RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ IBN-KHALDOUN DE TIARET

FACULTÉ DES SCIENCES APPLIQUEES

DÉPARTEMENT DE GENIE ELECTRIQUE

MEMOIRE DE FIN D'ETUDES

Pour l'obtention du diplôme de Master
Domaine : Sciences et Technologie
Filière : Génie Electrique
Spécialité : Informatique industrielle

THÈME

Placement mémoire et optimisation du contrôle

.

Préparé par : M.Hebara Fatima.

Devant le Jury	Grade	Qualité
Mr K.Hebri	**MAA**	**Président**
Mr D.Nasri	**MCB**	**Examinateur**
Mr A.Abdiche	**MCA**	**Encadreur**

Promotion 2016

REMERCIMENTS

Nous remercions en premier lieu le Dieu le tout puissant de nous avoir accordé la puissance et la volonté pour terminer ce travail.

Nous voulons remercier également le professeur Mr ABDICHE AHMED pour nous avoir fait l'honneur d'accepter d'être encadreur et qui nous a aidés avec ses spécieux conseils.

Nous remercions Mr K.HEBRI et Mr A.BENATIA qu'ils nous ont très honorés en acceptent de juger notre travail.

Notre profonde sympathie va à tous nos enseignants du département pour avoir participer à notre formation.

Nous voudrions aussi remercier toutes les personnes qui nous ont aidés de prés ou de lois à réaliser ce mémoire.

Dédicace

Je remercie ALLAH de m'avoir donné de la santé et du courage, à fin d'arriver jusqu'au là.

Je dédie ce modeste mémoire :

A celui qui a épuisé sa vie et sa jeunesse, et qui n'a vécu que pour me voir un jour réussir, à celui qui n'a jamais su dire non pour subvenir à mes besoins, à mon très

Cher père Abde

A celle qui a bercé mes nuit, à celle qui n'a jamais su dire non pour subvenir à mes besoins, à ma très

Cher mère Merieme

Une grande dédicace aussi à mes frère et à ma sœur Ainsi que tout ma famille.

Spécial dédicace à toutes mes amie et A tous les enseignants qui ont contribué à ma formation du primaire jusqu'à l'université.

Enfin, à tous ceux qui j'aime et qui sont dans mon cœur dans cette vie.

Hebara Fatima

TABLE DE MATIERE.

CHAPITRE I : GENERALITE SUR LES MEMOIRES.

CHAPITRE II : PLACEMENT MEMOIRE.

Table des figues.

CHAPITRE II

CHAPITRE III

L

LDPC : Low Density Parity Check Codes.

LTBPA : Low-Traffic Belief Propagation.

S

SAGE : Static Address Génération Easing.

SSA : Single Static Assignment.

R

RAM : Random Access Memory (mémoire vive).

ROM : Read Only Memory (mémoire morte).

RTL : Register Transfer Level.

Au cours des dix dernières années, la demande pour des systèmes de transmission numériques fiables s'est considérablement accrue. L'explosion de l'échange d'informations, et les nouvelles possibilités offertes par le traitement numérique du signal ont accentué cette tendance.

Les systèmes de communication modernes exigent des débits de plus en plus élevés afin de traiter des volumes d'informations en augmentation constante. Ils doivent être flexibles pour pouvoir gérer des environnements multinormes, et évolutifs pour s'adapter aux normes futures.

Pour ces systèmes, la qualité du service doit être garantie et ce malgré l'évolution des technologies microélectroniques qui augmente la sensibilité des circuits intégrés aux perturbations externes (impact de particules, perte de l'intégrité du signal, etc.). La tolérance aux fautes devient un critère important pour améliorer la qualité de service. Aux contraintes liées au traitement de l'information s'ajoute la nécessité de protéger les informations émises dans des environnements perturbés (par exemple erreurs de transmission) où traités dans des systèmes susceptibles d'être parasités par des fautes temporaires causées par les radiations cosmiques SEU (Single Event Upset) . En effet, la qualité du service rendu en termes de communication se décline en deux mots clés : fiabilité et rapidité. La qualité d'une transmission numérique dépend principalement de la probabilité d'occurrence d'erreur dans les symboles transmis. Cette probabilité étant fonction du rapport « signal sur bruit », une amélioration de la qualité de transmission peut être envisagée en augmentant la puissance d'émission et en diminuant le facteur de bruit du récepteur. Malheureusement, cette solution implique des coûts énergétiques et technologiques importants, ce qui en limite sensiblement l'emploi. Le contrôle des erreurs par codage est ainsi indispensable. L'utilisation de techniques de traitement numérique du signal, et notamment le codage des informations à transmettre, permet la détection et/ou la correction d'éventuelles erreurs de transmission. Comme ces techniques permettent de contrôler les erreurs induites par le bruit du canal de transmission, elles sont nommées « codages de canal ». Parmi les principales techniques existantes, les codages en bloc et les codages convolutifs sont prédominants. Les codages en bloc sont utilisés notamment dans les réseaux Ethernet, dans les normes de transmission sans fils telles que Bluetooth, et dans les normes de transmission HDTV (High Definition TeleVision) et DVB-C (Digital Video Broadcasting-Cable) . Le codage convolutif est très présent dans les systèmes de communication numérique sans fil. La stratégie de base du codage consiste à ajouter une quantité contrôlée de redondance à la série d'informations à envoyer. La procédure de

génération de redondance traite les informations, soit par blocs (codage en bloc) ou au contraire de manière continue (codage convolutif), soit comme entité indépendante ou à l'inverse en tant que structure concaténée avec un autre code, soit plus récemment sous forme d'élément constituant dans un code LDPC (Low Density Parity Check) ou turbo code. L'ajout de la redondance par le codeur permet au décodeur de détecter et de corriger le cas échéant un nombre fini d'erreurs de transmission. L'ensemble codeur/décodeur est considéré comme critique pour garantir le bon fonctionnement de la chaine de transmission. Le nombre d'erreurs affectant la transmission des informations dépend du moyen de transmission. Le débit des erreurs et leur distribution temporelle diffèrent selon que le moyen de transport est une ligne téléphonique, une ligne numérique, un lien satellite ou un canal de communication sans fil. Il est dès lors évident que l'introduction de techniques de codage de canal induit une augmentation de la complexité du traitement numérique du système de communication. L'importance de cette augmentation est fonction du niveau de protection envisagé par l'opération de codage : une protection plus efficace contre les erreurs de transmission implique l'utilisation de méthodes de codage plus complexes, rendant par conséquent les procédures de décodage plus onéreuses. Différents codes détecteurs d'erreurs EDC (Error Detecting Codes) ou codes correcteurs d'erreurs ECC (Error Correcting Codes) ont été utilisés pendant des années pour accroître la fiabilité des systèmes des transmissions. De nombreuses architectures parallèles-pipeline ont été conçues pour les codes correcteurs d'erreurs afin d'augmenter leur débit du fonctionnement.

Chapitre I :

Généralité sur les mémoires

I.1Introduction:

Dans ce chapitre on va voir les différents types de mémoires utilisés dans les ordinateurs.

Nous intéresserons maintenant au fonctionnement des mémoires vives (ou volatiles), qui ne conservent leur contenu que lorsqu'elles sont sous tension.

Ce type de mémoire est souvent désigné par l'acronyme RAM, Random Access Memory, signifiant que la mémoire adressable (on peut accéder à n'apport qu'elle information dans la mémoire) par opposition aux mémoires secondaires séquentielles comme les bandes (pour accéder à une information il faut passer sur toutes les informations qu'elles précédent).

I.2Définition:

On appelle « mémoire » tout composant électronique capable de stocker temporairement des données. On distingue ainsi deux grandes catégories de mémoires :

*la mémoire centrale (appelée également mémoire interne) permettant de mémoriser temporairement les données lors de l'exécution des programmes. La mémoire centrale est réalisée à l'aide de micro-conducteurs, c'est-à-dire des circuits électroniques spécialisés rapides.La mémoire centrale correspond à ce que l'on appelle la mémoire vive.

*la mémoire de masse (appelée également mémoire physique ou mémoire externe) permettant de stocker des informations à long terme, y compris lors de l'arrêt de l'ordinateur. La mémoire de masse correspond aux dispositifs de stockage magnétiques, tels que le disque dur, aux dispositifs de stockage optique, correspondant par exemple aux CD-ROM ou aux DVD-ROM, ainsi qu'aux mémoires mortes**[1].**

I.3 Historique :

Au cours de l'histoire, diverses technologies de mémoire ont vu le jour. L'amélioration des techniques de création a produit des mémoires toujours plus petites, moins couteuses, consommant moins d'énergie, avec une capacité toujours plus grande, et une vitesse plus élevée.

*L'usage de la mémoire dans les ordinateurs a été introduit par le concept de l'architecture de Von Neumann, en 1944.

Les premiers disques durs ont été construits en 1956. Le disque DEC RP07 construit en 1970 pesait 180 kg. Un disque dur des années 2000 pèse moins de1 kg, tout en ayant une capacité de stockage supérieure.

Les mémoires à tores de ferrite sont des mémoires vives non volatiles utilisées dans les années 1960 à 1970. Ces composants sont faits d'un réseau de fil de cuivre dans lequel sont entremêlés des anneaux en céramique ferromagnétique. Les mémoires utilisant cette technologie sont volumineuses et lourdes. Cette technologie a été remplacée par des semi-conducteurs et des circuits intégrés.

Les premières générations de mémoires vives consommaient beaucoup d'électricité. L'utilisation de la technologie CMOS a permis des composants beaucoup moins gourmands. Ces composants associés à une minuscule pile ont permis la construction de mémoires rémanentes, utilisées par exemple dans les cartes à puce.

La diminution du nombre d'électrons nécessaires au stockage d'un bit accroit la vitesse de la mémoire. La recherche vise des technologies qui n'utiliseraient qu'un seul électron (ou quelques-uns) à la place de près d'un demi-million nécessaire aujourd'hui au stockage d'un bit, et combineraient la grande miniaturisation et la vitesse des mémoires dynamiques actuelles, avec la rémanence des mémoires mortes**[2].**

I.4 Caractéristiques techniques:

Les principales caractéristiques d'une mémoire sont les suivantes :

***La capacité** : représentant le volume global d'informations (en bits) que la mémoire peut stocker.

***Le temps d'accès** : correspondant à l'intervalle de temps entre la demande de lecture/écriture et la disponibilité de la donnée.

***Le temps de cycle :** représentant l'intervalle de temps minimum entre deux accès successifs.

***Le débit**: définissant le volume d'information échangé par unité de temps, exprimé en bits par seconde.

*** Lanon volatilité :** caractérisant l'aptitude d'une mémoire à conserver les données lorsqu'elle n'est plus alimentée électriquement.

Ainsi, la mémoire idéale possède une grande capacité avec des temps d'accès et temps de cycle très restreints, un débit élevé et est non volatile.

Néanmoins les mémoires rapides sont également les plus onéreuses. C'est la raison pour laquelle des mémoires utilisant différentes technologiques sont utilisées dans un ordinateur, interfacées les unes avec les autres et organisées de façon hiérarchique **[1].**

I.5 Les technologies utilisées pour construire des mémoires principales:

Comme on l'a mentionné précédemment, plusieurs types de mémoires composent un ordinateur: la mémoire principale, les registres et les mémoires périphériques.

Nous allons pour l'instant nous pencher surtout sur la mémoire principale. Ceci nous amène à faire la distinction entre deux autres types de mémoire: la mémoire vive (RandomAcces Memory) etla mémoire morte (Read Only Memory). Par défaut, la mémoire morte, est une mémoire dont le contenu est fixé en permanence. Ce type de mémoire ne peut donc qu'être lu. Par contre, le contenu de la mémoire vive peut être changé à volonté, il s'agit donc d'une mémoire où on peut lire et écrire au besoin.

Jusqu'au milieu des années 70, les mémoires principales étaient constituées de tores magnétiques. Depuis, les mémoires à semi-conducteurs se sont imposées et différentes technologies de mémoires à semi-conducteurs ont été mises au point pour construire des mémoires vives et les mémoires mortes. Certaines recherches sont faites dans le but de mettre au point des procédés utilisant la perforation optique ou l'effet Jacobson, mais pour l'instant, il ne semble pas que les mémoires à semi-conducteurs soient vraiment menacées de disparaître; les dernières améliorations technologiques ont plutôt consisté à pousser de plus en plus l'intégration des circuits.

I.5.1 Les mémoires à tores magnétiques:

Le "tore" magnétique est en fait un petit anneau, fait de ferrite, qui peut prendre deux états: être aimanté dans un sens ou dans l'autre, selon la charge de courant qui lui est appliquée. Les tores, dont le diamètre est d'une fraction de millimètres, sont disposés en rangées et en colonnes pour former un plan où chaque tore représente un bit. Plusieurs plans peuvent être superposés, de façon à ce que les tores dont les coordonnées (rangée, colonne) sont les mêmes sur chaque plan forment des mots.

Chaque tore est traversé par trois fils: un décodeur de rangée et un décodeur de colonne qui permettent de localiser un tore (donc un bit) et finalement un fil détecteur dont la fonction est de détecter le sens de l'aimantation et de le modifier au besoin. Il suffit d'appliquer à un tore une certaine charge de courant pour changer le sens de l'aimantation, et donc, son état.

L'opération de lecture consiste à repérer le tore correspondant à l'adresse recherchée et à utiliser le fil détecteur pour savoir si le bit contient un 0 ou un 1. Comme cette opération se

fait en appliquant une charge sur les fils de rangée et de colonne, chaque opération de lecture a pour conséquence de mettre à zéro le contenu de la cellule lue. Il faut donc lire et réécrire le contenu de la mémoire lue à chaque lecture. Pour effectuer une opération d'écriture, il faut aussi deux étapes: d'abord mettre la cellule à zéro, puis procéder à l'aimantation de la cellule.

I.5.2 Les mémoires à semi-conducteurs:

Un semi-conducteur est un matériau dont la conductibilité électrique se situe entre celle des isolants et celle des métaux. Les plus utilisés sont le germanium et surtout, le silicium.

Les mémoires à lecture seulement, ou ROM (Read Only Memory) contiennent un contenu qui est enregistré de façon définitive lors de la construction, et par conséquent ce contenu est conservé même en cas de perte de tension électrique. Ce n'est pas le cas des mémoires vives ou RAM (pour RandomAcces Memory) qui voient leur contenu s'envoler dès que la tension électrique disparaît, par exemple, dès que l'appareil est éteint.

Les mémoires vives peuvent par contre être lues et écrites autant de fois que nécessaire, car leur contenu n'est pas "câblé".

Quoique laisse supposer l'appellation RAM (RandomAcces signifie accès aléatoire), les deux types de mémoire ont des accès "aléatoires", en ce sens que le temps nécessaire pour accéder à une information varie "au hasard", par opposition aux mémoires où l'accès se fait de façon séquentielle, comme sur les bandes magnétiques.

I.6 Les type des mémoires :

I.6.1 La mémoire vive (RAM) :

Les mémoires vives ou RAM sont aussi des circuits intégrés. Comme le contenu de chaque cellule peut être lu ou écrit, il doit pouvoir varier. Contrairement au cas des ROM, la sortie correspondant à une série de bits d'adresse donnée en entrée n'est pas fixée dès la construction, mais elle peut au contraire changer selon le programme utilisé et les données qui l'alimentent.

Les opérations diffèrent selon qu'on procède à une lecture ou à une écriture. Dans le cas d'une lecture, les bits qui constituent l'adresse sont "reçus" par un décodeur d'adresse qui localise la cellule recherchée.

Selon que cette cellule contient un 0 ou un 1, la donnée est acheminée en sortie sur la ligne delecture/écriture d'un 0 ou sur la ligne de lecture/écriture d'un 1.Pour une opération d'écriture, l'adresse est aussi décodée par le décodeur d'adresse qui localise la cellule

recherchée, et selon qu'on veut écrire un 0 ou un 1, la ligne de lecture/écriture d'un 0 ou la ligne de lecture/écriture d'un 1 est utilisée pour acheminer la donnée à la cellule désirée**[2].**

I.6.1.1 Mémoire vive statique:

Une mémoire vive statique est une mémoire vive qui n'a pas besoin de rafraîchissement.

***SRAM (Static Random Access Memory):**

Cette mémoire utilise le principe des bascules électroniques pour enregistrer l'information. Elle est très rapide, par contre, elle est chère et volumineuse. Elle consomme moins d'électricité que la mémoire dynamique. Elle est utilisée pour les caches mémoire, par exemple les caches mémoire L1, L2 et L3 des microprocesseurs.

***DPRAM (ported Random Access Memory):**

Cette mémoire est une variante de la Static Random Access Memory (SRAM) où on utilise un port double qui permet des accès multiples quasi simultanés, en entrée et en sortie.

***MRAM (Magnetic Random Access Memory):**

Cette mémoire utilise la charge magnétique de l'électron pour enregistrer l'information. Elle possède un débit de l'ordre du gigabit par seconde, un temps d'accès comparable à de la mémoire DRAM (~10 ns) et elle est non-volatile. Étudiée par tous les grands acteurs de l'électronique, elle a commencé à être commercialisée en 2006.

***PRAM (Phase-Change Random Access Memory):**

Cette mémoire utilise le changement de phase du verre pour enregistrer l'information. Elle est non-volatile. Elle a commencé à être commercialisée en 2012.

I.6.1.2 Mémoire vive dynamique:

Une mémoire vive dynamique est une mémoire vive qui a besoin de rafraîchissement.

La simplicité structurelle de la DRAM (un pico-condensateur et un transistor pour un bit) permet d'obtenir une mémoire dense à faible coût. Son inconvénient réside dans les courants de fuite des condensateurs : l'information disparaît à moins que la charge des condensateurs ne soit rafraîchie avec une période de quelques millisecondes. D'où le terme de dynamique. A contrario, les mémoires statiques SRAM n'ont pas besoin de rafraîchissement, mais utilisent plus d'espace et sont plus coûteuses**[3].**

I.6.2 La mémoire morte (ROM):

Originalement, l'expression mémoire morte (en anglais, Read-Only Memory : ROM) désignait une mémoire informatique non volatile (c'est-à-dire une mémoire qui ne s'efface pas lorsque l'appareil qui la contient n'est plus alimenté en électricité) dont le contenu était fixé lors de sa programmation, qui pouvait être lue plusieurs fois par l'utilisateur, mais ne pouvait plus être modifiée.

Avec l'évolution des technologies, la définition du terme mémoire morte a été élargie pour inclure les mémoires non volatiles dont le contenu est fixé lors de leur fabrication, qui peuvent être lues plusieurs fois par l'utilisateur et qui peuvent être modifiées par un utilisateur expérimenté. Ces mémoires sont les UVPROM, les PROM, les EPROM et les EEPROM. Seules les mémoires mortes de première génération sont vraiment mortes et vraiment readonly. Par contre, dans le vocabulaire informatique, la définition des termes mémoire morte (en français) et readonly memory (en anglais) a été élargie pour inclure les autres types, bien que ces mémoires ne soient ni mortes, ni readonly.

I.6.2.1 Types des mémoires mortes :

Les mémoires mortes sont classées selon la possibilité de les programmer et de les effacer :

***Les ROM** (Read Only Memory) dont le contenu est défini lors de la fabrication.

***Les PROM** (Programmable Read Only Memory) sont programmables par l'utilisateur, mais une seule fois en raison du moyen de stockage, les données sont stockées par des fusibles.

***Les EPROM** (Erasable Programmable Read Only Memory)sont effaçables et programmables par l'utilisateur.

***Les EEPROM** (Electrically Erasable Programmable Read Only Memory)sonteffaçables et programmables par l'utilisateur. Elles sont plus faciles à effacer queles EPROM car elles sont effaçables électriquement donc sans manipulations physiques.

***Les UVPROM or Flash EPROM** (Ultra Violet Programmable Read Only Memory) sont des mémoires programmables par l'utilisateur. Elles sont effaçables en les mettant dans une chambre à ultraviolet. Les UV Prom n'ont plus de raison d'être aujourd'hui carde

nouvelles mémoires (par exemple, mémoire Flash) bien plus pratiques les remplacent. Toutefois il est encore possible d'en rencontrer dans certains anciens appareils**[4].**

I.7 La mémoire flash :

*La mémoire flash est une mémoire de masse à semi-conducteurs réinscriptible, c'est-à-dire une mémoire possédant les caractéristiques d'une mémoire vive mais dont les données ne disparaissent pas lors d'une mise hors tension. Ainsi, la mémoire flash stocke les bits de données dans des cellules de mémoire, mais les données sont conservées en mémoire lorsque l'alimentation électrique est coupée.

*Sa vitesse élevée, sa durée de vie et sa faible consommation (qui est même nulle au repos) la rendent très utile pour de nombreuses applications : appareils photo numériques, téléphones cellulaires, imprimantes, assistants personnels (PDA), ordinateurs portables ou dispositifs de lecture et d'enregistrement sonore comme les baladeurs numériques, clés USB. De plus, ce type de mémoire ne possède pas d'éléments mécaniques, ce qui lui confère une grande résistance aux chocs**[5].**

I.8 la gestion de la mémoire :

La gestion de la mémoire est un difficile compromis entre les performances (temps d'accès) et la quantité (espace disponible). On désire en effet tout le temps avoir le maximum de mémoire disponible, mais l'on souhaite rarement que cela se fasse au détriment des performances.

La gestion de la mémoire doit de plus remplir les fonctions suivantes :

*permettre le partage de la mémoire (pour un système multitâche).

*permettre d'allouer des blocs de mémoire aux différentes tâches.

*protéger les espaces mémoire utilisés (empêcher par exemple à un utilisateur de modifier une tâcheexécutée par un autre utilisateur).

