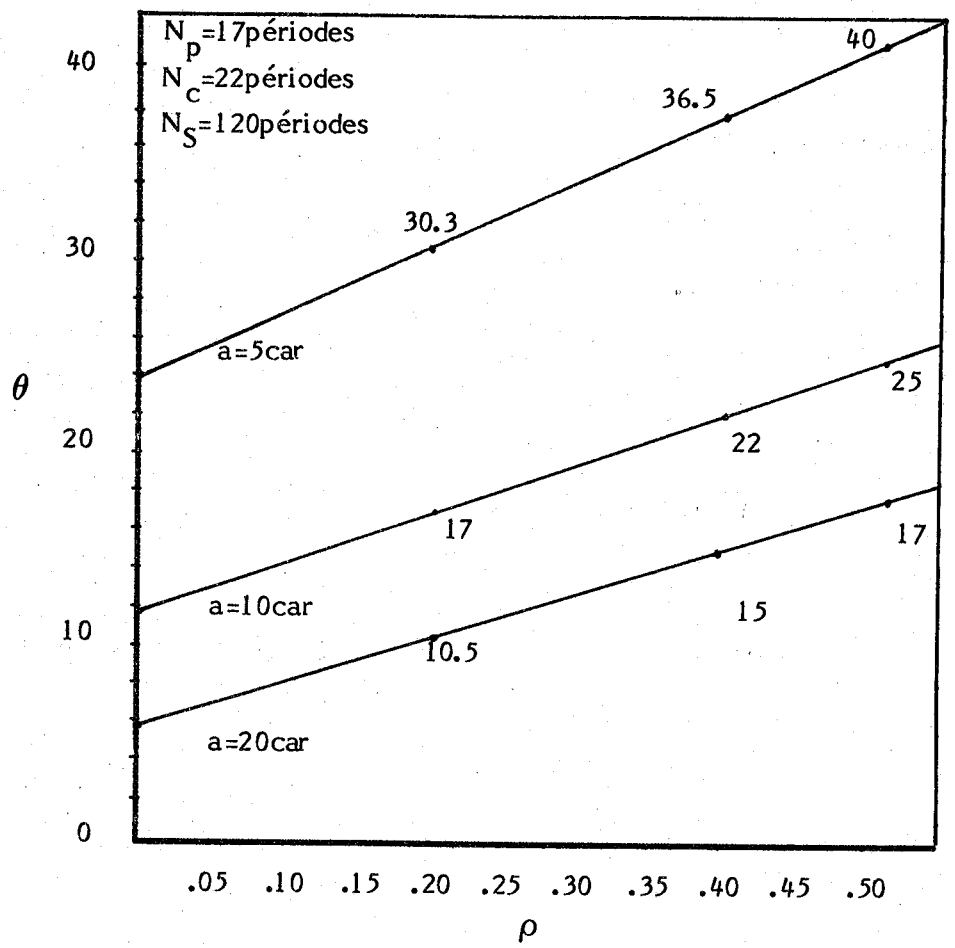


Figure 6.1. Temps moyen de cycle θ (en nombre de périodes) en fonction de la proportion d'attributs à comparer et projeter

$$\rho = \frac{S+P}{A}$$

6.1.3. Validation de l'estimation du temps de cycle moyen

Une estimation du temps de cycle moyen τ est obtenue de l'estimation θ ci-dessus par l'équation (1) : $\tau = \theta \times t_c$, où t_c est la durée d'une période élémentaire (t_c = 0,2μs pour le ME SASI).

En vue d'évaluer l'estimation de τ, on la compare (Figure 6.2) aux premières mesures de τ obtenues sur le ME SASI. Le filtrage est écrit en Pascal et consiste à effectuer une projection sur un fichier plat de 200 000 caractères environ, de 10 attributs par n-uplet et de taille moyenne d'attribut 4 caractères.

Pour obtenir le temps de cycle moyen (mesuré : Figure 6.2.b), on a divisé le temps total de filtrage par la taille du fichier et on fait varier le nombre d'attributs P qu'on projette.

Le temps de cycle moyen estimé pour un filtrage en assembleur (Figure 6.2.c) est obtenu des équations (1) et (2) avec les valeurs numériques suivantes des paramètres :

N_s=120, N_p=17, a=4, t_c=0,2μs, A=10

On a alors : $\tau = 6 + .34P$

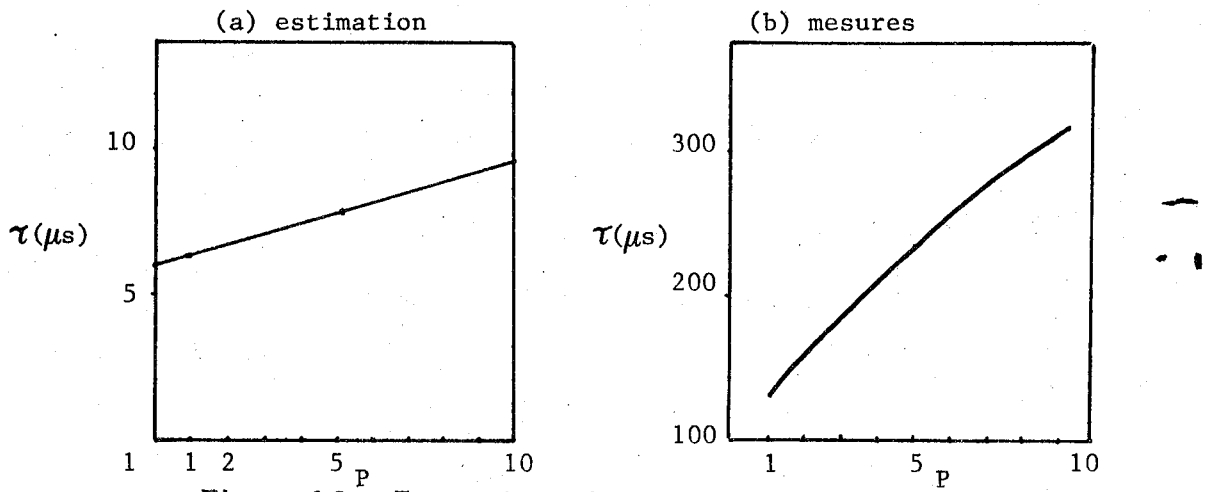


Figure 6.2. Temps de cycle moyen en Projection en fonction du nombre d'attributs projetés P sur le ME SASI

Une constatation s'impose à la lecture de la Figure 6.2. Le rapport entre temps mesuré et temps estimé est considérable puisqu'il est de l'ordre de 20 !

Est ce à dire que l'estimation ci-dessus est optimiste ? Certes on peut l'affirmer. Cependant il faut remarquer que :

(i) le temps de cycle mesuré correspond à une écriture du Filtrage en Pascal et non en Assembleur. On peut espérer gagner un rapport de 1 à 10 par une écriture Assembleur des parties critiques du code,

(ii) il s'agit d'une première implantation et l'algorithme de filtrage demande encore à être optimisé.

Il est à noter que le temps mesuré n'est pas tout à fait linéaire avec le nombre d'attributs à projeter. Ceci est dû au fait que la taille des attributs projetés n'est pas constante.

Bien que l'estimation de θ soit optimiste, c'est elle que nous garderons dans la suite du chapitre comme borne inférieure sur le temps de cycle que l'on peut atteindre avec le filtrage logiciel.

6.1.4. Vitesse relative du filtrage par rapport aux E/S

Soit r la vitesse relative du filtrage par rapport au transfert des données entre Disques et mémoire RAM. On a :

$$r = \frac{\tau}{\tau_D} \qquad (4)$$

où τ (équation (1)) indique le temps de cycle moyen en μs et τ_D le temps de transfert d'un caractère : τ_D = .8μs pour l'interface SMD, 6,2 μs pour l'interface SASI(*).

La figure 6.3 représente r en fonction de ρ (équations 1,3,4) pour 3 valeurs de la taille d'attribut dans le cas du module d'échange SMD (période t_c = 125 ns).

(*) La vitesse de transfert dépend du contrôleur plutôt que du transfert. Bien que des disques à interface SASI peuvent atteindre des vitesses de l'ordre de 10 Mbits/s, le débit utile dont nous disposons actuellement est considérablement diminué par le contrôleur utilisé (Voir paragraphe 6.2.1).

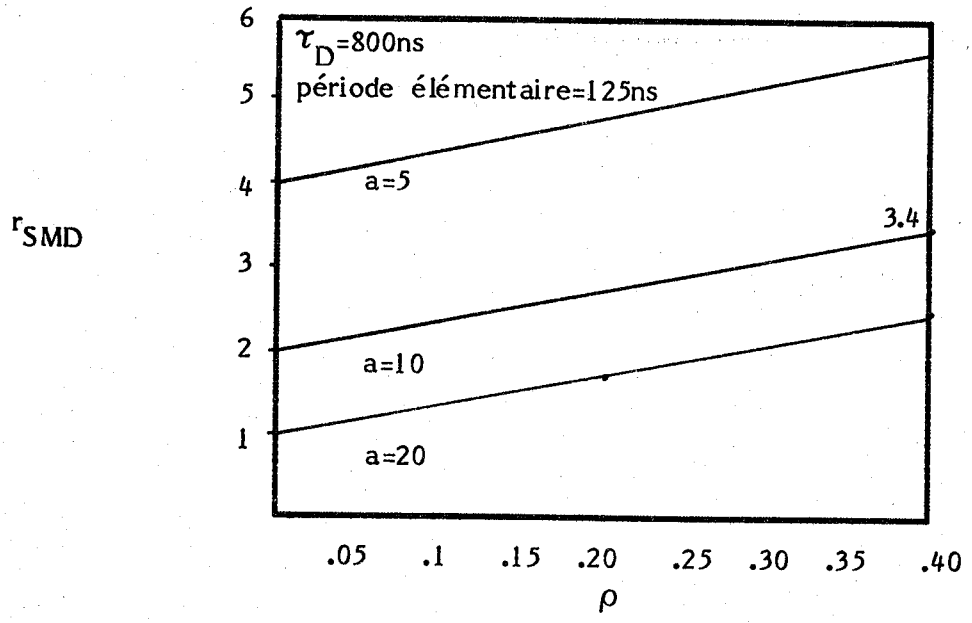


Figure 6.3. Vitesse relative de filtrage en fonction du nombre relatif d'attributs intervenant dans la requête pour 3 longueurs d'attributs (ME SMD)

Discussion

Alors que le filtre matériel va 2 fois plus vite que le transfert Disque/Mémoire (τ=400ns, r=1/2), le filtrage logiciel avec le ME SMD est considérablement plus lent. En particulier, pour une longueur d'attributs de 10 caractères et une valeur de ρ=.4, le temps moyen de filtrage d'un caractère est τ=2,75μs et le filtrage logiciel est trois fois plus lent que le transfert Disque/Mémoire (r=3.4) et presque 7 fois plus lent que le filtrage matériel.

Toutefois pour les applications "gros objets", où les attributs sont très longs, le filtrage est plus rapide (r=1.6 si a = 20 et ρ=.20). La figure 6.4 donne la borne inférieure de r, pour de gros objets, en fonction de ρ (toujours pour le ME SMD)

$$r = \left[\frac{N_p + N_c}{2} \right] \times \frac{t_c}{\tau_D} \times \rho$$

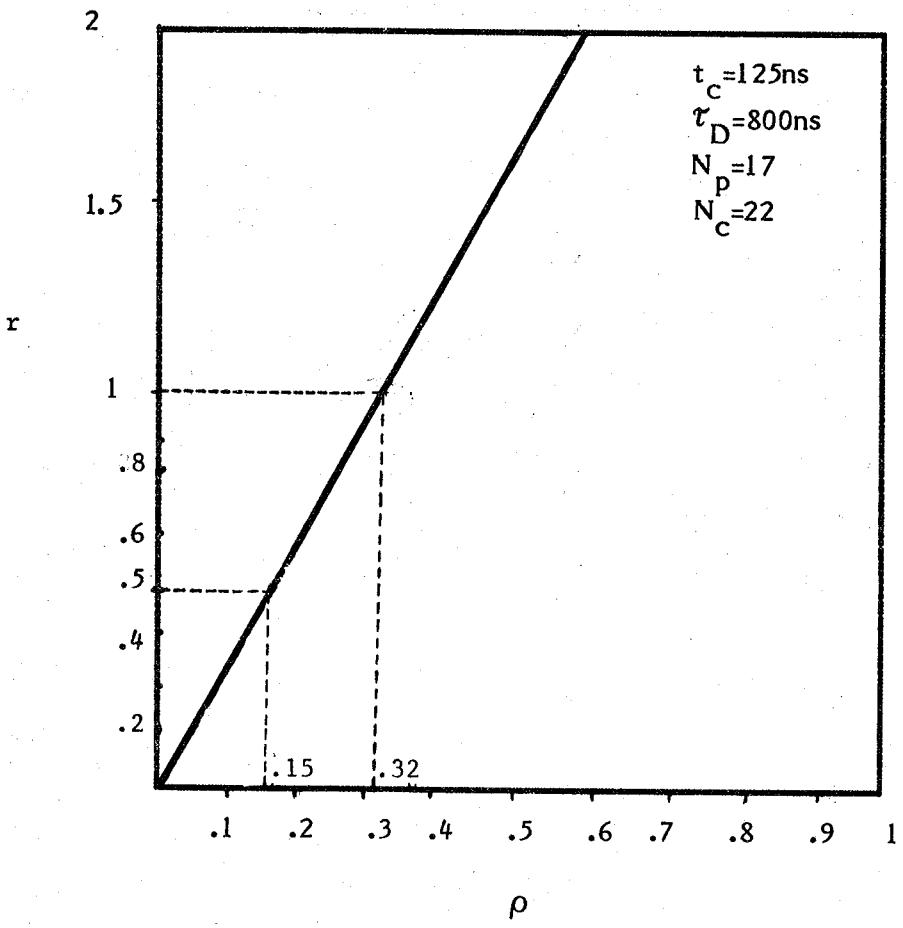


Figure 6.4. Borne inférieure de la vitesse relative de filtrage en fonction de la proportion d'attributs intervenant dans la requête, pour de gros objets.