*Optimiser la quantité de mémoire disponible, notamment par des mécanismes d'extension de lamémoire**[6].**

I.8.1 L'extension de la mémoire :

Il est possible d'étendre la mémoire de deux manières :En découpant un programme en une partie résidente en mémoire vive et une partiechargée uniquement en mémoire lorsque l'accès à ces données est nécessaire.

En utilisant un mécanisme de mémoire virtuelle, consistant à utiliser le disque dur comme mémoireprincipale et à stocker uniquement dans la RAM les instructions et les données utilisées par leprocesseur. Le système d'exploitation réalise cette opération en créant un fichier temporaire (appeléfichier SWAP, traduisez "fichier d'échange") dans lequel sont stockées les informations lorsque laquantité de mémoire vive n'est plus suffisante. Cette opération se traduit par une baisse considérabledes performances, étant donné que le temps d'accès du disque dur est extrêmement plus faible quecelui de la RAM.

Lors de l'utilisation de la mémoire virtuelle, il est courant de constater que la LED du disque dur restequasiment constamment allumée et dans le cas du système Microsoft Windows qu'un fichier appelé "fichierswap" d'une taille conséquente, proportionnelle aux besoins en mémoire vive, fait son apparition**[6].**

I.8.2 Les mécanismes de découpage de la mémoire :

La mémoire centrale peut-être découpée de trois façons :

***la segmentation** : les programmes sont découpés en parcelles ayant des longueurs variables appelées«segments».

***la pagination**: elle consiste à diviser la mémoire en blocs, et les programmes en pages de longueurfixe.

***une combinaison de segmentation et de pagination**: certaines parties de la mémoire sont segmentées,les autres sont paginées**[6].**

I.9 Conclusion :

Dans ce premier chapitre nous avons présentés les différents types de mémoires leurs utilisations leurs capacité de stockage d'informations ainsi que leurs volatilité/non volatilité pour la sauvegarde des données au sein des cellules mémoire puis la technologie utilisée pour la construction de mémoires principale et en fin la gestion et le découpage de la mémoire (pagination et segmentation).

[1] Web:http://www.commentcamarche.net/contents/751-ordinateur-introduction-a-la-notion-demémoire.

[2] Web : http://docplayer.fr/6681625-Historique-et-architecture-generale-des-ordinateurs.html

[3] Web : http://www.commentcamarche.net/contents/764-ram-memoire-vive.

[4] Web :http://www.commentcamarche.net/contents/765-la-memoire-morte-rom.

[5] Web :http://www.comment ca marche.net/contents/735-carte-mémoire-mémoire-flash.

[6] Web :http://www.commentcamarche.net/contents/1089-la-gestion-de-la-memoire.

Chapitre II :

Placement mémoire

II.1 Introduction :

Dans le chapitre précédent nous avons introduit une généralité sur les différents types de mémoires en particuliers les RAMs et les ROMs.

Dans ce chapitre nous allons envisagés les solutions existantes dans la littérature permettant de répondre au problème de placement mémoire pour des architectures de codeurs/décodeurs parallèles. Certains types d'approches consistent à trouver la bonne assignation des données aux bancs mémoire à la conception, de manière à éliminer tous risques de conflits pouvant survenir durant l'exécution de l'application. Dans ce chapitre, nous présentons l'approche de placement mémoire sans conflit sous contrainte de réseau. Pour ce faire nous devons avoir recours à une approche sur les deux principaux codes correcteurs des erreurs selon la manière avec laquelle on ajoute la redondance : les codes en blocs (Low Density Parity Check Codes, LDPC) et les codes convolutifs (Turbo-Codes) **[1]**. Dans le premier cas, le message est d'abord divisé en blocs de données et le codeur traite ces blocs séparément. Par conséquent, le codeur doit attendre la réception d'un bloc pour démarrer. Dans le deuxième cas, le codeur traite le message de façon continue et génère séquentiellement les bits de redondance sans avoir besoin du message complet.

II.2 Codes Convolutifs :

Le fonctionnement des codes convolutifs est semblable à une machine à état fini qui traduit un flux continu de X bits d'information en C bits codés (avec C>X). Ils sont appliqués dans plusieurs standards de communication grâce à leurs structures simples et à la relative rapidité de leur implémentation. On retrouve ces codes convolutifs dans de nombreux standards de communication.

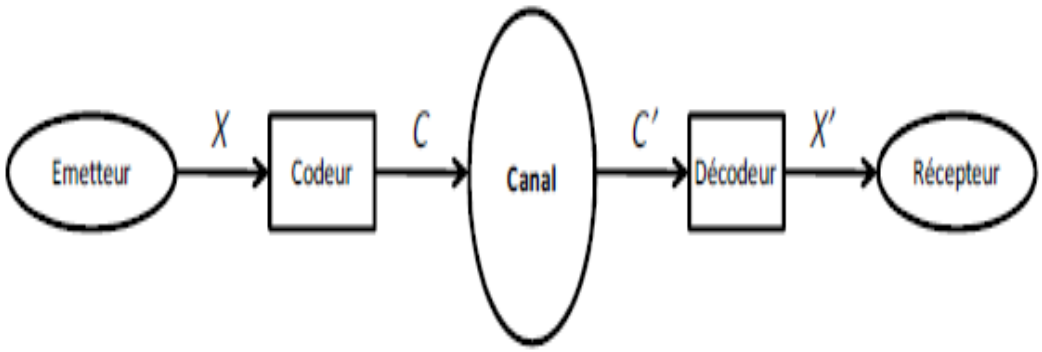


Figure II.1 : Système de communication.

II.3 Turbo-Codes :

Les Turbo-Codes **[1],** grâce à leurs excellentes capacités de correction d'erreurs, sont considérés comme une partie fondamentale des normes de télécommunication actuelles telles que LTE **[2],** HSPA **[3],** DVB-S **[4].** Le turbo-codeur est la concaténation parallèle de deux codeurs convolutifs séparés par un entrelaceur qui permute les séquences de bits afin de casser les relations de voisinage entre eux, ce qui permet d'accroitre les capacités de correction d'erreurs.

Le décodeur est constitué par la concaténation en série de deux décodeurs convolutifs qui sont séparés par deux entrelaceurs. Ces deux décodeurs partagent leurs informations de manière itérative afin de décoder le message reçu. Ce procédé itératif, introduit au niveau du décodage, permet d'obtenir des gains de performances considérables et de se rapprocher à la limite de Shannon. **[5].** En outre, grâce à la faible complexité des algorithmes du décodage itératif, l'implémentation matérielle du Turbo-décodeur est simple à mettre en œuvre. Nous expliquons en détails dans la section suivante les différents éléments qui constituent un turbo encodeur.

II.3.1 Turbo-Codeur :

Un turbo-codeur est la concaténation de deux codeurs convolutifs : le premier encode les bits d'informations X dans l'ordre naturel pour générer les bits de parité $p^{(1)}$, tandis que le second encode les bits d'informations (X) dans l'ordre entrelacé (le message original est introduit dans l'entrelaceur), pour générer les bits de parités $p^{(2)}$, comme c'est indiqué dans la Figure II.2.

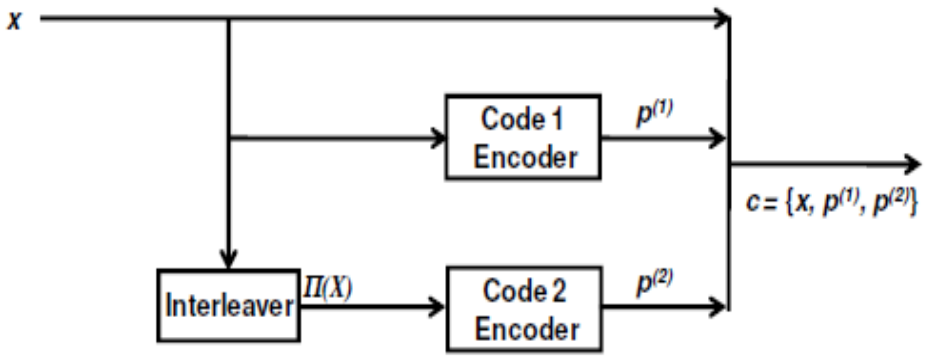


Figure II.2 : Turbo-Codeur.

Les Turbo-Codes étant des codes systématiques, le mot de code à la sortie du codeur est construit par la concaténation parallèle des bits d'information X et des bits de parités générés par les deux codeurs ($p^{(1)}$ et $p^{(2)}$, cf. Figure II.2). Les codeurs utilisés dans les Turbo-Codes sont généralement identiques, mais il est également possible d'utiliser des codeurs différents. Toutefois, du fait de la présence d'un entrelaceur, les bits de parité générés par les deux codeurs sont toujours différents même si les codeurs sont identiques. (cf. Figure II.2).

II.3.2 Entrelaceur :

L'entrelacement est exprimé à partir d'une séquence de permutation = {1, 2, 3,……, n} dont la séquence {1, 2, 3,……, n} représente la permutation des entiers de 1 jusqu'à n.

La fonction d'entrelacement consiste à générer deux bits de parités complètement différents une fois qu'ils sont introduits dans deux codeurs. La meilleure performance des Turbo-Codes est normalement réalisée par l'utilisation d'un entrelaceur qui effectue des permutations aléatoires pour des milliers de bits.

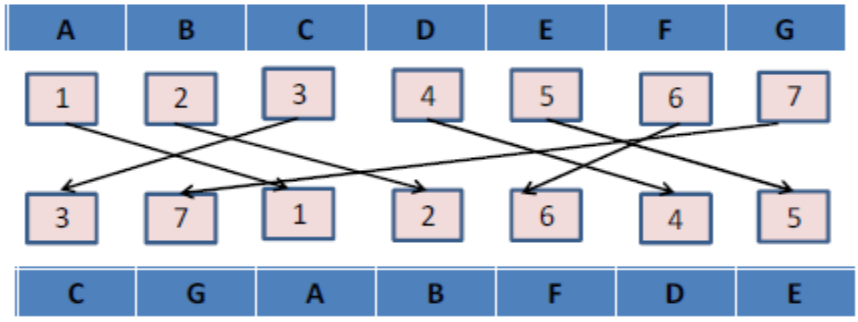


Figure II.3 : Fonction d'entrelacement.

La Figure II.3 montre un exemple de règle d'entrelacement représenté sous la forme d'une séquence de données A, B, C, D, E, F, G qui sera entrelacée selon la règle de permutation représentée par les séquences numériques. Le résultat de cette permutation sera le suivant : la séquence de données après permutation est la suivante C, G, A, B, F, D, E.

II.3.3 Turbo Décodeur :

Le décodage de chaque code convolutif est réalisé par l'utilisation de l'algorithme BCJR **[6]** en turbo décodage, les deux décodeurs partagent l'information sur les bits du message reçu. Cette information est appelée information extrinsèque. Chaque décodeur

fournit l'information extrinsèque à l'autre décodeur afin d'estimer les bits de mot de code transmis par le codeur.

Ensuite, dans le turbo décodage, l'information extrinsèque est mise à jour et partagée par les décodeurs durant plusieurs itérations. L'algorithme BCJR est donc appliqué plusieurs fois par chaque décodeur pour obtenir les bits du mot de code les plus probables. Le bloc diagramme du turbo décodeur est illustré dans la Figure II.4.Le décodeur reçoit des valeurs d'entrée $Y^{(u)}$, $Y^{(1)}$, $Y^{(2)}$ par le canal de transmission correspondant respectivement aux valeurs transmises par le codeur X, $p^{(1)}$, $p^{(2)}$, ces valeurs ont été définies dans la section 0.

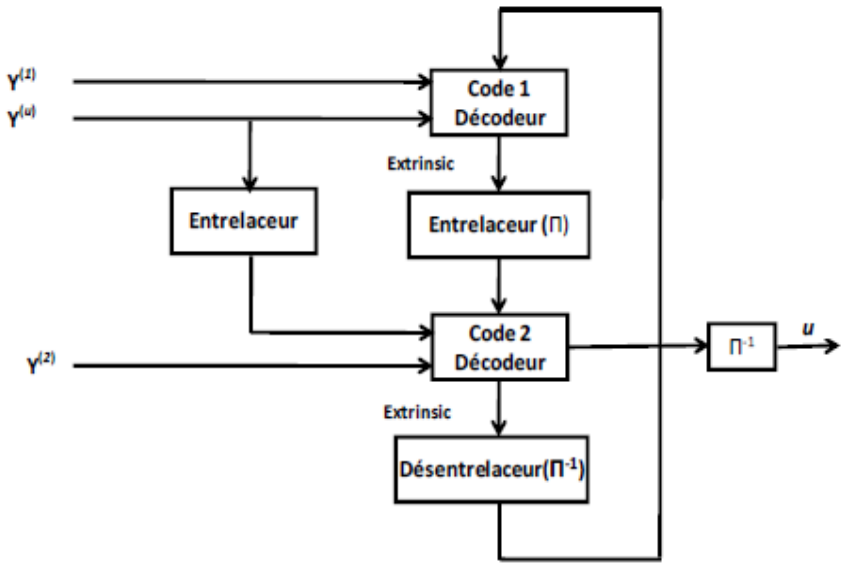


Figure II.4 : Turbo-Décodeur.

Une itération complète du turbo décodeur est faite en deux demi itérations : le premier décodeur reçoit les valeurs par le canal correspondant aux bits du message $Y^{(u)}$, les bits de parité $Y^{(1)}$ et la valeur extrinsèque désentrelacée par le second décodeur afin de générer une nouvelle valeur extrinsèque. Cependant, dans la première itération le premier décodeur ne reçoit pas de valeur extrinsèque par le deuxième décodeur, il utilise uniquement $Y^{(u)}$, $Y^{(1)}$ pour produire la valeur extrinsèque. Durant la seconde demi itération, l'autre décodeur crée la valeur extrinsèque à partir des bits du message entrelacés, les bits de parité $Y^{(2)}$ et la valeur extrinsèque du premier décodeur.

Après un nombre fixé d'itérations, la décision finale sur les bits du message est basée sur les valeurs extrinsèques des deux décodeurs et les valeurs du canal. Il est important de noter que seules les valeurs extrinsèques sont mises à jour à chaque itération, les valeurs du canal correspondant aux bits du message restent toujours fixes.

II.4 Les code en bloc :

Les codes en bloc font partie de la seconde classe de codes correcteurs d'erreurs qui sont utilisés pour transmettre des données numériques de façon fiable via des canaux de communication non fiables, en présence de bruit. Plusieurs types de codes en blocs sont utilisés dans différentes applications. Parmi les codes de blocs classiques on peut citer : le Reed-Solomon **[7]** que l'on retrouve dans les CD, DVD et disques durs, les code Golay **[8]** ou encore les codes de Hamming **[9].**Dans le codage en bloc, le flux de données est réparti en segments, ou blocs, de bits. Chaque bloc de message noté X contient x bits d'information : il en résulte 2x mots de codes possibles. La fonction de codage consiste à ajouter de la redondance à un message d'information X afin de générer un mot de code C de longueur c bits avec c>x. Pour utiliser ce code en bloc dans la pratique, il est nécessaire que les 2x mots de codes soient différents. Transformer le code en bloc en 2x mots de codes de longueur c est une opération coûteuse en termes de surface puisque le codeur doit stocker 2x mots de codes dans la mémoire. Afin de réduire cette complexité, les applications utilisent dans la pratique les codes en blocs linéaires.

II.4.1 Code en bloc linéaire :

Un code en bloc linéaire est une classe de codes en blocs dans lesquels la somme de deux mots de code modulo 2 est également un mot de code.

Dans les codes en blocs linéaires, les mots de codes sont générés par une matrice génératrice G. Cette matrice contient x mots de codes de longueur c et qui sont linéairement indépendants. Le message d'information est multiplié par G pour générer le mot de code correspondant. L'exemple suivant explique la construction d'un code linéaire (7, 4) dont c=7 et x=4. Cet exemple ainsi que les informations sur les codes blocs linéaires sont extraits de **[10].**

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Si X = (1 1 0 1) est le message d'information qui doit être codé alors son mot de code correspondant est :

C = X.*G*

C = **1**(1 1 0 1 0 0 0) ⊕ **1**(0 1 1 0 1 0 0) ⊕ **0**(1 1 1 0 0 1 0) ⊕ **1**(1 0 1 0 0 0 1)

C = (1 1 0 1 0 0 0) ⊕ (0 1 1 0 1 0 0) ⊕ (1 0 1 0 0 0 1)

C = (0 0 0 1 1 0 1)

Puisque le code en bloc linaire est spécifié par G, le codeur doit stocker les x lignes de G afin de pouvoir générer un mot de code de longueur c pour tout message d'information.

Message	Mot de code
(0000)	(0000000)
(1000)	(1101000)
(0100)	(0110100)
(1100)	(1011100)
(0010)	(1110010)
(1010)	(0011010)
(0110)	(1000110)
(1110)	(0101110)
(0001)	(1010001)
(1001)	(0111001)
(0101	(1100101)
(1101)	(0001101)
(0011)	(0100011)
(1011)	(1001011)
(0111)	(0010111)
(1111)	(1111111)

Tableau II.1 : Code en bloc linéaire avec x =4 et c =7.

Le Tableau II.1 montre tous les mots de code générés pour des messages de longueur x=4 et des mots de code de longueur c=7. Il existe plusieurs types de codes en blocs parmi lesquels nous citons les codes en blocs linéaires.

II.5 Les problèmes de conflits mémoires :

Comme décrit précédemment, l'implémentation d'architectures parallèles de décodeurs LDPC ou de turbo-code suppose un compromis entre le coût du matériel et le débit. La Figure II.5 montre une architecture typique d'une implémentation parallèle, dans cette figure, P processeurs (PE) dont le rôle est de traiter des données, communiquent avec B bancs mémoires (avec P=B) via un réseau d'interconnexion dédié.

Un contrôleur se charge de piloter le fonctionnement du système (pilotage du réseau, contrôle des multiplexeurs, des accès mémoire…).

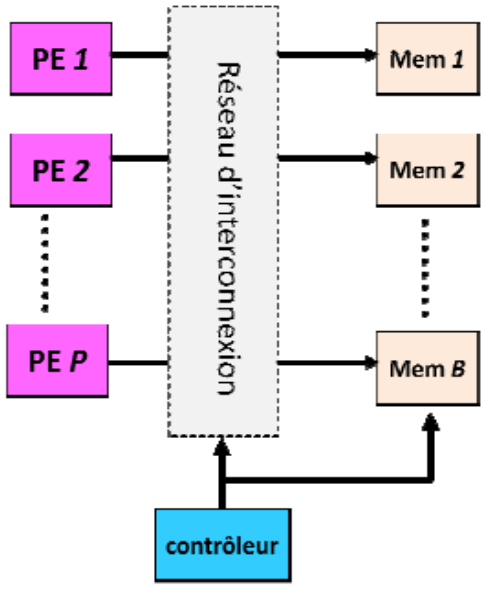


Figure II.5 : Architecture partiellement parallèle.

Un conflit mémoires est présent si deux, ou plus, processeurs accèdent simultanément à des données distinctes stockées dans un même banc mémoire. Les conflits mémoires sont une problématique majeure dans la conception des architectures parallèles. Dans le domaine d'application que nous avons présenté, puisque l'accès en parallèle aux données peut se faire selon de nombreux schémas différents selon les algorithmes et les standards utilisés, nous présenterons dans un premier temps le problème séparément pour les Turbo-Codes, et les codes LDPC.

II.5.1 Les problèmes de conflits mémoire dans les Turbo-Codes :

Comme expliqué précédemment, les entrelaceurs sont utilisés pour améliorer les performances de correction d'erreurs des Turbo-C odes. Ceci est obtenu en mélangeant les données de manière à ce que les bits de parités générés par les deux codeurs soient complètement différents.

Afin d'implémenter les Turbo-Codes parallèles présentés, les entrelaceurs doivent être également parallélisés, afin d'augmenter la bande passante de communication. Pour ce faire, la mémoire est divisée en blocs de mémoires plus petits, de telle sorte que plusieurs données puissent être extraites de la mémoire en même temps dans l'ordre naturel et que dans l'ordre entrelacé. De part le brassage des données qu'il induit, le parallélisme a pour conséquence d'augmenter le risque de conflit mémoire (plusieurs données peuvent être accédées en lecture ou en écriture au même instant dans le même banc mémoire).

Nous présentons le problème à partir d'un exemple d'entrelaceur. Soit un ensemble de D=16 données. Nous structurerons cet ensemble sous la forme d'une matrice d'ordre 4*4, les 4 premières données sont placées dans la première ligne de la matrice, les 4 suivantes dans la deuxième ligne … jusqu'à complétion de la matrice.

(La Figure II.6.a) montre la matrice d'entrelacement. Lorsque que les processeurs doivent accéder aux données en ordre naturel, on lit la matrice ligne par ligne, par contre, pour accéder aux données en ordre entrelacé, on lit la matrice colonne par colonne. Les deux ordres peuvent être présentés de la manière suivante :

Ordre naturel : Contenu de la première ligne, contenu de la deuxième ligne,... contenu de la dernière ligne.

Ordre entrelacé : Contenu de la première colonne, contenu de la deuxième colonne,… contenu de la dernière colonne.

Appliqué à notre exemple on obtient :

Ordre naturel : 0,1,2,3 4,5,6,7 8,9,10,11 12,13,14,15

Ordre entrelacé : 0,4,8,12 1,5,9,13 2,6,10,14 3,7,11,15

Dans un traitement parallèle utilisant la technique de « Sliding Window », le mot de code est divisé en 4 fenêtres dont chacune est traitée par un seul processeur, comme indiqué dans la Figure II.6.b.

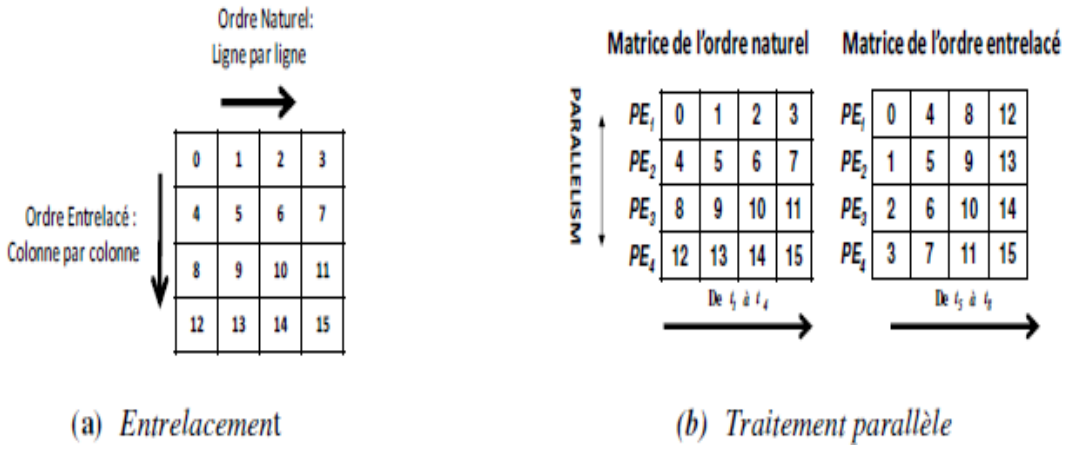
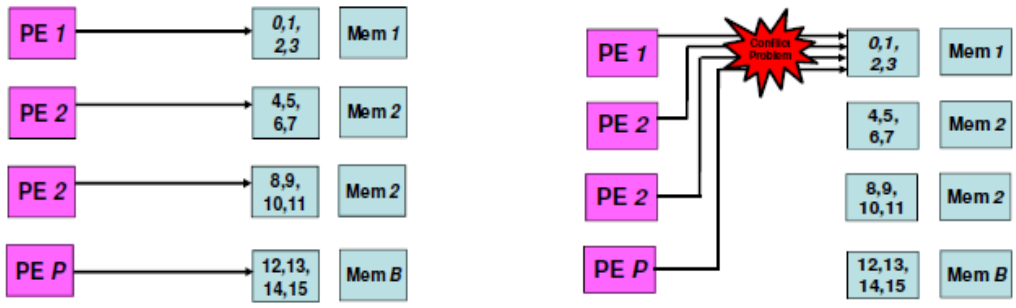


Figure II.6 : Traitement parallèle pour les Turbo Codes.

Afin d'augmenter la bande passante de la mémoire, 4 bancs mémoires sont utilisés de manière à ce que les 4 processeurs puissent tous accéder en parallèle aux données. Le système doit alors être conçu de telle façon que les données nécessaires aux processeurs soient accessibles par ces derniers à chaque instant, à l'ordre naturel. Ceci suppose donc qu'à chaque instant les processeurs accèdent tous à un banc mémoire distinct, comme indiqué dans la Figure II.7.a.



(a) Accès à l'ordre naturel sans conflit (b) Accès à l'ordre entrelacé avec conflit

Figure II.7 : Les problèmes de conflits mémoire dans les turbo-décodeurs parallèles.

Toutefois, pour les accès en ordre entrelacé, on s'aperçoit que le placement des données en mémoire implique ici que tous les processeurs accèdent alors au même banc mémoire à un instant donné (cf. Figure II.7.b). Ceci a pour conséquence de créer des conflits d'accès aux mémoires, et de dégrader la latence d'accès aux données dans la mémoire (à cause de la présence de mécanismes de gestion de ces conflits qu'il va falloir ajouter). Nous sommes là en présence d'un problème découlant directement du parallélisme de l'architecture.

II.5.2 Les problèmes de conflits mémoire dans les codes LDPC :

Nous l'avons évoqué précédemment, si le calcul des nœuds de variables et des nœuds de parité est mathématiquement relativement simple, l'implantation matérielle d'une telle architecture pose des difficultés, notamment en termes de routage des nœuds variable et des nœuds parité. Il s'avère qu'une architecture partiellement parallèle est la solution la plus adéquate pour l'implémentation des décodeurs LDPC **[11][12].** Malheureusement, cette architecture souffre elle aussi de problèmes de conflits d'accès aux mémoires.

Afin d'exposer le problème, nous introduisons une matrice d'affectation dans laquelle P lignes correspondant aux P processeurs, et N colonnes correspondant aux instants (ou cycles de calcul) t_i. Chaque colonne représente les données qui sont accédées en parallèle par les P processeurs à l'instant t_i. Les données situées dans une même ligne sont traitées par un même

processeur. La Figure II.8.a représente la matrice d'affectation dans laquelle D=6 données, P=B est le nombre des bancs mémoires, M=D/B=2 est la taille de chaque bancs mémoires et N=6 cycles de calculs.

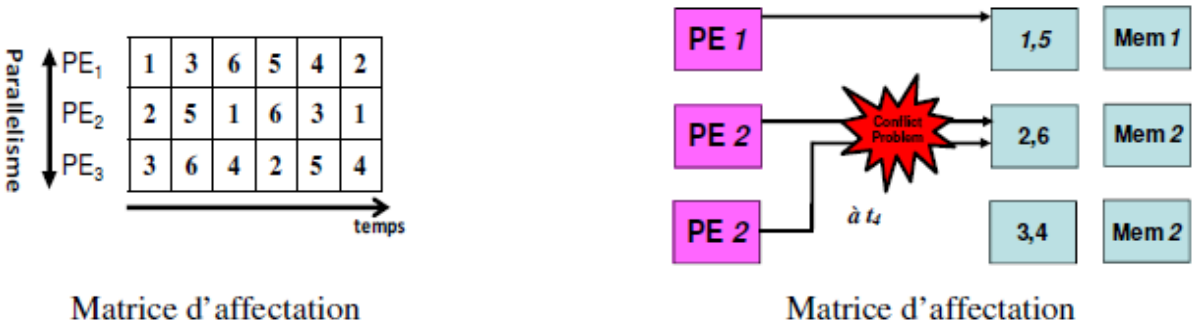


Figure II.8 : Les problèmes de conflits mémoires dans les décodeurs LDPC partiellement parallèles.

Si les données stockées dans les bancs Mem1, banc Mem2 et bancs Mem3 sont respectivement (1, 5), (2, 6) et (3, 4) alors aux instants t_4 et t_5, on se retrouve dans une situation où plus d'un processeur veut accéder au même banc mémoire (cf. Figure II.8.b). Ceci a, à nouveau, pour conséquence d'aggraver les problèmes de conflits mémoires, d'augmenter la latence ainsi que le coût du système.

II.6 problème de coloriage de graphe :

Fondamentalement, la problématique de placement de données en mémoires sans conflit se ramène à un problème de coloriage de graphe. Toutefois, comme nous allons le voir, les algorithmes existant dans le domaine de la théorie des graphes sont soit inefficaces, soit non pertinent dans la pratique s'ils sont utilisés tel quel **[13].** Toutefois avant de rentrer dans ces aspects théoriques, nous présentons les algorithmes et les définitions utilisées qui permettront de comprendre certaines approches utilisées pour concevoir des entrelaceurs sans conflits mémoires.

II.6.1 Définition d'un graphe :

Un graphe G = (V, E) est un ensemble de nœuds V, et un ensemble d'arcs E.

Si v,w ∈ V alors un arc (v, w) ∈ E est incident à v et à w, et les sommets v and w sont dits adjacents.

Un sous graphe de G est un graphe dont les sommets et les arcs sont dans G.

De nombreux problèmes théoriques animent la communauté scientifique autour de la théorie des graphes. Notamment le problème de coloriage minimal des nœuds d'un graphe qui, comme nous le verrons par la suite, permet de formaliser le problème de conflit d'accès à des bancs mémoire.

II.6.2 Les graphes de conflits :

Dans **[14],** les problèmes d'allocation mémoires sont modélisés par des graphes de conflit. Dans un graphe de conflit, toutes les données sont modélisées par des nœuds. Deux données sont reliées par un arc si elles sont accédées en parallèles (i.e. en même temps). Incidemment, cela signifie que ces deux données doivent être stockées dans deux mémoires différentes.

Soient deux données accédées en parallèle, si l'on assimile les bancs mémoires aux couleurs, alors cela revient à dire que les deux données doivent être colorées avec deux couleurs différentes puisqu'elles sont adjacentes dans le graphe de conflit.

***Graphe de conflit :**

Un graphe de conflit *G* (*N, E*) est un graphe dont l'ensemble de nœuds *N* = {n_1, n2, ..., n_n} est Un graphe de conflit G (N, E) est un graphe dont l'ensemble de nœuds N = {n_1, n2, ..., n_n} est défini de la manière suivante :

(a) n_i N, le nœud n_i correspond à une donnée utilisée dans le calcul avec i = {1,2,…,n}.

(b) e_{ij} E, l'arc e_{ij} est incident entre le nœud n_i et le nœud n_j si les données i et j sont conflictuelles.

Une fois le graphe de conflit construit, le problème d'allocation devient alors un problème de coloriage de graphe défini de la manière suivante :

Si on considère n couleurs, puisqu'on a n bancs mémoires qui sont accédés en parallèle, alors le problème de l'assignation des structures de données aux bancs mémoires, de telle sorte qu'aucun conflit ne se produit durant l'exécution de l'architecture parallèle, est un problème de coloriage de graphe.

II.6.3 Problème de coloriage des nœuds :

Etant donné un graphe *G*, est-il possible que les nœuds de G soient colorés avec *n* couleurs en respectant la condition suivante : les nœuds adjacents doivent avoir des couleurs différentes avec *n* le nombre minimal nécessaire de couleurs pour colorier les nœuds de G.

Nous le verrons par la suite, de nombreuses approches de l'état de l'art traitent les problèmes de conflits d'accès à des bancs mémoire en utilisant sous une forme ou une autre, de tels graphes de conflits. Dans **[15]** il a été prouvé que le coloriage des nœuds est un problème NP–complet, et nous verrons par ailleurs qu'utiliser des approches de résolutions basées sur des algorithmes de coloriage de graphes ne permet pas de répondre systématiquement au problème.

Dans une première partie, nous étudierons un ensemble d'approches consistant à construire une règle d'entrelacement qui soit intrinsèquement sans conflit autour d'un réseau d'interconnexion ciblé. Puis, nous introduirons des approches proposant de gérer les conflits d'accès aux mémoires directement dans le réseau d'interconnexion et à la volée, i.e. en cours d'exécution du système, via une architecture dédiée.

Enfin, une dernière famille d'approches consiste à utiliser des algorithmes de placement de données en mémoire qui assignent les données dans des bancs mémoires de manière à ce que tous les processeurs puissent accéder, en parallèles, aux bancs mémoires sans aucun risque de conflit.

II.7 Approches de placement mémoire pour les Turbo-codes :

II.7.1 Sans contrainte de réseau :

Un premier algorithme **[16],** a été proposé pour résoudre les problèmes des conflits mémoire des turbo-codes à la conception. Dans cette méthode l'auteur utilise un méta heuristique dit de « recuit simulé » pour déterminer un placement de données dans des bancs mémoires sans conflits. Les modèles mathématiques et les contraintes d'assignation reliées à ce problème sont détaillés dans **[16].**Par exemple dans la Figure II.9.a, les données 1, 6, 11, 16, 21 doivent être accédées en parallèle au premier cycle de traitement en ordre naturel. La matrice dans la Figure II.9.b représente les accès en ordre entrelacé ; à chaque cycle de traitement (i.e. à chaque colonne de la matrice) est associée une lettre appelée « tuile ». Dans **[16],** cette tuile est utilisée pour « recouvrir » les cases de la matrice d'accès en ordre naturel

(voir Figure II.9.c). Ainsi les données reposant sous une même tuile sont accédées en même temps en ordre entrelacé, et sous ces tuiles, puisque que nous avons là la matrice d'accès en ordre naturel, les données d'une même colonne sont accédées en même temps lors de traitement en ordre naturel. Par conséquent, afin d'avoir une allocation mémoire valide et donc sans conflits, tous les bancs mémoire assignés aux données d'une même tuile de la matrice tuilée et dans chaque colonne de cette matrice doivent être différents.

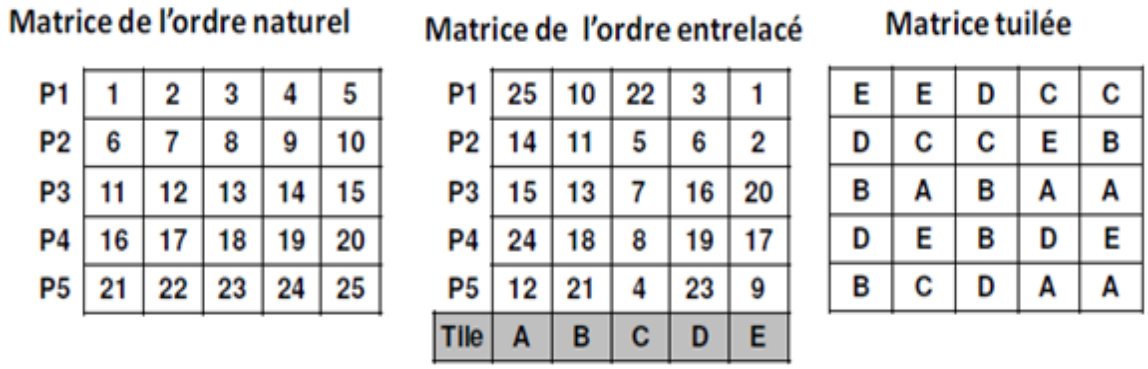


Figure II.9 : Principe de Tuilage.

L'algorithme présenté dans **[16]** est divisé en deux étapes :

Première étape : Il s'agit de la construction d'une matrice d'affectation préliminaire avec les propriétés suivantes : « Chaque colonne ainsi que chaque tuile contient au maximum un élément égal à chaque symbole dans {1,….,P} ». La construction de cette matrice préliminaire est encore améliorée par une initialisation gloutonne de la matrice d'affectation **[17]** de la manière suivante : « Pour i=1,….P, lire la $i^{ème}$ ligne de la matrice d'affectation de gauche à droite et affecter la valeur de l'élément courant à i de telle sorte qu'il y aura pas de collision avec les autres éléments de la même ligne, sinon garder la case vide ».

Deuxième étape : Dans cette étape, l'approche utilise un algorithme de recuit simulé sur la matrice d'affectation préliminaire afin de remplir toutes les cases vides. Pour ceci, l'algorithme injecte une collision soit dans une colonne ou dans une tuile, et résout cette collision à chaque étape en introduisant probablement une autre collision. L'idée principale est que l'ensemble va se stabiliser et l'on pourra ensuite continuer avec les conflits restants.

Le problème de cet algorithme se situe dans la seconde étape. Il n'est pas possible de déterminer le temps nécessaire pour avoir une matrice d'allocation mémoire complète sans conflit. De fait on ne sait pas prédire la convergence de l'algorithme, même si les auteurs disent avoir prouvé cette convergence. De plus, cette approche ne supporte aucune contrainte

sur l'architecture de réseau d'interconnexion cible et ne sait pas optimiser le coût de l'architecture de contrôle. Dans **[18],** l'auteur présente une réallocation optimisée des adresses mémoires (OPMM) dans laquelle des règles d'échanges sans collision sont définies afin de compléter la procédure du recuit simulée plus rapidement que celle utilisée dans la méthode traditionnelle présentée dans le document **[16].** L'OPMM accélère la procédure du recuit simulé de deux façons : d'abord l'algorithme sélectionne des paires de données qui ont besoin d'échanger leurs bancs mémoires afin d'effectuer plusieurs étapes d'OPMM en une seule itération, ensuite durant les futures itérations, les paires de données sélectionnées échangent leurs informations de bancs de telle sorte que les données peuvent seulement varier entre deux bancs mémoires au lieu de P bancs mémoire.

Les expériences réalisées montrent que la procédure OPMM est capable de trouver une allocation mémoire sans conflit plus rapidement. Cependant, la méthode est basée sur une méta heuristique et la complexité ainsi que le temps d'exécution de l'algorithme sont inconnus. De plus, comme **[16],** il ne gère pas la contrainte sur le réseau d'interconnexion, ni l'optimisation du contrôleur.

II.7.2 Avec contrainte de réseau :

Dans **[19],** les auteurs présentent une nouvelle approche simplifiée appelée « Static Address Génération Easing» (SAGE), cette méthode rajoute des contraintes supplémentaires permettant d'orienter l'allocation mémoire afin de cibler une architecture d'entrelaceur particulière. Dans cette approche, deux matrices vides appelées « SAGE Mapping Matrices » sont utilisées pour stocker les informations des bancs mémoire durant l'exécution de l'algorithme. Ces deux matrices (MAP_{Nat}, MAP_{Int}) sont de même ordres et elles sont illustrées dans la Figure II.10.

Afin de cibler une architecture particulière sans conflit mémoire, deux contraintes sont définies et doivent être respectées durant l'exécution de l'algorithme d'allocation mémoire. Premièrement, chaque colonne des matrices d'allocation doit contenir des bancs mémoire différents. Deuxièmement, chaque colonne respect e des règles de pilotage du réseau d'interconnexion ciblé si la loi de permutation le permet.

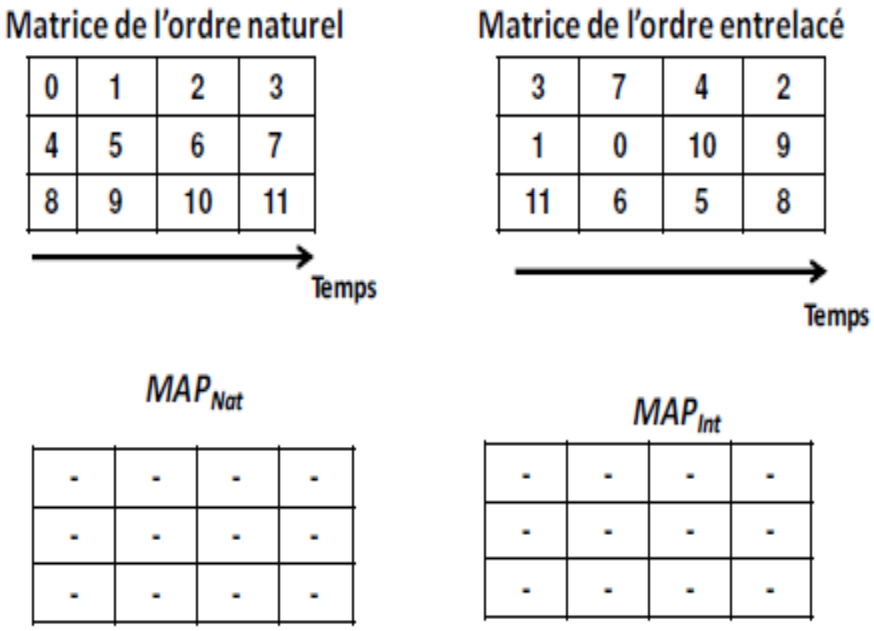


Figure II.10 : Matrices utilisées dans SAGE.

L'algorithme commence par effectuer une première assignation à la première colonne de MAP_{Nat}, puis l'algorithme met à jour dans MAP_{Int} les données qui viennent d'être assignées dans MAP_{Nat}. Ensuite, à chaque itération l'algorithme sélectionne la colonne la plus Contrainte dans l'une des deux matrices (la colonne ayant le plus des données déjà assignées), puis remplit la colonne avec des informations d'assignations respectant les contraintes définies et met à jour ces informations dans la seconde matrice jusqu'à ce que les colonnes des deux matrices soient complètement remplies.

Cette approche, si elle se révèle capable de trouver un placement des données en mémoire sans conflit doit faire face à plusieurs limitations : en premier lieu, si la règle d'entrelacement ne permet pas nativement de respecter l'architecture cible définie pour le réseau d'interconnexion, alors **[19]** ne sera pas capable de générer l'architecture voulues par le concepteur. De plus, cette approche n'offre pas de solution pour l'optimisation de l'unité de contrôle du système.

Dans **[20],** le problème est formulée au travers de deux matrices : une matrice qui exprime l'accès à l'ordre naturel et l'autre qui exprime l'accès à l'ordre entrelacé (voir Figure II. 11). Chaque matrice a P lignes qui montrent les éléments de traitements et T/2 colonnes qui montrent les instants d'accès. Le problème d'allocation mémoire est modélisé par un graphe biparti. Ensuite le graphe est transformé en un problème de transport auxquels divers

algorithmes de l'état de l'art peuvent être appliqués pour trouver l'allocation mémoire des données.

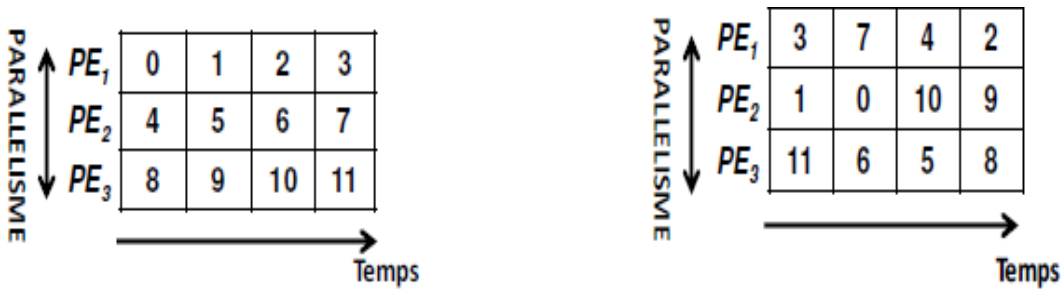


Figure II.11 : Matrice d'accès aux données pour les turbo-codes.