Cette borne n'est réaliste que si les attributs sont suffisamment longs (e.g. si ρ=.4, a > 60 caractères). Dans ce cas le filtrage logiciel peut non seulement être plus rapide que le transfert Disque/mémoire (r < 1 si ρ < .32), mais encore il peut être plus performant que le filtrage matériel si la proportion d'attributs dans la requête est petite (r < .5 si ρ < .15).

Dans la suite de ce chapitre, nous choisirons comme estimation de r_{SMD}, r=3 approximation de la valeur obtenue pour a =10, ρ=.4 : r=3.4 (Figure 6.3) : cette estimation correspond à une valeur de θ de 22 cycles (Figure 6.1). Cependant dans le cas du ME SASI (τ_D=6,2 μs, t_c=200 ns), la valeur θ=22 cycles donne pour vitesse relative r_{SASI} = 3/5 = .6. Le temps de transfert disque/mémoire est si grand, que malgré la lenteur du processeur, on est limité par les E/S ! Remarquons toutefois que la vitesse relative correspondant aux premières mesures (Filtrage en Pascal, ME SASI : Figure 6.2) est bien plus grande. En effet (Figure 6.2 et Equations 1 et 4) si p=.1 et a=4, on a r_{SASI} = 130/6,2 $\simeq$ 20 ! On est limité par le filtrage et non par les E/S.

6.2. Partage de la mémoire entre le processeur et le Disque*

1) Module d'échange 8086/SASI

Dans le module d'échange 8086/SASI un processeur 8089 et un processeur 8086 (horloge à 5MHz) se partagent l'accès à la mémoire partagée MP (Figure 6.5). Le 8089 lit un secteur dans le tampon du Contrôleur et l'écrit dans la mémoire MP, octet par octet (Figure 6.5). Avant ce transfert, quand un secteur a été lu du Disque dans le tampon, le contrôleur avertit le 8089 qui envoie un "Bus request" au 8086. Celui-ci termine l'instruction en cours et donne la main au 8089 qui la lui rendra une fois le secteur transféré en MP. Pour un transfert de la mémoire MP vers le Disque le mécanisme est similaire.

(*) Pour plus de détails, voir [28]

Le diagramme de la figure 6.6 indique les différentes phases au cours du temps lorsque plusieurs secteurs consécutifs sont transférés du Disque vers MP.

La première phase c'est la lecture d'un secteur (256 octets) du disque dans le tampon par le contrôleur et elle dure environ 530μs.

Il s'écoule alors un délai moyen de 220 μs avant que le 8089 commence le transfert du 1er octet du tampon vers MP (phase 2).

Ce transfert d'un secteur vers la MP dure 400 μs (3e phase). Entre temps plus d'un secteur a défilé devant la tête de lecture du disque. Il faut attendre en moyenne 440 μs (phase 4) jusqu'à la fin du secteur, avant que le contrôleur puisse à nouveau lire un secteur.

Le contrôleur lit donc un secteur sur 3. Il faut 3 tours de piste pour lire une piste entière (entrelacement 3).

Entre 2 lectures successives d'un secteur, 1590 μs s'écoulent environ. Au cours de ce cycle, le 8086 n'est arrêté que 400 μs.

Par conséquent lorsque le 8086 travaille en même temps qu'un transfert Disque $\leftrightarrow$ MP, le ralentissement γ du 8086 est :

$$\gamma = \frac{400}{1590} \simeq .25$$

On dénote par τ_{SASI} le temps de transfert d'un caractère entre le disque et la mémoire MP (en supposant que plus d'un secteur est transféré). On a :

$$\tau_{SASI} = \frac{1590}{256} \simeq 6{,}2 \ \mu s$$

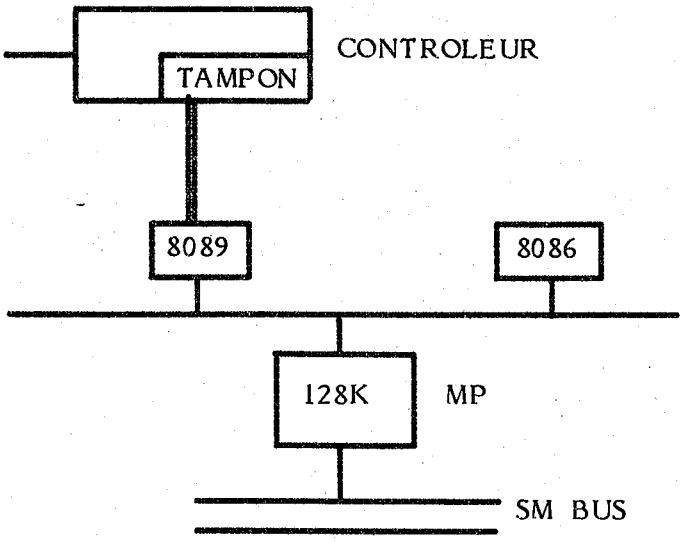


Figure 6.5. Architecture du ME 8086/SASI

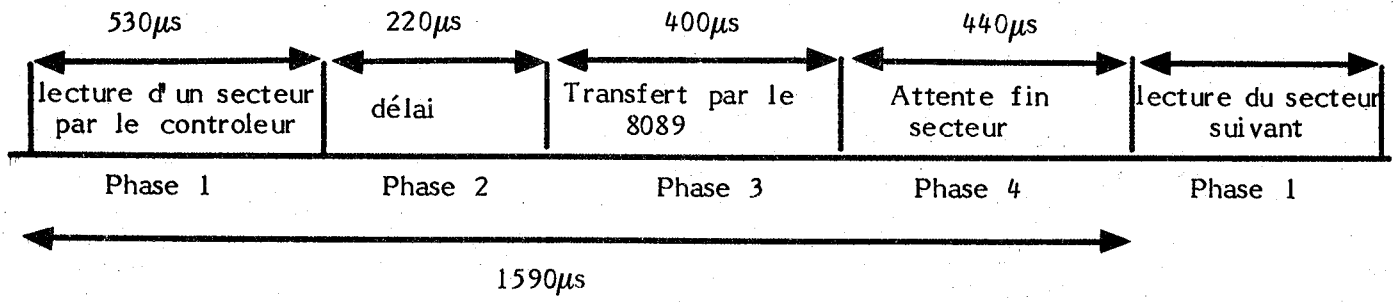


Figure 6.6. Cycle de transfert d'un secteur entre le Disque et la mémoire MP : Module 8086/SASI

2) Module d'échange 80186/SMD

L'architecture de ce module en cours de test [28] est décrite dans la Figure 6.7 : Un DMA câblé permet le transfert d'octets entre le Disque et la mémoire MP (512 Koctets de RAM dynamique) au moyen d'une FIFO de capacité maximale 6 octets et de 2 latches. Le processeur 80186 (horloge à 8MHz) accède à MP, à un mémoire RAM privée et à une mémoire EPROM.

L'arbitre de MP résoud les conflits entre 1) la demande d'accès de la part d'un module de traitement SM (Demande Globale), 2) la demande de rafraichissement (Refresh) de MP et 3) la demande locale : accès par le 80186 ou par le DMA. L'arbitrage est non préemptif c'est-à-dire que l'arbitre satisfait une demande plus prioritaire seulement lorsque la demande en cours a été satisfaite. La demande de rafraichissement (toutes les 14 μs, elle dure 250 ns) est la plus prioritaire. La demande locale est la moins prioritaire. La demande globale dure environ 600 ns.

Nous supposerons dans la suite que le MT SM est silencieux et négligerons la demande de rafraichissement. Nous étudions maintenant le partage de la MP entre le DMA et le 80186.

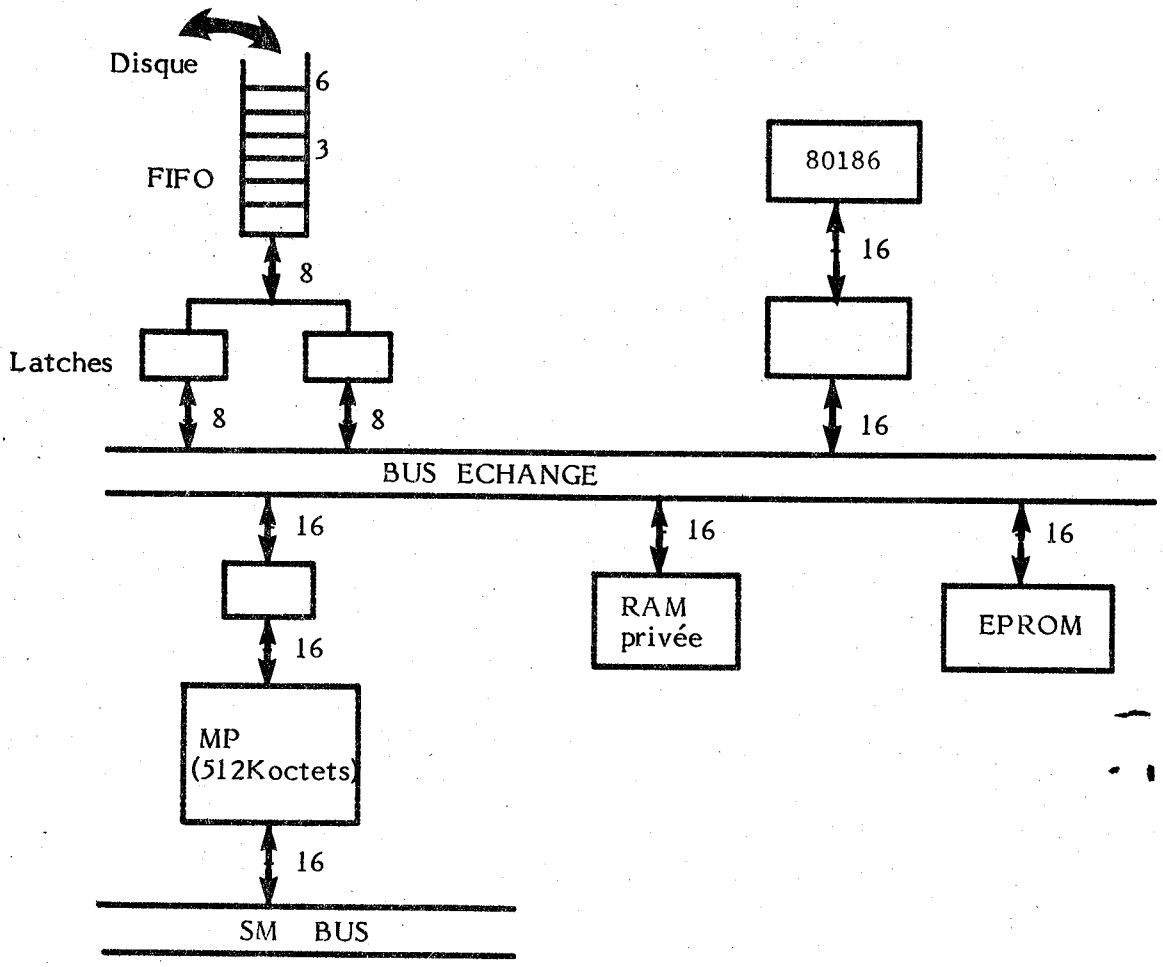


Figure 6.7. Architecture du ME 80186/SMD

On étudie pour cela la lecture du Disque vers MP (l'écriture de MP vers le Disque a un mécanisme tout à fait similaire).

Les octets lus du disque (standard SMD) à un débit de 1,25 Moctets (800 ns par octet) sont entrés dans la file FIFO. Lorsque celle-ci initialement vide a reçu 3 octets (Figure 6.8), un signal DREQ est envoyé au DMA lequel envoie une demande de BUS au 80186 (signal HOLD). Celui-ci libère le bus d'échange une fois que son instruction en cours est achevée (signal HLDA). Le DMA demande alors à l'arbitre l'accès à la MP qui la lui donne ($\overline{DLV}$) si le MT est silencieux et s'il n'y a pas de rafraichissement (au maximum au bout de 850 ns).