Le graphe biparti G = (T ∪ D, E) est illustré dans la Figure II.12, avec T l'ensemble des sommets qui représentent tous les instants d'accès aux données et D est l'ensemble des sommets qui représentent toutes les données utilisées dans le calcul. Un arc (t,d) relie la donnée d à l'instant t si d doit être traité à l'instant t.

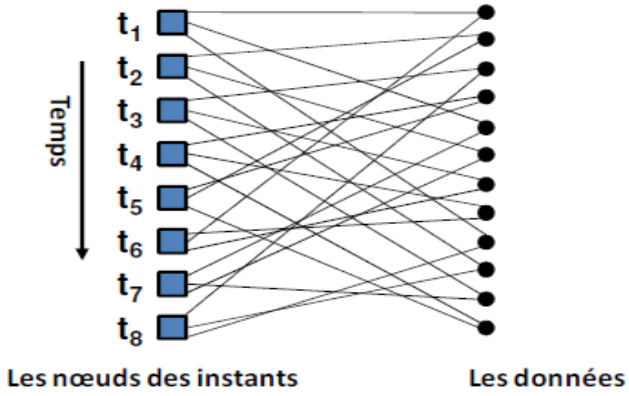


Figure II.12 : Représentation du graphe biparti pour la Figure II.11.

Le problème d'allocation mémoire est ensuite transformé en un problème de « transport ». Une matrice est construite en respectant un réseau d'interconnexion particulier. Une fois la matrice construite, une approche basée sur la programmation linéaire est utilisée pour trouver une allocation de mémoire sans conflit.

Cette approche permet de trouver une assignation mémoire sans conflits et de concevoir un réseau d'interconnexion simple à condition la règle d'entrelacement le permette. Ainsi, cette approche doit faire face aux mêmes limitations que **[19].**

II.8 Approches de placement mémoire pour les codes LDPC :

Les approches présentées dans les documents **[16]** et **[18]** peuvent également être utilisées dans le cas des codes LDPC, à condition (cf. **[16]**) de gérer les données en mode « Single Static Assignment » (SSA), c'est-à-dire que chaque accès à une donnée sera considéré comme indépendant des autres et devra être mémorisé dans une case mémoire spécifique. Autrement dit, si une donnée est accédée 4 fois, plutôt que de faire 4 accès dans une même case mémoire, il faudra utiliser 4 cases mémoires distinctes. Par conséquent, l'ajout des bancs mémoires supplémentaires devient indispensable afin de trouver une allocation mémoire sans conflit dans certains cas d'application.

L'implémentation de l'algorithme d'allocation mémoire présenté dans le document **[16]** est détaillée dans le document **[21].** L'architecture se compose de deux cross bars ou bien deux réseaux de Benes afin de pouvoir gérer l'échange des données entre les différents processeurs. Un cross bar est utilisé pour transmettre les informations aux nœuds de variable et un autre est utilisé pour transmettre les informations aux nœuds de parité.

L'algorithme «Traditional Belief Propagation » a été modifié pour avoir l'algorithme « Low-Traffic Belief Propagation » (LTBPA) dans le but d'utiliser un seul réseau d'interconnexion, néanmoins le LTBPA n'est pas l'algorithme standard utilisé dans les implémentations des décodeurs actuels.

Ces approches font quoi qu'il arrive face à un surcoût mémoire important du fait de leur mode de fonctionnement en SSA. De plus, elles ne peuvent pas être utilisées pour cibler un réseau d'interconnexion particulier, ni optimiser le coût du contrôleur.

Dans **[13],** les auteurs présentent une technique basée sur l'utilisation de bancs mémoire potentiellement distincts pour la lecture d'une donnée à un cycle de traitement donné et l'écriture du résultat correspondant. Ainsi selon cette approche, chaque donnée est assigné à deux bancs mémoires, un à partir desquels le processeur lit la donnée et un autre dans le quel le processeur écrit le résultat. C'est pourquoi dans l'exemple de la Figure II.13 deux cases sont associées à chaque donnée di. Afin d'obtenir une allocation mémoire valide, si une donnée est accédée plusieurs fois, son $i^{ème}$ accès en lecture doit être assigné au même banc mémoire que celui assigné à son $(i-1)^{ème}$ accès en écriture. Deux autres contraintes doivent également être vérifiées : dans chaque colonne tous les bancs mémoires doivent être différents

(accès mémoire sans conflit) et le premier accès en lecture d'une donnée di doit être assigné au même banc mémoire que celui assigné à son dernier accès en écriture.

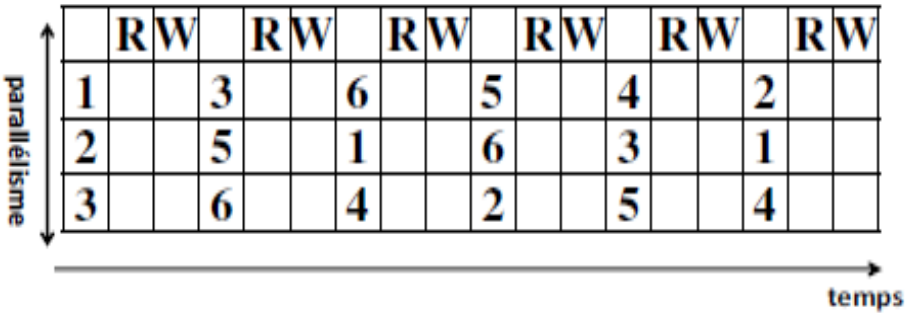


Figure II.13 : Matrice d'assignation de l'approche basée sur la lecture multiple et l'écriture Multiple.

L'algorithme commence par effectuer une première assignation des bancs mémoires aux données situées dans les colonnes de lecture et d'écriture du cycle de traitement t_1. Ensuite, l'algorithme continu l'assignation de l'accès en lecture et en écriture des données situées dans les colonnes qui suivent, tout en respectant les contraintes imposées.

Ceci se poursuit tant qu'aucun conflit n'apparait. En cas de conflit l'algorithme effectue un retour en arrière pour aller à l'occurrence la plus poche de la donnée conflictuelle afin de changer son assignation et résoudre ainsi le conflit. Ce processus continue jusqu'à l'obtention d'une assignation mémoire valide de toutes les données. Cet algorithme décrit une nouvelle approche pour résoudre des problèmes difficiles présents dans le cadre des codes LDPC. Cependant, cet algorithme est basé sur une approche récursive et le temps d'exécution est inconnu. De plus il ne sait pas cibler un réseau d'interconnexion particulier, ni optimiser le coût du contrôleur. Dans le document **[22],** l'auteur présente un algorithme permettant de résoudre le problème de conflits mémoire pour des codes LDPC en se basant sur les graphes bipartis. La formulation du problème est basée sur le concept « Double Memory Mapping » **[13].** Puis l'accès aux données est modélisé par un graphe biparti (cf. Figure II.14). Ensuite un algorithme d'allocation de mémoire est appliqué pour colorer les arcs de ce graphe tout en respectant les contraintes. Afin de faciliter le coloriage des arcs, le graphe biparti est divisé en sous graphes et chaque sous graphe est traité de manière séparée.

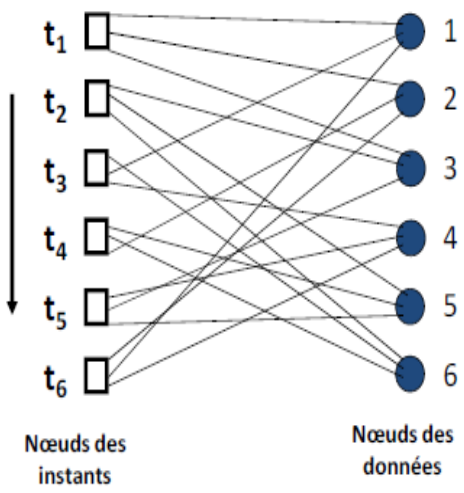


Figure II.14 : Modélisation biparti du problème d'allocation de la mémoire pour les codes LDPC.

Afin de diviser le graphe en sous graphes un algorithme de partitionnement est appliqué. Dans cet algorithme certaines contraintes de partitionnements sont imposées. L'algorithme est basé sur deux processus : un processus de parcours qui permet de construire un chemin dans le graphe de manière aléatoire et un autre processus d'élimination dont le rôle est d'éliminer tous les arcs qui contredisent les contraintes de partitionnement. Une fois le partitionnement effectué, un algorithme de coloriage des arcs du graphe biparti est appliqué pour chaque partition

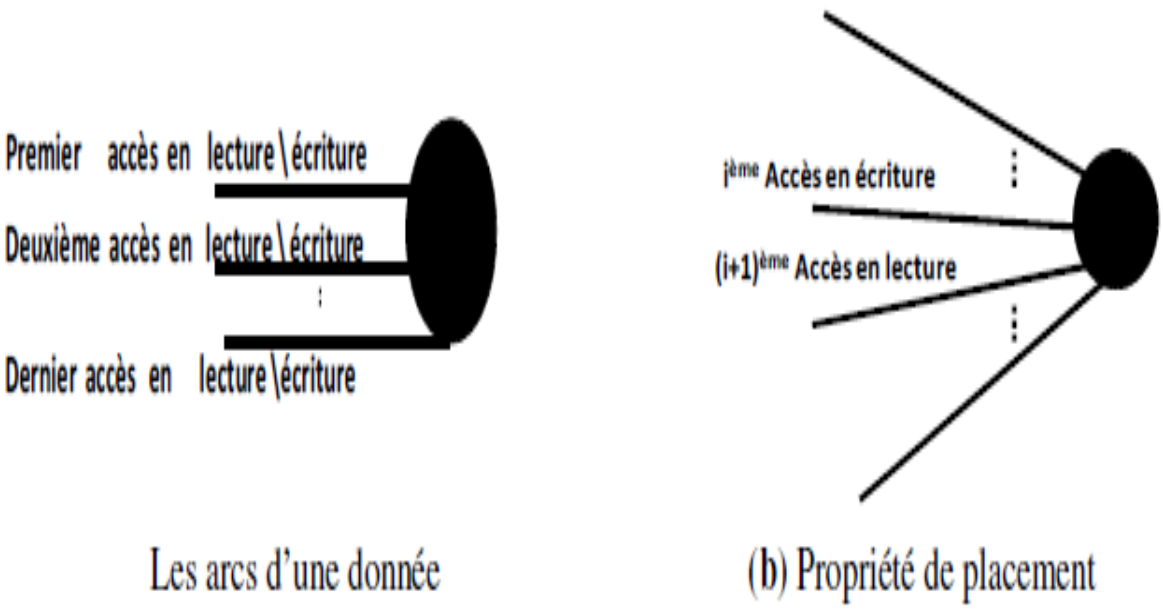


Figure II.15 : Représentation des arcs d'un nœud d'une donnée du graphe biparti.

Grâce à la modélisation du problème d'allocation mémoire par un graphe biparti, et à l'utilisation du concept de « allocation mémoire double », cette approche est capable de concevoir un entrelaceur sans conflits mémoire avec un nombre réduit de bancs mémoire. Néanmoins, elle ne cible pas un réseau d'interconnexion particulier, et n'est pas non plus capable d'optimiser le coût du contrôleur.

Deux autres approches **[23] [20]** ont été également proposées par les mêmes auteurs. Elles se basent sur l'utilisation d'un graphe triparti. L'avantage de ses méthodes réside dans le fait qu'un graphe triparti peut être converti en un graphe biparti sur lequel n'importe quel algorithme de coloriage d'arc peut être appliqué. Il est alors possible d'exploiter des algorithmes de coloriage d'arcs efficaces.

La première approche **[23]** est dédiée à la résolution les problèmes d'allocations mémoires des codes LDPC. La modélisation du graphe tripartie est basée sur la matrice d'accès aux données illustrée dans Figure II.11. La Figure II.16 illustre un graphe triparti. Chaque graphe construit est partitionné en différents sous graphes colorés de manière indépendante.

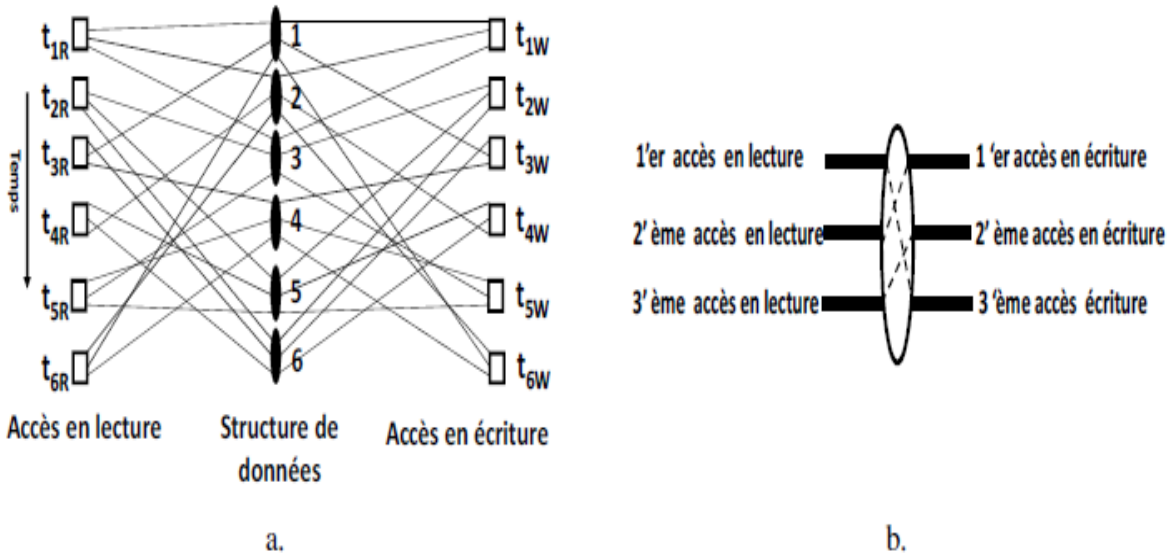


Figure II.16 : Graphe tripartie pour d'accès aux données illustrées dans Figure II.11.

Dans la seconde approche **[20],** les deux techniques d'allocation mémoire simple et double sont modélisées par un graphe triparti. Ensuite, les arcs corrélés (c'est-à-dire devant être assignés au même banc mémoire, cf. Figure II.16) dans ce graphe sont reliés et les nœuds de données sont supprimés afin de générer un graphe biparti.

Ainsi, n'importe quel algorithme de coloriage des arcs d'un graphe biparti peut être appliqué.

Une comparaison de la complexité de ces différentes approches, montre que **[20]** est la meilleure en termes de généricité et également de complexité puisqu'elle peut être exécutée en un temps polynomial. Cette approche est la plus rapide de toute mais elle ne sait pas cibler un réseau ni réduire la complexité du contrôleur.

II.9 Placement mémoire sous contrainte de réseau :

Il y a une nouvelle approche d'allocation mémoire basée sur une heuristique gloutonne qui génère une architecture RTL d'un entrelaceur mémoire sans conflit à partir d'une règle d'entrelacement. Notre contribution comparée aux méthodes existantes consiste à considérer le réseau d'interconnexion comme étant une contrainte définie par le concepteur et non plus comme un objectif ciblé (i.e. **[22], [20]**). L'approche que nous proposons peut être appliquée dans le cas des Turbo-Codes aussi bien que pour des codes LDPC, puisqu'elle est basée sur le concept « d'allocation mémoire double » (tiré de **[19]**) comme nous le verrons par la suite.

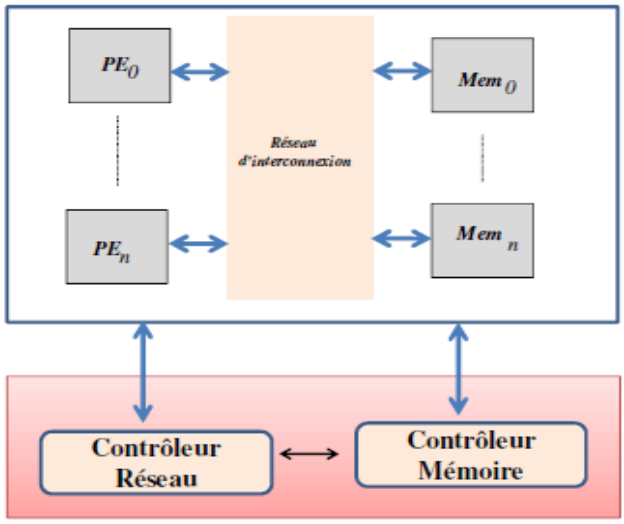


Figure II.17 : Architecture typique d'un entrelaceur mémoire.

Notre approche, au-delà de se conformer aux désidératas du concepteur (définition du réseau d'interconnexion), vise également à optimiser le coût de l'architecture d'entrelacement dans son ensemble, c'est-à-dire en minimisant le coût de l'unité de contrôle associée. Pour rappel, l'architecture d'un entrelaceur mémoire (cf. Figure II.17) est généralement composée d'un réseau d'interconnexion reliant des unités de calculs à des bancs mémoire et d'une unité de contrôle dont une partie est dédiée pour le contrôle du réseau et une autre pour le contrôle des bancs mémoire.

Les concepts théoriques et les modèles formels ainsi que les différentes étapes détaillées de deux approches d'allocation mémoire permettant la conception d'une architecture d'entrelacement sans conflit. La première méthode cible l'optimisation de la partie réseau en explorant l'ensemble des solutions d'assignation des données aux bancs mémoire (tout en respectant une topologie réseau bien définie) de façon macroscopique en traitant d'un bloc l'ensemble de données mis en jeu à chaque cycle de codage/décodage. Cette première solution qui présente les principes clefs de notre approche à toutefois été largement améliorée pour aboutir à une seconde approche qui consiste à placer les données dans les bancs mémoire d'une manière plus fine en explorant les solutions de placement mémoire individuellement pour chaque accès aux données .La problématique de l'optimisation de l'unité de contrôle sera toutefois développée dans le chapitre suivant.

II.10 Formulation du problème d'allocation mémoire pour les Turbo-codes et les codes LDPC :

Afin d'expliquer le problème, nous considérons un ensemble de structures de données $\{d_0,d_1\ldots\ldots,d_{D-1}\}$ et un ensemble d'éléments de traitement $\{PE_0,PE_1,\ldots..PE_{p-1}\}$ qui traitent les D structures de données en T cycles $\{t_0,t_1\ldots, t_{T-1}\}$. Afin de stocker les D structures de données et réaliser un traitement itératif parallèle des données, nous utilisons un ensemble de B bancs mémoire $\{b_0b_1,\ldots.,b_{B-1}\}$ avec B P.

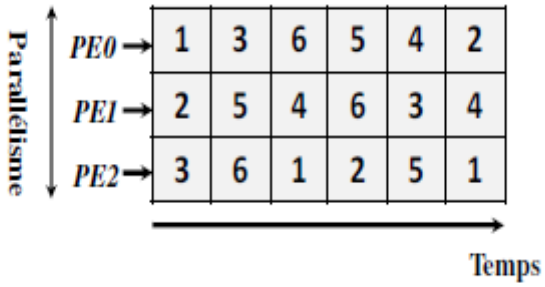


Figure II.18 : Matrice d'accès aux données.

La problématique d'allocation mémoire est classiquement modélisée par une matrice appelée matrice d'accès aux données (Figure II.18). Cette matrice a P lignes qui correspondent aux P éléments de traitement et T colonnes qui représentent les instants t_i . Elle contient 6 données dont chacune est accédées 3 fois, et ce dans un ordre pseudo-aléatoire (défini par une règle d'entrelacement).

II.10.1 Allocation mémoire double :

Afin d'appliquer le concept d'« allocation mémoire double», chaque colonne est divisée en deux sous colonnes. La première sous colonne contient les bancs mémoire dans lesquels les données qui sont accédées en parallèle à l'instant t_i sont lues et la seconde sous colonne montre les bancs mémoire dans lesquels les résultats du traitement de ces données sont écrits. En outre, les données de chaque ligne sont accédées par le processeur qui est connecté à cette ligne. La Figure II.19.b illustre un élément de la matrice d'affectation où la donnée e_i est lue dans le banc mémoire b_j et écrite après avoir été traitée par un processeur dans le banc mémoire b_k .

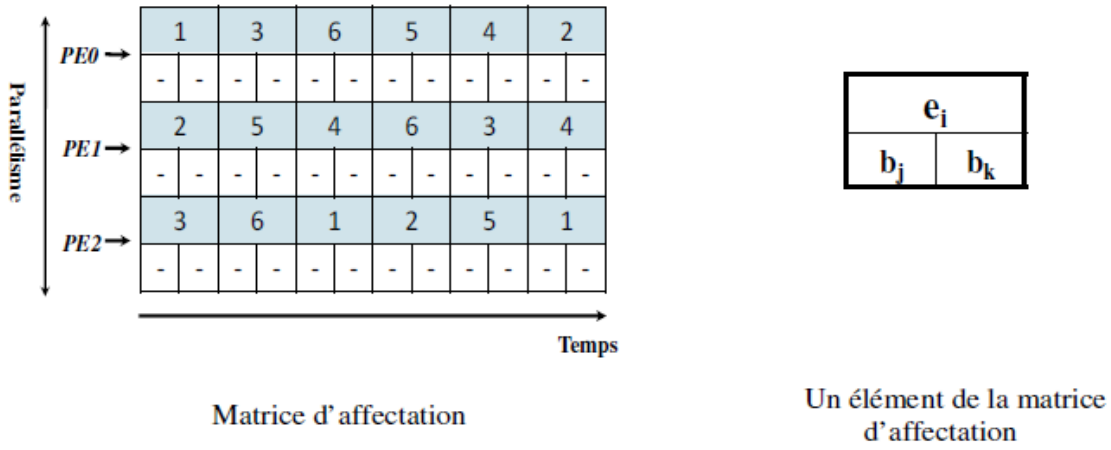


Figure II.19 : Matrice d'affectation pour la mémoire d'accès aux données.