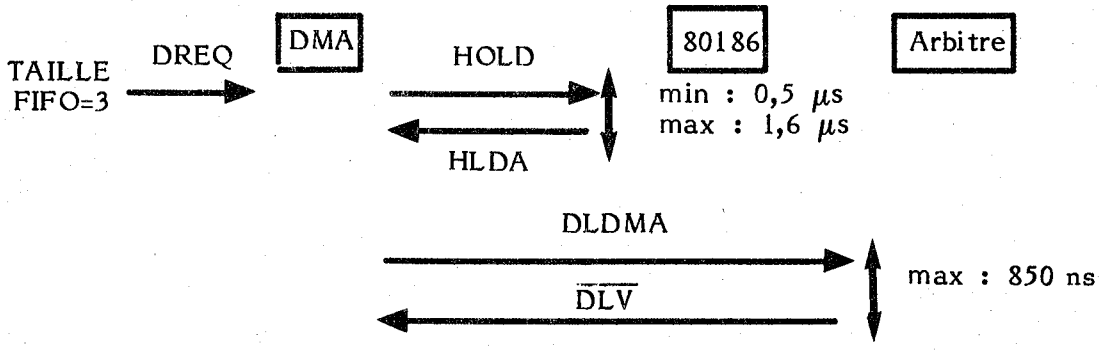


Figure 6.8. Demande d'Accès du DMA à la mémoire MP

Le temps au bout duquel le 80186 envoie le signal HLDA sur un signal HOLD dépend de l'instruction en cours. Il est estimé varier entre 0,5 μs et 1,6 μs.

Une fois que le DMA a reçu l'autorisation de l'arbitre ($\overline{DLV}$) il écrit un mot dans MP (270 ns : Entre temps les latches avaient été chargés à partir de la FIFO). On a alors 2 cas :

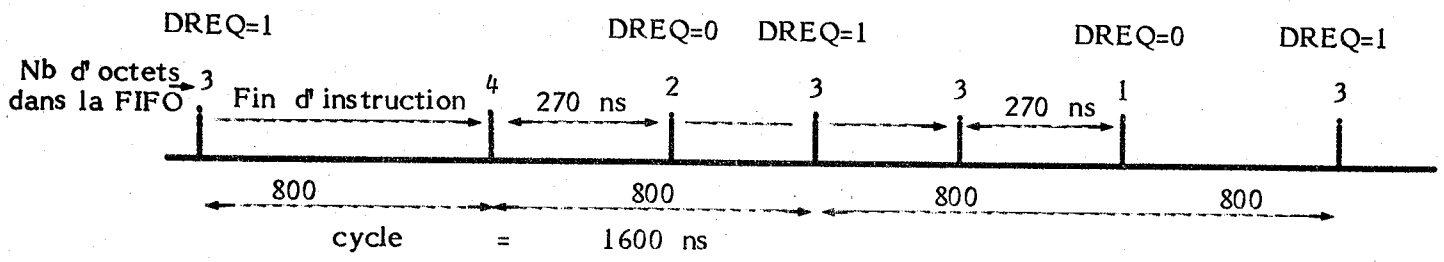
a) DREQ = 0 (Figure 6.9a). Après écriture de ce mot, la file contient moins de 3 octets. Le DMA perd l'accès au profit du processeur qui lance une nouvelle instruction.

b) DREQ = 1 (Figure 6.9b). Le cycle a été suffisamment long pour qu'à l'issue de l'écriture de ce 1er mot, DREQ est toujours à 1 : la file à au moins 3 octets. Le DMA relance l'écriture d'un deuxième mot. Celle-ci dure 570 ns : En effet le temps interécriture est de 300 ns. A l'issue de cette écriture, DREQ = 0, le DMA perd l'accès.

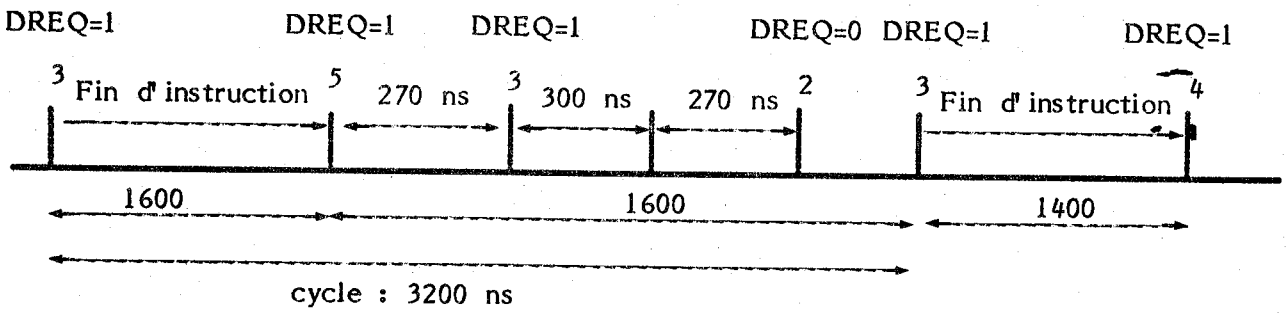
Observons que ce cas est fort improbable puisqu'il suppose que lors de l'événement DREQ=1, l'instruction en cours nécessite encore plus de 10 périodes élémentaires (1 période dure 125 ns). Or une instruction demande en moyenne 5 cycles.

Une étude plus fine devrait prendre en compte la distribution du nombre de cycles des instructions 8086.

On supposera par la suite qu'on est toujours dans le cas a), c'est-à-dire qu'à chaque autorisation (toutes les 1600 ns) un seul mot est écrit dans la MP (270 ns).



a) Fin d'instruction < 1,33 μs



b) Fin d'instrucion > 1,33 μs

Figure 6.9. Cycle de Transfert d'un mot entre la FIFO et la Mémoire MP : Module 80186/SMD

En conclusion, lorsque le processeur 80186 et le DMA travaillent en même temps, avec les hypothèses faites sur la durée d'exécution d'une instruction, au cours d'un cycle de 1600 ns, le 80186 est arrêté 270 ns. Il subit donc un ralentissement :

$$\gamma = \frac{270}{1600} \simeq .17$$

6.3. Algorithme pour la Sélection/Projection

Rappelons que lors du filtrage d'une relation de taille M blocs, la relation est transférée du disque bloc par bloc dans un (ou plusieurs) tampon(s) source de la MP par le processus CHARGEUR. Celui-ci s'arrête lorsqu'il n'y a plus de tampon disponible ou bien lorsque tous les blocs ont été transférés (rappelons que le bloc est l'unité de transfert et que sa taille est un paramètre, par exemple une piste du disque).

Le Processus FILTRE pour démarrer a besoin d'un bloc à filtrer (bloc chargé) et d'un tampon cible pour le résultat du filtrage. Une fois le bloc Source filtré, un tampon est disponible pour le chargement. Lorsque le tampon cible est plein, son adresse est mise dans la file des blocs à vider.

Une fois le bloc vidé vers le disque ou l'utilisateur (mémoire de la SM) par le processus VIDEUR, un tampon cible est disponible pour le processus FILTRE.

Ainsi, pendant que le chargeur (videur) charge (vide) un bloc, le Filtre peut travailler sur 2 autres blocs (Figure 6.10)

Le temps total de filtrage se compose d'un temps de chargement, d'un temps de filtrage proprement dit et d'un temps de vidage du résultat.

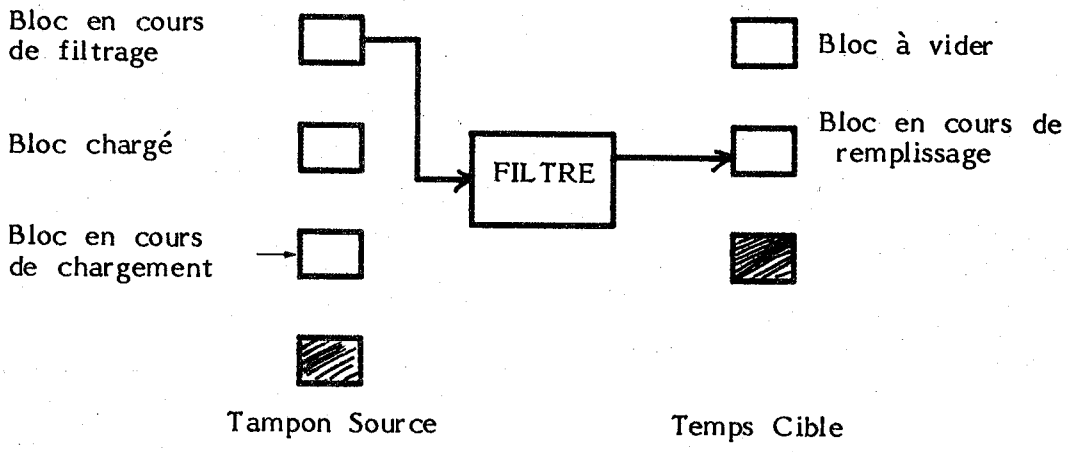


Figure 6.10 Partage des Tampons MP entre les 3 processus Chargeur, Filtre, Videur

Tampon disponible

Ce temps total dépendra du nombre M de blocs de la relation, de la sélectivité p de la requête (définie comme le rapport entre la taille du résultat et la taille M de la relation Source) et de la vitesse relative de filtrage r.

L'utilisation de plusieurs tampons permet un certain parallélisme entre le filtrage d'une part et le chargement/vidage d'autre part.

Nous nous placerons dans un contexte pessimiste, à savoir que la relation est très grande et le résultat tient beaucoup trop de place pour tenir dans la MP : $pM >> n_c$ où n_c est le nombre de tampons cible. Ceux-ci doivent donc être vidés sur le disque.

Une extension à cette étude consisterait à étudier le cas où le résultat est retourné vers l'utilisateur (SM 90). Il existe alors un parallélisme supplémentaire entre le chargement et le vidage et un ralentissement du Filtrage proprement dit dû à l'accès du module de traitement SM à MP lors du vidage (voir paragraphe précédent).

Avec les hypothèses ci-dessus, l'algorithme suivant à été choisi :

1) Charger un premier bloc. Ceci inclut un temps de mouvement de bras initial S + un temps de latence L pour que la tête de lecture soit positionnée en début de piste et enfin un temps T_D de transfert d'un bloc : on a :

$$T_D = b \times \tau_D$$

où b est le nombre de caractères d'un bloc (une piste) et τ_D le temps de transfert d'un caractère (τ_D = 800ns dans le cas SMD, τ_D = 6,2 μs dans le cas SASI). Dans tout ce qui suit, nous supposerons que :

$$S + L = T_D$$

2) Charger le bloc suivant si un tampon source est disponible et en même temps lancer le filtrage du 1er bloc si un tampon cible est disponible.

3) Si aucun tampon cible n'est disponible, i.e. n_c tampons sont pleins du résultat de filtrage, les vider :

   a) Attendre s'il y a lieu la fin de chargement du bloc en cours,

   b) Vider le 1er parmi les n_c blocs à vider (un tampon cible est alors disponible, le filtrage peut alors recommencer). Ceci inclut un temps de mouvement bras S, un temps de latence L et un temps de transfert proprement dit.

   c) Vider les autres blocs

   d) recommencer le chargement des blocs. Ceci inclut un mouvement de bras et un temps de latence.

Un tel algorithme a pour souci de minimiser les aller-retour entre chargement et vidage, i.e. de minimiser les temps de mouvement de bras et de latence.

Intuitivement, plus le nombre de tampons (Source + Cible) est grand, moins grand sera le temps de réponse à une requête de Sélection/Projection suivant l'algorithme ci-dessus.

L'objectif du paragraphe suivant est d'étudier l'influence du nombre de tampons source n_s et du nombre de tampons cible n_c sur le temps de réponse. Clairement si ce nombre de tampons est trop important l'objectif de minimisation du temps de réponse sera atteint au prix d'une très mauvaise occupation de la mémoire partagée MP.

6.4. Evaluation du nombre de tampons Source et Cible

Le paramètre le plus important est la vitesse relative r de filtrage proprement dit (cf paragraphe 6.1.3) :

a) Si le filtrage est rapide, le "filtre" attend toujours qu'un bloc soit chargé ou vidé. Le temps de réponse est alors essentiellement le temps de chargement + le temps de vidage ;

b) Si par contre le filtrage est lent, le temps de chargement/vidage disparait derrière le temps de filtrage.

Nous évaluons dans chacun des 2 cas le nombre de tampons nécessaires.