Afin de faciliter les explications de nos approches nous définissons une Semi-colonne de la manière suivante :

II.10.1.1 Semi-colonne :

On parle de Semi-colonne de lecture (resp. d'écriture) pour désigner l'ensemble des accès en lecture (resp. en écriture) réalisés à un cycle de traitement ti.

II.10.1.2 Contraintes d'assignation :

Afin de garantir une allocation mémoire valide, un ensemble de contraintes de placement mémoire et des contraintes de réseau doivent être respectées.

A- Contraintes de placement mémoire :

Les données qui sont traitées au même instant (lues par les éléments de traitement, ou bien écrites), i.e. qui appartiennent à une même Semi-colonne dans la matrice d'affectation, doivent être assignées à des bancs mémoire différents. Le $n^{ème}$ accès en lecture de la donnée e_i doit être effectué dans le même banc mémoire que son $(n-1)^{ème}$ accès en écriture, c'est-à-dire la donnée doit être lue dans le même banc mémoire où elle a été écrite. En outre, puisque les algorithmes de décodage sont basés sur des approches itératives, si le premier accès en lecture à la donnée e_i est effectué dans le banc mémoire b_j alors le dernier accès en écriture à la donnée e_i doit être effectué dans ce même banc mémoire b_j.

B- Contraintes de réseau :

L'allocation mémoire doit respecter une topologie de réseau définie par l'utilisateur. Ceci se traduit par un ensemble de permutations de données que devra pouvoir réaliser l'architecture. Nous pouvons citer l'exemple d'un « Barrel Shifter » qui peut réaliser seulement des permutations circulaires tandis qu'un « Cross bar » peut effectuer toutes les permutations.

II.10.1.3 Problème d'assignation :

Le problème à résoudre peut se formuler ainsi : « Stocker D données dans B bancs mémoire de telle sorte que les P éléments de traitement puissent accéder sans aucun conflit et en parallèle à chaque instant aux P bancs mémoire pour la lecture de ces P données et ensuite l'écriture des P résultats correspondant».

II.11 Allocation mémoire sans conflit sous contrainte de réseau d'interconnexion :

II.11.1 Première approche d'allocation mémoire :

Nous proposons dans cette section une première méthode d'assignation mémoire permettant de résoudre les problèmes de conflits mémoire lors de la conception d'applications dédiées aux Turbo-Codes et aux codes LDPC. Son point de départ est une règle d'entrelacement modélisée par une matrice d'accès aux données, et une contrainte architecturale qui consiste à respecter une topologie d'un réseau particulier défini par le concepteur.

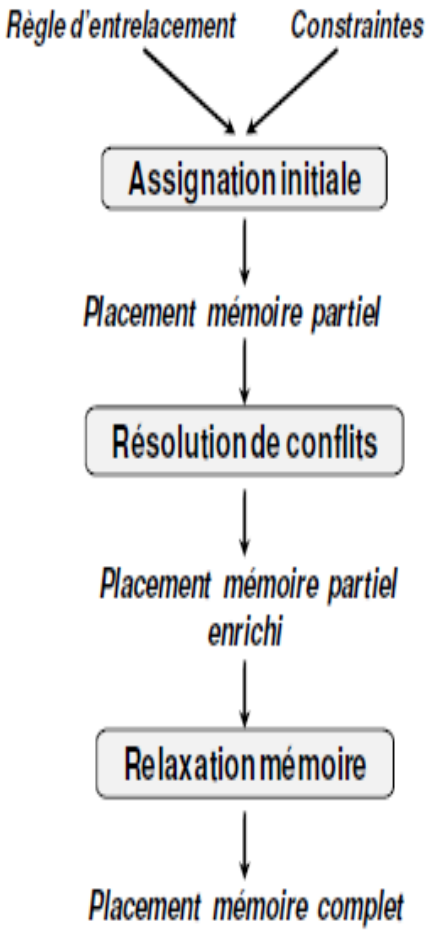


Figure II.20 : Flot de conception**.**

Relaxation :

On parle de relaxation mémoire lorsque l'on élargit l'espace mémoire disponible en lui rajoutant des éléments de mémorisation supplémentaires (registres, FIFO...).

A. Assignation initiale

Cette première étape a pour objectif de générer une première allocation mémoire sans conflit. Si un ensemble de données accédées en parallèle ne respecte pas les contraintes de la mémoire ou bien du réseau, alors les assignations de ces éléments conflictuels sont reportées, les différentes étapes de l'assignation initiale sont illustrées dans la Figure II.21.

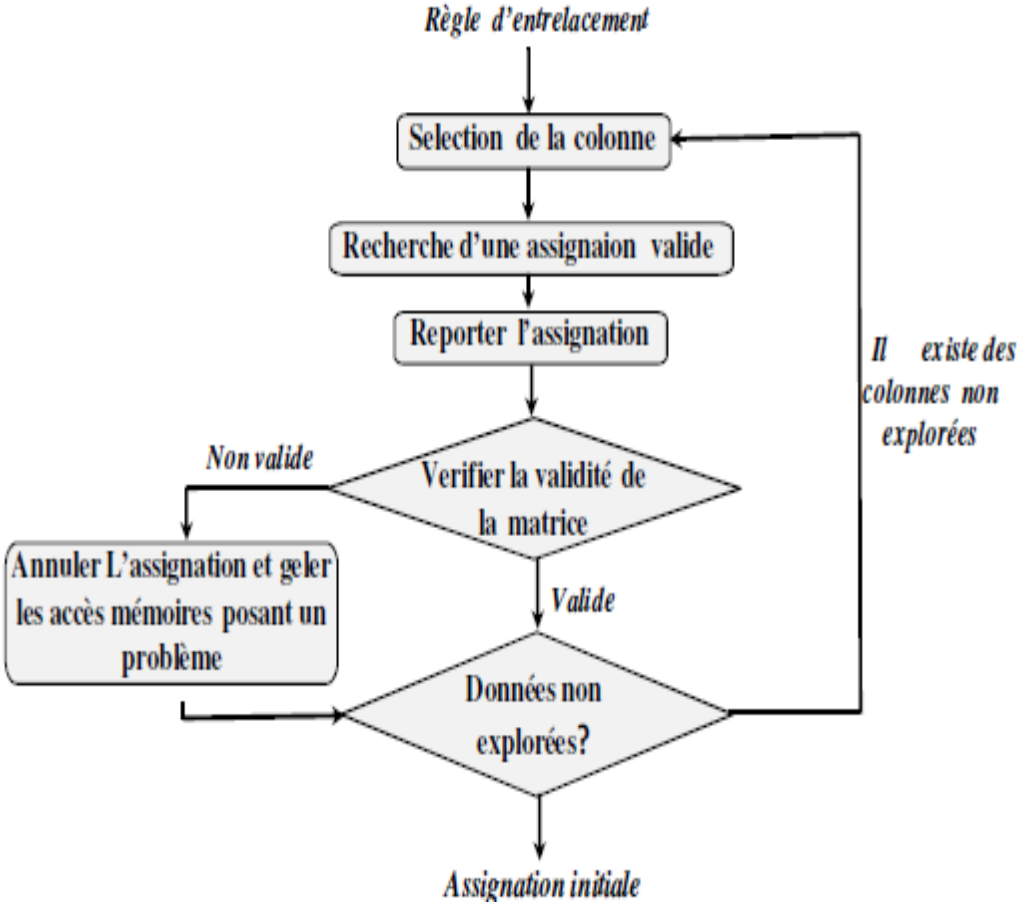


Figure II.21 : Assignation initiale.

L'assignation initiale est effectuée comme suit jusqu'à ce que toutes les colonnes aient étés traitées.

Sélection de la colonne :

Dans cette étape l'algorithme sélectionne la colonne la plus contrainte et qui n'a pas encore été explorée. La Semi-colonne la plus contrainte (lecture ou écriture) est celle qui contient le nombre minimum de données n'ayant pas encore été assignées à des bancs mémoire.

Assignation et report :

L'algorithme recherche une assignation mémoire valide respectant les contraintes de placement mémoire et de réseau pour la Semi-colonne sélectionnée. Il s'agit d'une solution valide localement à la Semi-colonne sélectionnée. Ensuite, l'algorithme met à jour la matrice d'affectation en reportant l'assignation de cette Semi-colonne sur les cellules concernées. La validité de ce report n'est pas vérifiée dans cette étape.

Vérification :

Dans cette étape, l'algorithme vérifie si les cellules qui ont été mises à jour dans l'étape « Assignation et report » respectent les contraintes de mémoire et de réseau. Si ce n'est pas le cas, les assignations des

données appartenant aux Semi-colonne qui ne respectent pas les contraintes seront annulées et gelées. En effet, ces accès mémoire ne seront plus pris en compte au cours de cette étape (d'assignation initiale).

Nous introduisons certains termes qui sont utilisées dans la majorité des algorithmes que nous détaillerons par la suite :

C_i : Une Semi-colonne de la matrice d'assignation.

L_{Ci} : Une solution d'assignation mémoire valide pour la colonne Ci. d_k : une donnée de la colonne C_i.

Ld_k : une solution d'assignation mémoire valide pour la donnée dk.

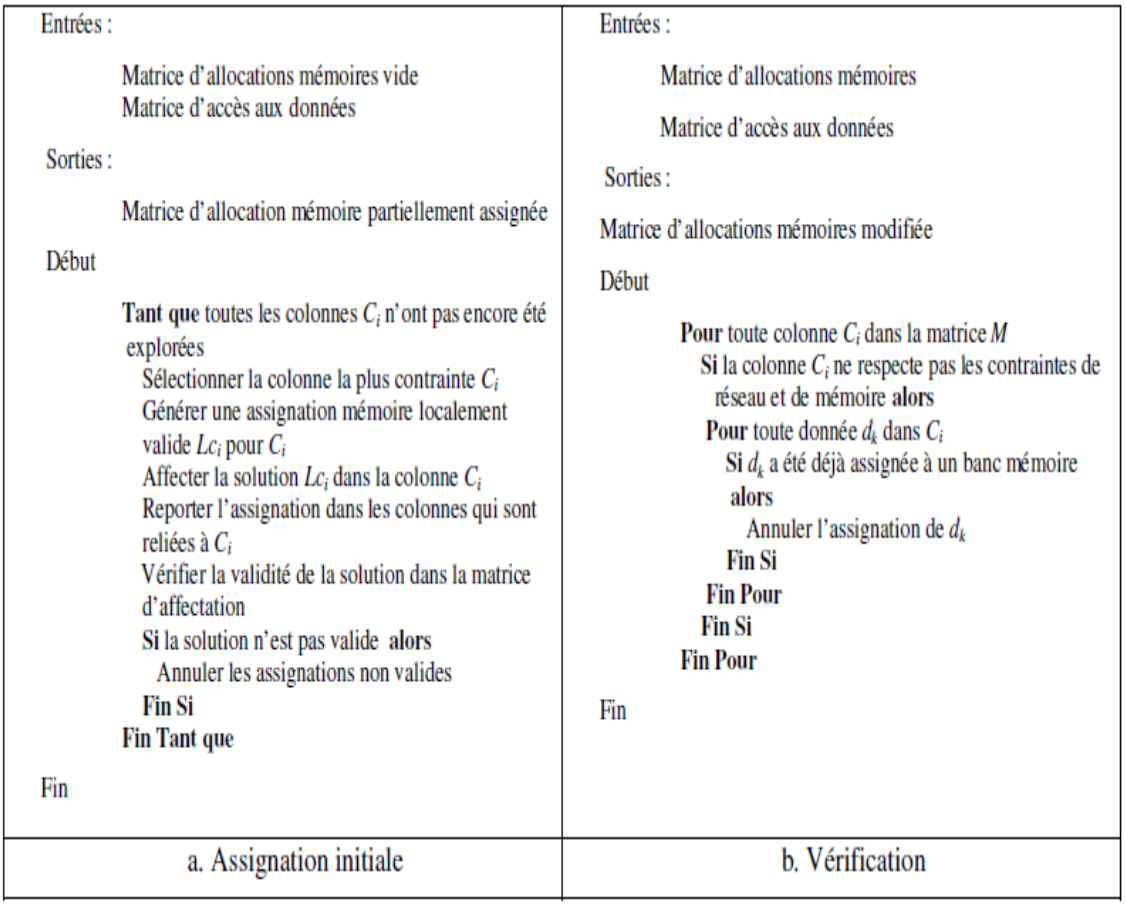


Figure II.22 : Algorithme de l'assignation initiale.

II.11.1.2 Résolution des conflits :

Cette deuxième étape part d'une matrice partiellement affectée, résultante de l'assignation initiale. À chaque itération, l'algorithme sélectionne lui aussi la colonne la plus contrainte,

ensuite pour chaque donnée il génère une solution d'assignation mémoire respectant les contraintes de réseau et de mémoire dans la matrice d'affectation. La génération d'une solution d'allocation mémoire valide pour une donnée dans la matrice doit respecter la propriété de consubstantialité :

Soient,

M une matrice d'affectation mémoire

C_i M une semi colonne de lecture non assignée

C_j M une semi colonne d'écriture non assignée

d_k une donnée non assignée et qui est accédée en lecture dans le cycle i et accédée en écriture dans le cycle j

Si l'ensemble de solutions localement valides pour d_k dans la colonne C_i S_j l'ensemble de solutions localement valides pour d_k dans la colonne C_j

Propriété Consubstantialité :

L'ensemble de solutions valides pour dk dans la matrice d'affectation M est : SM (dk) = {S_i _ S_j}

```
Entrées :
        Matrice d'allocation mémoire partiellement assignée
        Matrice d'accès aux données
Sorties :
        Matrice d'allocation mémoire totalement assignée
Début
        Tant qu'il existe des colonnes vides ou partiellement
         assignées
           Sélectionner la colonne la plus contrainte Ci
           Pour tout dk non assignées dans Ci
              Générer l'ensemble de solutions valides à dk
              Si l'ensemble de solutions est vide alors
                 Affecter un registre à dk
              Sinon
                Affecter une solution valide à dk
              Fin Si
              Reporter l'assignation dans la colonne concernée
           Fin Pour
        Fin Tant que
Fin
```

Figure II.23 : Algorithme de résolution de conflits.

Si certaines de ces données n'ont pas de solution mémoire valide (i.e. SM (d_k)= Ø) des registres supplémentaires sont mis en œuvre afin de stocker ces éléments conflictuels pour la relaxation de la contrainte mémoire. Il s'agit là d'un nouveau concept que nous introduisons dans le cadre de la conception d'entrelaceurs parallèles. Comme évoqué précédemment, le principe est d'accroitre l'espace d'exploration des solutions de placement mémoire. L'approche que nous proposons dans le cadre de cette thèse ne met pas en œuvre de rétroaction (i.e. recommencer l'ensemble du placement mémoire en utilisant le nouvel espace des solutions possibles).

II.12 Conclusion :

Nous avons en premier lieu présenté les concepts utilisés dans les codages et décodages des codes convolutifs et leur principe de fonctionnement et applications dans plusieurs standards de communication dans s'y trouve puis nous avons évoqué la concaténation parallèles de deux codes convolutifs séparés par des ENTRELACEURS – qui effectuent une séquence de permutation aléatoires pour des milliers de bits -présentés par des TURBO CODES ces derniers sont distingués par leur capacité de correction d'erreurs. Puis les codes en bloc qui font partie de la seconde classe de codes correcteurs d'erreurs qui sont utilisés pour transmettre des données numériques de façon fiable via des canaux de communication non fiables.

Le problème de conflit mémoires constitue la colonne vertébrale de cette étude et qui intervient si deux ou plusieurs processeurs accèdent simultanément à des données distinctes stockées dans un même banc mémoire et pour y remédier le code LDPC-Turbo Codes est introduit en ca d'accès parallèle aux données. Ces concepts sont utilisés pour expliquer les mécanismes à la base des Turbo-Codes (une sous classe des codes convolutifs) caractérisés par un taux d'erreurs binaires proche à la limite de Shannon. Nous avons également détaillé plusieurs techniques du décodage des Turbo-Codes pour des applications à haut débits. Puis, nous avons introduit une brève description des codes en blocs pour nous focaliser, sur le cas particulier des codes LDPC. Finalement, nous avons mis l'accent à travers des exemples sur les problèmes communs de conflits mémoires qui apparaissent lors de l'implémentation d'architectures parallèles pour ce type d'applications (Turbo-Code et LDPC).

[1] C.Berrou, A.Glavieux and P.Thitimajshima. « Near Shannon limit error-correcting coding and decoding : turbo-codes » in Proc. IEEE Int Conf. On communication (ICC93). Geneva.Switzerland, pp.1064-1070,1993.

[2] « Technical Specification Group Radio Access Network ; Evolved Universal Terrestrial Radio Access ; Multiplexing and Channel Coding (Release 8).» 3GPP Std. TS 36.212. Dec.2008.

[3] 3GPP, « Technical specification group radio access network ; multiplexing and channel coding (FDD) » (25.212 V5.9.0).

[4] ETSI EN 302-583 V1.1.1, (2008). « Digital Video Broadcasting (DVB) ; Framing structure, channel coding and modulation for satellite services to handheld devices (SH) below 3GHz ».March.2008.

[5] C. E. Shannon, A mathematical theory of communication, Bell System Tech.J.27 (July and October (1948) 397-423,623-656.

[6] L. R. Bahl, J. Cocke, F. Jelinek and J.L.R. Bahl, J. Cocke, F. Jelinek and J. Raviv, « Optimal decoding of linear codes for minimizing symbol error rate.» IEEE Trans.Inform.Theory, vol TT-20, no.2, pp.284-287, March.

[8] M.J.E. Golay, « Complementary series », IRE Trans. on Info. Theory, Vol. IT-7 , pp. 343-346.

[9] R. R. W.Hamming, « The Bell System Technical Journal.» Bell Syst. Tech. J, Vol. XXVI, no, 2, pp.147-160, Apr.1950.

[10] Shu Lin and Daniel J. Costello, Jr. « Error control Coding. » Pearson Education, Ine 2004.

[11] G. Masera, F. Quaglio and F. Vacca, « Implementation of a flexible LDPC decoder, » IEEE Trans. Circuits and Systems-II : Express Briefs, vol.54.no.6.pp.542-546.

[12] J.-Y. lee and H.-J. Ryn, «A 1-Gb/s flexible LDPC decoder supporting multiple code rates and block lengths," IEEE Trans. Consumer Electronics, vol. 54, no. 2, pp. 417–424, May 2008.

[13] C. Chavet, P. coussy, P. Urard and E. Martin, « A memory Mapping Approach for Parallel

Interleaver desing with multiples read and write accesses. » In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS) 2010.

[14] P. Keyngnaert, B. Demoen, B. De Sutter, B. De Sus, and K. De Bosschere. "Conflict Graph Based Allocation of Static Objects to Memory Banks" Informal proceedings of the First workshop on Semantic, Program Analysis, and Computing Environments, pages 131–142, September 2001.

[15] D. C.Kozen, « The Design and Analysis of Algorithms », Springer Verlag, USA, 1992.

[16] A. Tarable, S. Benedetto, and G. Montorsi, « Mapping interleaving laws to parallel turbo and LDPC decoder architectures", IEEE Trans.Inf.Theory, vol. 50, no.9, pp.2002-2009, Sep. 2004.

[17] A.Tarable and S. Benedetto, «Further results on mapping functions,» inProc. Inform.Th.Workshop 2005(ITW2005), pp.221-225.

[18] Jing-Ling, «Parallel Interleavers Through Optimized Memory Address Remapping »

IEEE Trans. VLSI Systems vol. 18, no.6, pp.978-987, June. 2010.

[19] C. Chavet, P. coussy, P. Urard and E Martin, « Static Adress Generation Easing : a Desing Methodology for Parallel Interleaver Architecture »In Proc.ICASSP 2010.

[20] Awais Sani, Philippe Coussy, Cyrille Chavet, Eric Martin « A methodology based on Transportation problem modeling for designing parallel interleaver architectures » ICASSP 2011, 1613-1616.

[21] F.Quaglio, F.Vacca, C.Castellano, A.Tarable, M.G.Asera. « Interconnection Framework for High- Throughput, Flexible LDPC Decoders.» Framework for High-Throughput, Flexible LDPC Decoders". In proceeding Design Automation and Test in Europe Conference and Exhibition, 2006.

[22] Awais Sani, Philippe Coussy, Cyrille Chavet, Eric Martin, "Design of parallel LDPC interleaver architecture: A bipartite edge coloring approach", ICECS 2010, 466-469.

[23] Awais Sani, Philippe Coussy, Cyrille Chavet, Eric Martin "An approach based on edge coloring of tripartite graph for designing parallel LDPC interleaver architecture" ISCAS 2011, 1720-1723.

Chapitre III :

Optimisation du contrôle

III.I Introduction :

Le placement de données en mémoire et l'optimisation de l'architecture résultante que nous proposons se base sur des modèles formels et des approches exploratoires que nous avons présentées dans le chapitre précédent. Ces modèles sont exploités par un flot de synthèse dédié à la génération d'architectures d'entrelacement de données (Réseau d'interconnexion, Mémoires et Unité de contrôle) de niveau transfert de registres (RTL, pour Register Transfer Level). Toutefois, afin de pouvoir optimiser le coût architectural du contrôleur, nous devons modifier quelque peu ces algorithmes, et proposer de nouveaux outils théoriques.