6.4.1. r petit : Filtre rapide

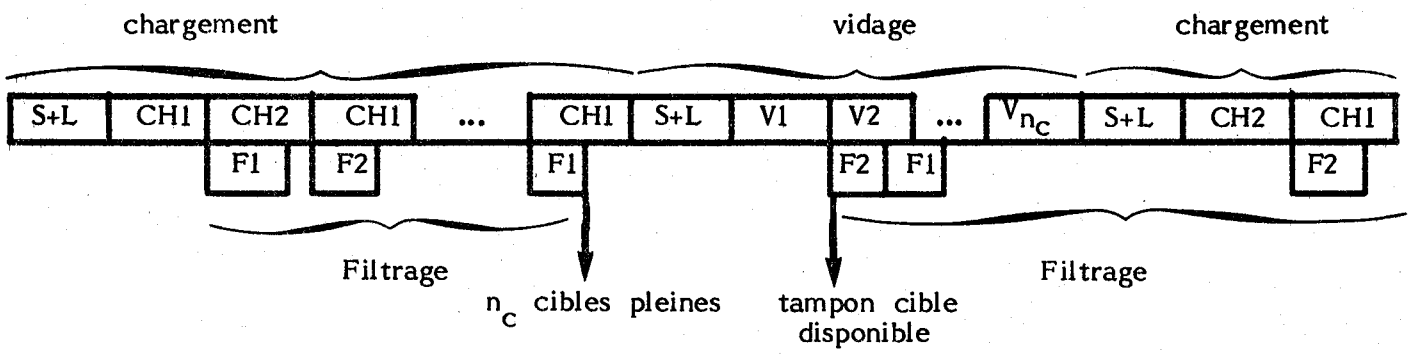
C'est le cas des très gros objets avec le ME 80186/SMD ou le cas général avec le ME 8086/SASI (avec nos estimations du paragraphe 6.1.3 : r=3/5).

n_s = 2 tampons source en bascule sont suffisants. Ceux-ci sont chargés en alternance jusqu'à ce que le filtrage s'interrompe

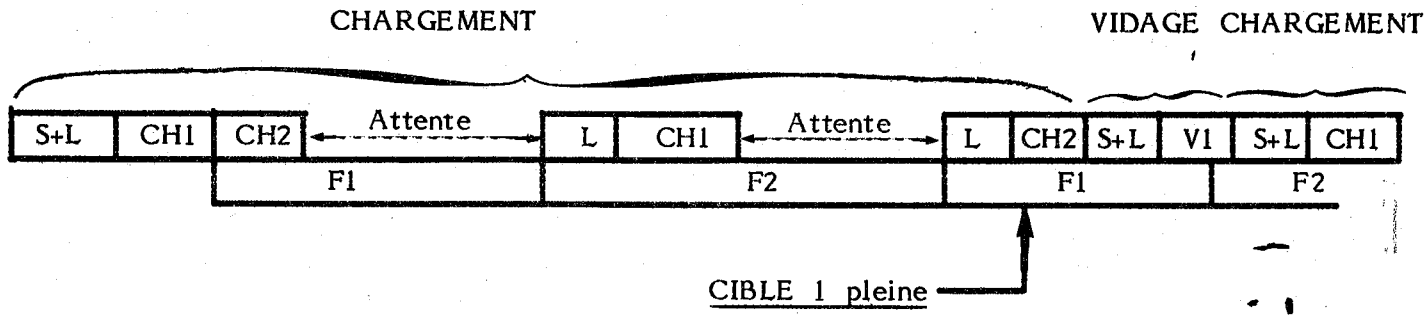
parce que les n_c tampons cible sont pleins. Lorsque le bloc en cours de chargement a été chargé, on vide alors les n_c blocs cible et on recommence alors à charger des blocs source. La Figure 6.11a illustre ce cas où le temps de Filtrage disparaît totalement derrière le temps de transfert.

Le temps perdu en mouvement de bras et latence est proportionnel à $\frac{2pM}{n_c}$.

Avec un grand nombre de tampons cible, on minimisera donc le temps de réponse.



a) r petit



b) r grand

Figure 6.11 Parallélisme entre Filtrage et Transfert

6.4.2. r grand : Filtre lent

C'est le cas du ME SMD. n_s=2 tampons source sont encore suffisants. Ceux-ci sont chargés en alternance jusqu'à ce qu'un tampon cible soit plein, disons le premier. Le filtrage continue et le résultat est écrit dans le deuxième tampon cible. Après le chargement du bloc suivant, on aura le temps de vider le premier tampon cible (Figure 6.11b). Deux tampons cibles suffisent. Excepté le chargement initial d'un bloc et le vidage du dernier bloc, le temps de transfert disparaît entièrement derrière le temps de filtrage.

En conclusion, 2 tampons source sont toujours suffisants. Il existe une valeur r_o de r au dessus de laquelle 2 tampons cible sont suffisants. On peut se demander alors ce qui se passe pour les valeurs de $r < r_o$. Il faut augmenter le nombre n_c de tampons cible. L'étude ci-dessous précise l'influence de n_c sur le temps de réponse en fonction de r et de la sélectivité p.

6.5. Temps de réponse (Filtrage logiciel)

Soit T_{rep} le temps de réponse à une requête de sélection/projection. On étudie les 2 cas : r petit et r grand.

a) r petit

On a : $$T_{rep} = MT_D + pMT_D + \frac{2p}{n_c} M(S+L)$$

Les deux premiers termes comptabilisent les temps de chargement et de vidage (T_D étant le temps de transfort d'un bloc), tandis que le dernier terme mesure le nombre d'aller-retour en temps de mouvement de bras (S) et de latence (L). On a négligé le temps de déplacement du bras et le temps de latence initiaux et on a supposé la relation source stockée sur des cylindres contigus (temps de mouvement de bras et de latence nuls lorsqu'on charge deux blocs successifs de la relation source, cf. figure 6.11a).

Comme on suppose que $S + L = T_D$, on a :

$$T_{rep} = MT_D \left(1 + p + \frac{2p}{n_c}\right) \qquad (5)$$

b) r grand

$$T_{rep} = MT_F + 2T_D$$

Le 1er terme indique le temps de filtrage tandis que le 2e comptabilise les temps de chargement du 1er bloc et de vidage du dernier bloc.

Le temps de filtrage d'un bloc T_F est donné par :

$$T_F = T_D\left[r + \frac{\gamma}{\gamma+1}(1+p)\right] \qquad (6)$$

En effet, au cours du filtrage des M blocs, durant $M(1+p)T_D$ secondes de transfert disque, le filtre travaille et est ralenti d'un facteur γ. Pendant le temps restant $M \times \alpha \times T_D$, le filtre n'est pas ralenti et sa vitesse relative est r. On a donc :

$$T_F = T_D\left[\alpha + 1 + p\right]$$

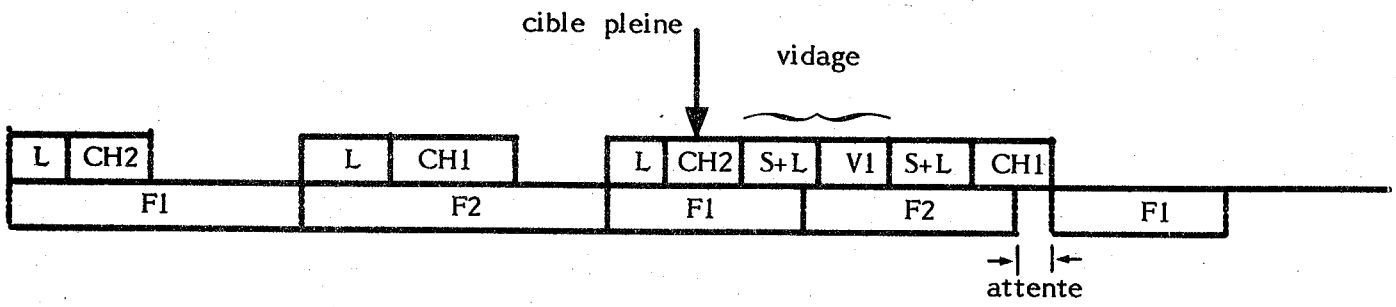
où est donné par $(r-\alpha)(1+\gamma) = 1+p$. D'où l'équation (6).

Il est à noter que :

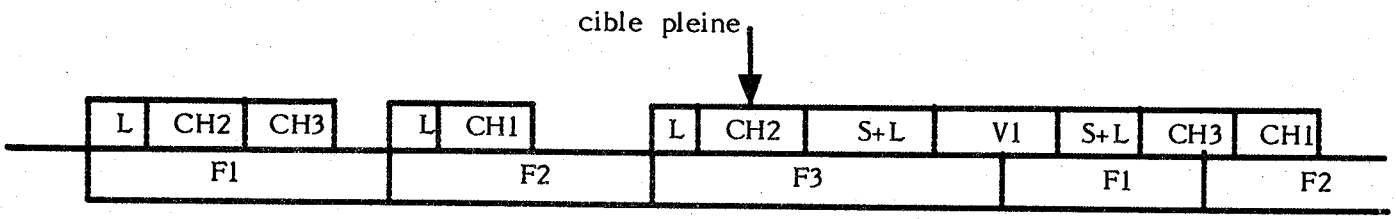
i) l'équation (6) est valable quelle que soit la valeur de $n_c \geq 2$.

ii) Ce n'est qu'une approximation qui suppose que lorsqu'une cible est pleine, on a toujours le temps lors du filtrage du bloc suivant de vider la (ou les) cible(s) et de charger un nouveau bloc (Figure 6.11b).

Un contre exemple est donné en Figure 6.12. Pour éviter ce problème, il suffit de lever la restriction $n_s=2$ et avoir un ou 2 tampons source supplémentaires. Il existe donc des valeurs intermédiaires de r pour lesquelles notre assertion "2 tampons source sont suffisants" est incorrecte. Seules une analyse probabilistique plus exacte ou des mesures indiqueraient le pourcentage du temps (la probabilité) que le nombre de tampons source nécessaires est supérieur à 2. Cette probabilité est d'autant plus faible que r est grand.



a) Attente du Filtre



b) Solution

Figure 6.12 Attente du Filtre même quand r est grand ;
Solution : on augmente le nombre de tampons Source

On a donc dans le cas r grand :

$$T_{rep} = MT_D \left[r + \frac{\gamma}{\gamma+1}(1+p)\right] + 2T_D$$

Nous supposerons par la suite que M >> 2 et nous négligerons le second terme. On a :

$$T_{rep} \simeq MT_D \left[r + [\gamma/(\gamma+1)](1+p)\right] \qquad (7)$$

Avec l'approximation précédente, il existe une valeur r_o pour laquelle le temps de filtrage d'un bloc est exactement égal à la somme des temps de chargement d'un bloc, de vidage de p blocs et de commutation (Figure 6.13).

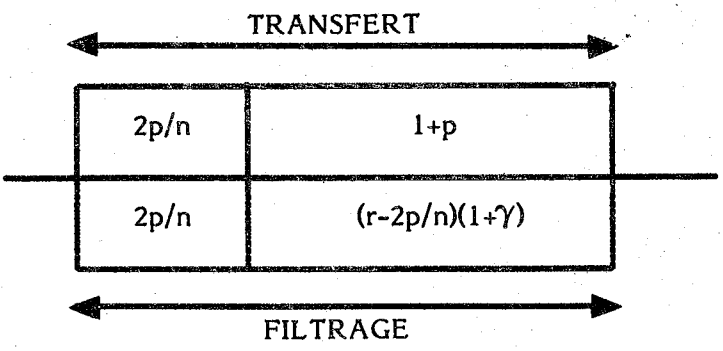


Figure 6.13 Parallélisme total entre Filtrage et Transfert

On a alors (equations (1) et (3)) :

$$r_o + \frac{\gamma}{1+\gamma}(1+p) = 1 + p + \frac{2p}{n}$$

ou :

$$\left(r_o - \frac{2p}{n}\right)(1+\gamma) = 1 + p$$

ou encore :

$$r_o = \frac{2p}{n} + \frac{1+p}{1+\gamma} \qquad (8)$$

Si $r > r_o$, le temps de transfert "disparaît" derrière le temps de filtrage.
Si $r < r_o$, c'est le temps de filtrage qui disparaît derrière le temps de transfert.

La figure 6.14 illustre l'influence du nombre de tampons n_c et de la vitesse relative r sur les temps de réponse.

La courbe représente, en fonction de n_c (equation (8)) la borne r_o pour laquelle le temps de filtrage est égal au temps de transfert.

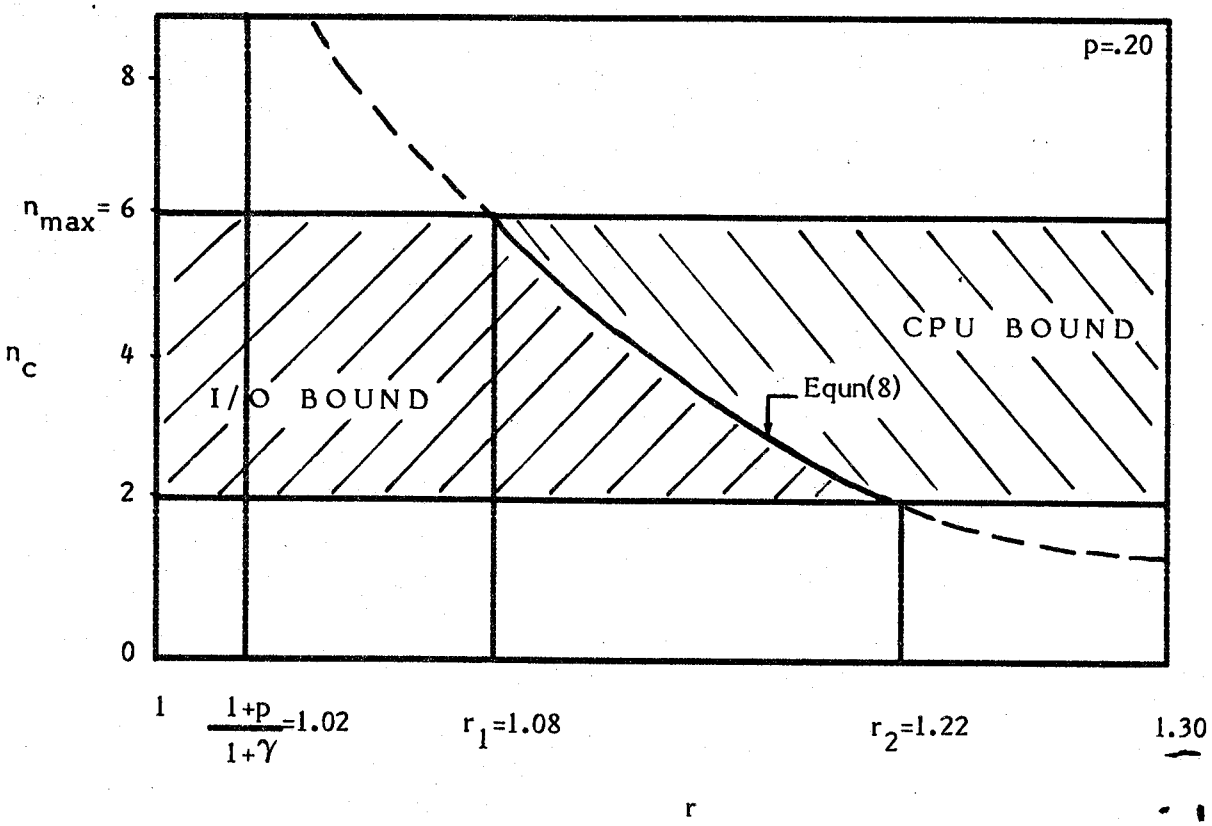


Figure 6.14 Influence de n_c et r sur le temps de réponse

Tout d'abord si r est petit : $r\ (1+\gamma) \leq 1+p$, on est toujours limité par les E/S quel que soit le nombre de tampons cible n_c. Le temps de réponse est alors donné par l'équation (5). Il est bien évidemment minimisé par la valeur maximale possible de n_c : n_{max}, et est indépendant de r.