Le chemin de données de l'entrelaceur mémoire que nous ciblons (cf. Figure III.1) est composé de processeurs communiquant en parallèle avec des bancs mémoire via un réseau d'interconnexion défini par le concepteur, des éléments de mémorisation supplémentaires permettant de mémoriser transitoirement les données conflictuelles, de la logique d'aiguillage afin de pouvoir assurer le partage de ces éléments entre les différents processeurs.

Le pilotage des mémoires pourrait être réalisé soit avec une machine à états finis qui détermine, à chaque cycle d'horloge, à quelle adresse le processeur accède à la mémoire, soit en effectuant un prétraitement hors ligne qui consiste à mémoriser les mots de commandes dans des ROMs dédiées. Il est alors nécessaire d'avoir au minimum une ROM par banc mémoire afin de garantir un traitement parallèle des données. Réalisé soit par une machine à état fini ou bien par des ROMs, la complexité du contrôleur mémoire augmente considérablement avec la taille de la trame de données traitées par chaque processeur et avec le degré du parallélisme de l'entrelaceur. Afin de pouvoir réduire cette complexité nous proposons un contrôleur mémoire basée sur des ROMs et nous introduisons dans ce contexte, dans une deuxième partie de ce chapitre une approche permettant d'effectuer un placement intelligent des données dans la mémoire ciblant l'optimisation de la taille et le nombre des ROMs utilisées.

La partie contrôle peut être divisée en trois unités : une unité qui est dédiée pour le contrôle du Réseau, les mots de commandes de celle-ci sont stockées dans une ROM qui est elle-même pilotée par un compteur. Une deuxième unité permettant le contrôle des registres ainsi que les multiplexeurs qui leurs sont associés. Et enfin une troisième unité dédiée au contrôle des bancs mémoire (également basé sur un ensemble de ROMs). Les ROMs mémorisant les séquences de mot de commande sont optimisées afin d'en minimiser leurs tailles. Naturellement une logique d'aiguillage dédiée est alors nécessaire afin de distribuer les

signaux de contrôle en sortie des ROMs vers les RAMs. Chaque ROM d'adressage peut alors être contrôlée par son propre compteur permettant d'incrémenter l'adresse de lecture dans la ROM à chaque fois que celle-ci est parcourue.

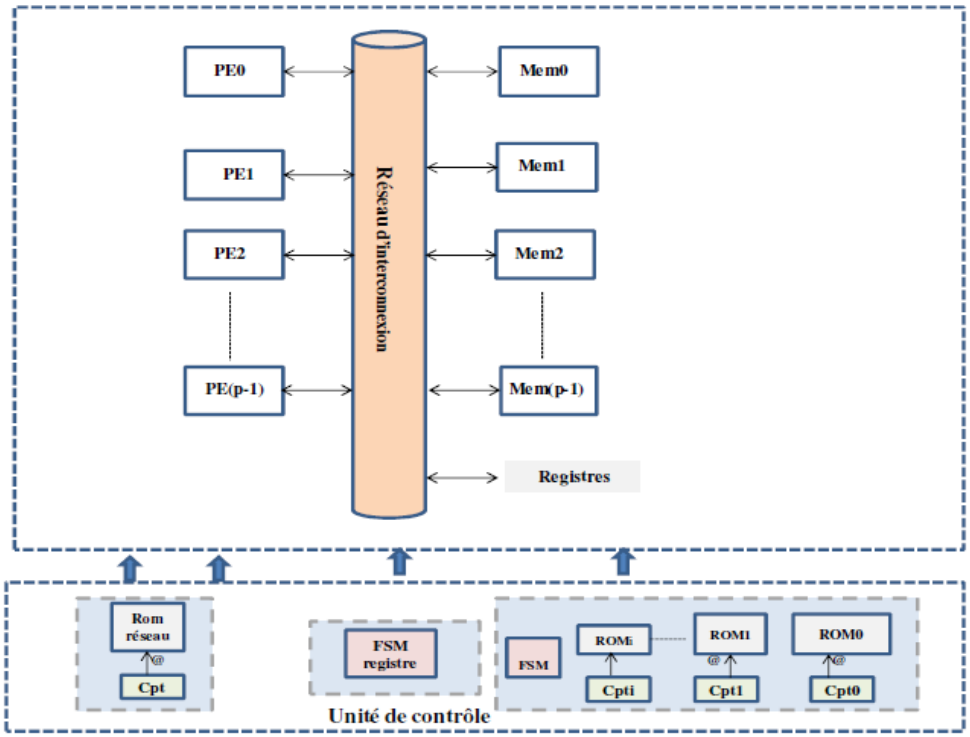


Figure III.1 : Architecture cible d'un entrelaceur mémoire parallèle.

C'est dans ce contexte que nous proposons l'automatisation de la génération et de l'optimisation de l'adressage mémoire. L'objectif est de pouvoir réduire la taille et le nombre des différentes ROMs utilisées. En effet, en essayant de placer les données dans la mémoire de manière intelligente, i.e. en permettant à plusieurs processeurs d'accéder dans le même cycle à la même adresse dans les différents bancs mémoire, il n'est plus nécessaire de mémoriser cette adresse dans toutes les ROMs. Dans ce cas il suffit qu'elle soit stockée dans une seule ROM et elle sera partagée par les différentes RAMs. Ainsi, la difficulté consiste à extraire une certaine régularité au niveau de placement des données dans les RAMs, autrement dit à maximiser l'occurrence d'une adresse mémoire qui sera mémorisée dans les différentes ROMs et lu par les RAMs dans un même cycle d'horloge. Plus cette régularité ne sera élevée, plus les tailles des ROMs seront réduites. Au final, nous aurons minimisé le nombre total de ROMs, celles-ci étant partagées par toutes ou partie des RAMs, et donc le coût du contrôleur associé.

Comme nous le verrons par la suite, l'optimisation de la surface du contrôleur mémoire peut également réduire d'une manière considérable le coût matériel de l'entrelaceur, cependant

dans la littérature, les concepteurs n'accordent leur attention qu'à l'optimisation du réseau d'interconnexion et au placement des données dans les bancs mémoires.

III.2 Placement des données en mémoire sans conflit pour la génération d'un contrôleur mémoire optimisé :

Compte tenu du modèle architectural visé, la problématique de l'optimisation du contrôle consistera dans un premier temps, à faire en sorte qu'à chaque cycle d'accès aux données, les adresses de lecture ou d'écriture soient les mêmes pour tous les bancs mémoires, dans la mesure du possible. Ainsi, il sera possible par la suite de fusionner les structures contenant les mots de commandes destinés à chaque banc mémoire, au sein d'une seule et même structure. Pour atteindre cet objectif, la première étape consistera à formaliser les accès aux adresses mémoire par un modèle de graphe spécifique.

III.2.1 Graphe de Conflit d'Adresse :

Dans le cadre de cette thèse, nous proposons un nouveau modèle de graphe de conflit destiné à la formalisation des conflits entre les accès aux adresses des données en mémoire : le Graphe de Conflit d'Adresse (ou ACG pour Address Conflict Graph). En premier lieu, nous introduisons certaines terminologies nécessaires pour la présentation du modèle formel.

A .Durée de vie d'une donnée :

Soient

$T_{écr}$ (d_i)= temps d'écriture de di par un processeur dans un banc mémoire B_i.

T_{lec} (di)= temps de lecture de di par un processeur à partir du banc mémoire B_i.

La durée de vie est alors définie par l'intervalle temporel

$\Delta_{di} = [t_{écr} (d_i), t_{lec} (d_i)]$

Figure III.2 : Durée de vie d'une donnée.

B. Chevauchement de durées de vie :

Soient :

Δ_{di} = durée de vie d'une donnée d_i.

Δ_{dj} = durée de vie d'une donnée d_j.

Deux données ont des durées de vie Δ_{di} et Δ_{dj} qui se chevauchent ssi,

$$\Delta_{di} \cap \Delta_{dj} \neq \emptyset$$

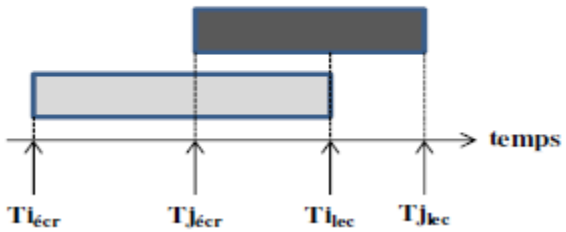


Figure III. 3 : Chevauchement de durée de vie de deux données.

III.2.1.1 Un Graphe de Conflit d'Adresse :

Un graphe de conflit d'adresse, ou ACG (pour Address Conflict Graph), est un graphe de Contraintes non orienté G (V, E) :

- L'ensemble des nœuds du graphe V, contenant les nœuds représentant les accès aux données.

- L'ensemble des arcs de compatibilité E = $\{(v_i, v_j)\}$ représente l'existence d'un conflit d'adresse entre les nœuds v_i et v_j (i.e. deux adresses différentes doivent être utilisée pour les nœuds v_i et v_j)

III.2.2 Exploitation d'un ACG pour l'optimisation de l'architecture de contrôle :

Comme nous venons de la voir, l'assignation de deux données dans un même banc mémoire peut créer un conflit d'adresse, selon les durées de vie de ces données dans ce banc mémoire. De ce fait, nous constatons que la manière avec laquelle les données sont assignées aux bancs mémoires a un impact sur la « complexité » du graphe de conflits d'adresses résultant. Le nombre d'arcs de conflits et le degré de chaque nœud appartenant au graphe sont des conséquences directes de l'assignation des données aux bancs mémoires (cf [01]).

III.2.2.1 Génération d'un Graphe de Conflits d'Adresses :

Pour obtenir un ACG, nous allons modifier l'algorithme d'assignation mémoire « donnée par donnée » proposé précédemment afin de définir un nouvel objectif que devront prendre en compte les étapes d'Assignation Initiale et de Résolution des Conflits. Ainsi, le placement des données en mémoire se fera en respectant les contraintes de placement mémoire et de réseau précédemment définies, mais également avec un objectif qui sera de minimiser la complexité du graphe ACG généré en parallèle. Cette minimisation se fera lors

de la sélection du banc mémoire à assigner aux données en élisant le banc qui créera le moins d'arcs de conflit dans le graphe ACG. Cette étape se fera par l'intégration d'une heuristique dédiée au cours de l'étape de placement mémoire. Le flot de conception que nous proposons (cf. Figure III.4 Flot de conception pour la génération de l'entrelaceur mémoire parallèle) permet à partir d'une règle d'entrelacement et d'une contrainte de réseau (i.e. le réseau que souhaite cibler le concepteur) de générer une architecture optimisée (VHDL de niveau RTL), d'un entrelaceur mémoire.

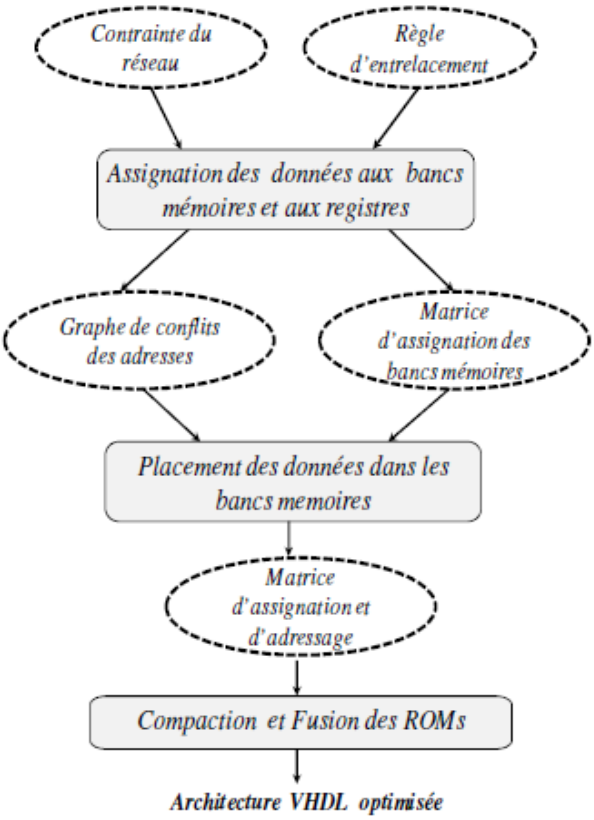


Figure III.4 : Flot de conception pour la génération de l'entrelaceur mémoire parallèle.

La première étape consiste à assigner les données aux bancs mémoires et aux registres, tout en générant le graphe ACG. Ainsi, après cette étape d'Assignation des données, nous obtenons (1) un graphe de conflit d'adresse et (2) une matrice d'assignation des données dans les bancs mémoires. Ces deux « modèles » serviront dans la seconde étape de notre flot à placer les données à leurs adresses dans les bancs mémoires.

Nota :

Dans la Figure III.4, les étapes Résolution de conflit et Relaxation mémoire de la figure II.20 ont été fusionnées dans le bloc Placement des données dans les bancs mémoires. En effet, ces deux étapes n'ont pas étés modifiées dans le cadre de la mise au point de notre nouvelle approche de placement mémoire avec optimisation de l'architecture de contrôle.

Enfin, à partir de la matrice d'assignation et d'adressage, une dernière étape d'optimisation permet de tirer partie de la régularité insérée dans les séquences d'adressage pour générer une unité de contrôle fortement optimisée. Nous détaillons par la suite l'implémentation des étapes de ce flot. Nous explorons les solutions que nous avons présentées.

III.2.2.2 Assignation Initiale :

Comme nous l'avons mentionné, l'approche de placement mémoire que nous proposons dans ce chapitre est basée sur l'approche d'allocation mémoire « donnée par donnée » Toutefois, plutôt que sélectionner la donnée la plus contrainte (i.e. la donnée ayant le minimum de solutions d'allocation mémoire valides), l'algorithme choisit le banc ayant le plus grand nombre de conflits d'adresses et lui affecte la donnée créant le minimum de conflits dans l'ACG.

Afin de minimiser les conflits des adresses dans chaque banc mémoire, à chaque itération l'algorithme traite en priorité le banc mémoire dont le graphe ACG lui correspondant contient le plus grand nombre de conflits d'adresses. Il lui associe l'arête de poids le plus faible, ce qui permet d'équilibrer les conflits entre les différents bancs mémoires.

Ensuite le graphe biparti est mis à jour en éliminant toutes les arêtes violant la validité de la solution en cour de construction. En effet, si une arête sélectionnée relie un banc bi et une donnée dj, la mise à jour consiste à éliminer d'une part toutes les autres arêtes issues de bi ou de di, ainsi que toutes les arêtes violant la contrainte de réseau. Ce processus est ainsi répété jusqu'au traitement de toutes les données accédées à l'instant courant t_k.

Etant donné un ensemble de données $\{d0,\ldots d_{p-1}\}$ accédées simultanément par P processeurs à l'instant courant t_k, nous associons une fonction de coût à chaque solution d'allocation mémoire Si affectant l'ensemble de données à l'ensemble de bancs mémoires $\{b0,\ldots..b_{p-1}\}$.

Si représente un banc mémoire possible. Cette fonction de coût exprime le nombre de conflits des adresses créés suite à cette assignation (i.e. le nombre d'arcs de conflit créé dans l'ACG correspondant au banc mémoire Si).

***Coût d'assignation d'une donnée à un banc mémoire :**

- Soient Δd_i et Δd_j deux intervalles de durées de vie associées, respectivement, aux données d_i et d_j.
- Soit le poids W_{ij} associé aux deux données di et dj, défini de la manière suivante

Si $\Delta di \cap \Delta dj \neq \emptyset$ alors $W_{ij}(d_i, d_j) = 1$

Sinon $W_{ij}(d_i, d_k) = 0$

Le c**oût d'assignation d'une donnée** d_i à un banc b_j, noté par C_i correspond au nombre d'arcs de conflits crées suite à l'affectation de la donnée *di* au banc mémoire b_j.

$$C_i = \sum_{d_k \in b_j} w_{ik}(d_i, d_k) \qquad \textbf{(1)}$$

Cout de la solution S_i

$$\text{Cout}(s_i) = \sum_{K=0}^{p-1} C_k \qquad \textbf{(2)}$$

A partir de l'ensemble des coûts d'assignation de chacun des bancs respectant les contraintes de placement mémoire, nous proposons une heuristique avec pour objectif de minimiser les conflits d'adresses par banc mémoire en se basant sur cette fonction de coût.

Pour ce faire, nous modélisons le problème par un graphe biparti G = (D, B, E) :

D est l'ensemble des P données à assigner dans le cycle courant t_k.

B est l'ensemble des bancs mémoires disponibles.

E est l'ensemble des arêtes pondérées.

La Figure III.5 illustre un exemple de graphe biparti (cf. [04]) pondéré avec $\{d_0,\ldots d_{p-1}\}$ qui représentent les nœuds des P données (pouvant être accédées en même temps au sein d'une Semi-colonne) et $\{b_0,\ldots b_{p-1}\}$ qui représentent les nœuds de bancs mémoire. Une arête existe entre un nœud de banc mémoire et un nœud de donnée, si le banc mémoire fait parti de l'ensemble de solutions d'allocation mémoire valides respectant les contraintes de placement mémoire et de réseau. Le poids associé à chaque arête représente le coût C_i défini précédemment. Notre heuristique cherche une solution S valide minimisant la fonction de coût(S_i).

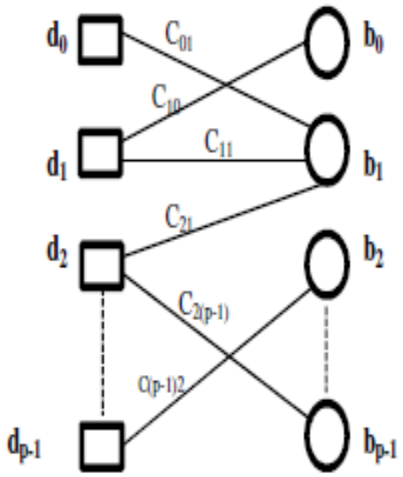


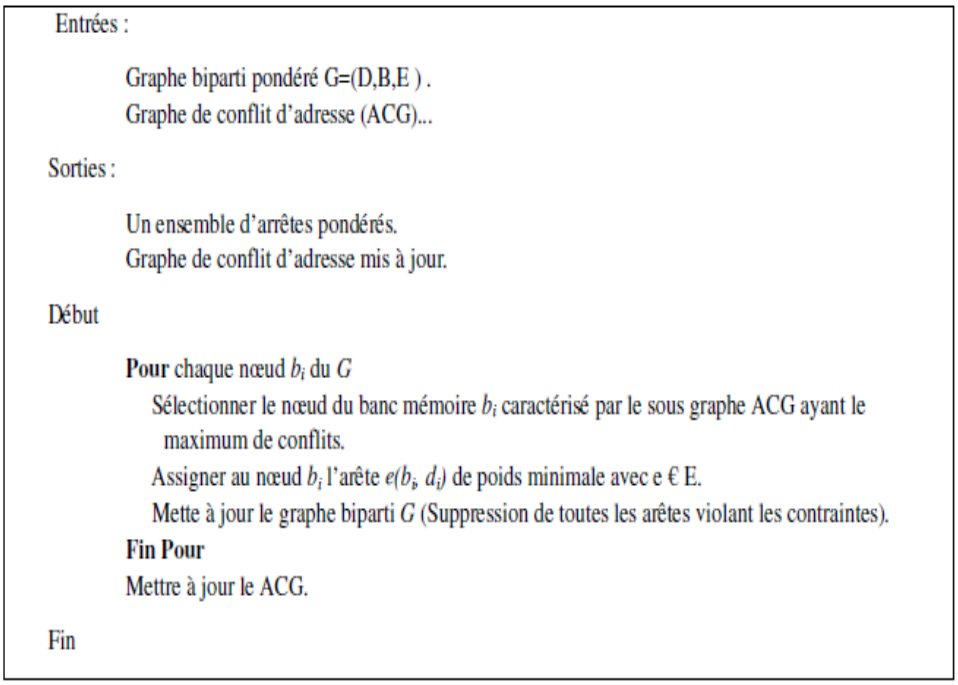
Figure III. 5 : Modélisation du problème d'allocation mémoire dans un cycle particulier par un graphe biparti.

Nota :

Une solution S est valide si elle respecte les contraintes de placement mémoire et de réseau imposées par le concepteur.

$$\text{Cout(s)} = \min(\text{Cout}(si)) = \min \sum_{K=0}^{P} C_k \qquad \textbf{(3)}$$

Cette heuristique est appliquée pour chaque ensemble de données non assignées et accédées simultanément par les différents processeurs soit en écriture, soit en lecture. Dans la phase de vérification les affectations qui ne respectent pas les contraintes dans la matrice sont supprimées et les données en question ne peuvent plus être assignées aux bancs mémoires durant cette étape. Par la suite e ACG est mis à jour.


Entrées :

Graphe biparti pondéré G=(D,B,E) .
Graphe de conflit d'adresse (ACG)...

Sorties :

Un ensemble d'arrêtes pondérés.
Graphe de conflit d'adresse mis à jour.

Début

Pour chaque nœud b_i du G
Sélectionner le nœud du banc mémoire b_i caractérisé par le sous graphe ACG ayant le maximum de conflits.
Assigner au nœud b_i l'arête $e(b_i, d_i)$ de poids minimale avec e € E.
Mette à jour le graphe biparti G (Suppression de toutes les arêtes violant les contraintes).
Fin Pour
Mettre à jour le ACG.

Fin

Figure III.6 : Algorithme d'assignation initiale tenant compte de la génération d'ACG.

III.2.2.3 Phase de Résolution de Conflits :

Durant la phase de résolution des conflits, le modèle est légèrement différent. En effet, une solution d'allocation mémoire valide n'exige pas nécessairement d'avoir assigné tous les bancs mémoires ou bien toutes les données puisque l'algorithme s'autorise à mémoriser certaines données dans des registres. Le principe de notre heuristique reste donc le même que ce que nous venons de décrire, à la différence près qu'une fois tous les bancs mémoire explorés, et en cas de présence de données n'ayant pas de solution d'allocation mémoire valide (i.e. équivalent à des nœuds de données non reliées à des arêtes) des registres seront mis en œuvre afin de mémoriser les données non assignées à des bancs mémoire.

Entrées :

Graphe biparti pondéré G= (D, B, E).
Graphe de conflit des adresses (ACG).

Sorties :

Un ensemble d'arrêtes pondérées et un ensemble de données assignées à des registres.
Graphe de conflit d'adresse mis à jour.

Début

Tant que tous les nœuds de bancs mémoires n'ont pas encore étés explorées
Sélectionner le nœud de banc b_i ayant le maximum de conflits et relié au moins à une arête.
Assigner au nœud b_i l'arête $e(d_k, b_i)$ de poids minimal.
Mette à jour le graphe G (Suppression de toutes les arêtes violant les contraintes).
Fin Tant que
Mettre à jour le graphe de conflit des adresses ACG.
Affecter des registres aux restes des données non assignées à des bancs mémoires.

Fin

Figure III.7 : Résolution des conflits dans un cycle particulier.

III.2.2.4 Adressage des données au sein des bancs mémoire :

Nous introduisons dans cette partie une approche automatisant le placement des données dans la mémoire (i.e. à quelle adresse mémoire sera placée la donnée) à partir d'un graphe de conflit d'adresse et d'une matrice d'assignation mémoire.

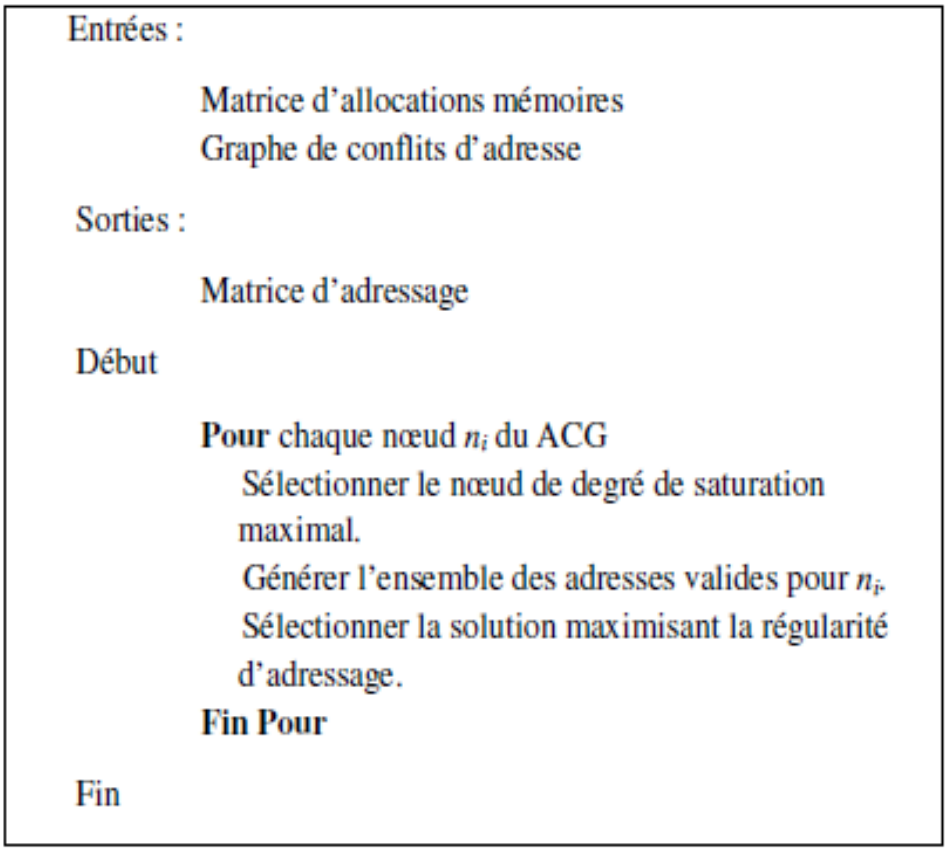


Figure III.8 : Algorithme d'adressage.

Pour ce faire, nous proposons une approche exploitant le graphe de conflits d'adresse et la matrice d'assignation mémoire. Il est important de noter que l'étape de placement mémoire décrite précédemment avec pour objectif la minimisation de la complexité des graphes ACG de chaque banc mémoire, mais ne vise pas en elle-même à minimiser la taille des ROMs. L'idée est plutôt d'offrir un espace de solution plus large pour l'étape d'adressage des données dans leur banc que nous allons maintenant décrire.

A chaque itération, l'algorithme sélectionne une donnée non assignée à une adresse mémoire, génère l'ensemble de solutions possibles pour cette donnée et enfin choisit la solution maximisant la fonction de coût. Le nœud sélectionné est le nœud le plus contraint dans le graphe de conflit d'adresse, on parle ici de degré de saturation d'un nœud.

Soient :

- ACG (V,E) un graphe de conflit d'adresse
- n_i un nœud du graphe de conflit

***Degré de saturation :**

n_i = Nombre des nœuds qui ont déjà été assignés à leurs adresses mémoire et qui sont connectés à ni dans ACG (V,E) + Degré de ni dans ACG(V,E).

Une solution d'adressage valide est une solution ne violant pas les contraintes imposées par le graphe de conflit d'adresse : deux nœuds reliés par un arc dans l'ACG doivent être affectés à deux adresses mémoire différentes. Nous associons une fonction de coût à chaque solution d'adressage possible afin de favoriser d'une façon heuristique la régularité au niveau des séquences d'adressage mémoire.

Plus concrètement, la fonction de coût associé à une solution d'assignation d'une adresse $@_j k$ (adresse k dans le banc mémoire j) à une donnée ni est composée par deux termes :

- le premier «NbrOcc» exprime le nombre de données déjà assignées à l'adresse $@_f k$ (avec $f \in [b0..bP-1]$ et $f \neq j$) et qui sont accédées simultanément avec la donnée ni (voir 4.a).

Puisque la donnée est écrite et puis lue dans deux cycles d'horloge différents, le terme «NbrOcc» correspond aux accès à l'adresse $@_f k$ des différents bancs mémoires effectués dans ces deux cycles d'horloges (voir 4.c).

Soient,

- n_i est un nœud de donnée dans l'ACG.
- $@_j k$ est une adresse mémoire valide pour la donnée n_i.
- $@_f k$ est une adresse mémoire avec $f \in [b_0...b_{P-1}]$ et $f \neq j$
- T_accès (n_i) correspond aux temps d'accès à la donnée n_i.
- T_accès_Conf (n_i) correspond aux temps des accès à toutes les données non assignées à des adresses mémoires et qui sont conflictuelles à n_i.
- T_lecture ($n_{i)}$ correspond au temps de lecture de n_i.
- T_écriture (n_i) correspond au temps d'écriture de n_i.
- T_écriture_Conf (n_i) correspond aux temps d'écriture des données conflictuelles à n_i
- T_lecture_Conf (n_i) correspond aux temps de lecture des données conflictuelles à n_i.
- NbrOcc ($@_f k$, T_accès (n_i)) correspond au nombre des accès à l'adresse mémoire $@_f k$ effectué dans les cycles T_accès (n_i).
- MoyOccConf ($@_f k$, T_accès_Conf (n_i)) correspond à la moyenne des accès à l'adresse mémoire $@_f k$ effectué dans les cycles T_accès_Conf (n_i).

- MoyOccConf($@_f$k,T_écriture_Conf(n_i)) correspond à la moyenne des accès à l'adresse mémoire $@_f$k effectué dans les cycles T_lecture_Conf (n_i).

- MoyOccConf($@_f$k,T_écriture_Conf(n_i)) correspond à la moyenne des accès à l'adresse mémoire @fk effectué dans les cycles T_écriture_Conf (n_i).

$W(n_i,@_j k) = NbrOcc (@_f k, T\_accès (n_i)) – MoyOccConf (@_f k, T\_accès\_Conf (n_i))$ **(4 .a)**

$MoyOccConf (@_f k, T\_accès\_Conf (n_i))= MoyOccConf(@_f k,T\_écriture\_Conf(n_i))$ **(4.b)**

+

$MoyOccConf(@_f k,T\_lecture\_Conf(n_i))$

$NbrOcc(@_f k,T\_accès(n_i))= NbrOcc (@_f k,T\_écriture (n_i)) + NbrOcc(@_f k ,T\_lecture (n_i))$ **(4.c)**

III.3 Compaction des ROMs d'adressages optimisées :

III.3.1 Transformation des adresses mémoires en matrice de banc et placement des adresses mémoires dans des ROMs dédiées :

Pour simplifier nos explications concernant l'optimisation de l'architecture de contrôle à base de ROM, nous réorganisons les séquences d'adressage par banc mémoire et non plus par processeur (cf. Figure III.9). Par exemple la première ligne correspond aux séquences d'adressage du banc mémoire A.

Nota :

Lorsqu'un banc mémoire n'est pas accédé (i.e. un processeur va accéder à un registre à la place) alors au cycle correspondant aucune adresse n'est fournie à ce banc mémoire (e.g. accès en écriture au banc mémoire A au cycle 3.). Ainsi dans les séquences d'adressage que nous construisons (cf. figure III.10) feront apparaître une « case vide ».

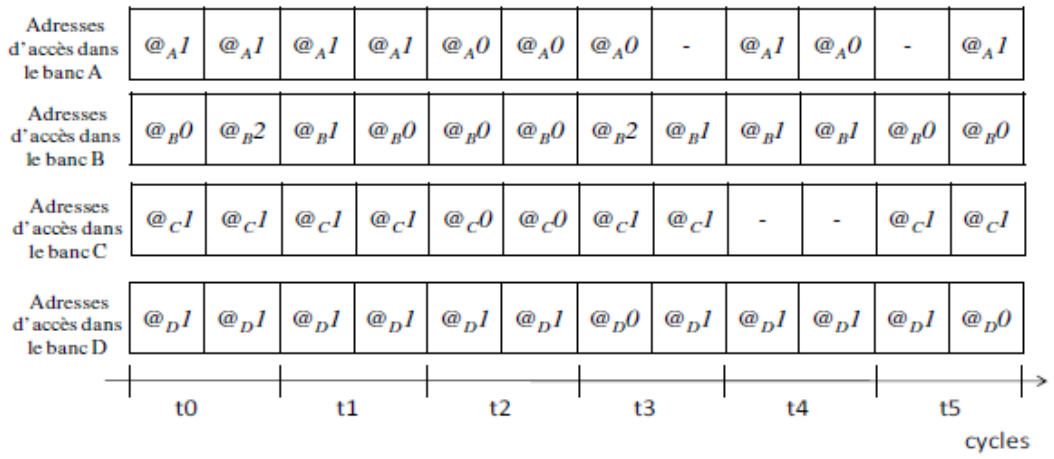


Figure III.9 : Séquences d'adressage aux bancs mémoires.

Afin de contrôler les différents bancs mémoire, ces séquences d'adressages sont stockées dans des ROMs ; par exemple la ROM contrôlant le banc mémoire A mémorise à son adresse 0 l'adresse 1 d'accès à la $RAM_A(@_A1)$ (cf. Figure III.9). La présence de cases vides dans la Semi-colonne d'écriture du cycle 3 et la Semi-colonne de lecture du cycle 5 montrent que durant ces périodes la RAM_A n'est accédée par aucun processeur. Nous verrons ultérieurement que ces cases vides seront exploitées pour favoriser la compaction de ces ROMs d'adressage.

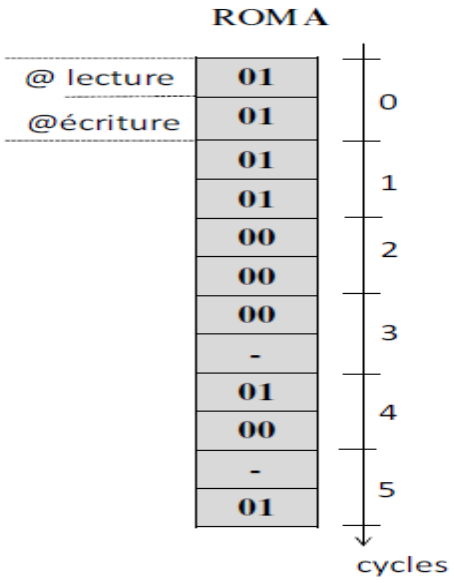


Figure III.10 ROM d'adressage pour le contrôle du banc mémoire A.

Nous avons choisit de stocker dans ces ROMs d'adressages uniquement les adresses mémoires sans leur associer les signaux de contrôles (write enable). En effet, puisque dans notre approche certains accès mémoires sont déportés des bancs mémoires vers les registres additionnels, les signaux d'écriture des bancs mémoire (RAMs) correspondants ne seront dans ce cas pas activés : le signal « write enable » de ces RAMs sera forcé à '0' dans certains cycles d'écritures (à priori un cycle sur deux) quand ces RAMs ne sont accédées par aucun processeur. Notre objectif est ainsi de favoriser la régularité des séquences d'adressage mémorisées dans les ROMs. Il s'agit là d'une première solution que nous souhaitons évaluer, mais l'algorithme de compaction et de fusion que nous proposons est capable à partir d'un niveau de régularité même minimal d'optimiser la taille des ROMs.

III.3.2 Compaction et Fusion des ROMs :

Suite à l'analyse que nous venons de présenter, nous savons que 4 ROMs sont nécessaires, à priori, pour contrôler les accès aux mémoires RAMs de l'architecture. Pour ce faire, nous proposons un flot dédié à la réduction de la taille et du nombre de ces ROMs (cf. Figure III.11). Ce flot s'insère à la suite du flot proposé section II.

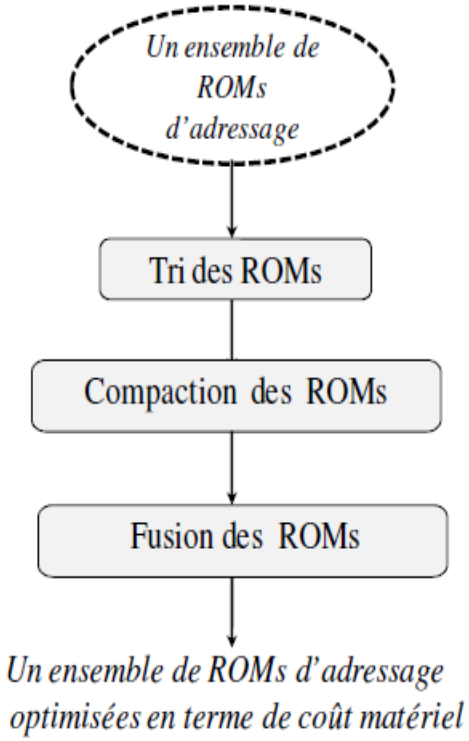


Figure III.11 : Flot de conception de la Compaction&Fusion des ROMs.

A) Tri des ROMs :

Nous définissons tout d'abord des terminologies qui seront utilisées pour trier les ROMs d'adressage.

***Degré de Compatibilité entre deux ROMs :**

Le degré de compatibilité de deux ROMs i et j est évalué de la manière suivante :

$$\text{Degré de Compatibilité } (ROM_i , ROM_j) = \sum_{t\in[0,T-1]} R_{ij}\ (t, ROM_i, ROM_j)$$

Avec,

T représentant le nombre total de cycles de traitements durant lesquelles les processeurs traitent les données.

R_{ij} est une métrique dont la valeur dépend des adresses stockées dans ROM_i et ROM_j. Plus concrètement, cette métrique exprime la compatibilité des adresses stockées dans ROM_i avec les adresses stockées dans ROM_j et qui sont lues respectivement par RAM_i et RAM_j dans le même cycle de traitement t.

Nota :

Nous supposons que durant un cycle de calcul t, les processeurs effectuent en parallèles deux accès : le premier est pendant [0 .. t/2] pour lire une valeur et le deuxième est pendant [t/2 .. t] pour écrire un résultat.

***Degré de compatibilité entre une ROM et un ensemble de ROMs :**

$$\text{Degré Compatibilité } (ROM_i, sROM) = \sum_{k=0\ ;k\neq i}^{k=p} \text{compatibilité } (ROM_i, ROMj)$$

Avec $sROM = \{ROM_0,\ldots ROM_{p-1}\}$.

Entrées :

Deux ROMs: ROM_i et ROM_j contrôlant respectivement deux RAMs: RAM_i et RAM_j.
Le cycle de traitement t.

Sorties :

Valeur de R_{ij} : La compatibilité des adresses stockées respectivement dans ROM_i et ROM_j et lus par RAM_i et RAM_j dans le cycle t.

Début

Si RAM_i n'est accédée en lecture par aucun processeur dans le cycle t (présence d'un registre) **alors**
$R_{ij}[0 .. t/2]=1$
Sinon
Si RAM_i et RAM_j sont accédées en lecture dans la même adresse mémoire **alors**
$R_{ij}[0 .. t/2]=1$
Sinon
$R_{ij}[0 .. t/2]=0$
Fin Si
Fin Si
Si RAM_i ou RAM_j n'est accédée en écriture par aucun processeur dans le cycle t (présence d'un registre) **alors**
$R_{ij}[t/2 .. t] = 1$
Sinon
Si RAM_i et RAM_j sont accédées en écriture dans la même adresse mémoire **alors**
$R_{ij}[t/2 .. t] = 1$
Sinon
$R_{ij}[t/2 .. t] = 0$
Fin Si
Fin Si

Fin

Figure III.12: Compatibilité des adresses mémoires envoyées par deux ROMs dans un cycle.

De traitement particulier, A partir de ces définitions, la prochaine étape va donc consister à trier les ROMs selon leurs degrés de compatibilité entre elles (leur similitude les unes par rapport aux autres). Nous considérons comme ROM principale celle qui a une compatibilité maximale avec le reste des ROMs.

Entrées :

Un ensemble de ROMs $sROM=\{ROM_0,.....ROM_{p-1}\}$ non trié.

Sorties :

sROM trié.

Début

Pour toutes $ROM_i \in sROM$

Calculer le degré de compatibilité de ROM_i avec le reste des ROMs dans *sROM*

Fin Pour

Trier les différentes ROMs suivant l'ordre décroissant de leurs degrés de compatibilité avec *sROMs*.

Fin

Figure III.13 : Tri des ROMs.

b) Compaction des ROMs :

L'identification de la ROM principale va nous permettre d'éliminer toutes les séquences d'adresses identiques, ordonnancées au même cycle (i.e. à la même adresse dans la ROM), répétées entre les ROMs. Au final, pour chaque séquence d'adressage répétée dans plusieurs ROMs aux mêmes cycles, il n'y aura plus qu'une seule ROM la mémorisant.

L'opération de tri expliqué précédemment permet de définir une certaine priorité dans ce classement, dans le sens où si une même séquence d'adressage est stockée dans deux ROMs et envoyée dans le même cycle d'horloge, alors c'est la ROM de rang inférieur qui mémorisera la séquence de contrôle. C'est pourquoi la ROM caractérisée par le degré de compatibilité le plus élevée par rapport aux autres est considérée comme la ROM principale : c'est elle qui pilotera le plus grand nombre de RAMs. Par conséquent, cette ROM ne subira aucune modification à ce stade de l'algorithme de compaction.

L'optimisation sera ainsi appliquée aux ROMs secondaires en se basant sur le principe suivant: Si un ensemble de ROMs envoient la même séquence de contrôle aux RAMs qui leur sont associées, dans le même cycle d'horloge, alors c'est la ROM possédant le degré de compatibilité le plus élevé qui stockera la séquence de contrôle, et c'est elle qui contrôlera les différentes RAMs concernées. Par voie de conséquence, les séquences de contrôle en question seront supprimées des autres ROMs, réduisant potentiellement leur taille. La Figure III.14 détaille l'algorithme d'élimination des séquences de contrôle répétées.

```
Entrées :
        Un ensemble de ROMs trié: sROM={ROM0,……..ROMp-1} avec
    rang(ROMi)<rang(ROMj) si i<j
Sorties :
        sROM optimisé en termes de taille
Début
        Pour tout cycle t €[0, T-1]
          Pour tous i allants de 1 jusqu'à p-1
            Pour tout j allant de 0 jusqu'à i-1
              Si il existe une adresse de lecture envoyée par ROMi dans le cycle t alors
                Si l'adresse de lecture envoyée par ROMi est la même que l'adresse de lecture
                envoyée par ROMj alors
                  Supprimer l'adresse stockée dans ROMi
                Fin Si
              Fin Si
              Si il existe une adresse d'écriture envoyée par ROMi dans le cycle t.
                Si l'adresse d'écriture envoyée par ROMi est la même adresse d'écriture
                envoyée par ROMj alors
                  Supprimer l'adresse stockée dans ROMi
                Fin Si
              Fin Si
            Fin Pour
          Fin Pour
        Fin Pour
Fin
```

Figure III.14 : Algorithme de compaction des ROMs.