La borne en dessous de laquelle on est limité par les E/S dépend de n_{max} :

$$r_1 = \frac{2p}{n_{max}} + \frac{1+p}{1+\gamma}$$

Ensuite, si $r > r_2 = p + \frac{1+p}{1+\gamma}$, on est limité par le filtrage, quelle que soit la valeur de n_c, le temps de réponse est donné par l'équation (7) et n_c= 2 tampons sont donc suffisants.

Enfin si $r_1 \leq r \leq r_o$, on est limité

- soit par les E/S : n_c petit, on perd du temps dans les commutations chargement/vidage,

- soit par le filtrage : n_c grand ; il ne sert à rien d'avoir une valeur de $n_c > n_o$ où n_o est donné par l'équation (8).

Le temps de réponse est minimisé par $n_o(r)$. Dans ce cas le temps de filtrage est exactement le temps de transfert.

Ceci est également illustré par la Figure 6.15 où le temps T_F/T_D (temps de réponse normalisé par rapport à $M \times T_D$: équations (5) et (7)) est tracé en fonction de r pour p=.5 (Figure 6.15a) et p=.2 (Figure 6.15b).

Prenons n_{max}=10. Si $r < r_1$ (r_1=1.4 si p=.5, 1.08 si p=.2) l'écart des temps de réponses entre n_c=2 et n_c=6 est non négligeable puisqu'il avoisine 10% pour p=.2 et 22% pour p=.5.

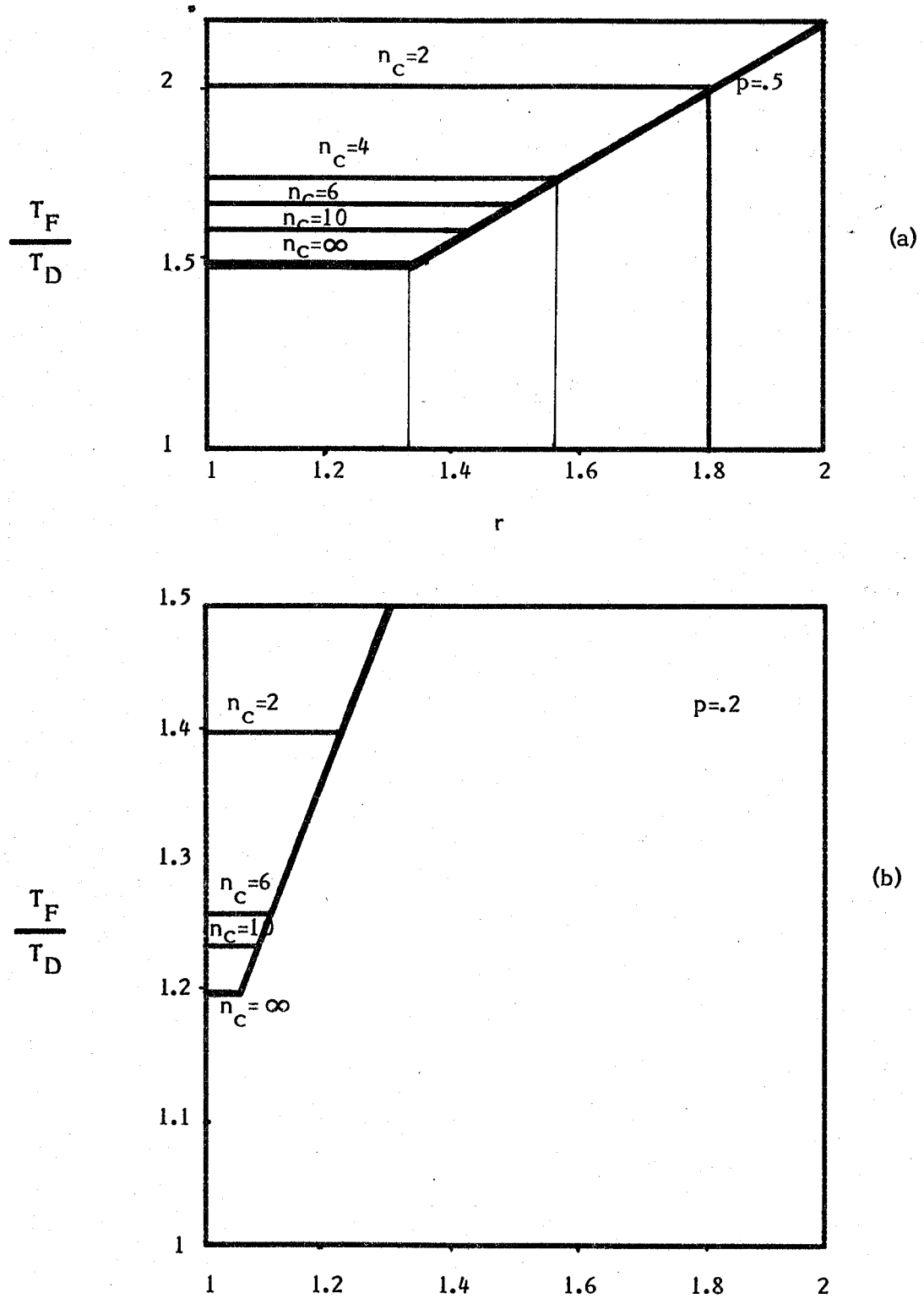


Figure 6.15 Temps de réponse (normalisé à MT_D) versus r
La courbe en traits gras indique en fonction de r l'enveloppe où le temps est minimum

Si $r_1 < r < r_2$ (r_2 = 1.8 si p=.5 et 1.21 si p=.2), le minimum est atteint sur la courbe en trait gras. En particulier si p=.5 (Figure 6.15a) et si r=1.55, le minimum est atteint pour n_c=5.

Observons que dans les 2 cas, cette plage $[r_1,r_2]$ est très étroite.

Observons également que $r_2 \leq 1 + \frac{2}{1+\gamma} \simeq 2.7$, la borne étant atteinte quant p=1.
De même $r_1 \geq \frac{1}{1+\gamma} \simeq .85$, la borne étant atteinte pour p=0.

Enfin, si $r > r_2$, le temps est indépendant de n_c. 2 tampons cible sont suffisants.

6.6. Conclusion sur le nombre de tampons

Nous avons montré que 2 tampons source sont suffisants. Par contre le nombre de tampons cible dépendra de la valeur de r. Concrètement si r est grand, (r>2), i.e. si le filtrage est limité par le processeur (CPU bound), 2 tampons cible sont suffisants.

Si par contre r est petit (r<2 : filtre rapide) on aura intérêt à choisir le nombre maximum possible n_{max} de tampons cible pour minimiser les temps de commutation, bien qu'il existe une plage de valeurs de r dépendant de la sélectivité p, où les n_{max} tampons sont sous-employés.

En particulier avec le ME 186/SMD r a été estimé à 3. On est limité par le processeur, 2 tampons cible sont suffisants; avec le ME 8086/SASI, par contre, on est limité par les entrées/sorties, il faudra utiliser le nombre maximum de tampons cibles autorisé(*). D'après notre estimation numérique ci-dessus, ces résultats sont valables quelle que soit la sélectivité p de la requête.

(*) Remarquons toutefois qu'avec le filtrage actuel en Pascal sur ME SASI r est grand (voir paragraphe 6.1.3) et qu'on est limité par le processeur.

6.7. Comparaison des performances des 3 architectures pour la Sélection/Projection

L'étude de 2 architectures pour filtrage logiciel nous a permis :

1) d'évaluer le ralentissement γ du filtrage logiciel par les Entrées/Sorties, lorsque le processeur partage la mémoire avec le processeur d'Entrées/Sorties,

2) d'évaluer le nombre optimal de tampons en mémoire partagée en vue de minimiser le temps de réponse, ce nombre de tampons dépendant du rapport r des vitesses du processeur et du Disque.

De ces 2 évaluations, on a tiré une estimation des temps de réponse T_{SASI} et T_{SMD} dans le cas des 2 architectures. Ce temps est fonction de la sélectivité p et de la taille M de la relation Source.

L'objectif de ce paragraphe est de comparer les performances de ces deux architectures entre elles ainsi qu'avec celles du filtrage matériel.

Soit T_{DSASI} et T_{DSMD} les temps de transfert d'un bloc entre le Disque et MP respectivement pour le ME SASI et le ME SMD.

$$\text{On a } \frac{T_{DSASI}}{T_{DSMD}} = \frac{\tau_{SASI}}{\tau_{SMD}} = \frac{6,2}{.8} = 7,75$$

où rappelons que τ_{SASI} et τ_{SMD} sont les temps de transfert d'un octet.

En normalisant le temps de réponse par rapport à $M \times T_{DSMD}$ (où $T_{DSMD} = T_D$ est le temps de transfert d'un bloc) et en se rappelant que pour le ME SMD (r=3) on est limité par

le processeur, tandis que pour le ME SASI (r=.6) on est limité par les E/S, on a pour temps de réponse normalisés (Equations (5) et (7)) :

$$T_{SASI} = \frac{\tau_{SASI}}{\tau_{SMD}} (1 + p + \frac{2p}{n_c}) \qquad (9)$$

$$T_{SMD} = r + \frac{\gamma}{\gamma+1} (1+p) \qquad (10)$$

où r=3 et γ=0.17.

La figure 6.16 montre l'influence mineure du nombre de tampons cible n_c sur le temps de réponse pour le ME SASI. n_c est choisi égal à 4 dans la suite des évaluations numériques.

6.7.1. Temps de réponse avec le Filtre Matériel

Il nous reste à évaluer T_{MAT}, le temps de réponse pour le Filtre matériel (cf. Chapitre 3). Celui-ci est évalué avec précision dans [25] pour plusieurs variantes architecturales (filtrage au vol, 2 tampons en bascule, etc...).

On utilise ici l'algorithme choisi dans [25] et implanté avec l'architecture considérée comme la plus performante. Dans un souci de simplification, on en utilise la légère variante suivante :

1) La Relation est chargée du Disque alternativement dans le 1er ou le 2ème tampon Source ;

2) Le 1er tampon est filtré tandis que le 2e tampon est chargé du disque et vice-versa ;

3) Quand les 2 tampons Cible sont pleins, ils sont vidés sur le Cache (après terminaison du chargement du bloc en cours, s'il y a lieu) ;

4) Quand le Cache est plein, il est vidé sur le Disque.

Le temps de filtrage d'un caractère étant $\tau = 400$ ns, c'est-à-dire la moitié du temps τ_{SMD} de transfert d'un caractère, on est limité par les E/S : le temps de filtrage disparaît derrière le temps de transfert. Le temps de transfert d'un caractère du tampon cible dans le Cache est de 400 ns.

En normalisant par rapport à $M \times T_D$, on a alors :

$$T_{MAT} = 1 + p/2 + p/4 + (1 + \frac{2}{c})\, p \times \mathbb{1}_{Mp > c} \qquad (11)$$

où le premier terme correspond au temps de chargement, le deuxième terme exprime le temps de vidage du tampon cible dans le Cache, le troisième terme est le temps de latence après vidage des cibles dans le Cache. Enfin le dernier terme traduit le temps de vidage du Cache vers le disque ainsi que le déplacement du bras et le temps de latence.

Rappelons que la somme des deux derniers temps est $S + L = T_D$.

c dénote la taille du Cache, tandis que $\mathbb{1}_{Mp > c}$ vaut 1 si $Mp > c$, 0 autrement.