Nous appliquons ainsi l'algorithme de tri et de suppression de séquences contrôle répétées sur notre exemple. Le calcul des degrés de compatibilités des différentes ROM donnent les résultats suivants :

Soit : sROM= {ROMA, ROMB, ROMC, ROMD},

Degré Compatibilité (ROMA, sROM)=22

Degré Compatibilité (ROMB, sROM)=12

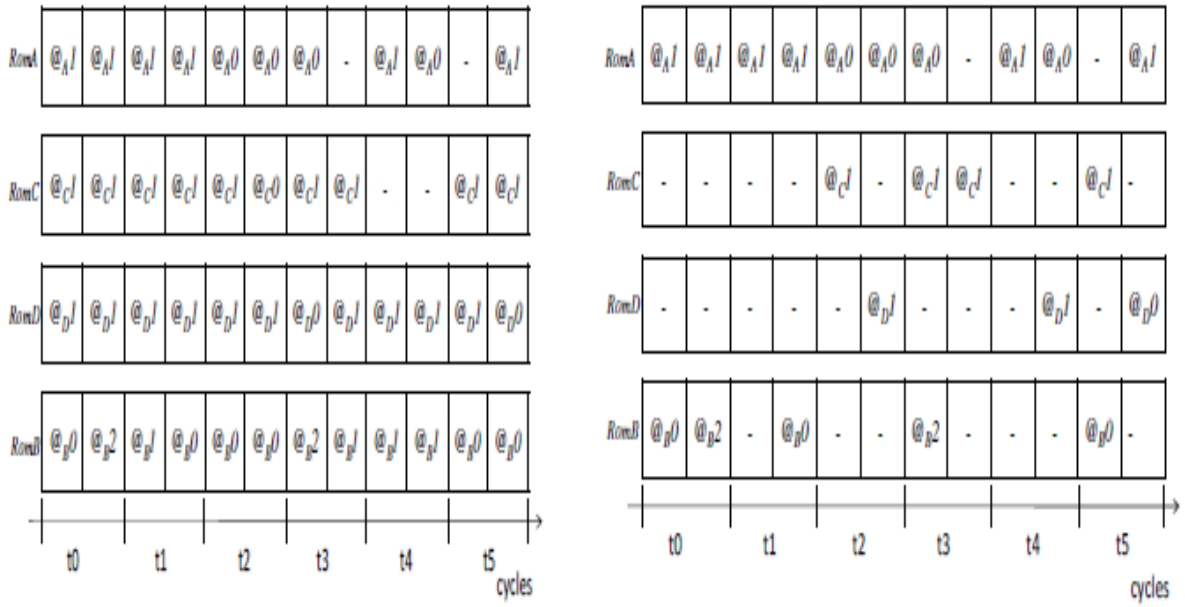
Degré Compatibilité (ROMC, sROM)=22

Degré Compatibilité (ROMD, sROM)=17

On obtient :

sROM_trié= {ROMA, ROMC, ROMD, ROMB}

Dans ce cas de figure, ROMA est la ROM principale.



(a) ROMs avant optimisation. (b) ROMs après optimisation.

Figure III.15 : Application de la compactions des ROMs sur l'exemple pédagogique.

La Figure III.15. b illustre le résultat de la compaction des ROMs. On peut ainsi observer que la ROMA est considérée comme la ROM principale. C'est cette ROM qui ici stock le plus grand nombre de séquences d'adressage. Plusieurs séquences d'adressage ne sont donc plus nécessaires dans les autres ROMs. Ainsi, grâce à la maximisation de la régularité des séquences d'adressage on peut minimiser la taille des ROMs.

c) Fusion des ROMs :

L'élimination des séquences de contrôles répétées crée des « cases vides » dans les différentes ROMs. Ces cases mémoires peuvent elle-même être utilisées pour compacter encore l'ensemble des ROMs de contrôle. À titre d'exemple, une ROM peut être éliminée si il est possible de déplacer toutes les adresses mémoires quelle stocke vers des ROMs de rangs inférieurs. Par exemple, l'ensemble des adresses contenues dans ROMD peut être regroupé dans le banc ROMC.

L'objectif par cette étape est de garder au finale un minimum de ROMs pour contrôler toutes les RAMs. Pour ce faire, nous proposons un algorithme dédié à la fusion des ROMs (cf. Figure III.16).

Celui ci commence par déplacer les adresses de la ROM classé deuxième vers la ROM principale, ensuite il traite à chaque fois la ROM de rang juste supérieur et commence en priorité par vérifier s'il est possible de déplacer ses adresses vers la ROM principale si ce

n'est pas le cas il vérifie la possibilité dans les ROMs secondaires et ainsi de suite jusqu'à vérifier la possibilité de faire un déplacement des adresses mémoires vers toutes les ROMs de rangs inférieurs (cet algorithme s'inspire d'un Left-Edge).

Entrées :

Un ensemble de ROMs triée et compactée $sROM = \{ROM_0 ..... ROM_{p-1}\}$, avec $rang(ROM_i) < rang(ROM_j)$ si i<j

Sorties :

Un ensemble de ROMs fusionnée

Début

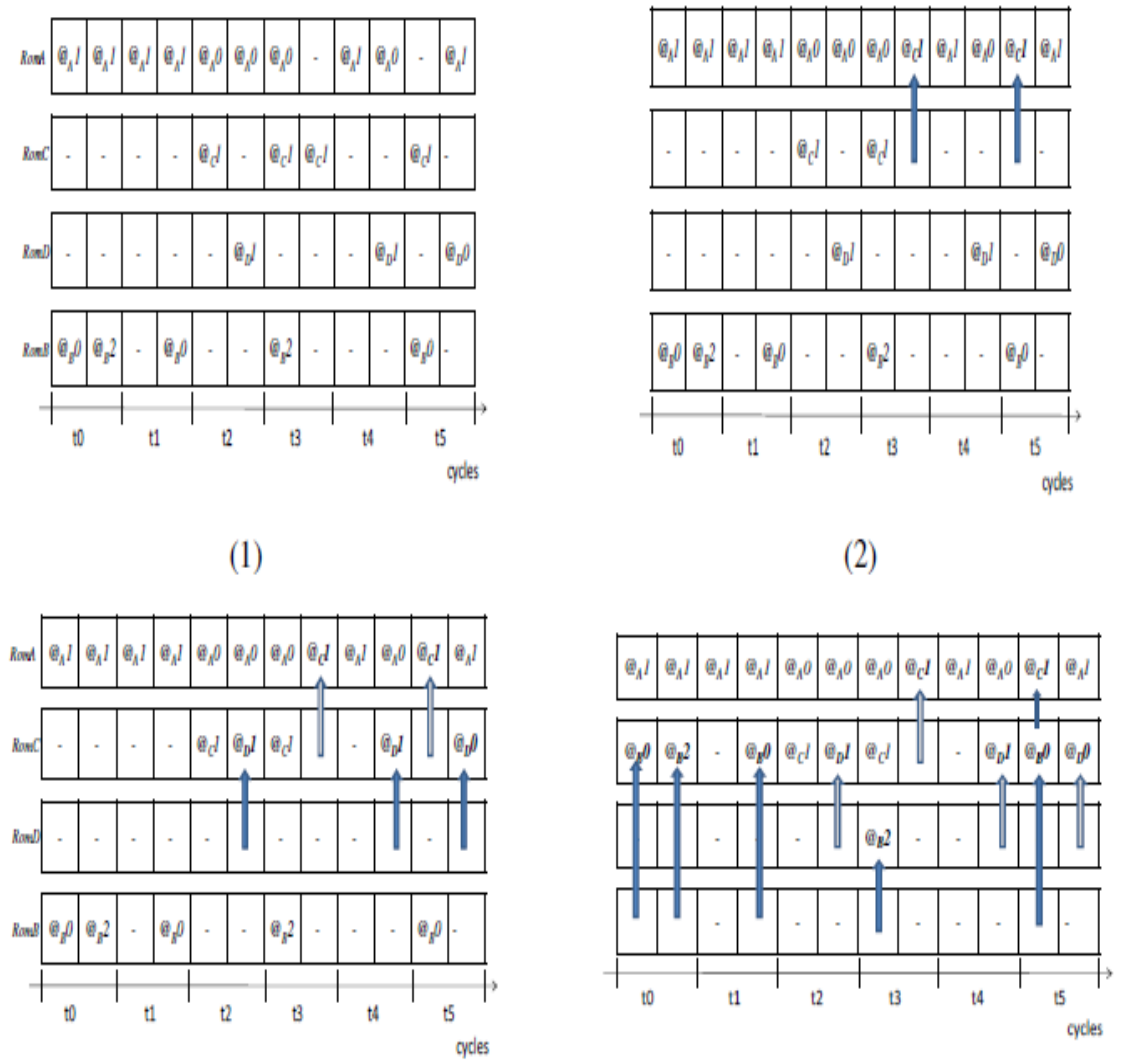
Pour tout *i* allant de 1 jusqu'à *p-1*
Pour tout *k* allant de 0 jusqu'à *i-1*
Pour tout $t \in [0, T-1]$
Si aucune adresse de lecture n'est lu à partir de la ROM_k et il existe une adresse de lecture $@i_t$ lu à partir de la ROM_i dans le cycle *t* **alors**
Déplacer l'adresse de lecture $@i_t$ de la ROM_i vers la ROM_k
Fin Si
Si aucune adresse d'écriture n'est lu à partir de la ROM_k et il existe une adresse d'écriture $@i_t$ lu à partir de la ROM_i dans le cycle *t* **alors**
Déplacer l'adresse d'écriture $@i_t$ de la ROM_i vers la ROM_k.
Fin Si
Fin Pour
Fin Pour
Fin Pour

Fin

Figure III.16 : Algorithme de fusion ROMs.

La Figure III.17 illustre les différentes étapes de l'algorithme de fusion des ROMs appliquées à notre exemple. Dans (2) deux séquences d'adresses initialement stockées dans la ROM_C ont été déplacées vers la ROM principale grâce à la présence dans celle-ci de deux cases vides. Dans le (3) trois séquences d'adresses ont été déplacées de la ROM_D vers la ROM_C puisque la ROM principale est complètement remplie. Finalement dans (4), après avoirs libéré plusieurs cases dans ROM_D et ROM_C, toutes les séquences d'adresses mémorisées dans la ROM_B ont été déplacées vers la ROM_C et la ROM_D.

Nous avons au final compacté les 4 ROMs initiales en 3 ROMs dont une principale de taille maximale et deux autres secondaires de tailles inférieurs.



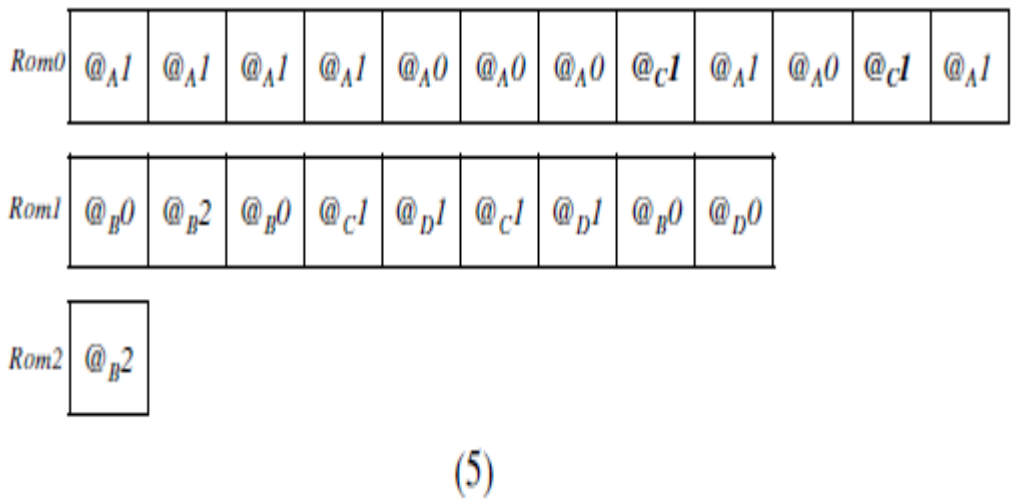


Figure III.17 : Application de l'algorithme de fusion des ROMs sur l'exemple.

III.4 Architecture RTL schématique :

La Figure III.18 illustre l'architecture RTL schématique générée dans le cadre de notre exemple. Elle se compose de 4 bancs mémoires (Mem_0, Mem_1, Mem_2, Mem_3), 4 processeurs (PE_0, PE_1, PE_2, PE_3) et de deux réseaux papillons pour la communication entre les processeurs et les bancs mémoires (accès en écriture et en lecture). Un registre R_0 est également présent permettant de mémoriser transitoirement les données conflictuelles.

L'unité de contrôle est divisée en trois unités, une contrôlant la mémoire, une autre est dédiée au contrôle du registre et finalement une qui pilote les deux réseaux. Le contrôleur mémoire est composé d'une ROM principale qui pilote toutes les mémoires et de deux ROMs secondaires de tailles plus petites pilotant chacune un sous-ensemble de mémoires. Chaque ROM est pilotée par son propre compteur qui s'incrémente si celle ci est utilisée pour piloter au moins une mémoire. Un automate (FSM, pour Finité State Machine) est ainsi nécessaire pour piloter la lecture des ROMs. Les signaux de contrôles pilotant les deux réseaux sont stockés dans une ROM dédiée. Enfin, un second automate se charge du contrôle du registre et du multiplexeur associé.

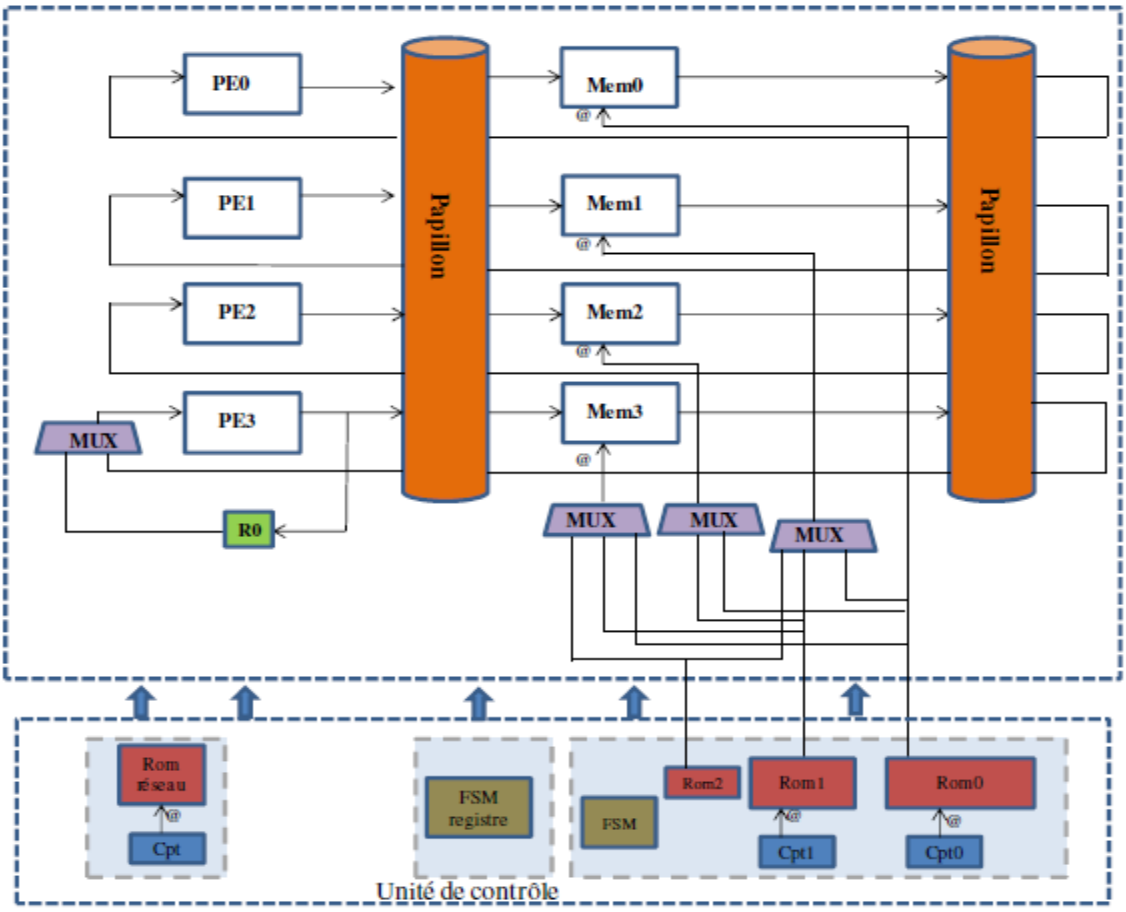


Figure III.18 : Architecture RTL générée.

L'approche que nous proposons permet d'automatiser l'adressage des données dans leur banc mémoire tout en garantissant des accès aux bancs mémoires sans conflit et en respectant un réseau d'interconnexion cible. L'objectif de cette approche est de maximiser la régularité au niveau de des séquences d'adressage en se basant sur une fonction de coût.

III.5 conclusion :

Dans ce chapitre nous avons présenté le flot de conception permettant la génération automatique des entrelaceurs mémoires parallèles sans conflits, sous contrainte de réseau et optimisés en terme de coût du contrôleur de l'architecture. Nous avons proposé également une stratégie permettant d'optimiser l'architecture du contrôleur via compaction et fusion des ROMs.

Ainsi que nous l'avons déjà évoqué, l'approche de placement mémoire basée sur la relaxation de la contrainte mémoire, combinée à l'optimisation à la volée du coût du contrôleur permet d'obtenir des gains significatifs en terme de coût architectural. Toutefois, si le concepteur souhaite obtenir une architecture basée sur un réseau d'interconnexion trop éloigné de ce que peut supporter nativement la règle d'entrelacement ciblée, le surcoût apporté par l'ajout de registres (et la logique d'aiguillage associée) peut dans certain cas faire perdre tout intérêt à l'approche.

[1] A. Briki, C. Chavet and P. Coussy, "A Memory Mapping Approach for Network and Controller Optimization in Parallel Interleaver Architectures", In Proceedings of the 23th ACM Great Lakes Symposium on VLSI (GLSVLSI) 2013, page XX-YY, Paris, France, may 2013.

[2] Awais Sani, Philippe Coussy, Cyrille Chavet, Eric Martin, "Design of parallel LDPC interleaver architecture: A bipartite edge coloring approach", ICECS 2010, 466-469.

[3] A. Hashimoto et J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in Proceedings of Design Automation Workshop, pp. 155-169, 1971.mur00

Conclusion générale

Les Turbo-Codes et les codes LDPC sont au cœur des normes de télécommunication actuelles grâce à leurs excellentes capacités de correction d'erreurs. La complexité croissante des algorithmes mis en œuvre et l'augmentation continue des volumes de données et des débits applicatifs requièrent souvent la conception de circuits intégrés dédiés (ASIC). Aujourd'hui, la complexité et le coût de ces systèmes sont très élevés; les concepteurs doivent pourtant parvenir à minimiser la consommation et la surface total du circuit, tout en garantissant les performances temporelles requises. Pour répondre à ces défis, les concepteurs jouent notamment sur le parallélisme de traitement dans la mise en œuvre de l'application. Toutefois, cela suppose que les éléments de calcul qui fonctionnent en parallèle s'échangent des données efficacement i.e. qu'il n'y ait pas ou peu de conflits d'accès aux mémoires. Ainsi, comme nous l'avons vu, de nombreux travaux de recherche tentent de résoudre ces problèmes. Malheureusement, aucune des approches de l'état de l'art ne cherche à trouver un placement de données en mémoire supprimant les conflits d'accès tout en respectant une cible architecturale (i.e. un réseau d'interconnexion) et en optimisant le coût de l'architecture du contrôleur du système.

Dans ce contexte, nous avons présenté un ensemble d'approches de placement de données en mémoire répondant à ces défis. Dans un premier temps nous avons présenté deux méthodes capables pour toute cible de réseau d'interconnexion et pour toute règle d'entrelacement (même si elle est incompatible avec le réseau ciblé), de trouver un placement mémoire et exploitant le principe de relaxation de la contrainte mémoire (i.e. l'ajout de registres additionnels).

Dans un second temps, nous avons présenté une évolution de la deuxième méthode. Cette nouvelle approche permet d'optimiser l'architecture du contrôleur de l'application, en prenant en considération, dès l'étape de placement mémoire, la problématique du coût du pilotage des mémoires dans lesquels sont assignées les données. Pour ce faire un modèle de graphe dédié à la formalisation des conflits d'adressage entre les données (ACG) a été défini. Le flot de conception exploitant ce modèle a ensuite été formellement présenté.

Résume

Notre contribution dans le cadre de ce mémoire porte sur deux axes principaux qui sont développés dans les contextes suivants :

*Définir une approche de placement de données en mémoire sans conflit et garantissant au concepteur que l'architecture qu'il souhaite pour le réseau d'interconnexion sera respectée.

*Parvenir à optimiser le coût de l'architecture du contrôleur du système résultant.

ملخص

تركز مذكرتنا كجزء من هذه الأطروحة على مجالين رئيسيين التي يتم تطويرها في السياقات التالية

* تعيين نهج الاستثمار بيانات الذاكرة دون صراع والتأكد من أن العمارة مصمم يحبه لشبكة الربط سيتم احترامها.

*تحقيق الاستفادة المثلى من تكلفة العمارة تحكم في نظام الناتجة عن ذلك.

Abstract

Our contribution as part of this thesis focuses on two main areas that are developed in the following contexts:

* Set a memory data investment approach without conflict and ensuring that the designer architecture he wishes for the interconnection network will be respected.

* Achieving optimize the cost of the controller architecture of the resulting system.