On suppose la relation rangée sur des cylindres contigus (temps de déplacement de bras supposé nul) et on néglige le temps de déplacement du bras en début de chargement.

6.7.2. Comparaison entre les trois architectures

Les temps de réponse sont tracés en fonction de la sélectivité p dans la Figure 6.17 pour les 3 architectures : Filtrage logiciel avec le ME 8086/SASI (Equation (9)), avec le ME 80186/SMD (Equation (10)) et Filtrage Matériel (Equation (11)). Au delà de $p_o = c/M$, la relation résultat ne tient plus dans le cache. p_o est choisi égal à .2.

Comme il fallait s'y attendre pour le filtrage matériel et le filtrage avec le module 8086/SASI, architectures limitées par les E/S,

le temps de réponse est sensible à la sélectivité : le temps de vidage est proportionnel à la taille du résultat.

Dans le cas du filtrage avec le module SMD, le temps de transfert disparaît derrière le temps de filtrage. Toutefois le filtrage est légèrement ralenti (dans un rapport $\gamma/1+\gamma = .15$), lorsqu'un transfert a lieu en même temps, ce qui explique que la droite n'est pas horizontale.

En conclusion, si le filtrage logiciel SASI est très lent (jusqu'à 10 fois plus lent que le filtrage matériel si $p \simeq .1$), le filtrage logiciel SMD est considérablement plus rapide. Il est trois fois plus lent que le filtrage matériel lorsque la taille du résultat est petite ($p \leq .1$) ; mais il est presqu'aussi rapide que le filtrage matériel si la taille du résultat est grande ($T_{MAT}/T_{SMD} = .7$ si $p = .8$).

La comparaison précédente entre Filtrage materiel et filtrage logiciel amène les remarques suivantes :

(i) Les performances du module SASI sont extrêmement limitées, étant donné le débit réel du disque avec un tel module : 1,2 Mbits/s seulement.

(ii) même s'il n'atteint pas la vitesse du filtre matériel, le module SMD permet un filtrage performant.

(iii) le temps de réponse du filtre logiciel SMD dépend cependant considérablement de la vitesse relative de filtrage r_{SMD}. Celle-ci a été estimée dans un paragraphe précédent à 3. La figure 6.18 donne une idée plus précise de l'influence de r_{SMD} sur le rapport entre temps de filtrage logiciel et temps de filtrage matériel.

En particulier, si r_{SMD} vaut 30 (ce qui correspond à un temps de cycle moyen de 37μs, valeur réaliste qui se rapproche des

mesures obtenues avec une première version non optimale du filtrage Pascal, cf. Figure 6.2), le filtrage logiciel est 27 fois plus lent que le filtrage matériel pour p=.1 et 17 fois plus lent que p=.4.

Le filtrage matériel est limité par les Entrées/Sorties : le filtre est deux fois plus rapide que le disque (r_{mat}=1/2).

Le temps de filtrage logiciel par contre est limité par le processeur. Il est proportionnel à la vitesse relative r_{SMD} (Equn 6).

Cependant pour une vitesse relative r_{SMD} = 30, alors que le temps moyen de traitement d'un caractère est 60 fois plus grand qu'avec le filtre matériel, le temps de réponse du filtrage logiciel est à peine 20fois plus grand que celui du filtrage matériel.

Ceci confirme les études dans [25] qui montrent que la puissance du filtre matériel est mal utilisée (il est trop souvent oisif) et qu'un filtre 2 à 3 fois plus lent serait amplement suffisant pour les vitesses de disque actuelles. Si l'on définit le taux d'utilisation du filtre U_F comme le rapport entre le temps passé en filtrage et le temps de réponse total, U_F est sensiblement égal à 1 pour le filtrage logiciel SMD alors qu'il décroît de 0,5 (p=0) à 0,18 (p=1) pour le filtrage matériel (r = 1/2). Pour une étude détaillée du filtrage matériel, le lecteur est reporté à [25].

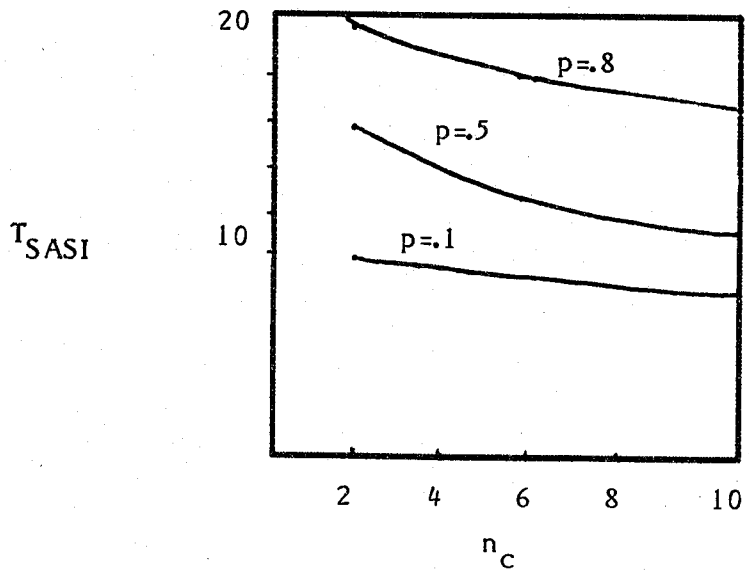


Figure 6.16 Influence du nombre de tampons cible sur le temps de réponse : ME SASI

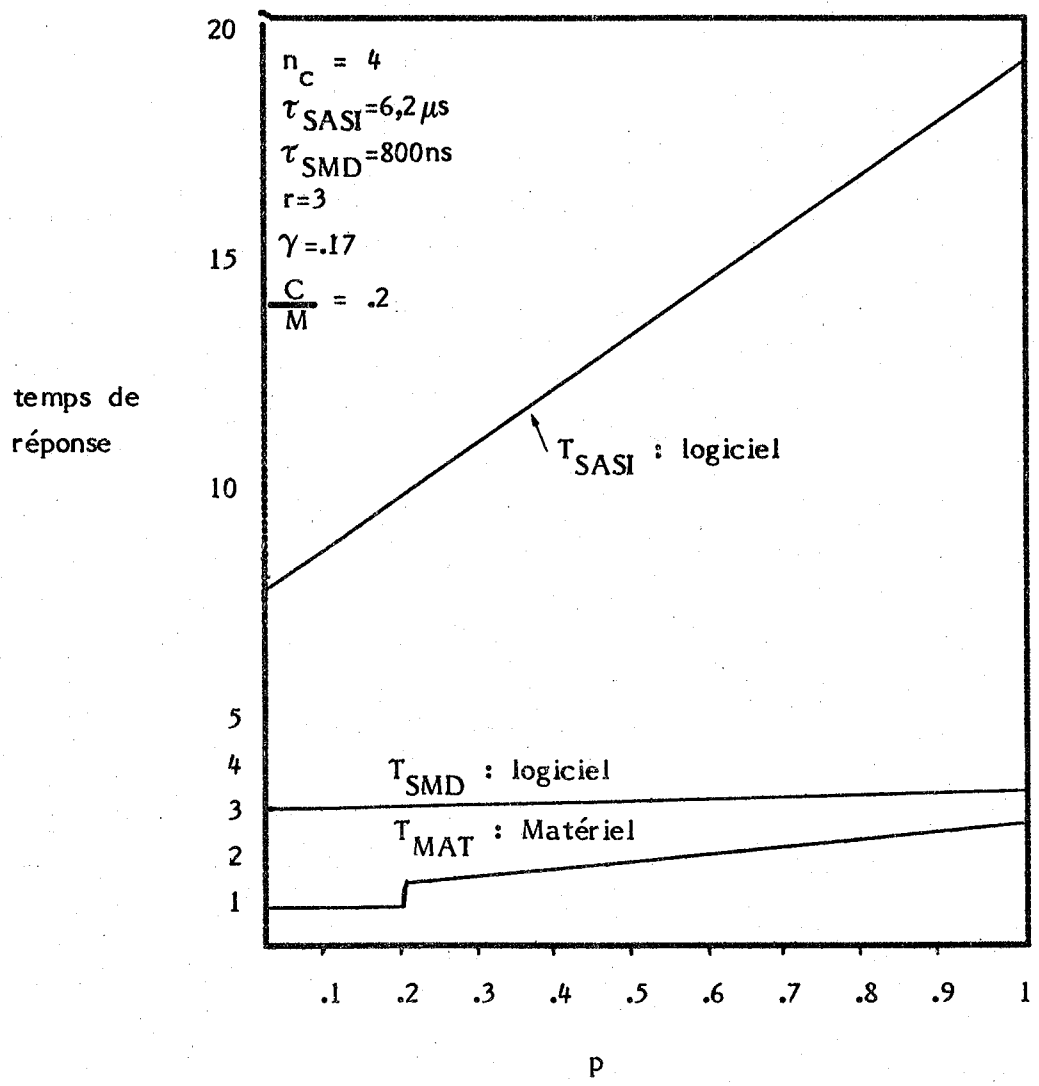


Figure 6.17 Temps de réponse normalisé à M × T_D en fonction de la sélectivité pour les trois architectures

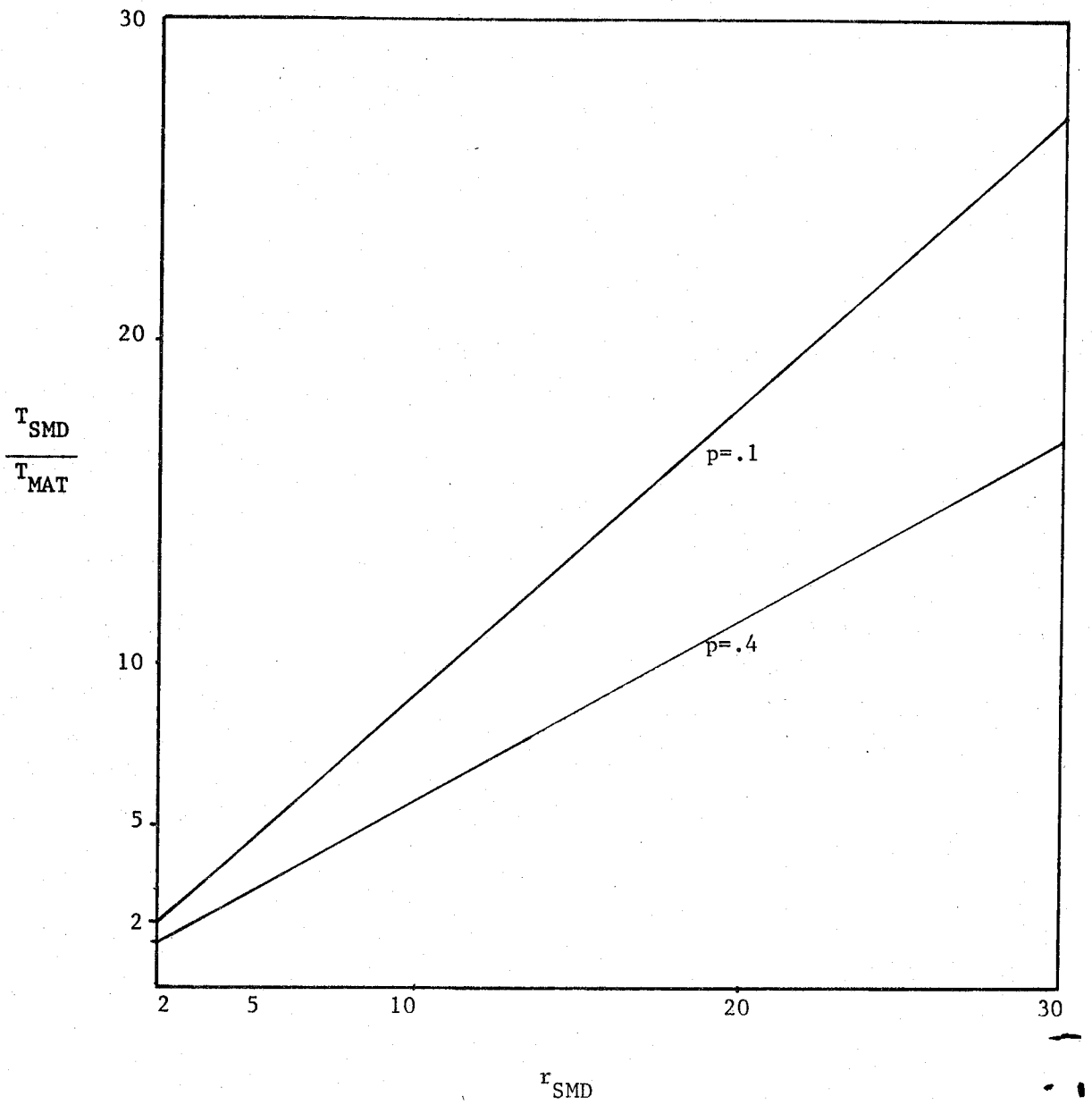


Figure 6.18 Influence de la vitesse relative du filtrage logiciel sur le rapport des temps de réponse entre le filtrage logiciel et le filtrage matériel

6.8. Conclusion

L'étude des deux architectures (SASI) et SMD) pour filtrage logiciel nous a permis de mettre en évidence les résultats suivants :

1) Le Module d'échange actuel SASI est extrêmement lent. Ceci est dû au débit réel du Disque extrêmement faible.

2) Le Module d'échange futur SMD est par contre bien adapté au filtrage logiciel. Nous avons montré

a) que le processus 80186 qui effectue le filtrage est peu ralenti lorsqu'un transfert simultané a lieu avec le Disque ($\gamma = .17$)

b) que le filtrage est limité par le processeur (CPU bound) étant donné que le débit réel du Disque est celui de l'interface SMD qui est élevé (10 Mbits/s)

c) que 2 tampons source et 2 tampons cible étaient suffisants, chaque tampon ayant la taille d'une piste disque (soit 64 Koctets),

d) qu'une borne inférieure de la vitesse relative du filtrage par rapport à celle du Disque devrait être $r_{SMD} = 3$.

3) Bien qu'on doit s'attendre à ce que le filtrage logiciel soit relativement lent, (de 3 à 20 fois plus lent que le filtrage matériel, cf. Figure 6.18, suivant l'estimation de r_{SMD}, i.e. de l'efficacité du code de filtrage), il faut noter que :

a) on n'a pour l'instant qu'une estimation optimiste - elle aussi - des temps de filtrage matériel ;

b) si on gagne un ordre de grandeur avec un filtre matériel, cette puissance est mal utilisée. En effet les évaluations de ce chapitre considèrent le cas le pire en supposant toujours qu'on filtre de longues relations du Disque vers le Disque. Quelle est la proportion des requêtes nécessitant un long filtrage ? En fait l'accès à un Index réduit considérablement le nombre de blocs à filtrer. Qu'advient-il des temps de réponse lorsque la Source ou la Cible n'est pas le disque mais l'utilisateur ? Il est probable alors qu'on est limité par ce transfert utilisateur ⟷ filtre.

En résumé, le filtrage logiciel est une solution relativement performante. On peut s'étonner que la solution matérielle présentée dans le Chapitre 3 ne permette pas de gagner 2 ordres de grandeur. Ceci est essentiellement dû à 2 raisons :

a) Souplesse d'une architecture logicielle

La transcription exacte de l'algorithme de filtrage matériel en logiciel aurait donné des temps de filtrage désastreux. Bien au contraire, l'introduction de longueurs de champs (et donc du comptage et du saut d'attribut) entre autres modifications, a permis d'optimiser le filtrage. On peut de la même façon, optimiser la jointure en implantant plusieurs algorithmes.

b) Mauvaise utilisation de l'architecture matérielle

Tout d'abord, l'utilisation du filtre matériel est très mauvaise : elle est inférieure à 1/2. Il est montré dans [25] qu'un filtre ayant la même vitesse que le Disque (ou même 2 fois plus lent) serait suffisant. Ensuite, si le Cache permet d'améliorer légèrement les temps de réponse [25], le parallélisme entre d'une

part les transferts Disque ⟷ Cache et d'autre part le filtrage ou les accès aux tampons Source et Cible - parallélisme que permet l'architecture - n'est pas utilisé.

Par contre, une mémoire cible plus grande aurait amélioré les performances. En effet r est petit donc il faut un nombre de tampons cibles $n_c > 2$ (cf. paragraphe 6.5 sur l'estimation du nombre de tampons), même si le temps d'accès à une mémoire cible plus grande aurait été plus grand et de ce fait la vitesse du filtre aurait été diminuée.

Enfin on a montré dans [25] que le mécanisme de mémoire à "double accès" n'est pas nécessaire : l'utilisation de tampons en bascule donne des performances comparables sinon meilleures. Le même mécanisme dans l'architecture logicielle ne ralentit le processeur 80186 que d'un facteur $\gamma = 0,17$.

Pour terminer, notons que l'architecture matérielle est mal adaptée aux opérations binaires [24] Une étude des performances de l'architecture logicielle pour les opérateurs binaires reste à faire qui permettrait d'infirmer ou de confirmer qu'une architecture logicielle de type 80186/SMD est bien adaptée au filtrage des données pour toutes les opérations de l'algèbre relationnel.

CHAPITRE 7 –

CONCLUSION

7. - CONCLUSION

Ce travail avait pour but de contribuer à l'amélioration des systèmes de Gestion de Bases de Données. Etant donnée leur complexité, toute amélioration passe par une remise en cause de l'architecture mais s'appuie sur la technologie courante.

Par remise en cause, il faut entendre qu'il faut repenser l'architecture en tenant compte de la spécificité de l'application SGBD. C'est dans cet esprit que l'approche Machine Base de Données (MBD) a donné lieu à de nombreuses recherches depuis 10 ans aboutissant souvent à des prototypes dont la caractéristique essentielle est l'implantation des fonctions d'accès aux données par un mécanisme matériel spécialement conçu à cet effet, à une époque où les microprocesseurs standards avaient des performances insuffisantes.

Parallèlement, la technologie a considérablement évolué pendant 10 ans. Les architectures actuelles de microordinateurs du commerce permettent avec une relative souplesse de s'adapter aux applications, d'introduire un parallélisme minimal avec des performances augmentant avec le temps.

C'est dans cette perspective que se situe l'essentiel du travail présenté ici.

S'appuyant sur l'expérience acquise par la réalisation successive de deux prototypes de SGBD appelés VERSO le premier pouvant s'intituler MBD, le second SGBD portable, l'étude développée ici avait pour soucis :

(1) de justifier une nouvelle méthode de traitement des données (consultation et mise à jour) dans les SGBD relationnels. Cette méthode appelée Filtrage par automate fini est un balayage séquentiel des fichiers bien adapté à la structure des fichiers et au type de requêtes sur ces fichiers (opérations de l'Algèbre relationnelle),

(2) de comparer deux architectures utilisant le filtrage à automate fini.

Partant d'une hypothèse courante que les performances sont limitées par les entrées/sorties et que la fonction à optimiser est l'accès aux données, de nombreux concepteurs de MBD ont eu pour souci de décentraliser des fonctionnalités près de la mémoire de masse, séparant ainsi les couches supérieures du SGBD implantées sur un processeur central d'une part, et la couche physique implantée près du disque d'autre part.

Dans cette optique, les prototypes VERSO ont l'architecture suivante :

Ayant choisi une configuration SM 90, on a implanté sur un module de traitement (Micrcoprocesseur Motorola à base de 68000) les couches de haut niveau (décomposition, chemins d'accès, gestion des transactions et contrôle de concurrence).

Ce module de traitement partage le bus SM avec un module d'échange chargé des accès au disque et de la couche physique, c'est-à-dire du filtrage par automate fini des relations physiques.

Si le premier volet de cette thèse a consisté à mettre en valeur le filtrage par automate fini, le deuxième avait pour but de permettre le choix entre l'approche MBD et l'approche SGBD portable.

Alors que l'approche MBD dans VERSO, consiste à implanter le filtrage par un ME matériel (7 cartes) spécialement conçu à l'INRIA à cet effet, l'approche SGBD portable a consisté à implanter l'algorithme de filtrage sur un Module d'échange de la SM 90 ayant un processeur 8086 et 128 K de mémoire.

La conclusion essentielle de cette étude est qu'un matériel spécialement conçu pour le filtrage a une puissance et une rapidité très mal utilisées dans l'application "Bases de Données", et qui serait

plus performante dans des applications nécessitant systématiquement le balayage de longs fichiers.

L'utilisation de microprocesseurs du commerce, par contre, permet l'implantation des opérations relationnelles par filtrage de façon efficace : dans le pire des cas, c'est-à-dire si le fichier à filtrer est très long, on devrait perdre moins d'un ordre de grandeur sur les temps de réponse à un balayage de longs fichiers en utilisant la solution logicielle pour le filtrage.

Des mesures sont en cours qui permettront de comparer plus précisément les solutions matérielle et logicielle et de valider les évaluations présentées dans cette thèse.

D'autre part, il faut remarquer que l'approche matérielle est coûteuse et risquée : développer et réaliser une maquette prend un temps non négligeable au bout duquel arrivent sur le marché de nouveaux processeurs aux performances considérablement meilleures.

Par ailleurs, il ne faut pas oublier l'optimisation des couches de haut niveau d'un SGBD relationnel : dans une Machine à Filtre telle que VERSO, il ne sert à rien de concevoir un traitement efficace de bas niveau (filtrage) si le goulet d'étranglement se situe au niveau des couches supérieures.

La démarche de conception rapportée dans cette étude est caractéristique d'une tendance actuelle dans le monde de la recherche : on a tout d'abord conçu des projets ambitieux, mémoires de masses sophistiquées filtrées par plusieurs filtres, architectures multiprocesseurs complexes, (aucune n'a été commercialisée), puis la tendance a été vers la simplification tant pour les mémoires de masse (utilisation d'un filtre connecté au bon vieux disque à tête mobile) que pour les processeurs (commercialisation de "machines" Bases de Données utilisant peu de parallélisme et peu ou pas du tout de processeurs spécialisés).

Si la recherche en matière d'architectures spécialisées à la gestion des données semble marquer le pas au profit de la diffusion de plus en plus importante de SGBD portables sur des microordinateurs de plus en plus performants, ce n'est que pour un temps.

Il faut s'attendre à un regain d'activité dans ce domaine : Un souci encore plus crucial de performances motive des organismes comme le japonais ICOT et l'américain MCC dans leurs recherches d'architectures spécialisées pour les systèmes dits de la 5e génération (Bases de connaissances, Bases déductives, etc...).

S'il est difficile de prédire quelle sera l'architecture des SGBD des années 1990, d'une part à cause de l'évolution technologique (la diminution exponentielle du coût de la mémoire à accès direct avec le temps va certainement révolutionner les architectures futures), d'autre part à cause de l'évolution des nouvelles applications dont les besoins en gestion de données sont encore mal connus et mal spécifiés, on peut cependant essayer de parier sur les tendances à 5 ans.

Les Machines à Filtre telles que VERSO, de conception relativement simple traitent les requêtes séquentiellement et utilisent pour traiter une requête deux processeurs, l'un chargé des couches de haut niveau, l'autre décentralisé près d'une unité à disques pour effectuer le filtrage.

Pour augmenter les performances, il faudra introduire un certain parallélisme. On peut (i) filtrer des données en parallèle et donc utiliser plusieurs unités à disques chacune étant connectée à un filtre, (ii) traiter plusieurs requêtes simultanément en multipliant les processeurs chargés des couches de haut niveau. Une société américaine TERADATA commercialise un système basé sur une telle architecture à base de processeurs du commerce. Dans ce type d'architecture prometteuse à court terme, on peut s'adapter à la charge, en connectant à la demande un processeur supplémentaire ou

une nouvelle unité à disques et son filtre. La difficulté principale qui réside dans la conception du réseau de communication entre tous ces processeurs est moins matérielle que logicielle par manque d'outils (logiciels de bas niveau distribués).

REFERENCES BIBLIOGRAPHIQUES

[1] Abiteboul S. et Bidoit N. : "Non First Normal Form Relations to represent hierarchically organized data", PODS 84, Atlanta, Mars 1984, pp. 191-200.

[2] Armisen J.P.., Caleca J.Y. : "A commercial Back-End Data Base System", Proc. Inter. Conf. on VLDB, Cannes, 1981.

[3] Auer H., Hell W., Leilich H.O., Schweppe H., Stiege G., Seehusen S., Lie J.S., Zeidlet H.Ch., Teich W. : "RDBM - A Relational Data Base Machine", Information Systems, Vol. 6, n° 2, 1981.

[4] Babb E. : "Implementing a Relational Database by Means of Specialized Hardware", ACM Trans. on Database Syst., Vol. 4, n° 1, March 1979.

[5] Bancilhon F., et Scholl M. : "Design of a Backend Processor for a DataBase Machine", Proc. ACM-SIGMOD, Santa Monica, May 14-16, 1980, pp. 93-93g.

[6] Bancilhon F., Richard P., Scholl M. : "The Database Machine VERSO : Binary Operators", Proc. 6th Workshop on Computer Architecture for Non Numeric Processing, Hyères, June 1981.

[7] Bancilhon F. et al. : "Les V-relations : Définition, Modification, Interrogation", Note Technique VERSO n°1, Avril 1982.

[8] Bancilhon F., Richard P., Scholl M; :"On Line Processing of Compacted Relations", Proc. Inter. Conf. on VLDB, Mexico, 1982.

[9] Bancilhon F. : "Machines Bases de Données : Une introduction", Journées Informatiques de Nice, Septembre 1982.

[10] Bancilhon F. et al : "VERSO : A Relational Back End Data Base Machine", Proc of the 7th International Workshop on Database Machines, Sans Diego, Sept. 1982 ; also in Advanced Database Machine Architecture, D.K. Hsiao editor, Prentice-Hall 1983, pp. 1-18.

[11] Banerjee J., Hsiao D.K., Kannan K. : "DBC - A Database Computer for very Large Databases", IEEE Trans. on Computers, Vol. 28 n° 6, 1979.

[12] Berra P.B., Oliver E. : "The Role of Associative Array Processors in Database Machine Architecture", Computers, Vol. 12, n° 3, March 1979, pp. 53-61.

[13] Bidoit N. : "Un Modèle de Données Relationnel Non Normalisé : Algèbre et Interprétation", Thèse 3e cycle, Université Paris-Sud, Juin 1984.

[14] Boral H., Dewitt D.J. : "Database Machines : An Idea whose Time has passed. A critique of the future of Database Machines", in Database Machines, H.O. Leilich and M. Missikoff editors, Springer-Verlag, 1983, pp. 166-187.

[15] Cabart C. : "Conception et réalisation d'un Filtre Logiciel Unaire", Rapport de DEA, Université de Paris-Sud, Orsay, Sept. 84.

[16] CAFS 800, (A short overview of the CAFS system), ICL World Headquarters, Putney, London, England SW15 1SW.

[17] Copeland G.P., Lipovski G.J., Su S.Y. : "The Architecture of CASSM : A Cellular system for Non-numeric Processing", Proc. 1st Annual Symposium on Computer Architecture, Dec. 1973, pp. 121-128.

[18] Delobel C. : "Normalization and Hierarchical Dependencies in the Relational Data Model", ACM Trans. Database Systems, Vol. 3, n° 3, Sept. 1978, pp. 201-202.

[19] Delobel C., Adiba M. : "Bases de Données et Systèmes Relationnels", Dunod Informatique, Bordas, Paris, 1982.

[20] Dewitt D.J. : "DIRECT - A Multiprocessor organization for Supporting Relational Database Management Systems", IEEE Trans. on Computers Vol. C.28 n°6, June 1979, pp. 395-405.

[21] Dewitt D.J., Hawthorn P. : "A Performance Evaluation of Database Machine Architectures", Proc. Inter. Conf. on VLDB, Cannes, 1981.

[22] El Masri A., Rohmer J. et Tusera D. : "A machine for information retrieval", Proc. 4th Workshop on Comp. Arch. for Non-numerical Processing, Syracuse, New York, Aug. 1978, pp. 117-120.

[23] Fischer P.C., Thomas S.J. : "Operations for Non First Normal Form Relations", Proc. IEEE Compsac, 1983.

[24] Gamerman S., Salza S., Scholl M. : "Analysis of filtering Efficiency in the Database Machine VERSO", in Database Machines, H.O. Leilich and M. Missikoff Editors, Springer-Verlag, 1983, pp. 81-105.

[25] Gamerman S. : "Où l'on découvre que les performances des Filtres dans les Machines Bases de Données ne sont pas celles qu'on croyait", thèse de 3e cycle, Université de Paris-Sud, Juin 1984.

[26] Gardarin G. et al. : "SABRE : A Relational Database System for a Multimicroprocessor System", Proc. of the 7th International Workshop on Database Machines, San Diego, Sept. 1982, also in "Advanced Database Machine Architecture", D.K. Hiaso Editor, Prentice-Hall 1983, pp. 19-35.

[27] Gonzales-Rubio R., Rohmer J., Terral D. : "The SCHUSS filter : A processor for non-numerical Data Processing", International Symposium on Computer Architecture, Ann Arbor, June 4-7, 1984.

[28] Guebo A. : "Conception et réalisation d'un module d'échange universel pour la SM 90. Application aux échanges disque hautes performances et à la téléphonie", Thèse de 3e cycle, Université de Paris VI, 19 Décembre 1984.

[29] Hull R., Yap C.K. : "The Format Model : A Theory of Database Orgnaization", Proc. of Symp. on Princ. of Database Systems, 1982.

[30] Kakuta T. et al. : "The Design and Implementation of Relational Database Machine Delta", Proc. of the 4th International Workshop on Database Machines, Grand Bahamas Island, March 6-8, 1985, pp. 13-34.

[31] Lacroix M., Pirotte A. : "Generalized Joins", ACM SIGMOD Record, Vol. 8, N° 3, Sept 1976, pp. 14-15.

[32] Lecalvé A., Kalfon P. : "S.A.R.I. : Système Associatif de Recherche d'Informations", Actes des Journées Machines Bases de Données, Sophia-Antipolis, Sept. 10-12, 1980, pp. 85-104.

[33] Leilich H.O., Stiege G., Zeidler H.Ch. : "A Search Processor for Database Management Systems", Proc. Inter. Conf. on VLDB, 1978, pp. 280-287.

[34] Lin C.S., Smith C.P., Smith J.M. : "The design of a Rotating Associative Memory for Relational Database Applications", ACM Trans. on Database System, Vol. 1 n° 1, March 1976, pp. 53-65.

[35] Maryanski F. : "Backend Database Systems", Computing Surveys, Vol. 12, n°1, 1980.

[36] Missikoff M., Scholl M. : "Relational Queries in a Domain Based DBMS", Proc. ACM SIGMOD, 1983.

[37] Missikoff M., and Terranova M. : "The Architecture of a Relational Database Computer known as DBMAC", Proc. of the 7th International Workshop on Database Machines, San Diego, Sept. 1982, also in "Advanced Database Machine Architecture", D.K. Hsiao Editor, Prentice-Hall 1983, pp. 87-108.

[38] Mostardi T. : "A Compact index for a relational data base machine : design and implementation" submitted to ACM-ICS, Florence, Italy, 1985.

[39] Naffah N. : "Editing Multitype Document", Office Information Systems, N. Naffah (ed) INRIA/North-Holland Publishing Company, 1982, pp. 125-148.

[40] Navaux P.O.A. : "Processeur de Base de Données MAGE Aspect Matériel", Thèse de Docteur Ingénieur, INPG, Grenoble, 27 Novembre 1979.

[41] Ozkarahan E.A., Schuster S.A., Smith K.C. : "RAP - An Associative Processor for Data Base Management", AFIPS Conf. Proc., Vol. 44, 1975, pp. 370-387.

[42] Plateau D. : "Une structure compacte pour indexer un fichier totalement ordonné : évaluation et mise en oeuvre", Thèse de 3e cycle, Université de Paris-Sud, 20 Sept. 1983.

[43] Roberts D.C. : "A Speicalized Computer Architecture for Text Retrieval", Proc. 4th Workshop on Comp. Arch. for Non-Numerical Processing, Syracuse, New York, Aug. 1978, pp. 51-59.

[44] Rohmer J. : "Machines et Langages pour traiter les ensembles de données (Textes, Tableaux, Fichiers)", Thèse d'Etat, INPG, Grenoble, Décembre 1980.

[45] Scheck H.J., Scholl M.H. : "An Algebra for the Relational Model with Relation-valued attributs", Technical Report DVSI-1984-TI, Technische Hochschedule Darmstadt, Fachbereich Informatik, 1984.

[46] Schweppe H., Stiege G. : "Database Machines : A state of the Art Report", Université de Braunschweig, Informatik-Berichte, 1982.

[47] Schweppe H. et al. : "RDBM - A dedicated Multiprocessor System for Database Management", Proc. of the 7th International Workshop on Database Machines, San Diego, Sept. 1982, also in "Advanced Database Machine Architecture", D.K. Hsiao Editor, Prentice-Hall 1983, pp. 36-86.

[48] Shemer J.E., Necks P.M. : "The Genesis of a Database Computer", Computer, Nov. 1984, pp. 42-56.

[49] Slotnick D.L. : "Logic per track devices", in "Advances in Computers", Vol. 10, Academic Press, New York, 1970, pp. 291-296.

[50] Su S.Y. et al. : "MICRONET : A Microcomputer Network System for Managing Distributed Relational Databases", Proc. Inter. Conf. on VLDB, Berlin, 1978.

[51] Tanaka Y. : "A Data-stream Database Machine with Large Capacity", Proc. of the 7th International Workshop on Database Machines, San Diego, Sept. 1982, also in "Advanced Database Machine Architecture", D.K. Hsiao Editor, Prentice-Hall, 1983, pp. 168-202.

[52] Trehel M. : "Contribution à la théorie des catalogues", Thèse de 3e cycle, Université de Grenoble, 26 Novembre 1965.

[53] Ullman J.D. : "Principles of database Systems", Computer Science Press, 2nd Edition, Maryland, 1983.

[54] Verroust A. : "Characterization of Well-Behaved Database Schematas and their Update Semantics", Proc. Inter. Conf. on VLDB, Florence, 1983.

ANNEXE

Evaluation du temps nécessaire pour sauter un attribut N_S (Filtrage Logiciel)

On calculera le nombre de périodes élémentaires d'un processeur 80186 nécessaires pour effectuer le saut d'attribut. Pour ce faire on écrit une procédure en langage assembleur 8086/80186 et on évalue le temps d'exécution de chaque instruction, en tenant compte des temps d'attente mémoire (WAIT STATE), sachant que les données et le code objet sont stockés dans la mémoire partagée du Module d'échange (voir paragraphe 6.2.2 pour l'architecture de ce Module d'échange).

L'algorithme de saut consiste à

1) lire le mot de la matrice de transition (automate) pointé par les mémoires CAR (caractère courant) et ETAT (état courant).

2) à tester si le 16e bit est à 0 ; dans ce cas il y a saut d'attribut.

3) à mettre à jour ETAT avec la valeur lue dans le 1er octet du mot de la matrice de transition (état suivant).

4) à incrémenter d'un octet le pointeur (RAB) sur la source (TAMPON). On pointait au départ sur l'identificateur d'attribut. On pointe maintenant sur la longueur de l'attribut (2 octets).

5) à incrémenter le mot RAB de cette longueur +2. RAB pointe maintenant sur l'identificateur de l'attribut suivant

6) à mettre à jour le registre CAR avec l'identificateur d'attribut lu dans la source.

La procédure ci-dessous déroule cet algorithme. Son exécution dure

N_S = 119 périodes élémentaires

Instruction		Nombre de périodes
	PUSH AX	5
	PUSH DS	5
	MOV AX, ETAT	6
	MOV DS, AX	2
	MOV AX, WORD PTR [DS] [CAR]	10
	TEST AX, 10H	4
	JNZ FIN	6
	MOV ETAT, AL	6
	MOV AX, TAMPON	6
	INC AX	2
	MOV DS, AX	2
	MOV AX, WORD PTR [DS] [RAB]	10
	INC AX	2
	INC AX	2
	ADD AX, RAB	9
	MOV RAB, AX	6
	MOV AX, WORD PTR [DS] [RAB]	10
	MOV CAR, AX	6
FIN	POP DS	5
	POP AX	5
	RETURN	10
	TOTAL	119

LEXIQUE

Motorola 68000 : microprocesseur VLSI 16 bits. Le bus données (16 bits) est séparé du bus adresses (24 bits).

Intel 8086, 80186 : microprocesseurs VLSI 16 bits. Les données et les adresses sont multiplexées sur un seul bus (24 bits).

Microprocesseurs en tranches : microprocesseurs pouvant être combinés pour s'adapter à des instructions de tailles (tranches) variables en nombre de bits.

ALU : unité arithmétique et logique. Dans un microprocesseur, c'est l'unité chargée du traitement (exécution d'une instruction).

Séquenceur : c'est la partie d'un microprocesseur qui enchaîne les opérations.

RAM : "Random Access Memory". Mémoire à accès direct en lecture et écriture, volatile.

RAM dynamique : RAM généralement en technologie CMOS (consommation faible et encombrement réduit) qu'il faut rafraichir de temps en temp (toutes les 14 s environ) : la lecture d'un octet recharge la capacité CMOS et donc conserve l'information.

EPROM : "Erasable programmable read only memory". Mémoire à accès direct accessible en lecture seulement et non volatile ; mais on peut l'effacer aux rayons ultra violets et la réutiliser.

Arbitre : mécanisme matériel permettant de partager une ressource (bus, mémoire) entre plusieurs processeurs.

DMA : "Direct Memory Access". C'est un processeur spécialisé qui transfère sans interruption un nombre entier d'octets entre une mémoire RAM et une mémoire périphérique (Disque par exemple). On lui donne une adresse mémoire et le nombre d'octets à transférer.

Latch : tampon

Formattage d'un disque : Structuration physique du disque en secteurs.

SCSI(SASI), SMD : Normes de communication (Interface) avec un Disque.

RÉSUMÉ:

Le premier objectif de cette thèse est de mettre en valeur la notion de filtrage des données par automate fini, décentralisé près de la mémoire de masse comme moyen efficace de traiter les opérations élémentaires de l'algèbre relationnelle.

Le deuxième objectif de ce travail est d'aider le concepteur à choisir la meilleure architecture pour un tel filtre décentralisé près du disque. Pour ce faire, on tire les enseignements de la conception et la réalisation de deux versions d'un tel filtre pour le système relationnel VERSO développé à l'INRIA.

La première version utilise une architecture matérielle "ad hoc", tandis que le filtrage est implanté en logiciel sur un processeur standard dans la deuxième version.

MOTS CLÉS: SGBD Relationnel – Relations non normalisées – V-Relations – Machines Bases de Données – Disques Intelligents – Filtrage des Données – Filtre – Filtrage par Automate Fini.

ISBN 2 - 7261 - 0423 - 1

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 5 de l'arrêté du 16 avril 1974

VU les rapports de présentation de Messieurs

. C. DELOBEL, Professeur
. A. PIROTTE, Ingénieur de recherche
. ROHMER, Ingénieur de recherche

Monsieur Michel SCHOLL

est autorisé à présenter une thèse en soutenance en vue de l'obtention du grade de DOCTEUR D'ETAT ES SCIENCES.

Fait à Grenoble, le 28 mai 1985

Le Président de l'U.S.M.G

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique