الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جامعة وهران
Université d'Oran
كلية العلوم
Faculté des sciences

Mémoire de Fin d'Etudes

Pour l'Obtention du Diplôme d'Ingénieur d'Etat en Informatique

Présenté par :

MOSTEFA MERIEM

Option : *Système distribué*

Session Juin 2009

THEME

PLACEMENT DES TACHES REPETITIVES SUR UNE ARCHITECTURE REGULIERE EMBARQUEE

Encadré par : **Mr A.E. BENYAMINA**
Co-encadré par : **Mr P. BOULET**

Jury

Président : **Mme M. BOURENANE**

Examinateur : **Mme F. BELKHEIR**

Code PFE :87/2009

Examinateur : **Mme Z. BEDAI**

Promotion 2008/2009

DEDICACE

Au nom de Dieu Allah

Avec toute ma reconnaissance Je dédie ce modeste mémoire :

A

mes parents

mes sœurs et à mon frère

mes grands parents

tous mes professeurs

Et a tous mes proches pour leur participation

Mostefa meriem...

REMERCIEMENTS

Ce travail a été réalisé dans le cadre de préparation de mémoire d'ingénieur pour l'obtention du diplôme d'ingénieur d'état en Informatique option système distribué. Il n'aurait pas pu voir le jour sans le soutien de nombreuses personnes que je tiens à remercier.

JE tiens tous d'abord à remercier mon encadreur **Mr.BENYAMINA abou el hassen et** mon co-encadreur **Mr BOULET pierre** qui m'ont encouragé à poursuivre mes travaux de recherche de ce mémoire, qui grâce a leur disponibilité et rigoureux conseils, j'ai pu entamer, développer et mener à terme ce travail .Qu'il trouve ici l'expression de toute ma gratitude.

Je remercie sincèrement touts ceux qui ont bien voulu prendre part à ce jury.
A Mme BOURENANE qui nous a fait l'honneur de présider le jury.
A Mme BELKEIR et **Mme BEDAI** qui ont accepté de juger ce travail.

Je les remercie pour leur intérêt à ce travail et la bonne formation que nous avons eu au département d'informatique.
Je remercie également tous mes amies et amis de la promotion pour leur soutien et les moments agréables que nous avons passé ensemble.

Sommaire

CHAPITRE III: contribution au mapping multiobjectifs Des taches répétitives

CHAPITRE IV: implementation et mise en oeuvre

LISTE DES FIGURES

Résumé

Notre projet consiste à concevoir et développer dans une plateforme JAVA Eclipse une application pour résoudre le problème du Placement de taches Répétitives sur une Architecture Hardware représentée par une Grille Torique Bidirectionnelle Homogène. Pour traiter ce problème on a proposé une approche exacte basée sur le branch and bound multi-objectif hybride. Cette approche nous a permis d'explorer d'une manière exhaustive tout le domaine des solutions et peut retourner une solution exacte répondant aux objectifs fixés. De plus dans le cadre de Conception et développement du logiciel notre contribution pour construire un modèle comprenant les valeurs de temps d'exécution de chaque tache sur chaque connexion en fonction du volume de données transportés où ces valeurs sont fournies à l'Algorithme Branch & Bound d'Optimisation Parallèle multicritères où le fait de trouver l'Algorithme d'Optimisation est toute la question du PFE et plus globalement le but du projet de L'Equipe Dart /West en vérifiant le critère des contraintes temporelles pour le choix du motif Optimal.

Mots clé : Placement, Ordonnancement, BnB, parallélisme , graphe .

Introduction générale

L’intitulé de ce mémoire est placement des taches répétitives DSP sur une architecture régulière embarquée.

Les systèmes embarqués désignent généralement les systèmes électroniques et informatique autonomes qui sont dédié à une tache bien précise. Ses ressources disponibles sont généralement limitées(taille mémoire limitée et consommation énergétique limitée).

Les domaines d'applications concernés par ces systèmes deviennent de plus en plus nombreux : Multimédia, Télécommunications, Automobile, Avionique, Santé, Marine...

Les particularités de ces systèmes sont nombreuses : étroite cohabitation entre matériel et logiciel, faible consommation, portabilité, temps réel, sûreté de fonctionnement et cout minimum.

Notre travail se propose de faire l’implémentation de l’algorithme Branch & Bound pour l’optimisation du placement d’un ensemble de taches répétitives sur une architecture régulière embarquée.

Pour l’introduction de ces systèmes et la présentation de notre travail, nous avons organisés ce mémoire en quatre chapitres.

Ainsi dans le premier chapitre nous présenterons les systèmes embarqués avec ses contraintes et ses flots de conception .

Dans le deuxième chapitre nous étudierons le traitement de signal et applications intensives(DSP) avec le langage ARRAY-OL.

Dans le troisième chapitre nous nous intéresserons au mapping multi objectifs des taches répétitives DSP .

Enfin le quatrième chapitre sera consacré à l’implémentation et les résultats de l’expérimentation.

CHAPITRE I

SYSTEMES EMBARQUES

ET FLOTS DE CONCEPTION

1 - introduction :

Dans ce chapitre, nous présentons une étude concernant la modélisation à haut niveau et la conception des systèmes sur puce. Cette approche est basée sur la méthode d'Ingénierie Dirigée par les Modèles (IDM ou MDE en anglais) qui permet de faciliter l'étude et le développement de ces systèmes. L'IDM, apparue vers la fin des années 1990, s'est particulièrement développée lors de la parution de l'initiative Model Driven Architecture de l'Object management Group. Comme son nom l'indique, l'IDM est une approche dont l'élément clé est la notion de modèle. Après avoir introduit les principes de base de cette démarche, nous allons présenter l'environnement de développement GASPARD 2, principalement conçu pour la conception des systèmes sur puce défini autour des applications de traitement de signal intensif et d'architectures massivement parallèles.

1.1 - systèmes embarqués

Un système embarqué peut être défini comme un système électronique et informatique autonome dédié à une tâche bien précise et répondent souvent à des contraintes temps réel. Ses ressources disponibles sont généralement limitées, elles sont généralement d'ordre spatial (taille limitée) et énergétique (consommation restreinte).

La majorité de ces systèmes, centralisés ou distribués sont critiques pour la sécurité, soit parce qu'ils sont au cœur du comportement du système, soit parce qu'ils interagissent avec l'être humain dans des situations critiques. Assurer leur fiabilité et leur sûreté de fonctionnement est un défi majeur.

La complexité croissante des systèmes embarqués nécessite des méthodes de conception globale qui tiennent compte des fonctionnalités et des constituants (capteurs, actionneurs, contrôleurs, réseaux), des perturbations et des fortes contraintes de coût ou environnementale (retards, incertitudes,...).

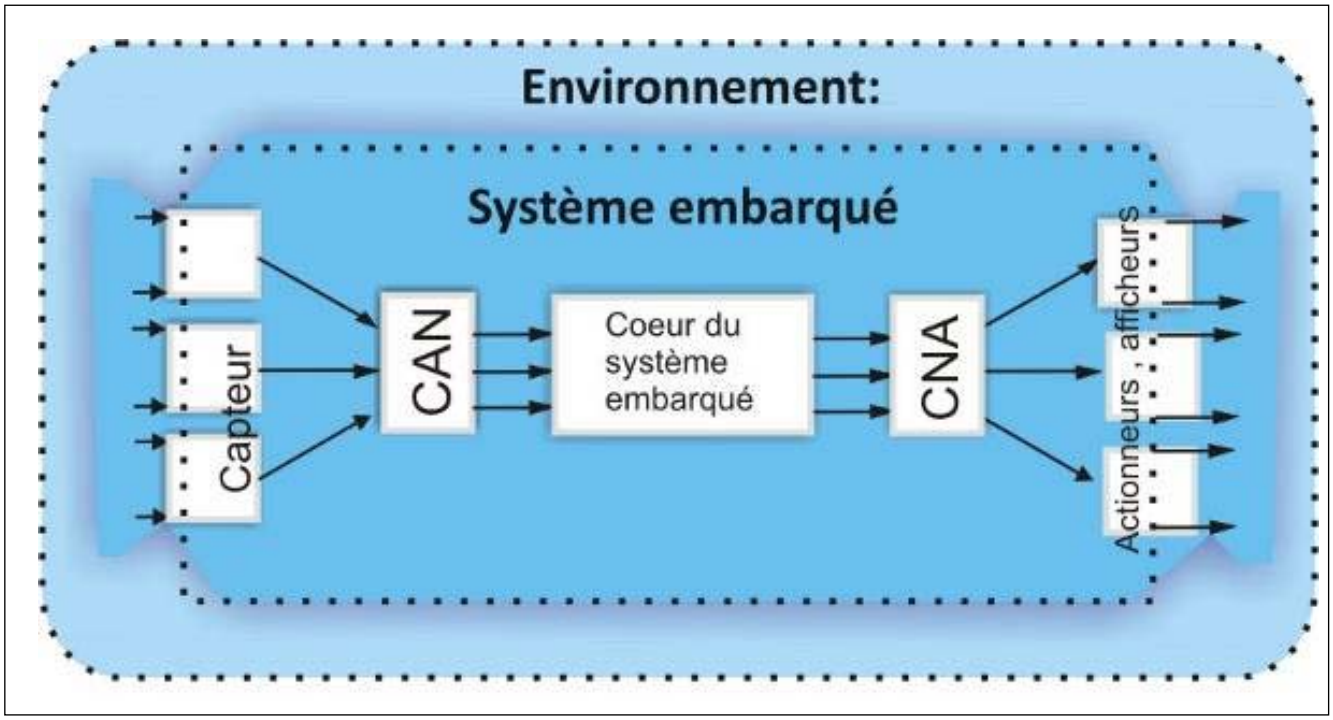


Figure 1 un système embarqué dans son environnement

1.2- historique

Les premiers systèmes embarqués sont apparus en 1971 avec l'apparition du Intel 4004. développé en 1971. Ce premier microprocesseur, était le premier circuit intégré incorporant tous les éléments d'un ordinateur dans un seul boîtier: unité de calcul, mémoire, contrôle des entrées / sorties.

Alors qu'il fallait auparavant plusieurs circuits intégrés différents, chacun dédié à une tâche particulière, avec ce type de microprocesseur un seul pouvait assurer autant de travaux différents que possible. Très rapidement, des objets quotidiens tels que fours à micro-ondes, télévisions et automobiles à moteur à injection électronique ne tardèrent pas à être équipés de microprocesseurs.

Ce sont alors les débuts de l'informatique embarquée. [1]

1.3- contraintes

- De coût.
- Taille (ou dimension),
- De puissance de calcul.
- De consommation énergétique
- Temporel, dont les temps d'exécution de tâches est déterminé
- De sécurité et de sûreté de fonctionnement. [1]

1.4- domaine d'application

- Transport : Automobile, Aéronautique (avionique), etc.
- Astronautique : fusée, satellite artificiel, sonde spatiale, etc.
- Militaire : missile
- Télécommunication : Set-top box, téléphonie, routeur, pare-feu, serveur de temps, téléphone portable, etc.
- Électroménager : télévision, four à micro-ondes
- Impression : imprimante multifonctions, photocopieur, etc.
- Informatique : disque dur, Lecteur de disquette, etc.
- Multimédia : console de jeux vidéo, assistant personnel [1]
- Guichet automatique bancaire (GAB)
- Équipement médical
- Automate programmable industriel
- Métrologie

1.5 - architectures embarquées

Les systèmes embarqués utilisent généralement des microprocesseurs à basse consommation d'énergie ou des microcontrôleurs, dont la partie logicielle est en partie ou entièrement programmée dans le matériel, généralement en mémoire dans une mémoire morte (ROM), EPROM, EEPROM, FLASH, (on parle alors de firmware) [4].

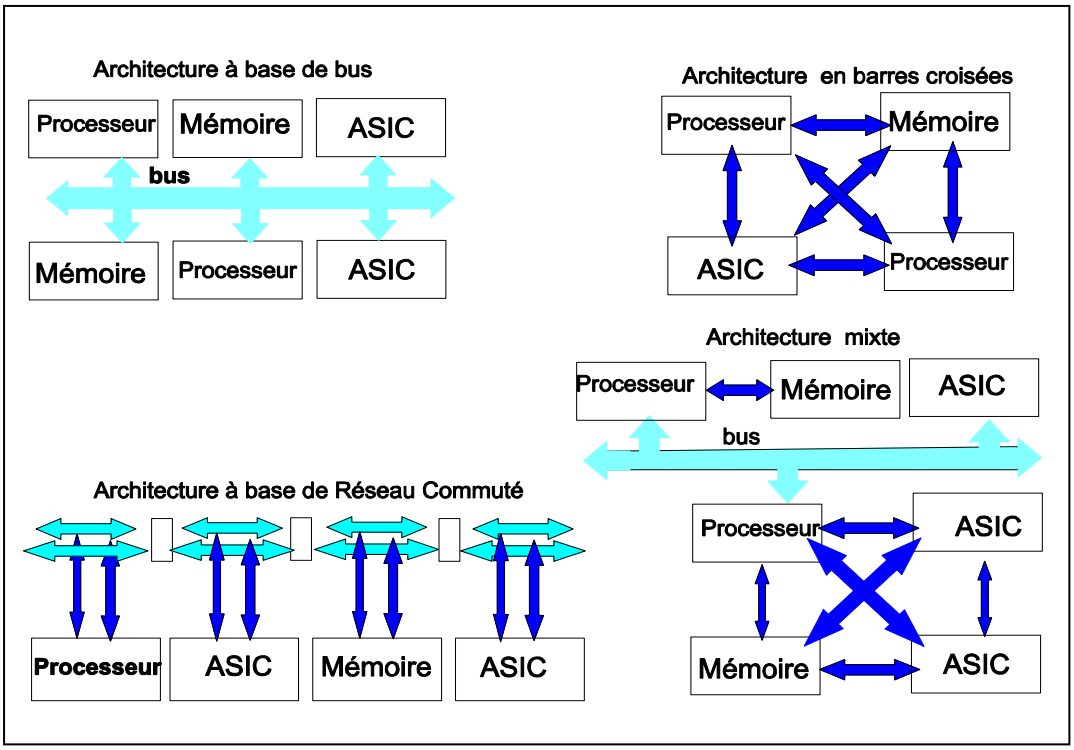


Figure 2 Architecture embarqué de troisième génération

1.5.1 - les systèmes mono puces (SoC)

Les progrès réalisés par les fondeurs de circuits permettent maintenant d'envisager l'intégration sur une même puce d'un système embarqué complet. Ces systèmes mono puces (SoC : System on Chip) apportent des changements importants dans les flots de conception classiques.

Dans les systèmes électroniques classiques, les grandes dimensions des cartes électroniques entraînaient des problèmes électriques. L'intégration d'un système complet sur une seule pièce de silicium [4].

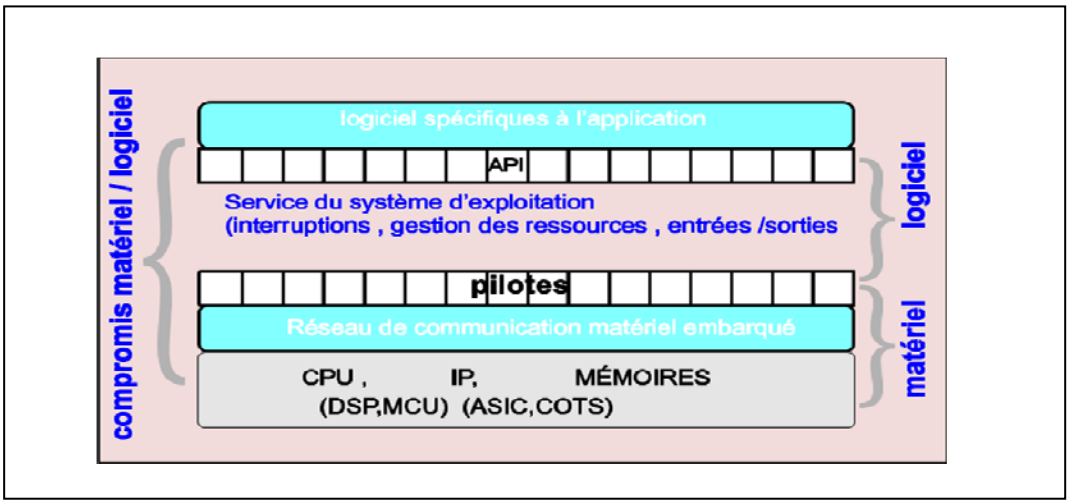


Figure 3 : Architecture logicielle /matérielle d'un système monopuce

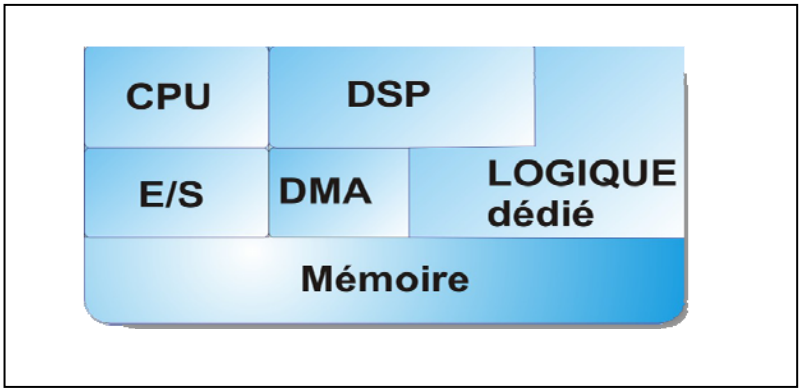


Figure 4 : Un exemple de système mono-puce (SOC).

1.5.1.1 critères de qualité de conception d'un soc

- Métriques usuelles
- Coût à l'unité: coût de fabrication d'une unité sans inclure ,les coûts non récurrents.
- Coûts non récurrents: coût de conception d'un système (coût de mise en place de la première pièce).
- Taille
- Performance
- Consommation
- Évolutivité: possibilité de faire évoluer le système pour en avoir des versions dérivées.
- Temps de prototypage : temps de mise en place d'un premier système fonctionnel
- Temps de mise sur le marché : système suffisamment fiable pour être commercialisé

- Maintenance : possibilité de modifications du système par rapport à sa première version
- Fiabilité, sûreté de fonctionnement, . . .

1.5.1.2 choix en fonction des contraintes

Une grande souplesse de réalisation est possible ,il n'y a pas de solution unique pour Implémentation contrainte de coût , implémentation uniquement logicielle de Performance Coût[4].

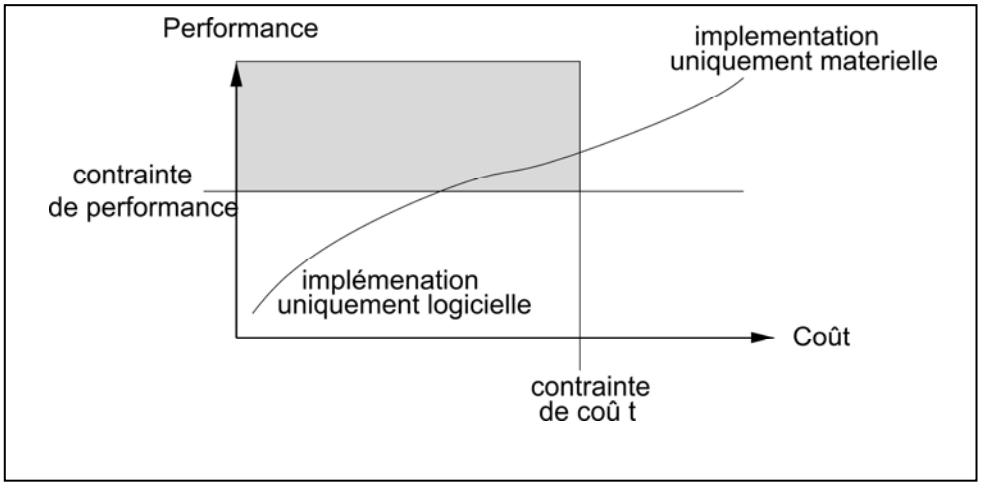


Figure 5 : illustration du choix en fonction des contraintes

1.5.2- Network On Chip (NoC) [17]

Circuit-intégré spécialisé, renfermant les fonctionnalités nécessaires à une connexion réseau, y compris celles liées aux protocoles. L'augmentation de la complexité des systèmes embarqués dans le domaine des télécommunications et la réduction des temps de mise sur le marché des produits les intégrant sont des facteurs nécessitant l'adoption d'une méthodologie de conception adéquate. Deux approches permettent de gérer cette complexité : l'augmentation du niveau d'abstraction de la description du système et la réutilisation de blocs déjà conçus (IP). Ces deux approches sont en général complémentaires. Une difficulté majeure lors de la conception de tels systèmes est la communication entre les différents modules composant le système. Une solution performante et économique est de se baser sur une infrastructure de communication configurable préétablie, à savoir un **NoC**. Le recours à un **NoC** s'impose pour les circuits de grande dimension, aussi bien du point de vue

technologique (asynchronisme entre les parties éloignées d'une même puce), que du point de vue fonctionnel (réutilisation de fonctions IP).

Pour faire face aux nouveaux problèmes introduits par la nanotechnologie, liés aux interconnexions sur puce et pour y accommoder l'intégration future de plusieurs centaines de composants, le concept de réseau sur puce (Network on Chip) a été proposé très récemment.

Il s'agit d'adapter les modèles, les techniques et les outils du domaine des réseaux d'ordinateurs au contexte d'intégration sur silicium. Ce nouveau concept intervient pour la conception des futurs systèmes embarqués.

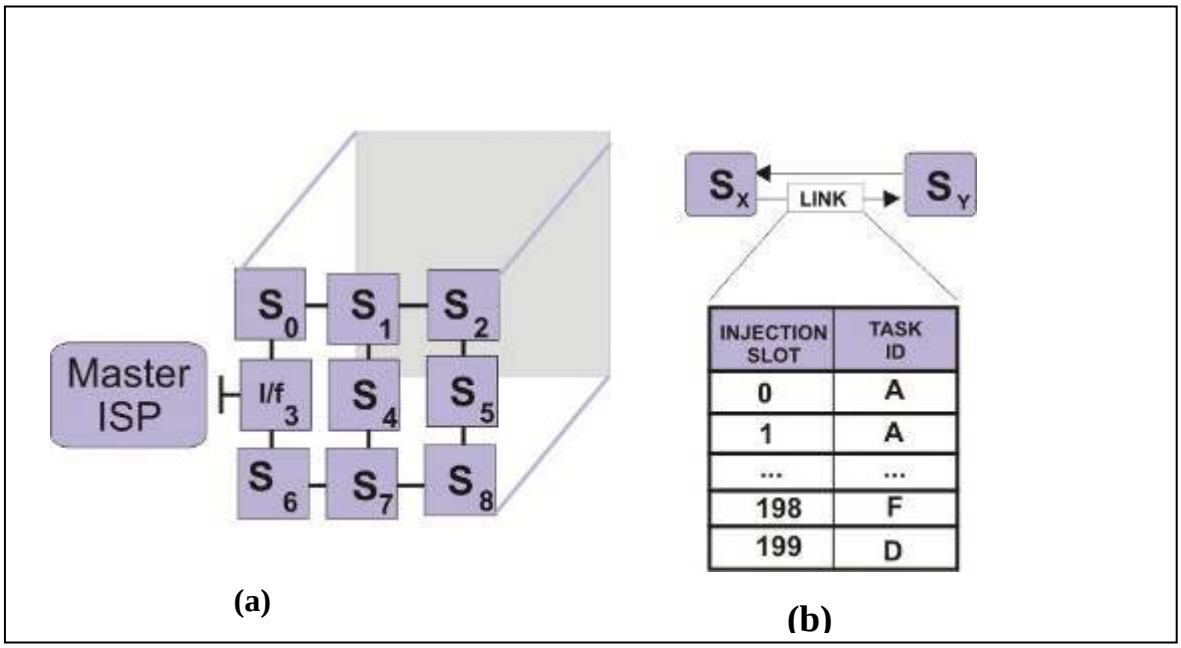


Figure 6 : (a) NoC platform . (b) communication ressource management

Aux côtés de processeurs universels, le recours à des circuits spécialisés dont l'architecture est optimisé pour la réalisation de certaines fonctions pour gagner en performance. Dans le cas des systèmes embarqués, ce gain de performance permet l'intégration d'un même circuit, des divers composants matériels et logiciels pour former ce qu'il convient d'appeler un Soc (System On Chip), ces divers composants pouvant par ailleurs être regroupés en réseau au sein d'un même circuit (Network On Chip).[**2]**

1.5.3 - topologies [17]:

1.5.3.1 - Topologies régulières et irrégulières

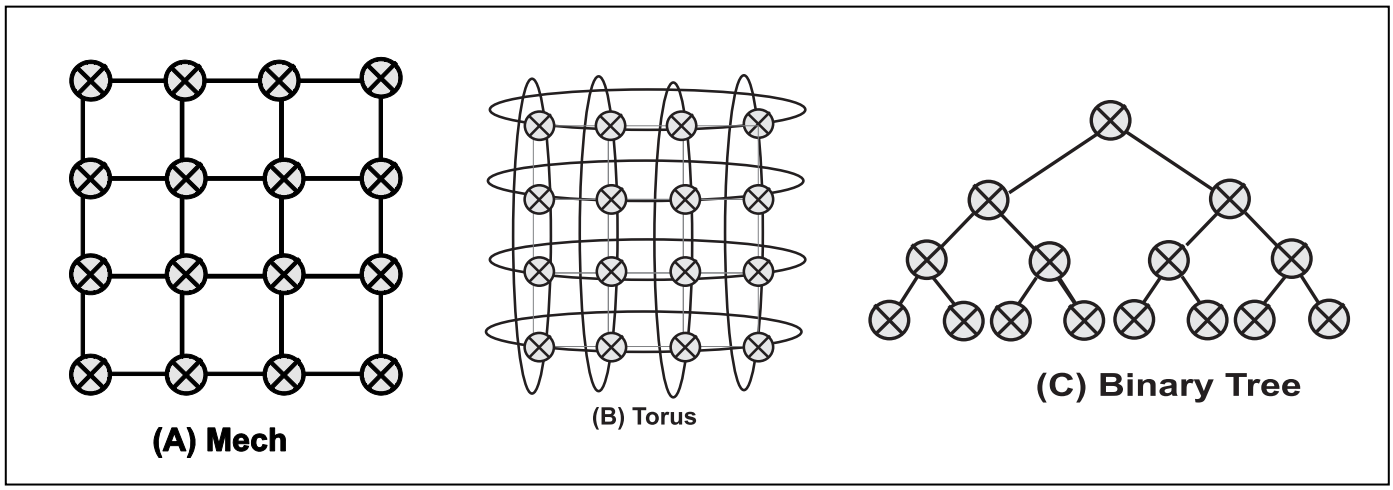


Figure 7 : Des formes régulières de topologies prévisibles.

Les exemples sont (A) Mech 4 array de 2 cubes, (B) Torique 4 array de 2 cubes et (C) Arbre binaire (2-array)

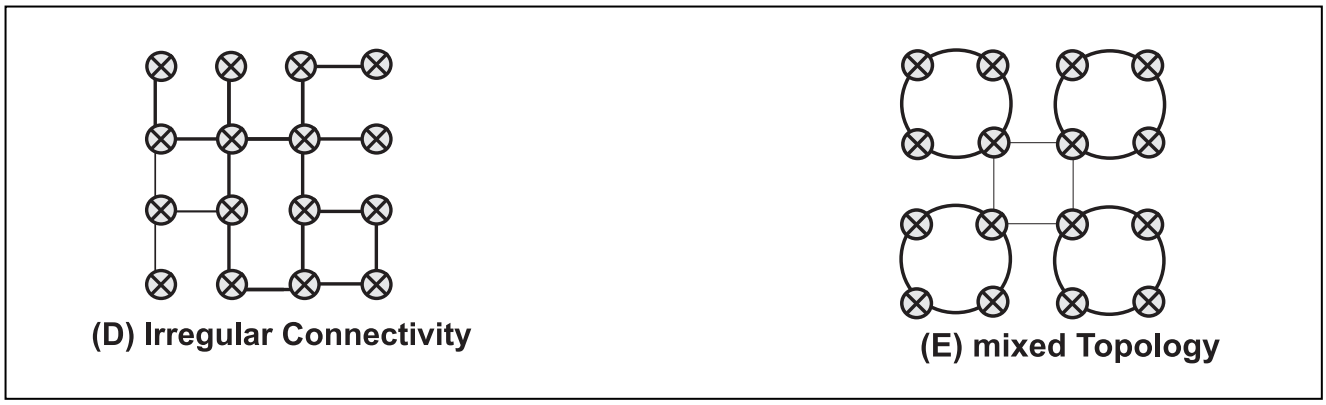
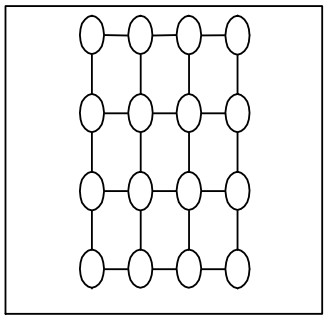
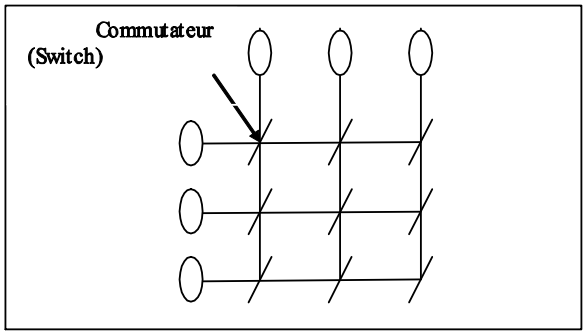


Figure 8 : formes irrégulières de topologies

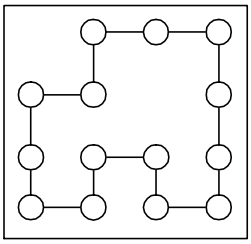
D'autres formes irrégulières de topologies sont dérivées en changeant la connectivité d'une structure régulière qui sont indiquée dans (D) où certains liens du Mech ont été enlevés et pour (E) les différentes topologies hybridées où un anneau coexiste avec Mech .



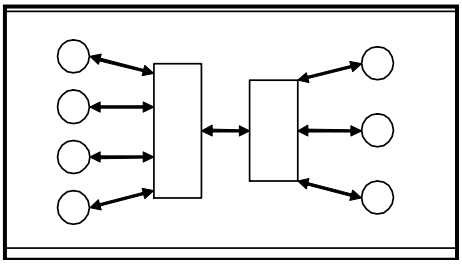
Maille complètement connecté



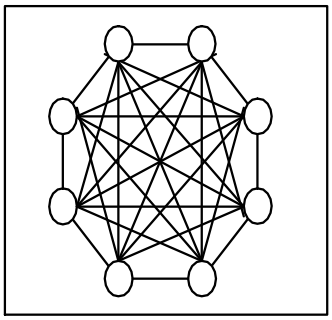
Crossbar complet



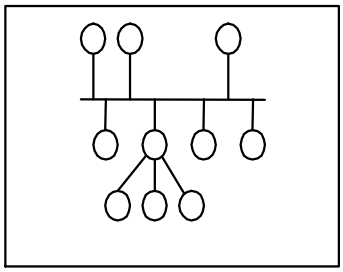
Anneau Simple



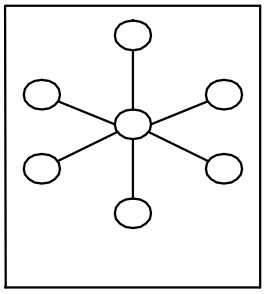
Liaison des Nœuds à des Commutateurs



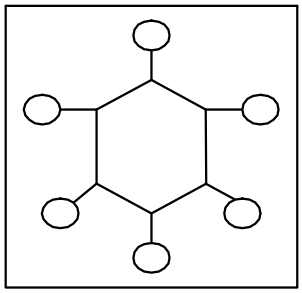
Interconnexions totales Bus+Etoile.



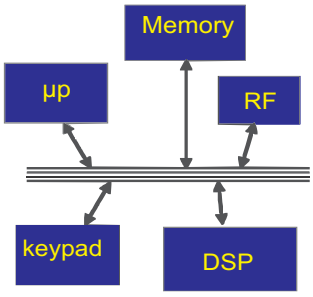
Topologie Hybride



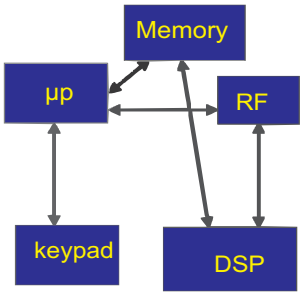
Etoile



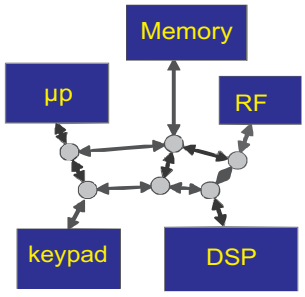
Anneau découplé



Bus centralisé



Point to point



Réseau

Figure 9 : Illustrations des différentes topologies

2. Flots de conception sur les SOC et environnement GASPARD

2 .1 - Spécification en «y» [16] :

Cette méthodologie de spécification en « Y » autorise par construction la réutilisation aussi bien de l'application que de l'architecture dans un formalisme UML , notre souci de respect des standards nous a orienté vers une construction « Y » dans un formalisme UML et suivant une philosophie de type MDA (Model Driven Architecture) [19].

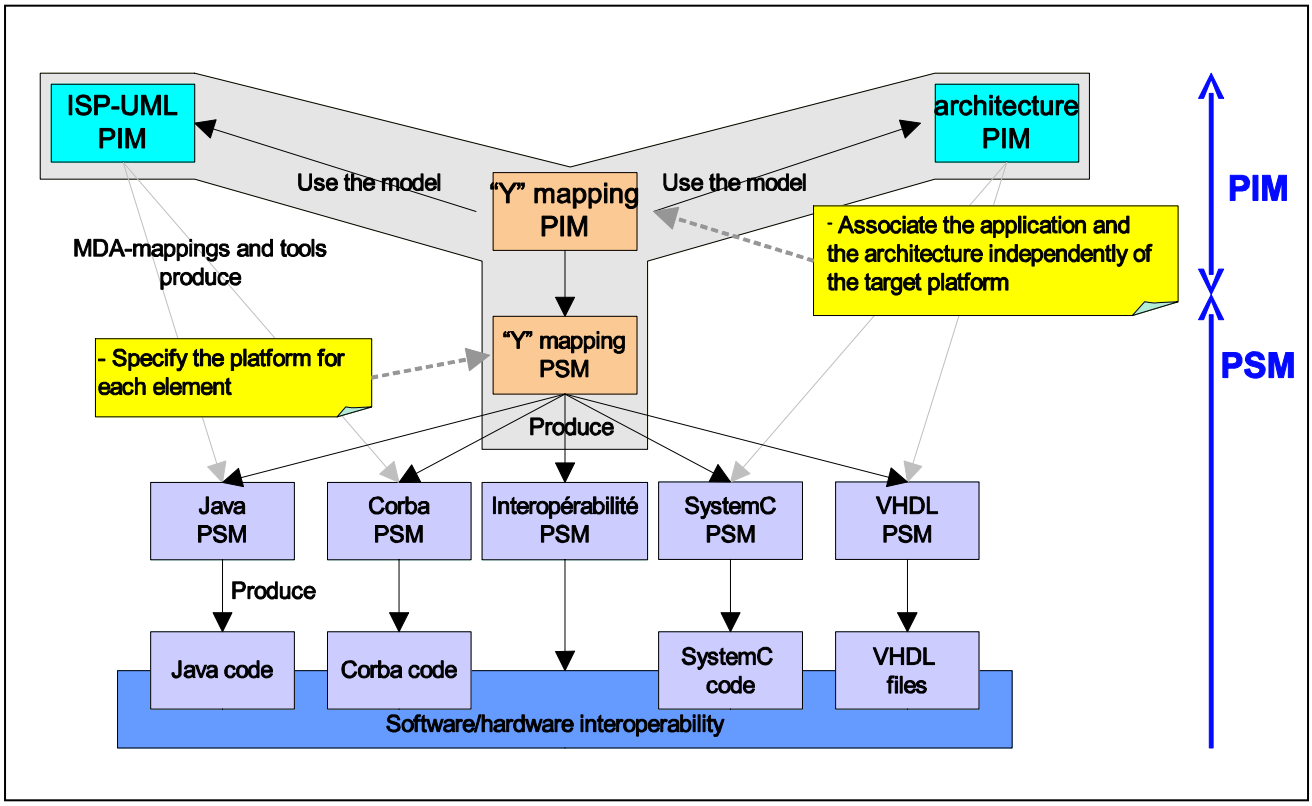


Figure 10 : Spéciations avec philosophie de type MDA

2.2 - L'Ingénierie Dirigée par les Modèles (IDM)

2.2.1- Principes de l'IDM

Généralement, l'IDM peut être défini autour de trois concepts de base :

- modèles
- les méta- modèles
- transformations

Les Modèles sont des abstractions de la réalité et un méta-modèle est la définition des concepts et relations à utiliser pour exprimer des modèles.

Un autre point clé de l'IDM est la transformation de modèles, elle permet de passer d'un modèle source décrit à un certain niveau d'abstraction à un modèle destination décrit éventuellement à un autre niveau d'abstraction. Ces modèles source et destination sont conformes à leur méta-modèle respectif (méta-modèles source et destination) et le passage de l'un à l'autre (i.e. la transformation) est décrit par des règles de transformation.

- ✓ **Avantages**
  - Réduire la complexité (Abstraction supplémentaire)
  - Réduire le temps de développement
  - Réutilisation des modèles
  - Séparation des concepts métiers des concepts technologiques

2.3.- GASPARD

2.3.1- Le flot de conception de Gaspard :

Gaspard [3] est un environnement de développement permettant la co-conception de système sur puce dans un cadre d'ingénierie dirigée par les modèles (IDM). Il est développé au sein de l'équipe DaRT et est orienté vers les applications de traitement de signal intensif. Ces applications sont rencontrées souvent dans les SOC et sont caractérisées par leur gourmandise en termes de puissance de calcul ce qui nécessite la parallélisation des traitements.

Gaspard se situe à la frontière de deux domaines de recherche : la conception des systèmes sur puce, et l'ingénierie dirigée par les modèles.

Il est important de noter que le "Y" de Gaspard correspond à un flot tel qu'on le définit généralement dans le domaine de la conception de systèmes sur puce (ou plus généralement dans le domaine de la conception de systèmes embarqués incluant des parties logicielles et matérielles) et parmi les spécifications on trouve :

- **Application** : Spécification de la fonctionnalité attendue d'un système,
- **Architecture matérielle** : Spécification de l'architecture matérielle qui sera utilisée pour supporter l'exécution d'une application et réaliser ses fonctionnalités attendues
- **Association** : Spécification du placement d'une application sur une architecture Matérielle donnée.

2.3.2- L'environnement GASPARD

L'environnement de spécification visuel GASPARD est construit à partir des caractéristiques héritées du modèle ARRAY-OL. Le modèle global est représenté par une description visuelle d'un composant logiciel. La métaphore que nous utilisons repose sur le « design » d'un composant électronique de type circuit imprimé. Les tableaux sont des informations circulant sur des fils qui sont connectés à différents slots sur lesquels sont branchés d'autres composants logiciels. Cette phase de branchement permet de spécifier les informations d'ajustage et de pavage qui indiquent comment le composant ainsi connecté référencera les tableaux liés au slot. Les motifs obtenus par ce pavage et ajustage deviennent les tableaux en entrée/sortie du composant ainsi branché.

De cette façon chaque composant peut être spécifié indépendamment des composants qu'il utilise et vice versa. En conséquence, les composants sont tous réutilisables.

Les composants de GASPARD sont assimilés à des classes telles que définies dans des langages à objets. Seules les instances de ces classes seront dites exécutables. Des bibliothèques de composants prédéfinis, éventuellement génériques, peuvent être crées pour des types d'applications particulières.

Les dépendances entre différents composants connectés dans le même composant Logiciel (niveau global) sont exprimées par les fils entre les slots.

2.3.2.1 - Notion de Tiler

Le nombre de lignes de la matrice correspond au nombre de dimensions du tableau et le nombre de colonnes correspond au nombre de dimensions du motif que l'on veut récupérer. Chaque élément e_i de la tuile est déterminé par la somme des coordonnées du point de référence et de la multiplication de la matrice d'ajustage par les coordonnées i de l'élément du motif, comme exprimé dans la formule suivante : i $0 \leq i < S_{pattern}$, $e_i = ref + F.i \bmod S_{array}$, $S_{pattern}$ représentant la forme du motif, S_{array} représentant la forme du tableau et F étant la matrice d'ajustage.

Dans la figure 11 , on présente un exemple de Tiler dans une application. Le connecteur situé entre les ports m1 et A, par exemple, le port m1 a une dimension 128x128 et le port A a une dimension de 8x8, le Tiler permet de décomposer le tableau situé sur m1 par bloc de 8x8 éléments contigus.

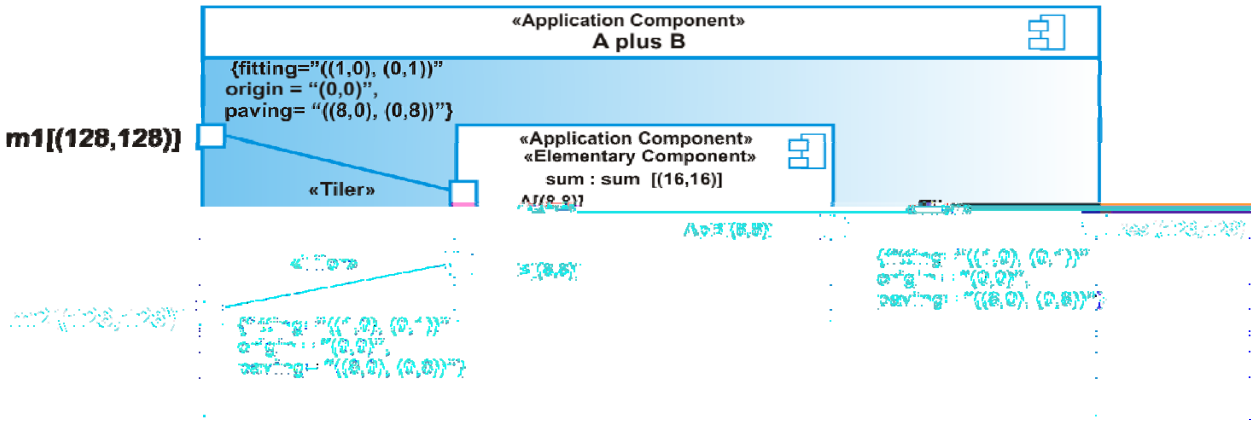


Figure 11 : Illustration de la notion de Tiler

2.3.2.2 - La notion de Reshape

La figure 12 illustre un exemple de Reshape. Dans cet exemple, il y a deux tâches t_1 et t_2, chaque ligne du tableau de sortie t_1 est rangée dans une colonne du tableau d'entrée t_2. On passe donc d'un tableau de dimension 3x2 à un tableau de dimension 2x3.

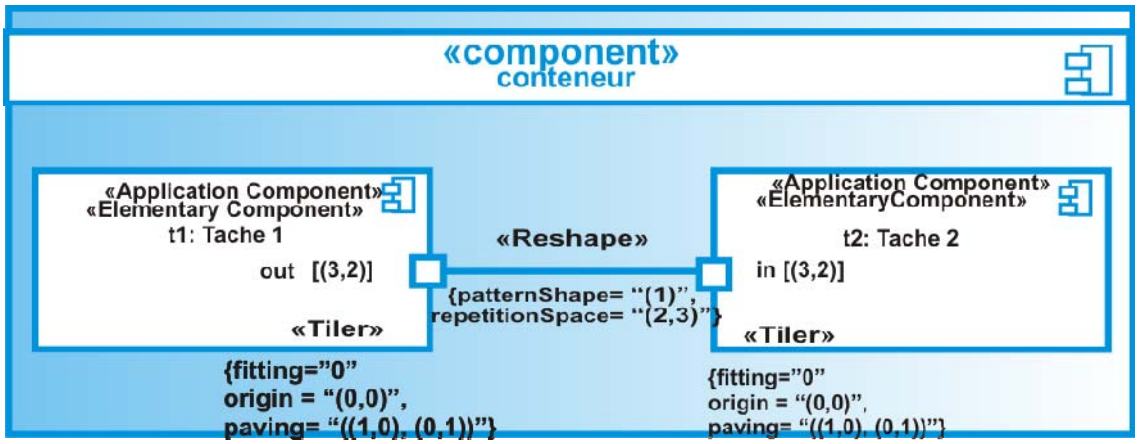


Figure 12 : Illustration La notion de Reshape

2.3.2.3 - les notions d'inter répétition et de default link

Le concept Default Link permet d'exprimer les connexions irrégulières pouvant se situer sur les bords d'un tableau d'éléments. [20]

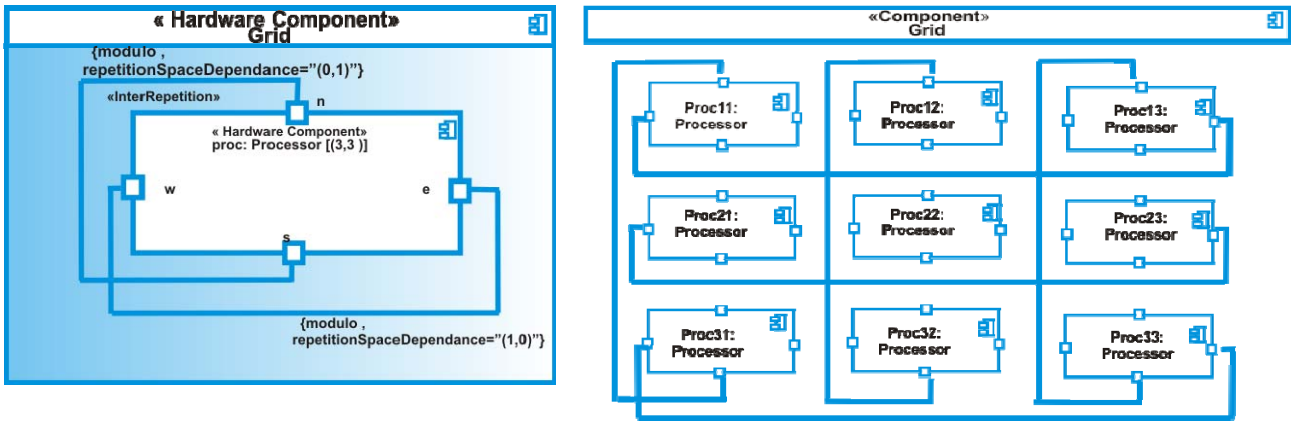


Figure13 : Illustration des liens Inter –Répétition, à gauche le composant Gaspard représentant une grille à droite son équivalent en UML.

CONCLUSION :

Pour la modélisation d'un système embarqué c'est le flot gaspard qui s'y prête le mieux. Le système et son application doivent permettre de prendre en considération les taches répétitives et les données intensives régulières pour cela dans le chapitre suivant nous verrons les applications de traitement de signal intensif.

CHAPITRE II

TRAITEMENT DE SIGNAL ET APPLICATIONS INTENSIVES

Introduction

Les applications temps réel sont depuis longtemps une préoccupation primordiale dans plusieurs domaines. Elles occupent une place de plus en plus importante dans le monde actuel, elles étaient réservées aux équipements industriels lourds et actuellement ,on les trouve dans des différents produits grand public et particulièrement dans les systèmes de télécommunication. La qualité de ces systèmes est jugée par leur temps d'exécution ce qui conditionne leurs situations au marché.

Les applications temps réel reposent principalement sur des algorithmes de traitement de signal qui nécessitent une quantité importante de calculs. Pour cela, il a fallu utiliser des calculateurs de fortes puissances dont le but d'accélérer les calculs spécifiques au domaine du traitement de signal. Pour cette raison, un type spécial de processeur a été créé, il s'agit des processeurs de traitement numérique du signal (DSP - Digital Signal Processing).

Une application temps réel est composée de deux parties :

- La première correspond à un système informatique composé d'un calculateur exécutant un ensemble de programmes qui forment son logiciel, et qui renferment les algorithmes de l'application.
- La seconde partie correspond à son environnement physique, dont les changements d'état, sont perçus par le calculateur au moyen de capteurs. Le calculateur réagit à ces stimulis pour maintenir l'environnement dans un état déterminé au moyen d'actionneurs.
- Ces applications doivent réagir dans un temps très court afin d'assurer que l'environnement est bien contrôlé, sinon on risque d'avoir des mauvaises conséquences.

Afin de pouvoir présenter le placement dans sa globalité (le graphe TG), nous devons commencer par présenter une de ses parties importantes qui est le traitement intensif. Dans ce chapitre on introduira le principe général des applications de traitement de signal intensif, les domaines faisant appel à ce type de traitement et les principaux modèles de calcul existants pour leur spécification. Par la suite, nous étudierons le principe du modèle Array-Ol, principalement conçu pour l'expression du parallélisme de données et des dépendances de tâches pour ce type d'applications.

1. Historique DSP :

Au niveau historique, vers les années 70, les ***DSP*** ont été initialement développés pour des applications de radars militaires et de télécommunications cryptées, et C'est Texas Instruments pour la première fois qui en 1978 a introduit un ***DSP*** pour la synthèse de la voix pour des applications très grand public.

Quinze ans plus tard, les ***DSP*** deviennent des composants incontournables dans le domaine de l'électronique grand public. Les domaines d'application du traitement numérique du signal sont nombreux et variés (Tableau 1).

Un ***DSP*** est un type particulier de micro processeur. Il se caractérise par le fait qu'il intègre un ensemble de fonctions spéciales. Ces fonctions sont destinées à le rendre particulièrement performant dans le domaine du traitement numérique du signal. Comme un microprocesseur classique, un ***DSP*** est mis en œuvre en lui associant des mémoires (RAM, ROM) et des périphériques. Un ***DSP*** typique a plutôt vocation à servir dans des systèmes de traitements autonomes. Il se présente donc généralement sous la forme d'un microcontrôleur intégrant, selon les marques et les gammes des constructeurs, de la mémoire, des timers, des ports séries synchrones rapides, des contrôleurs DMA, et des ports d'E/S divers [26].

2- Application intensive et données intensives :

2.1 Généralités sur les DSP

2.1.1 Caractéristiques des DSP :

Tous les systèmes à bases de DSP bénéficient des avantages suivants [26] :

- Souplesse de programmation : Un DSP est avant tout un processeur exécutant un programme de traitement du signal. Ceci signifie que le système bénéficie donc d'une grande souplesse de développement. De plus, les fonctions de traitements numériques peuvent évoluer en fonction des mises à jour des programmes, et cela pendant toute la durée de vie du produit incluant le système. Ainsi, modifier par exemple tel ou tel paramètre d'un filtre numérique ne nécessite pas un changement matériel.

Une autre qualité issue de la souplesse des programmes. Il est possible d'adapter une fonction de traitement numérique en temps réel suivant certains critères d'évolutions du signal (par exemple les filtres adaptatifs).

- Implémentation d'algorithmes adaptatifs : des possibilités propres au système de traitement numérique du signal. Certaines fonctions de traitement du signal sont difficiles à implanter en analogique, voir irréalisables (exemple : un filtre à réponse à phase linéaire).
- Stabilité : En analogique, les composants sont toujours plus ou moins soumis à des variations de leurs caractéristiques en fonction de la température, de la tension d'alimentation, du vieillissement. Une étude sérieuse doit tenir compte de ces phénomènes, ce qui complique et augmente le temps de développement. Ces inconvénients n'existent pas en numérique.
- Répétitivité, reproductibilité : Les valeurs des composants analogiques sont définies avec une marge de précision plus ou moins grande. Dans ces conditions, aucun montage analogique n'est strictement reproductible à l'identique, il existe toujours des différences qu'il convient de maintenir dans des limites acceptables. Un programme réalisant un traitement numérique est par contre parfaitement reproductible,

« à l'infini ».

Domaines	Applications
Général	✓ Filtres numériques ✓ Filtres adaptatifs ✓ Corrélation ✓ FFT DCT…
Instrumentation	✓ Analyse de transitoire ✓ Analyse spectrale ✓ Suppression de bruit …
Acoustique	✓ Réduction du bruit ambiant…
Médical	✓ Monitoring ✓ Echographie ✓ Imagerie médicale…
Contrôle	✓ Asservissements, robotique, vision artificielle…
Télécommunication	✓ Modems, multiplexeurs, FAX ✓ Annuleurs d'échos, répéteurs de ligne ✓ Egaliseurs auto-adaptatifs ✓ cryptage de données ...
Multimédia et Imagerie	✓ traitement d'image ✓ reconnaissance de forme ✓ reconnaissance et synthèse de la parole ✓ compression des images et du son ✓ suppression de bruits parasites ...
Militaire	✓ traitement d'image ✓ radar, sonar ✓ guidage de missile, calculateurs de trajectoires ✓ communication et sécurité ✓ navigation ...
Grand Public	✓ automobile, électroménager, jeux ✓ synthétiseurs musicaux, navigation, téléphone …

Tableau 1 : Différents domaines et différentes applications d'utilisation des processeurs DSP.

2.1.2 Domaines et applications des DSP

Les DSP sont généralement utilisés dans les systèmes qui comportent une quantité importante de calculs et nécessitant une rapidité dans l'exécution de ces algorithmes. Le tableau 1 représente en détail dans les différents domaines ainsi que les différentes applications des DSP.

2.2 Formats des DSP [26]

Dans les systèmes de traitement de signal, la numérisation des signaux intervient dans les dispositifs de conversion analogique/numérique ainsi que dans les processeurs DSP.

Lorsque le convertisseur et le processeur utilisent des représentations différentes, le type de quantification est imposé par le système de traitement.

2 .2.1 Numérisation d'un signal

Les signaux physiques sont généralement analogiques. Pour les traiter avec un DSP, il faut les échantillonner et convertir chaque échantillon en une donnée numérique. Cette conversion est réalisée par un convertisseur analogique/numérique (CAN). La conversion se décompose en une quantification et une numérisation (codage) de la valeur quantifiée. Réciproquement, les résultats numériques fournis par un DSP pourront être convertis en signaux analogiques à l'aide d'un convertisseur numérique /analogique (CNA).

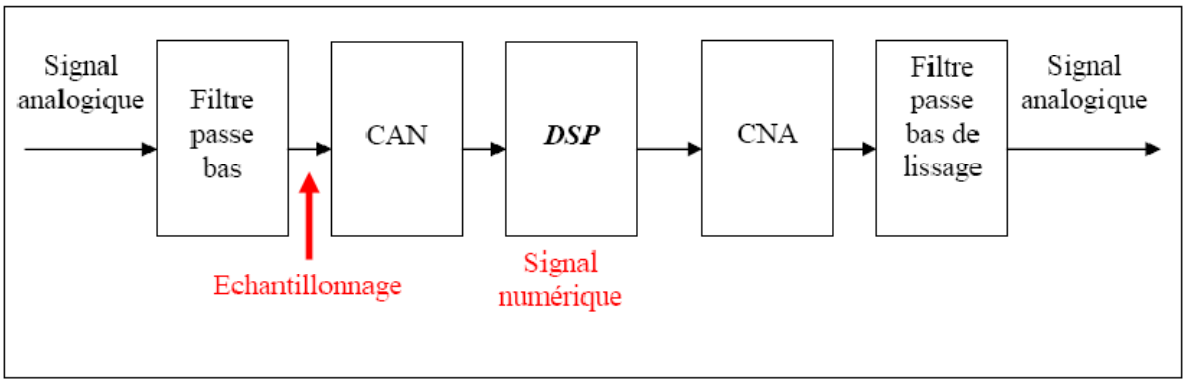


Figure 14 : Interface CAN-DSP-CAN.

La figure 14 représente l'interface entre un DSP et ces convertisseurs.

Les convertisseurs CAN fournissent une représentation numérique de chaque échantillon. Cette représentation peut être ensuite transformée par le DSP selon le type d'arithmétique utilisée.

2.3 Exemples d'applications (hautement régulières) [16] :

De nombreuses applications permettent la décomposition du TSI en TSS et TDI

2.3.1 Radar anticollisions :

La prévention des collisions dans un système nécessite l'utilisation d'une antenne qui renvoie le signal sous la forme d'un flux de données contenant des informations relatives à la présence d'obstacles, un écho. Ces informations sont masquées pare du bruit (des parasites) et sont étalées sur le flux de données temporel. Une phase de prétraitement qui relève du traitement du signal systématique, filtre ce signal de manière à en faire ressortir les caractéristiques intéressantes (la présence d'obstacles). La régularité se trouve dans l'étape suivante qui consiste à analyser ce nouveau flux de données de manière à valider la présence d'un obstacle, à déclencher un freinage d'urgence où à le poursuivre si nécessaire . Cette application est illustrée par la figure 15

.

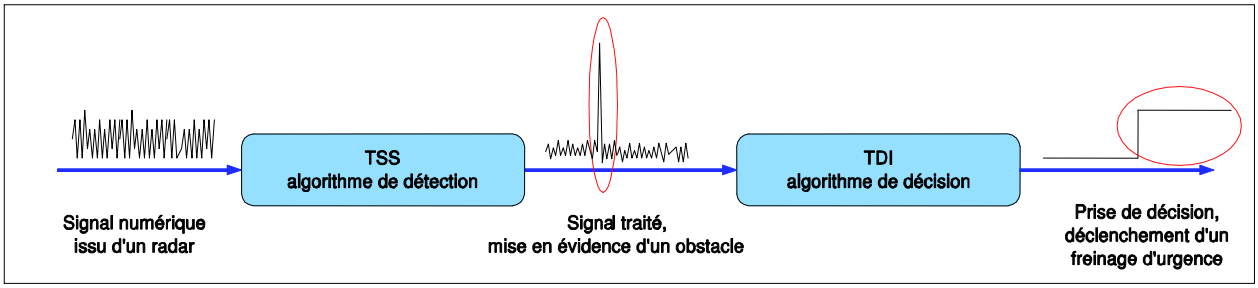


Figure 15 : Représentation de l'application radar anticollision.

La phase de traitement du signal systématique précède celle du traitement de données intensif.

2.3.2 Traitement sonar [16]:

Une vision plus large de la scène est nécessaire dans l'analyse d'un environnement maritime (qui est similaire à celui d'un routier). Dans le cas d'un sous-marin, ce n'est pas une antenne qui est utilisée mais plusieurs hydrophones répartis autour de ce sous-marin. La première phase de traitement est systématique et est composée d'une FFT (Transformations de fourrier).

Le traitement est dédié à la détection d'objets et à leur poursuite au cours du temps.

2.3.3 filtre de KALMAN :[25]

Le filtre de Kalman – Bucy conçu en 1961 est une solution des problèmes d'estimation qui ont été développés à l'origine par R.Kalman en 1960.

En utilisant la notion d'état, ce filtre est représenté sous forme d'un ensemble d'équations linéaires différentielles ou récurrentes. Ces équations présentent un caractère séquentiel plus facile à résoudre sur des calculateurs numériques. Ce filtre est bien adapté au traitement numérique en ligne, par conséquent, il fournit une estimée optimale dans le sens de la minimisation de l'erreur quadratique moyenne d'estimation. Il est très utilisé dans plusieurs domaines en particulier le domaine aérospatial où il a été particulièrement appliqué avec succès.

Le filtre de Kalman généralise le filtrage optimal aux systèmes non stationnaires en présence des conditions initiales et des entrées déterministes. Il est stable sous certaines conditions de contrôlabilité et d'observabilité.

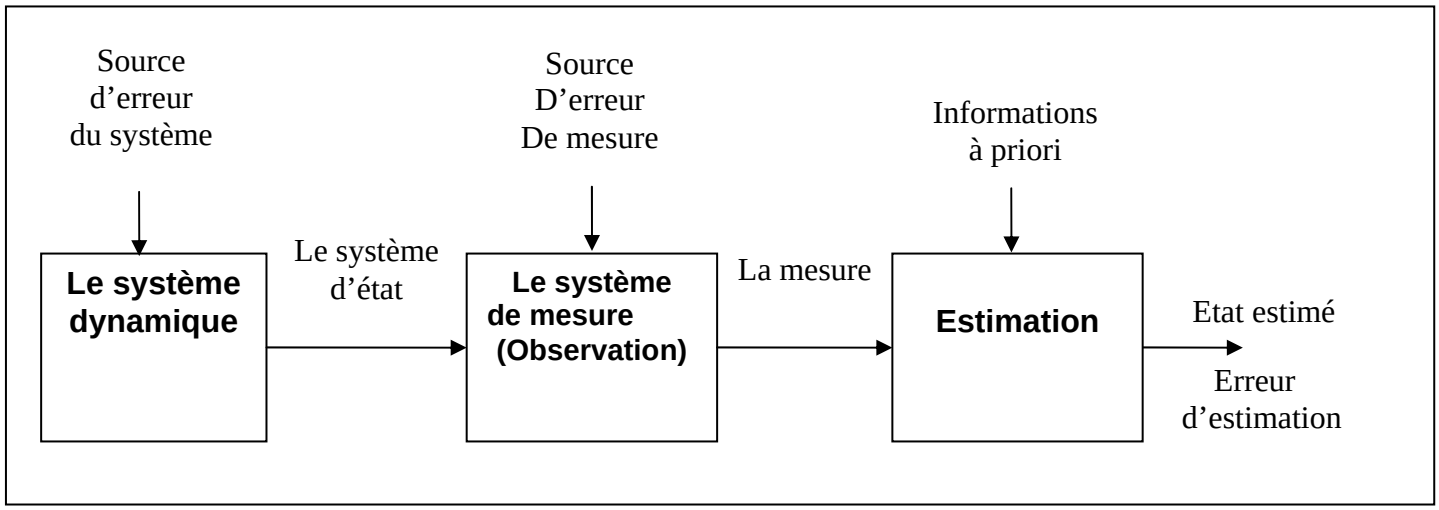


Figure 16 : La procédure d'estimation d'un état d'une cible

2.3.4 Convertisseur 16/9-4/3 :

Souvent devant notre télévision et en utilisant une télécommande on peut passer d'un format à un autre avec une certaine facilité. En pratique ce processus se décompose en deux phases. La première consiste à créer des pixels à partir du signal vidéo au format 16/9 via une interpolation, il en résulte un nouveau signal vidéo. La seconde phase supprime certains pixels de ce signal de manière à ne conserver qu'une partie de l'information, cela permet le passage

au format 4/3. Cette application ne considère à aucun moment la valeur des pixels, il s'agit de traitement de signal. D'autres exemples encore illustrent l'utilisation du TSI : le traitement de la vidéo décompressé dans une télévision HD.

JPEG 2000, récepteur de radio numérique, Chacune de ces applications réalise des traitements réguliers sur les données, certaines exploitent le résultat de ces traitements pour prendre des décisions. Nous pouvons par ailleurs remarquer la diversité de la sémantique portée par les dimensions des tableaux de données manipulées par ces applications (temporel, spatial et fréquentiel) et leur nombre.

Gaspard[1] n'exprime que des dépendances de données, c'est-à-dire l'ordre minimal que doivent respecter les calculs pour calculer le résultat attendu.

2.4 Exemple typique d'Application intensive [20]

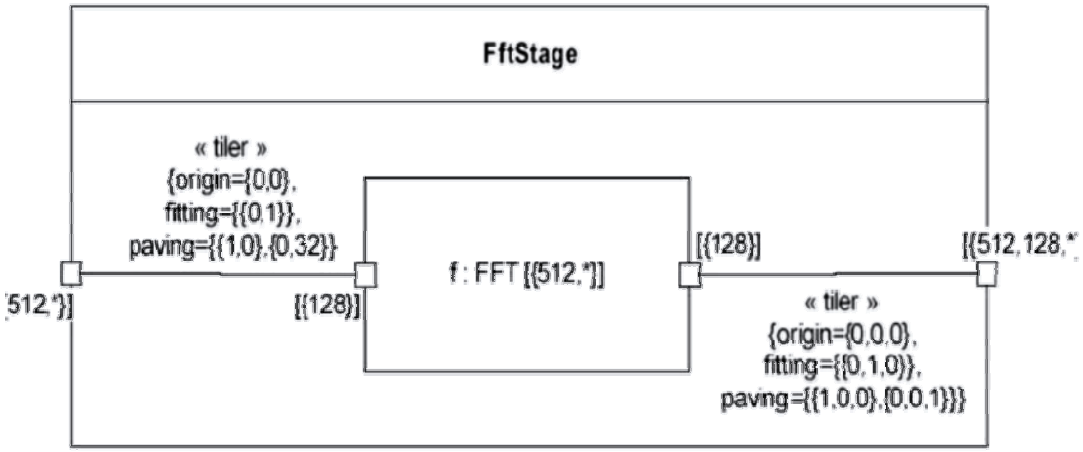


Figure 17 : Echantillon à partir de 512 hydrophones à l'infini autour d'un sous-marin.

Le graphe de l'application factorisé et met en valeur tâche et parallélisme des données :

- Echantillons de Format des données en entrées = 512 x infinie..
- répétition de FFTs pour chaque hydrophone.
- fenêtre glissante de 128 échantillons chaque 32 fractions de temps.

[1] Graphical Array Specification for Parallel And Regular Distributed computing

On représente ici l'ensemble des valeurs possibles et les dépendances de données qui les relient. Une implémentation de l'application sur une plate-forme matérielle devra projeter les tableaux infinis sur une architecture finie [20] .

3- Spécification Multidimensionnelle et Modèles de Calcul Pour Le TSI :

3.1 MDSDF : Multidimensional Synchronous Dataflow

Il existe plusieurs méthodes de modélisations on peut citer :

- SDF (Synchronous Data Flow) [8],
- MDSDF (MultiDimensional Synchronous Data Flow) [5],
- GMDSDF (Generalized MultiDimensional Synchronous Data Flow) [6].

Malgré la diversité de ces modèles, leur occupation principale consistant à mieux modéliser les grandes quantités de données multidimensionnelles dont le but est de faciliter la description et l'étude des applications de traitement intensif à parallélisme de données massive.

Le modèle SDF représente un cas spécial des modèles flot de données qui sont généralement utilisés pour décrire des applications de traitement du signal par des graphes, en représentant les fonctions par des nœuds et les données par les arrêtes du graphe.

Une application dans SDF, est décrite par un graphe orienté acyclique où chaque nœud consomme des données en entrée et produit des résultats en sortie. Donc les calculs sont représentés par des noeuds [Actor] et les données [Tokens] par des arcs.

MDSDF a été proposé par Edward Lee en 2002 [5]. Il étend le concept du modèle SDF (Synchronous DataFlow) [8] pour l'appliquer dans un contexte multidimensionnel. L'objectif est de permettre la spécification des applications de traitement du signal manipulant des données multidimensionnelles tels que le traitement d'image et de la vidéo.

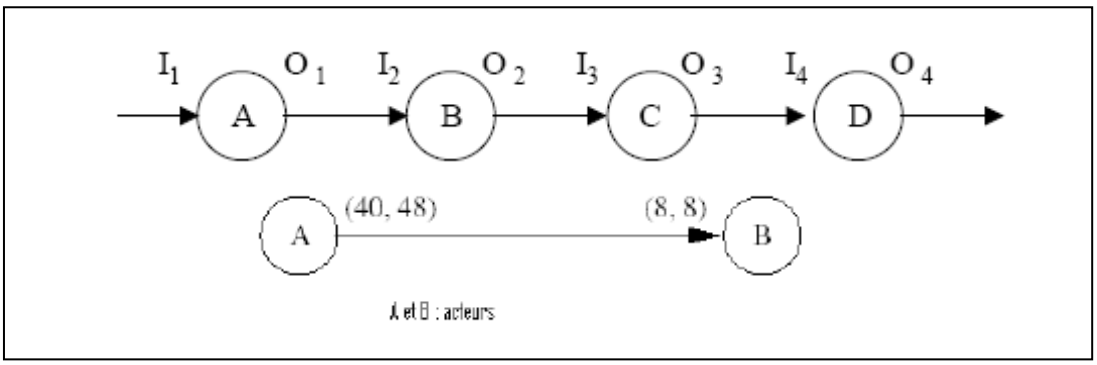


Figure 18 : Exemple d'un graphe d'une application en SDF

3.2 MATLAB/SIMULINK :

MATLAB [26] est un puissant logiciel de calcul et peut être vu comme un système interactif et convivial de calcul numérique et de visualisation graphique où beaucoup de ses fonctions sont basées sur un calcul matriciel simplifié. Il permet de distinguer plusieurs types de matrices : Monodimensionnelles (vecteurs), bidimensionnelles (matrices), tridimensionnelles . Le Logiciel de l'extension graphique de matlab est Simulink .

4. ARRAY-OL (ARRAY ORIENTED LANGUAGE) [22] :

Pour Array-Ol , c'est une sorte de formalisme de description des applications de traitement de Signal mis en place par l'Entreprise Thales.

Ce type d'application est caractérisé par une manipulation de grande quantité de données qui sont traitées par un ensemble de tâches de façon régulière. C'est un langage à assignation unique où seules les dépendances de données sont exprimées et où les dimensions spatiales et temporelles des tableaux sont banalisées.

Donc le modèle Array-Ol est multidimensionnel et permet d'exprimer un parallélisme potentiel complet sur des applications, que ce soit un parallélisme de données ou un parallélisme de tâches. Avant de comprendre l'apport de ce langage, on doit comprendre le contexte dans lequel est utilisé le Array-Ol, en commençant par ces applications qui présentent plusieurs difficultés [22].

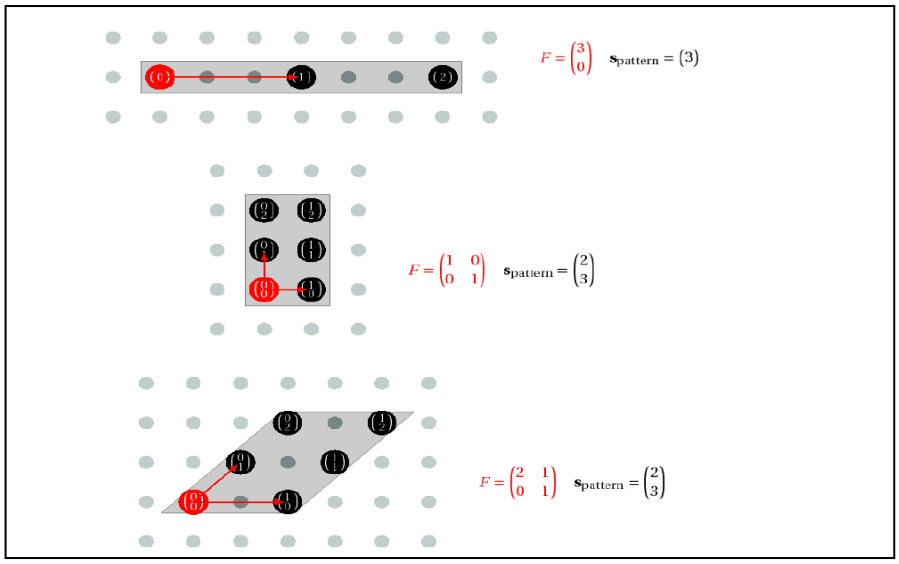


Figure 19 : Un exemple simple d'ajustage des elements d'un tableau par leurs indexes dans le motif (i), illustrant la formule i tel que 0 ≤ i <Spattern, ei=ref+F x i

Du fait que dans Array-Ol toutes les dimensions de tableaux sont toroïdales, et toutes les coordonnées des éléments du motif soient calculées modulo la taille des dimensions du tableau. La figure 16 fournit des exemples de motifs obtenus à partir d'éléments référence fixe (O comme origine dans la figure) dans des tableaux à taille fixe.

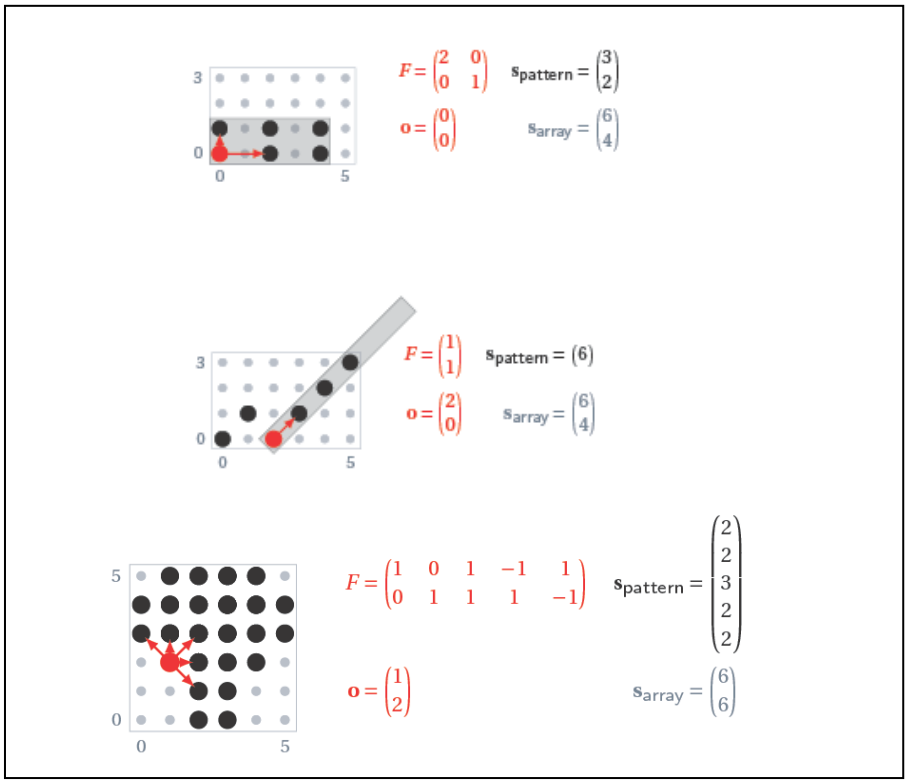


Figure 20: Un exemple complexe d'ajustage illustrant la relation i , 0 ≤ i <Spattern, ei=(ref+F x i)modSarray

Pour chaque répétition on doit spécifier les éléments référence des entrées et sortie. Les éléments référence de la répétition référence sont donné par le vecteur origine O de chaque Tiler. Les éléments référence des autres répétitions sont obtenus à partir du vecteur o. les coordonnées sont obtenues par combinaison linéaire de la matrice de pavage :

r tel que 0 ≤ r <Srepetition, refr=(0+P x r)modSarray Où Srepetition est le profil de l'espace répétition, P étant la matrice de pavage et Sarray le profil du tableau (figure 21). Ainsi on peut résumer toutes ces explications par les deux formules suivantes :

r tel que 0 ≤ r <Srepetition, refr=(0+P x r)modSarray , donne les éléments de référence des motifs.

i tel que 0 ≤ i <Spattern, ei=(ref+F x i)modSarray , Enumère tous les éléments d'un motif d'une répétition r.

Où Sarray est le profil du tableau, Spattern le profil du motif, Srepetition est le profil de l'espace de répétition, o est les coordonnées de l'élément référence du motif référence.

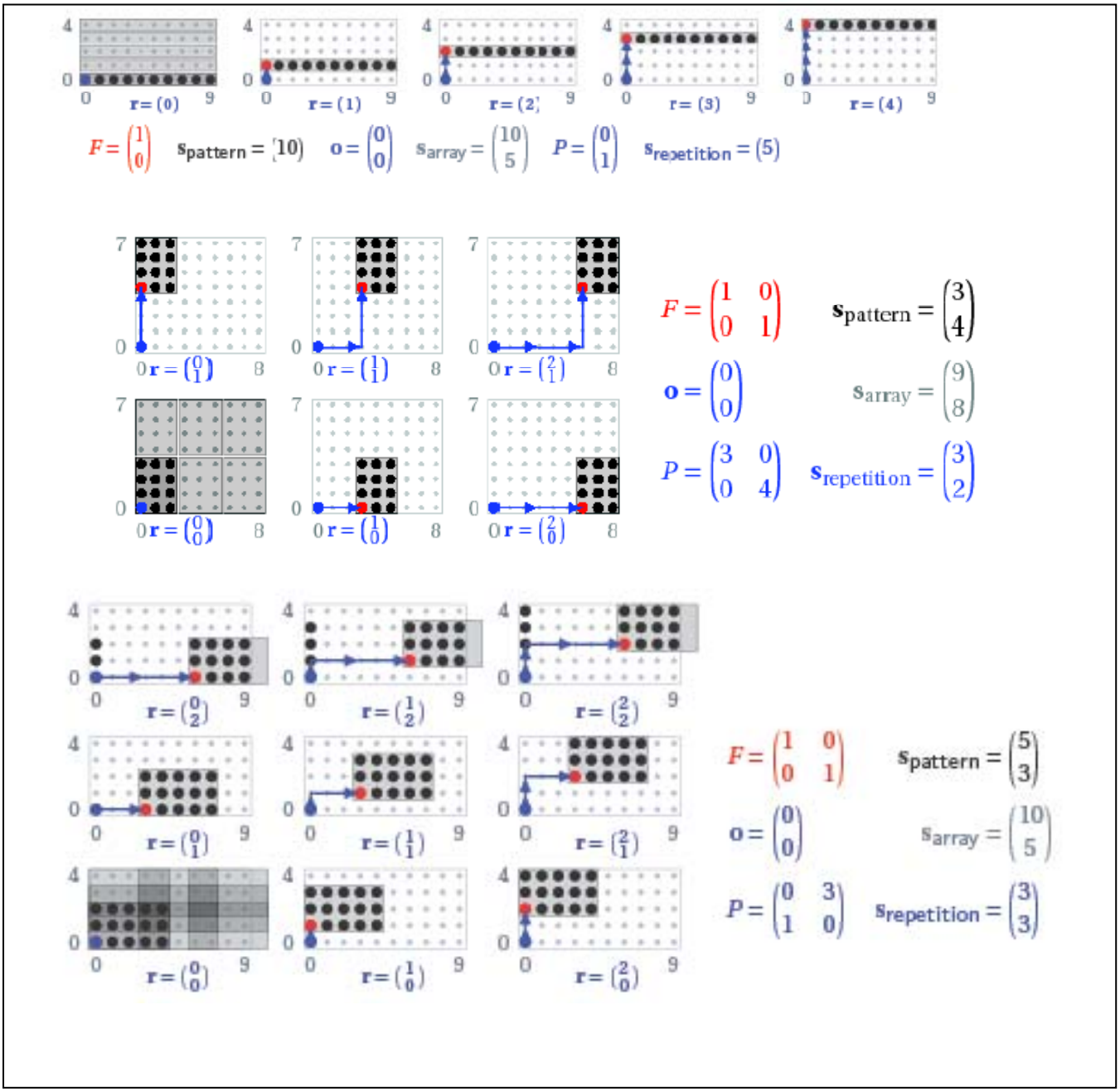


Figure 21 : Exemple de pavage illustrant la formule ;

r tel que $0 \leq r <$Srepetition, refr=(0+P x r) modSarray l'origine P est la matrice de pavage dont les vecteurs colonnes sont appelés vecteurs de pavage, et qui représentent l'espacement régulier entre les motifs, F est la matrice d'ajustage dont les vecteurs colonnes appelés vecteurs d'ajustage, représentent l'espacement régulier entre les éléments d'un motif du tableau. Les formules précédentes explicitent quel élément en entrée ou sortie d'un tableau va

être consommé ou produit par une répétition. Le lien entre les entrées et sorties est donné par l'index de répétition r.

Pour une répétition donnée, les motifs de sortie (pour un index r) sont produits par la tâche répétitive à partir des motifs en entrée (pour l'index r). Ainsi l'ensemble des Tilers, des profils des motifs et l'espace de répétition détermine les dépendances entre les éléments des tableaux en sorties et les éléments des tableaux en entrées d'une répétition.

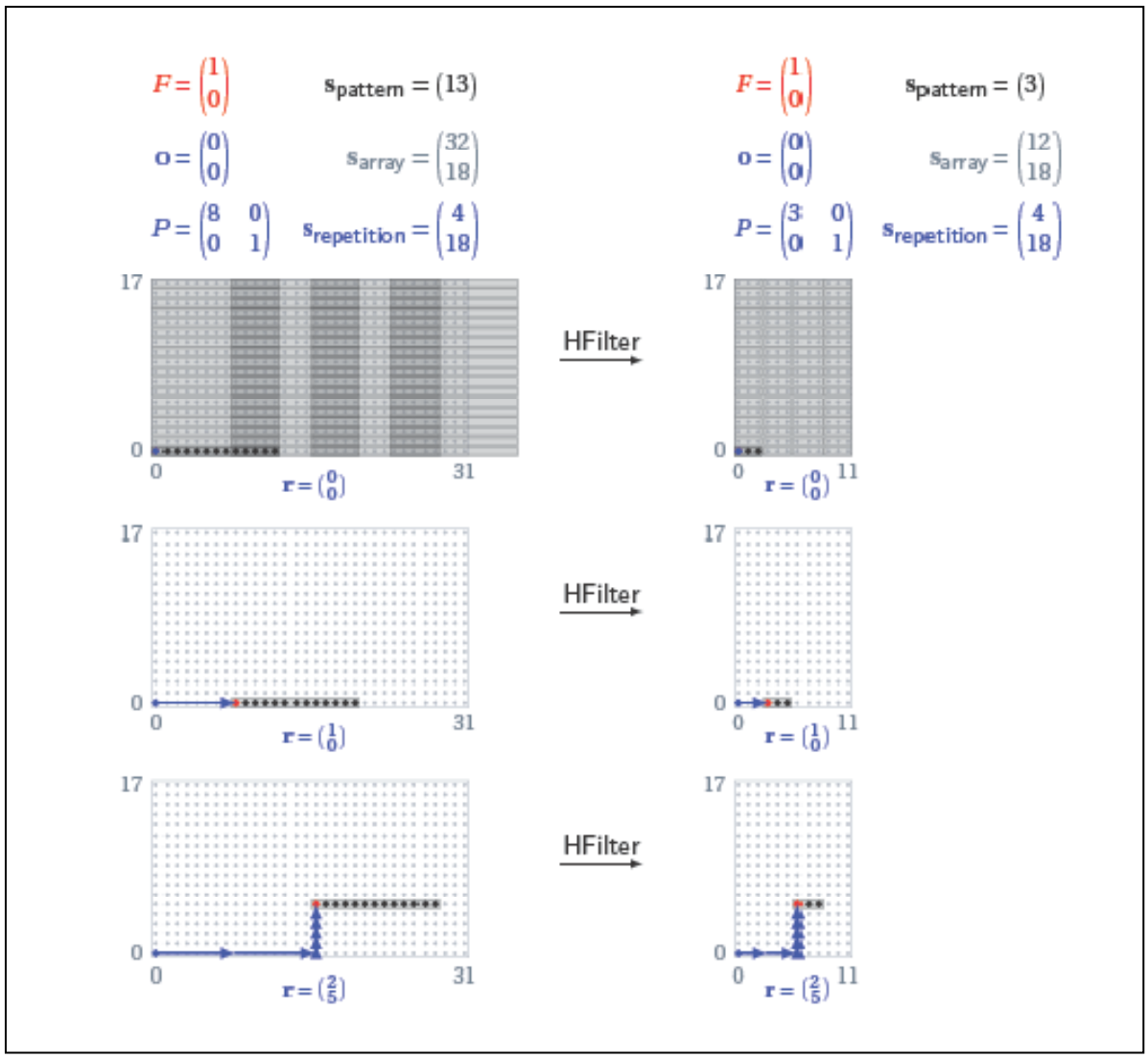


Figure 22: Quelques répétitions du filtre horizontal pour illustrer les dépendances entre les entrées/sorties d'une répétition .

A partir d'un élément référence (ref) dans le tableau on peut extraire un motif par l'énumération de ses éléments par rapport à l'élément référence. En utilisant la matrice d'ajustage, les coordonnées des éléments d'un motif (ei) sont obtenues en faisant la somme

des coordonnées de l'élément référence avec une combinaison linéaire du vecteur d'ajustage comme suit : i , $0 \leq i <$Spattern, ei=(ref+F x i)modSarray

Où Spattern est le profil d'un motif, modSarray est le profil du tableau et F la matrice d'ajustage.

La description d'une application en ARRAY-OL fait successivement appel à deux niveaux. Le premier niveau définit l'enchaînement des différentes parties de l'application dans son ensemble alors que le second précise les actions élémentaires à effectuer. Le modèle de fonctionnement sous-jacent déclenche les différentes étapes de calcul en fonction des dépendances exprimées au premier niveau ; chaque étape de calcul s'exécute suivant le modèle de fonctionnement SPMD[2].

Array Ol propose deux sortes d'approches :

- Le premier Model global définit Ordonnancement des tâche sous la forme de dépendance entre la tâche et les arrays .

-Le Second Modèle est Modèle local détaille les actions élémentaires les tâches réalisent les éléments issue d'un Array .

4.1 Le Modèle Global (parallélisme de tâches)

Le modèle global permet de nommer et de définir les tableaux. Ce sont des tableaux multidimensionnels non nécessairement définis. Ils sont les seules entités utilisées pour agencer le graphe de dépendances des tâches : chacune des tâches prend ses entrées parmi les tableaux définis. La spécification de la tâche ainsi que le balayage des tableaux ne sont pas visibles à ce niveau.

Les applications de TS sont organisées autour d'un flot de données potentiellement Infini. ARRAY-OL capture ce flot infini, qui représente généralement le temps, dans des Tableaux multidimensionnels dont l'une des dimensions peut être infinie. De plus, il est Fréquent que certaines dimensions spatiales correspondent à des capteurs qui peuvent, Par exemple, être organisés en cercle. Les dimensions finies des tableaux ARRAY-OL sont donc toriques[22].

4.2 Le modèle local (parallélisme de données)

ARRAY-OL bénéficie de la simplicité des opérations du TS. Une tâche ARRAY-OL met en relation des tableaux d'entrée et de sortie. Une tâche est toujours composée d'un constructeur

[2] Single Program Multiple Data

d'itérations, où chaque itération est indépendante et appliquée sur un sous-ensemble fini de points des différents tableaux en entrée et en sortie. Ces sous-ensembles, ou motifs, doivent être réguliers (ajustage du tableau) et se répartir régulièrement sur les tableaux (pavage de tableaux).

La cohérence globale d'une tâche est assurée par le fait que les motifs des tableaux qui sont attachés à une même tâche sont en nombre identique et reliés un pour un entre chacun des tableaux. Ainsi le rôle d'une tâche consiste, pour chaque itération de pavage, à extraire les motifs d'entrée, à leur appliquer une fonction donnée qui produit les valeurs associées aux motifs en sortie, et à finalement ranger ces derniers dans les tableaux résultats [22].

4.3-Fusion de deux tâches répétitives

La fusion vise à réduire deux tâches simples. Cette nouvelle tâche est hiérarchique et appelle les deux tâches originales en tant que tâches secondaires.

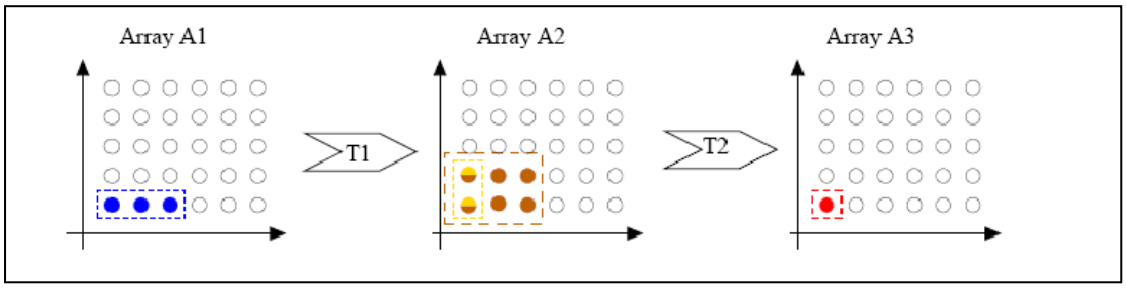


Figure 23 : avant fusion.

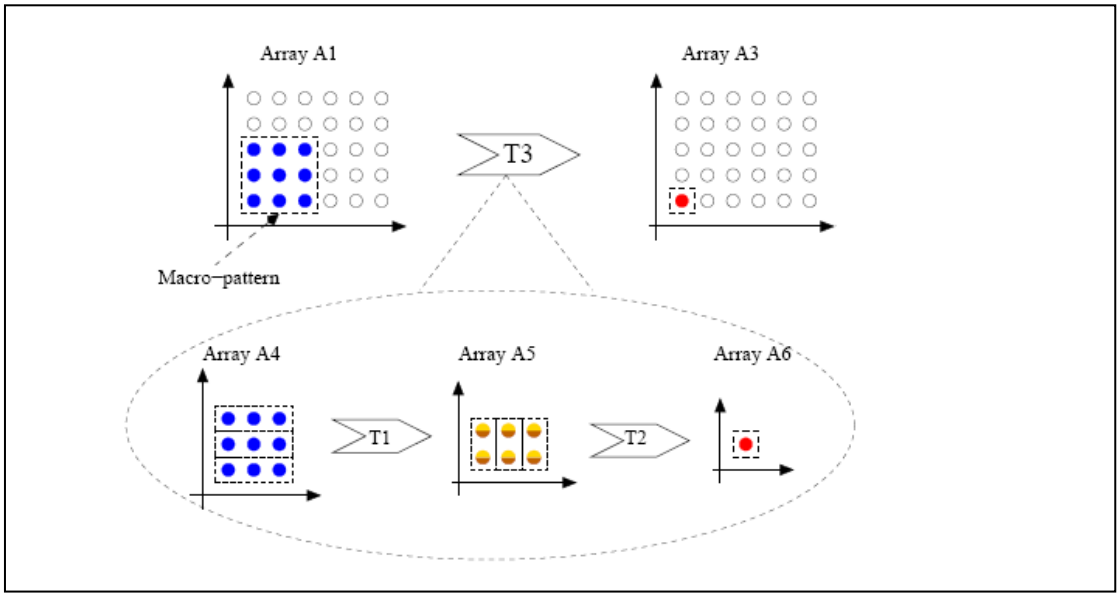


Figure 24 : après fusion

Dans un schéma d'exécution basé sur les KPN[3] on possède une FIFO qui permet de stocker plusieurs éléments.

La fusion permet de proposer un modèle d'exécution. Si on prend une application Array-OL classique, sa description permet seulement de définir les données en entrée et en sortie et comment se fait la correspondance entre elles.

Une application Array-OL est parallélisable car les données calculées sont indépendantes entre elles.

Dès qu'un macro motif est utilisé en entier, il peut être supprimé. C'est le principe des Kahn Process Networks, et c'est aussi ce que la fusion permet de faire dans le cas d'applications Array-OL. [9].

Conclusion

Dans ce chapitre nous avons vu les domaines ou on utilise du calcul intensif, nous avons défini les applications temps réel qui reposent sur les algorithmes de traitement de signal, une telle application sera plus efficace, si elle est implémentée sur des processeurs spécialisés.

Les processeurs de traitement numérique de signal (Digital Signal Processing, DSP) ont été présentés dans ce chapitre et plus spécialement à la famille C6000 de Texas Instruments (TI) et qui est un DSP à virgule fixe.

Plusieurs langages de spécification existent qui nous permettent de mettre en évidence le parallélisme des taches liés à ces applications.

Le langage ARRAY-OL, nous a permis de faire la démonstration du parallélisme des taches répétitives sur des architectures régulières embarquées. Le chapitre suivant va nous permettre d'étudier les différentes approches pour solutionner la problématique générale du placement sur les NOC (Network On Chip) .

[3] Khan Process Network

CHAPITRE III

CONTRIBUTION
AU MAPPING MULTI OBJECTIFS
DES TACHES REPETITIVES.

1.- Introduction :

Dans le chapitre présent nous allons présenter le flot de Gaspard ainsi que différents Algorithmes qui peuvent être implémentés à ce dernier.

Comme l'algorithme de Branch and Bound est l'algorithme que l'on doit utiliser pour l'élaboration de notre projet, nous avons répartis ce chapitre en deux parties :

Partie 1: nous parlerons de l'un des algorithmes d'implémentation, qui est le Branch and Bound d'une façon très détaillée et du problème de placement et ordonnancement ou AAS dans un premier terme.

Partie 2 : nous parlerons du Mapping des Tâches Répétitives dans un deuxième terme.

Notre projet consiste à trouver le meilleur placement de taches répétitives sur une architecture régulière cible. Ceci revient à trouver le nombre de ressources adéquates pour exécuter un sous ensemble de taches. Ce qui justifie notre choix pour l'algorithme Branch & Bound .

2. l'algorithme Branch and Bound :

Pour plusieurs problèmes, en particulier les problèmes d'optimisation, l'ensemble de leurs solutions est fini (c.-à-d dénombrable). En utilisant l'algourithme de Branch and Bound, on a la possibilité d'énumérer toutes ces solutions puis de prendre celle qui nous arrange. L'inconvénient majeur de cette approche est le nombre excessif du nombre de solutions : il n'est pas du tout évident d'effectuer cette énumération.

La méthode de branch and bound (procédure par évaluation et séparation progressive) consiste à énumérer ces solutions, en utilisant certaines propriétés du problème en question, cette technique arrive à éliminer des solutions partielles qui sont dominées selon les critères Energie Totale et Durée Totale et aussi les contraintes temporelles suivant la solution que l'on recherche d'un placement ou de plusieurs placements du problème.

Pour ce faire, cette méthode se dote d'une fonction qui permet de mettre une borne sur certaines solutions qui représentent les solutions du Pareto soit les exclure par la suite si ces dernières ne vérifient les contraintes temporelles comme troisième critère de minimisation du totale Slack , soit les maintenir comme des solutions potentielles.

2.1 Séparation et évaluation :

Un algorithme par séparation et évaluation est une méthode d'énumération implicite: toutes les solutions possibles du problème peuvent être énumérées mais l'analyse des critères du problème permet d'éviter l'énumération de larges classes de solutions dominées.

2.2 Principe de parcours de l'arbre :

Le principe utilisé pour la l'agorithme Branch and Bound est celui de la profondeur d'abord. Les règles de sélection des nœuds à explorer sont fixées à priori et ne dépendent pas des évaluations obtenues pour les différents nœuds. De même, il n'y a pas de liste des nœuds à explorer car ceux-ci le sont selon l'ordre dans lequel ils sont obtenus.

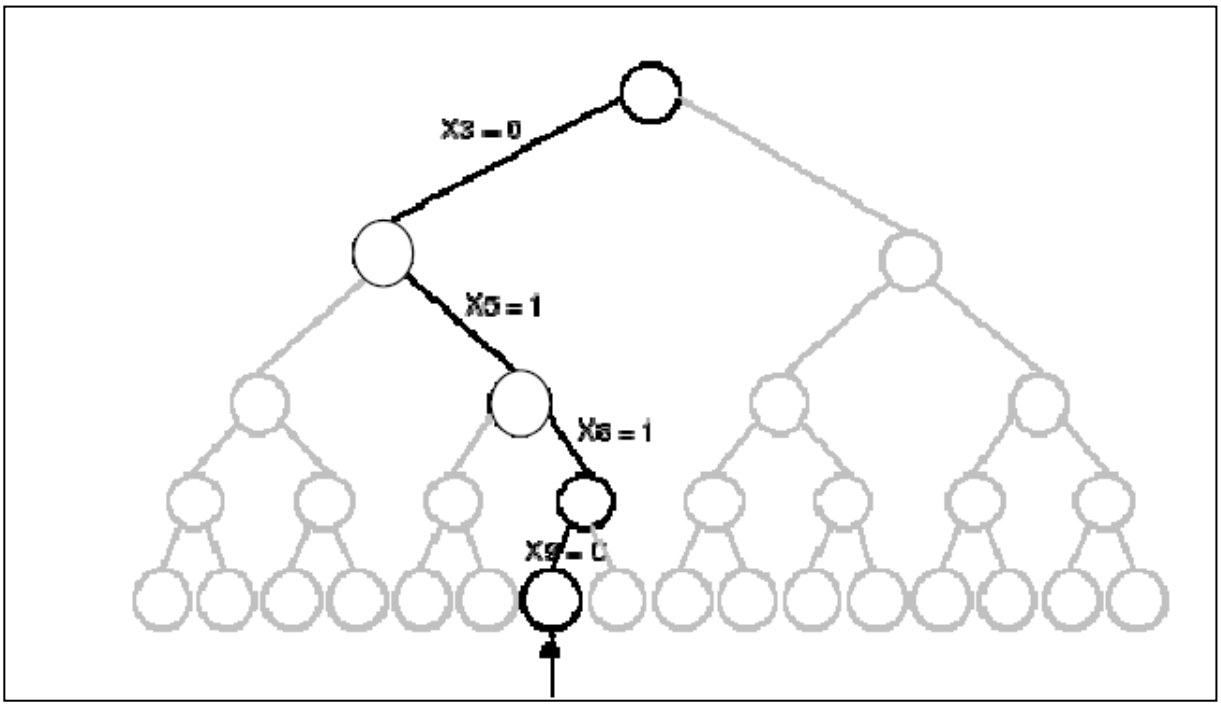


Figure 25 : Exemple d'un nœud Banch & Bound

3.- Placement et Ordonnancement (ou AAS) :

Le placement dans GASPARD

on entend par association , c'est le fait d'associer des taches logicielles (Application) Aux différents composants de l'architecture Hardware, et dans l'association aussi on affecte les données aux mémoires et les chemins de donnée aux fils du réseau.

3.1- Algorithmes de placement et d'ordonnancement :

Le placement et ordonnancement sont des problèmes NP-hard, de ce fait les problèmes de taille importante peuvent être résolu qu'avec des heuristiques qui sont des méthodes de recherche pseudo aléatoires et des techniques d'optimisation multi-objective . Par contre les problèmes à taille limitée peuvent être solutionné par des méthodes déterministes en explorant d'une manière exhaustive tout le domaine des solutions et peuvent retourner un placement qui représente l'optimum théorique « exact » ou plusieurs placements [27] . Lors

de l'exploration de l'arborescence BnB a pour but de chercher à chaque niveau i de l'arborescence le placement de la tache Ti sur l'ensemble des processeurs en deux modes d'exécution du même niveau et suivant le principe AAS ,on va essayer de voir la dominance entre nœuds fils (ou chaque nœud représente Ti sur Pj en mode m) et ceci ne pourra se faire qu'une fois avoir fais Assignation et Affectation en prenant aussi en compte le facteur de cout de communication et la saturation des liens de la grille Torique homogène et après la phase de dominance, on passe à la phase d'élimination des nœuds qui étaient dominées et une fois de plus on relance le processus d'Elimination aussi au sous fils qui résulte du nœud en question où chaque sous Fils devient le nœud en question en allant en Profondeur de l'arborescence BnB et pour chaque niveau i de l'arborescence ,on place Ti en prenant en considération le placement de Ti-1 en Profondeur d'abord jusqu'au i=n (niveau plus bas qui contient des nœuds feuilles) après épuisement de tout le nombre de nœuds ,on aura plusieurs placements non dominés et pour notre Solution BnB on a opter pour la distribution sur une Grille Torique bidirectionnel de SIMD Unit avec des Processeurs Elémentaires ou chaque composant de la Grille Torique est représenté comme un élément extrait du composant matériel GASPARD et avec Inter-répétitions .

Avec des heuristiques constructives qui permettent de construire graduellement des solutions partielles valides jusqu'à atteindre l'optimum par exemple LS (List Scheduling) . Les heuristiques de transformation alternent des solutions existantes en cherchant à élargir l'espace d'exploration de l'espace des solutions tout comme le cas GA (Genetic Algorithms) sans oublier d'autres approches rarement utilisées dans notre domaine comme la programmation mathématique.

Tant dit que pour l'ordonnancement ,on est amener à construire un modèle comprenant les valeurs de temps d'exécution de chaque tâche sur chaque processeur en fonction du voltage ainsi que les performances des réseaux d'interconnexion en fonction du volume de données transportées. Ces valeurs sont fournies à l'algorithme d'optimisation Branch and Bound avec Multi Critères qui propose généralement dans notre application un ou des placements intéressants par rapport à ces performances en vérifiant aussi les contraintes temporelles.

3.2- Maximiser les performances de temps :

La vérification suivant les contraintes temporelles des deadlines hard et le Total Slack pendant l'ordonnancement ce n'est pas les contraintes mémoires mais c'est le temps global pour chaque Sckeduling tout en minimisant le Total Slack de l'application .

3.2.1- La minimisation de la consommation d'énergie :

Vu qu'on peut connaître la consommation d'Energie d'un processeur quand il exécute une instruction (en mode exécution donc le cas du placement dynamique) et quand il est en repos (Cas du placement statique) par conséquent ,on peut essayer de minimiser l'énergie totale consommée en jouant sur ces deux aspects et aussi les communications durant le placement de tâches et la sélection de voltage. D'où on distingue placement de tâche selon l'énergie et le DVS ou SVS des tâches.

3.3- Le Probleme du Placement et d'ordonnancement d'une Application basée sur les Khan Process Network (KPN) :

- Pour représenter le parallélisme et la distribution de l'application, ils ont choisi d'utiliser le modèle des réseaux de processus proposé par Kahn [14,15]. Le problème du placement d'une application basée sur les KPN doit prendre en compte deux paramètres essentiels qui sont le parallélisme intrinsèque et Les communications entre processus doivent être optimisées.

- Par ailleurs pour faire face à la question de l'ordonnancement des KPN qui est cruciale, le modèle théorique suppose que les FIFO ont une taille infinie . Or dans la réalité la mémoire est bornée. Donc la question de l'ordonnancement (statique ou dynamique) et des tailles maximales des FIFOs sont deux autres paramètres à prendre en compte.

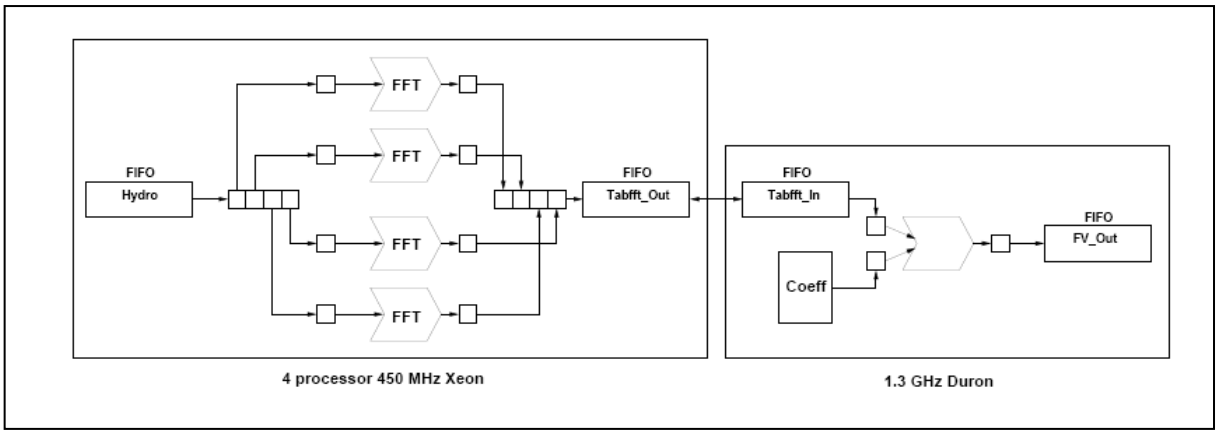


Figure 26 : Example distributed process network

Example distributed process network qui illustre la projection du langage de specification ARRAY-OL avec principe de fusion entre 2 taches évoquées au paragraphe du Chapitre II distribué sur Khan Process Network .

-La fusion qui était présentée au paragraphe du chapitre II, nous a permis de proposer un modèle d'exécution qui casse les limites infinies de certaines données (généralement le temps).

Cette application sera déployée sur deux stations de travail sur le même réseau Ethernet. La première machine était un quadri-processeurs, la nature même de l'application fait qu'il y a une dépendance de données entre les tâches [9].

4.- Les Problèmes D'optimisation Multi-Objectifs :

4.1- Définition :

La principale difficulté que l'on rencontre en Optimisation Multi- Objectif vient du fait que modéliser un problème sous la forme d'une équation unique peut être une tâche difficile.

L'optimisation multi-objectif autorise ces degrés de liberté qui manquaient en optimisation mono-objectif. Cette souplesse n'est pas sans conséquences sur la démarche à suivre pour chercher un optimum à notre problème enfin modélisé. La recherche ne nous donnera plus une solution unique mais une multitude de solutions. Ces solutions sont appelées solutions de Pareto et l'ensemble de solutions que l'on obtient à la fin de la recherche est la surface de compromis.

C'est après avoir trouvé les solutions du problème multi-objectif que d'autres difficultés surviennent : il faut sélectionner une solution dans cet ensemble. La solution qui sera choisie par l'utilisateur va refléter les compromis opérés par le décideur vis-à-vis des différentes fonctions objectif [23].

Le décideur étant "humain", il va faire des choix et l'un des buts de l'optimisation multi-objectif est de modéliser les choix du décideur ou plutôt ses préférences. Pour modéliser ces choix on pourra s'appuyer sur deux grandes théories :

-la théorie de l'utilité multicritères

- la théorie de l'aide à la décision .

Ces deux théories ont des approches différentes [23].

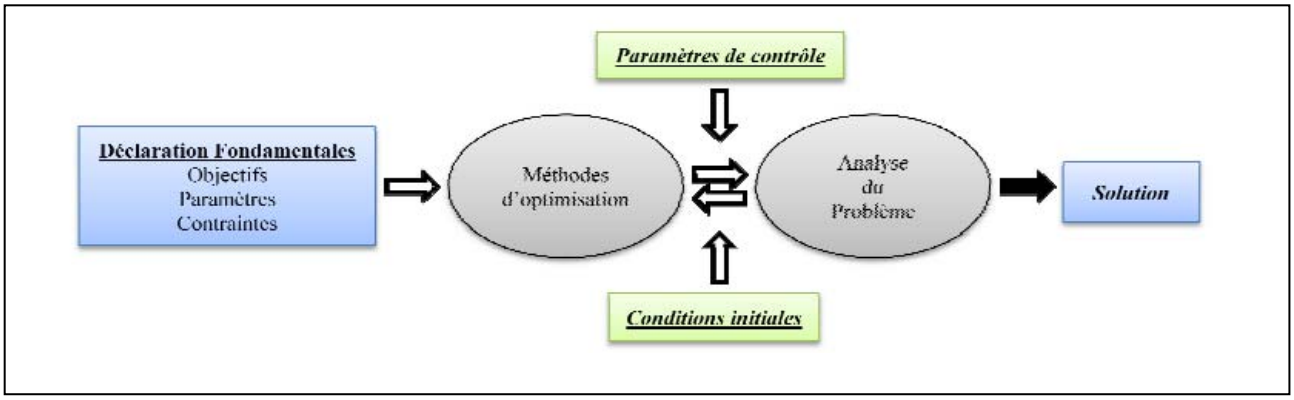


Figure 27 - Illustration de la disposition des divers composants d'une méthodologie

4.2- Problème multi-objectif :

Un problème multi-objectif ou multicritère peut être défini comme un problème dont on recherche l'action qui satisfait un ensemble de contraintes et optimise un vecteur de fonctions objectives. Les problèmes d'optimisation ont en général plusieurs solutions car la définition d'un optimum ne peut pas être établie.

Par définition mathématique :

Une action (ou un vecteur de décisions) est notée par :

$$x = (x1, x2,\ldots,xn) \quad \ldots\ldots\ldots\ldots\ldots \quad 1)$$

avec xi les variables du problème et n le nombre de variables.

Les contraintes sont exprimées:

$$gi(x) \text{ avec } i = 1,\ldots,m \quad \ldots\ldots\ldots\ldots... \quad 2)$$

avec m le nombre de contraintes ;

Le vecteur de fonction objectif est noté f :

$$f(x)=(f1(x),\ldots\ldots,fk(x)) \quad \ldots\ldots \quad 3)$$

avec fi les objectifs ou critères de décisions et k le nombre d'objectifs.

Les objectifs sont des fonctions de minimisation (dans le cas de fonction f de maximisation, il suffit de minimiser f).

Un problème d'optimisation recherche l'action x* telles que les contraintes gi(x*) soient satisfaites pour i=1,…,m et qui optimise la fonction f : f(x*) = (f1(x*),…….,fk(x*)) ;
L'union des domaines de définition de chaque variable et les contraintes définies en 2) forment un ensemble E que nous appelons l'ensemble des actions réalisables.

5- Modèle :

5.1 Définition1 :

Les Communications entre Tache Diffuseur T1 et les Composants répétitives sont représentés par un Graphe Orienté.
- Le graphe d'application (TG : Task Graph) est un graphe orienté G(T,E) où chaque sommet $T_i \in T$, avec i>1 représente un module (tâche) de l'application et l'arc Orienté $(T1,Ti)$ noté $e_{ij} \in E$ représente les communications entre les modules $T1$ et Ti. Le poid de l'arc e_{ij} noté par Q_{ij} représente le volume de Communications entre les modules $T1$ et Ti.

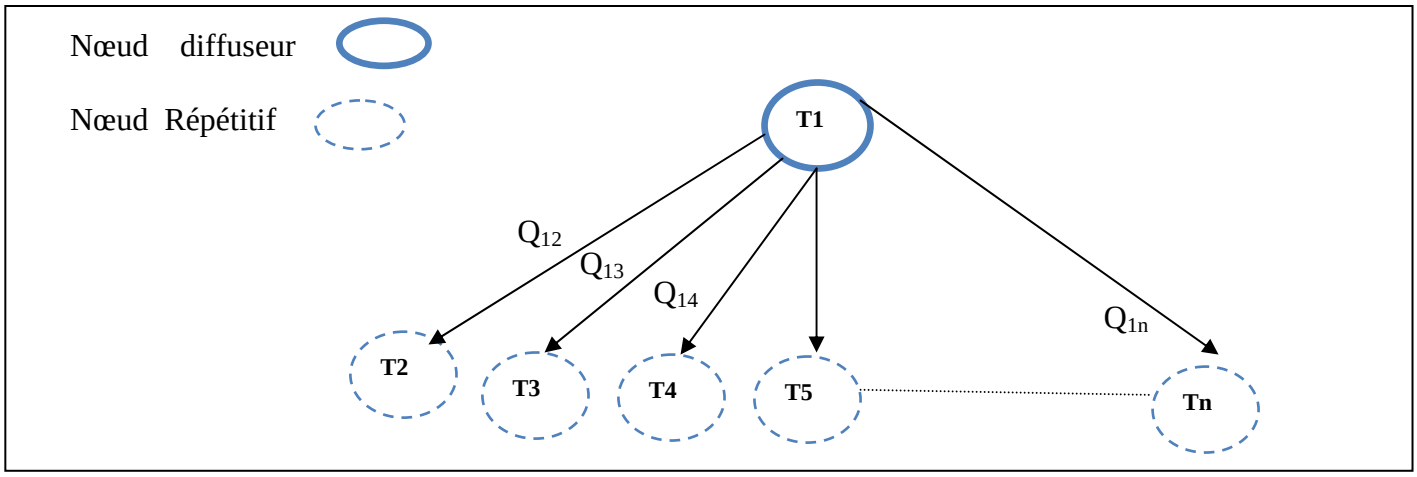


Figure 28: TG Repetitif .

5.2 Definition 2

le graphe d'architecture (NoC Topology graph) est un graphe orienté P(U,F) où chaque sommet $u_i \in U$ représente un noeud de la topologie et l'arc (u_i, u_j) noté par $f_{ij} \in F$ représente un lien physique direct ou bien dans les extrémités par des tores entre les éléments de u_i et u_j de l'architecture. Le poids de l'arc f_{ij} noté par bw_{ij} représente la bande passante disponible à travers le lien physique f_{ij}.

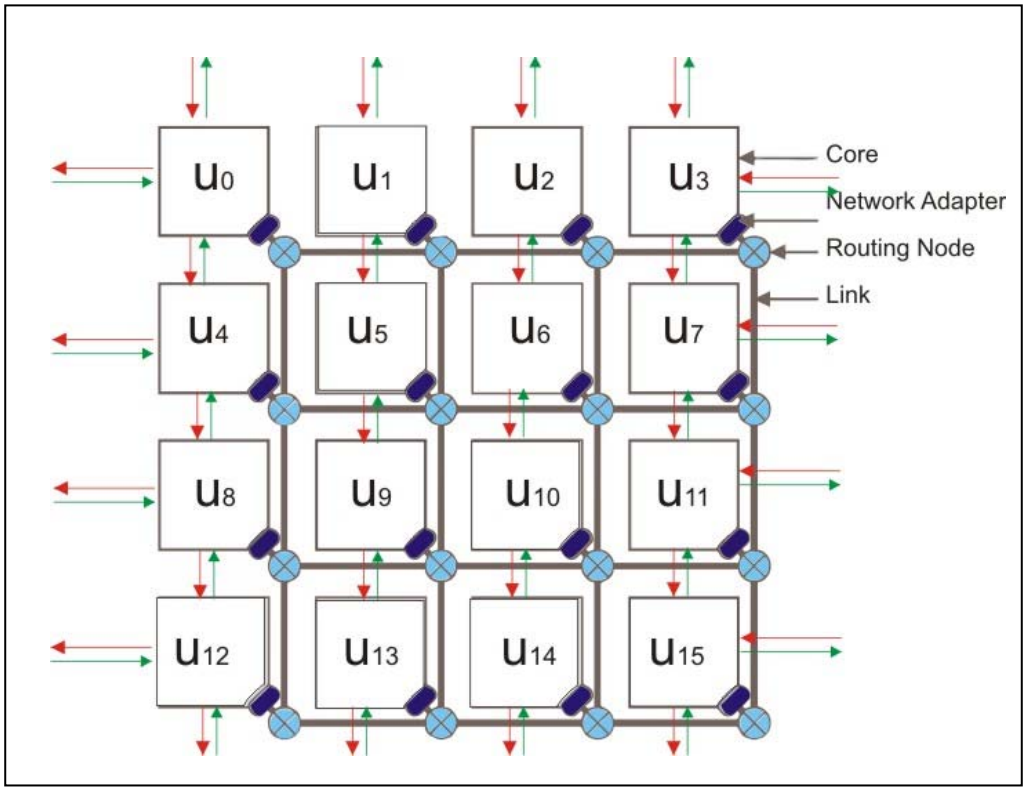


Figure 29 :Grille Torique bidirectionnelle

5.3 Nœud répétitif :

Ces nœuds permettent de représenter les répétitions de données parallèles des sous nœuds. Ces répétitions prennent comme entrées/sorties des parties de tableaux en entrée et en sortie du nœud répétitif. Ces parties de tableaux ont toutes la même forme et sont appelés Motifs (patterns). Les Motifs en sortie sont produits en appliquant le sous-nœud sur le Motif du tableau en entrée. Toutes les répétitions des sous-nœuds sont indépendantes et définies dans un espace de répétition Q. Pour chaque entrée/sortie du nœud répété, on a les informations nécessaires. Dans le chapitre 2, on a décrit comment on obtient ces informations grâce à Array-Ol.

6.- Mapping des Tâches Répétitives :

Le mapping ou l'affectation des communications au réseau consiste à placer une communication entre deux tâches t_i et t_j représentée par l'arc $e_{ij} \in E$ sur le réseau de communication. Ce placement consiste à trouver le chemin qui souvent est constitué de plusieurs liens physiques contigus utilisés pour envoyer les données de t_i à t_j.

Mais avant de considérer le mapping dans sa globalité ,on doit tout d'abord proposer une solution pour un mapping multi-objectif d'un seul graphe TG sur un seul graphe NT dans un même niveau.

Le mapping du graphe d'application G(T,E) sur le graphe d'architecture P(U,F) est défini par la fonction mapping map: $map(T_i)= u_j / T_i \in T$ et $u_j \in U$ (Le mapping est défini quand $|T| \geq |U|$).

7.Formulation Mathématique :

Pour un TG chaque nœud ou sommet représente une tâche avec ses caractéristiques ou propriétés. Soit $T=\{t_1,t_2,.....t_n\}$ l'ensemble des tâches représentées par l'ensemble des sommets V dans TG. $P= \{p_1,p_2,.....p_s\}$ l'ensemble des processeurs représentés par l'ensemble des nœuds U dans NT. On considère que chaque processeur p peut s'exécuter à des modes différents m_1, m_2 or m_3.

Comme variable de décision on prend des variables binaires X^m_{ij} tels que [07].

X^m_{ij}= 1 si la tâche i est placée sur le processeur j et s'exécute au mode m, o sinon.

d'^m_{ij} = la durée d'exécution de la tâche i placée sur p_j s'exécutant au mode m (sans prendre en compte les communications) .

$d'^m_{ij}=WCN^m_{ij}/f^m_j$ où WCN^m_{ij} est le nombre de cycles nécessaires à la tâche i pour s'exécuter sur le processeur j au mode m.

f^m_j est la fréquence d'horloge du processeur j au mode m.

dl_i est le deadline de la tâche i. C'est le temps auquel la tâche i doit être terminée. Ainsi $dl_{final}=dl_n$ est le deadline de la dernière tâche n et aussi celui de l'application.

Q_{ij} est le volume de données échangées entre les tâches i et j.

$d'Q^m_{ijpq}$ est la durée des communications entre les tâches i et j si elles sont placées respectivement sur les processeurs p et q au mode m.

q^m_{pq} est la durée de communication unitaire (bit, octet ou paquet) entre p et q au mode m.

e^m_{pq} est l'énergie consommée par une unité de donnée (bit, octet ou paquet) durant son transfert de p à q au mode m.

S'il n'existe pas un lien direct entre p et q soit $\mu(p,q)= \{(p_i,p_j)\}$ le chemin (succession de liens) reliant p à q [24].

Alors la durée d'utilisation des liens est $d'Q^m_{ijpq} = \sum Q_{ij} * q^m_{plpk}$ où $(p_l,p_k) \in \mu(p,q)$

La consommation d'énergie due à la communication des tâches i et j si elles sont placées sur les processeurs p et q au mode m en utilisant le même chemin est :

$$E'^{m}_{ijpq}=\sum Qij*e^{m}_{plpk} \quad \text{où} \quad (p_i,p_j)\in \mu(p,q)$$

Donc le temps total d'exécution d'une tâche i en prenant en considération les communications est :

$$D_i = \sum_{m=m1}^{mn} d'^{m}_{ip} + \sum_{j=1}^{N}\sum_{p=1}^{S}\sum_{q=p+1}^{S}\sum_{m=m1}^{mn} X^{m}_{ip} * X^{m}_{jq} * dQ^{m}_{ijpq} \qquad j\neq i$$

Cette équation peut être simplifiée car p est fixe et on considère par hypothèse que les communications intra-processeur sont négligeables. Pour un i fixé, $\sum X^{m}_{ip} = 1$.

Pour une tâche i, si D_i^{start} est son temps de début, alors son temps de terminaison est D_i^{fin}, il est donné par la formule suivante :

$$D_i^{fin} = D_i^{start} + D_i$$

L'ordonnancement est obtenu par LS où le début de l'application correspond avec le début de sa première tâche t_1 et sa fin correspond à la fin de la dernière tâche t_n. Ainsi, pour la tâche t_i son temps de début D_i^{start} correspond au temps de fin de la dernière des tâches qui la précèdent. Donc la liste LS est obtenue en utilisant la stratégie ASAP (As Soon As Possible).

$D_n^{fin} = D_n^{start} + D_n$ est le temps de fin de la dernière tâche t_n.

Ce temps est donc le temps de fin de l'application D^{fin} qui est donné par la formule suivante :

$$D^{fin} = D_n^{start} + D_n$$

Donc si D_{start} est le temps de début de l'application, qui correspond au temps de début de la première tâche D_1^{start}, et D_{fin} le temps de fin de l'application qui aussi celui de la dernière tâche D_n^{fin}, alors la durée d'exécution totale de l'application est :

$$D= D_{fin} - D_{start}$$

C'est ce temps qu'on doit minimiser et s'assurer qu'il vérifie la contrainte temps réel hard, c'est-à-dire : $D \leq dl_{final}$.

La consommation d'énergie totale est donnée par la formule suivante en prenant en considération la consommation des processeurs, des liens, des switch et des routeurs.

$$E=\sum_{i=1}^{N}\sum_{p=i+1}^{S}\sum_{m=m1}^{mn} X^{m}_{ip} * ET^{m}_{ip} + \sum_{i=1}^{N}\sum_{j=i+1}^{N}\sum_{p=1}^{S}\sum_{q=p+1}^{S}\sum_{m=m1}^{mn} X^{m}_{ip} * X^{m}_{jq} * E^{m}_{ijpq} \quad j \neq i \quad (*)$$

ET^{m}_{ip} est la consommation de la tâche i affectée au processeur p au mode m.

X^{m}_{ij}= 1 si la tâche i est placée sur le processeur j et s'exécute au mode m, o sinon.

$d'^{m}_{ij}=WCN^{m}_{ij}/f^{m}_{j}$ où WCN^{m}_{ij} est le nombre de cycles nécessaires à la tâche i pour s'exécuter sur le processeur j au mode m. là aussi WCN^{m}_{ij} varie uniquement selon le mode d'exécution des processeurs.

f^{m}_{j} est la fréquence d'horloge du processeur j au mode m. Elle est aussi la même pour tous les processeurs à un même mode.

Ainsi si dans un motif, après ordonnancement, la tâche k est la dernière dans l'exécution et dl_k est son deadline alors $dl_{finalmotif} = dl_k$. Il est le temps avant lequel se termine le motif et qui correspond au temps de fin de la dernière tâche du motif.

En essayant de minimiser la consommation d'énergie tout en exploitant au maximum les ressources. Le critère qu'on a adopté ici, en premier, pour l'exploitation maximal des ressources est $total_{slack}$ Où $total_{slack}$ correspond au temps de repos ou relâchement des ressources. C'est ce temps qu'on doit minimiser pour augmenter l'exploitation des ressources. On voit bien que ce critère est conflictuel avec le deuxième objectif qui correspond à la minimisation de la consommation d'énergie. En effet exploiter plus les ressources nous amène à consommer plus d'énergie. La minimisation de la consommation d'énergie revient à optimiser l'objectif donné par la formule (*) .Donc si on a S ressources d'exécution et N tâches constituant le motif. Notant par D^{S}_{n} la durée d'exécution d'un motif constitué de N tâches et s'exécutant sur S processeurs.

Alors $T_{dis}= D^{S}_{n} * S$ donne le temps de disponibilité des ressources.

Le temps d'occupation T_{oc} des S processeurs par les N tâches du motif peut être obtenu à par :

$$T_{oc}= \sum_{i=1}^{i=N} D_i$$

Alors le temps total de relâchement (ou des slack) des S processeurs utilisés pour l'exécution d'un motif de N tâches est obtenu par :

$$total_{slack} = T_{dis} - T_{oc}$$

C'est ce temps qu'on doit minimiser en jouant sur le nombre de tâches constituant le motif.
Pour généraliser l'expression notant par $total^{n,s}_{slack}$ le temps total de relâchement dû au placement de n tâche sur S processeurs du NoC.

Etant donnée que le nombre de processeurs est donnée dans une architecture régulière et qui est souvent raisonnable, alors le nombre de tâches constituant le motif est lui aussi raisonnable par rapport au nombre total des tâches répétitives. Pour ce placement on a proposé une approche basée une méthode exacte pour atteindre les objectifs [24].

Algorithme de placement des tâches répétitives [24]

1. Début ;
2. Introduire le nombre de PE, S et la matrice de topologie ;
3. Introduire les caractéristiques de l'architecture et des tâches répétitives ;
4. $Motif_{taches} = 0$; $total_{slack} = \infty$;
5. Pour N=S à N=S*S faire
6. utiliser la méthode exacte pour trouver le placement au mode max des processeurs
7. Utiliser l'algorithme d'optimisation des communications
8. calculer $total^{N}_{slack}$
9. Si $total^{N}_{slack} < total_{slack}$ alors $total_{slack} = total^{N}_{slack}$ et $Motif_{taches} = N$ sinon rien fsi
10. fin pour
11. faire changer le mode des processeurs
12. Calculer pour chaque placement la durée d'exécution et l'énergie
13. Retenir le front Pareto ou les solutions non dominées
14. Fin

8- Génération du plus court Chemin :

Le problème de génération de chemin consiste à trouver un chemin entre deux sommets tels que la somme des arcs le constituant soit minimale. La méthode NORD-WEST proposé dans le cadre de notre projet est basée sur la distance ha miltonienne avec la seule différence lors du parcours c'est la prise en compte que certains liens peuvent être déjà occupés.

Nous cherchons à retrouver le chemin le plus court dans une grille torique bidirectionnelle (chaque nœud a quatre voisins même ceux qui se retrouvent sur les limites de la grille comme le (0,0)) . Le modulo évoqué est nécessaire pour passer d'un nœud sur la limite de la grille à son voisin qui se retrouve sur l'autre limite (par exemple un nœud sur la limite haute aura un voisin sur la limite sud de la grille).

-S'il n'existe pas un lien direct entre p et q soit $\mu(p,q)= \{(pi,pj)\}$ le chemin (succession de liens) reliant p à q alors la durée d'utilisation des liens est $d'Q^{m}_{ijpq} = |\mu(p,q)| * q^{m}$ $|\mu(p,q)|$ est la longueur du chemin $\mu(p,q)$.

9.Mapping tâches repetitive avec BnB parallel [20] :

L'architecture du down scaler avec 4 MIPS code l'image en parallèle.

La figure suivante illustre une application Encoder H263 Encoder sur une HW Architecture du Down Scaler .

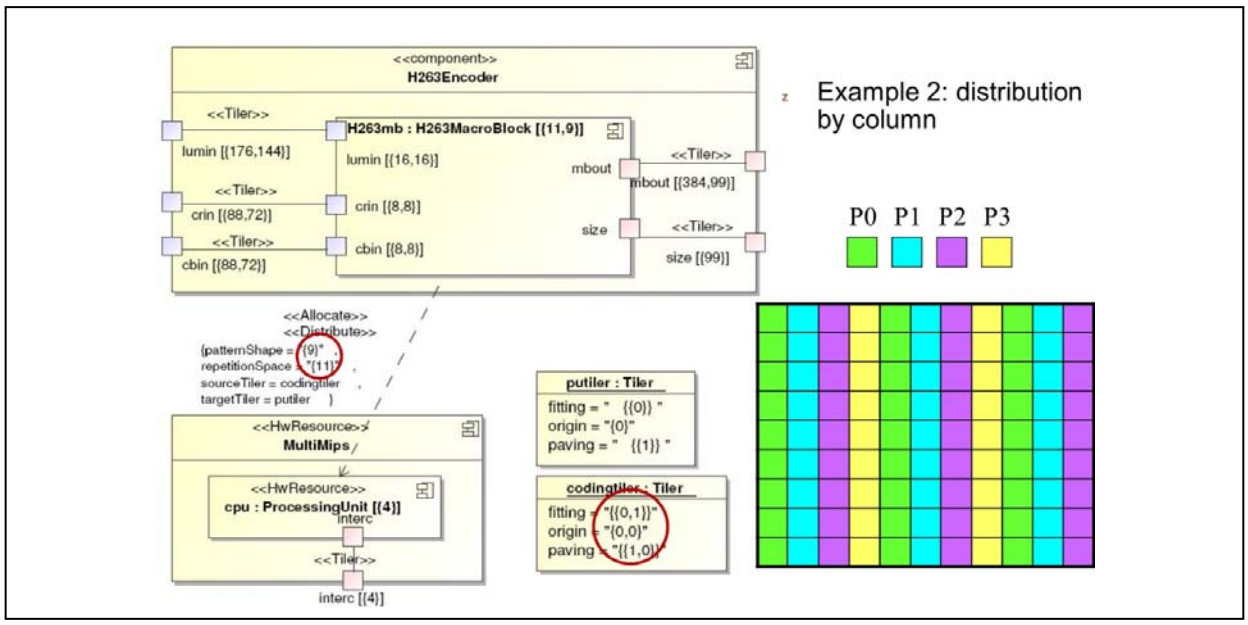


Figure 30 : application Encoder H263 Encoder sur une HW Architecture du Down Scaler .

l'exemple d'application du H263 est divisé en 9 X 11 macro blocs .c'est un composant composé les 3 tilers signifient juste qu'il y a 3 inputs différents .En effet la plateforme d'exécution est constituée elle aussi d'élément d'exécution et de communications identiques. C'est pour cette raison qu'on désigne ce type d'architecture par l'appellation « régulière ».

Ce type d'application régulière est très bien décrit et spécifiée par le langage array-ol présenté auparavant dans le chapitre II .

L'idée derrière le motif est basée sur deux objectifs de base, premièrement faire de la technique d'AAS indépendamment du nombre total des tâches. Deuxièmement, minimiser la

complexité du placement. Dans les applications DSP, on a généralement un flot de données infini qui sont traitées par des tâches répétitives. Considérer cette infinité de tâches répétitives lors de la recherche du placement optimal va accroître la complexité de traitement du processus AAS. C'est pour cette raison qu'on génère des motifs et pour générer le placement total (AAS) de toutes les tâches ($Total_{taches}$) ,on a qu'à répéter le placement du motif X fois Ainsi un placement de motif est généré sans prendre en considération le nombre total de tâches .

De plus le choix du motif idéal est déterminer après n itérations en s'inspirant du BnB Parallèle dans le cadre de notre Projet qui nous fournit un motif optimal qui présente un bon compromis Taches/Processeurs tout en Prenant en compte Architecture HW cible de SIMD Unit Grille bidirectionnelle torique.

Cet Algorithme d'Optimisation BnB Parallel verifient les Performances et les contraintes temporelles.

Comme résultat de son placement ,on obtient le nombre de tâches qui le constituent ($Motif_{taches}$) et X peut être calculé selon la formule suivante : $X = Total_{taches} / Motif_{taches}$.

10- FRONT PARETO [28] :

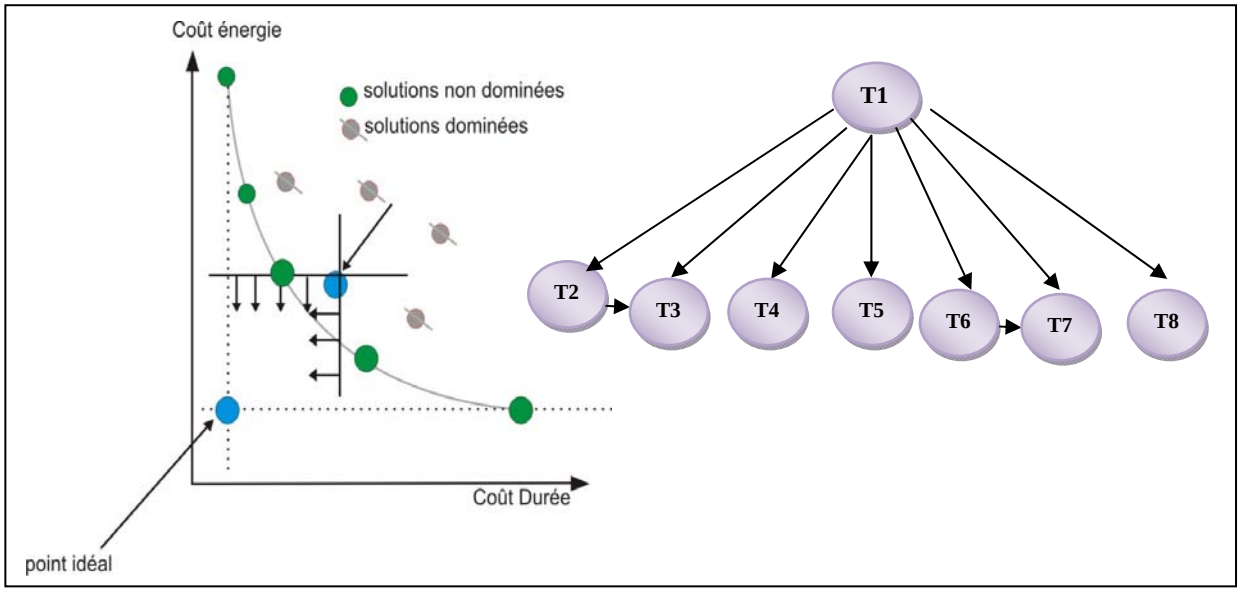


Figure 31: Exemple illustrant placement de motif 8 Tâches sur 5 Processeurs.

DIAGRAMME DE GANTT :

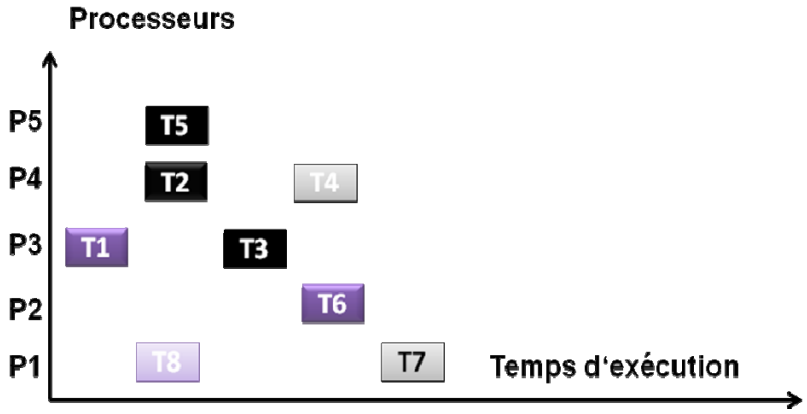


Figure 32 : Diagramme Gantt qui illustre la figure 31

La figure 32 montre le placement et l'ordonnancement des différentes tâches de la figure 31.

Conclusion

Le placement sur les NOC désigné par Assignation, affectation et ordonnancement (AAS) consiste à atteindre certains objectifs comme le temps réel, consommation d'énergie, la mémoire, qui sont souvent recherchés ensembles parce qu'on ne peut minimiser le temps de calcul global sans minimiser le coût de communication global, ou augmenter les performances du NOC sans penser à minimiser la perte de l'énergie, ce qui nous permet de conclure que ce problème ne peut être solutionné en cherchant des objectifs séparément mais d'essayer d'au moins d'atteindre quelques objectifs pertinents ensembles c'est ce qu'on appelle l'optimisation multi-objectifs.

CHAPITRE IV

IMPLEMENTATION
ET MISE EN ŒUVRE

1. Introduction :

Dans notre contribution nous avons constaté qu'il faut construire un modèle comprenant les valeurs de temps d'exécution de chaque tâche sur chaque processeur en fonction du voltage ainsi que les performances des réseaux d'interconnexion en fonction du volume de données transportées. Ces valeurs sont fournies à l'algorithme Branch & Bound d'optimisation Parallèle multicritères qui propose un placement qui représente l'optimum théorique « exact » ou plusieurs placements intéressants par rapport à ces performances et aussi en vérifiant le critère des contraintes temporelles pour le choix du motif Optimal . Dans ce chapitre, on passe aux différentes étapes de conception et développement du logiciel.

2.Méthodologie de conception UML.

UML (Unified Modeling Language, que l'on peut traduire par "langage de modélisation unifié) est une notation permettant de modéliser un problème de façon standard. Ce langage est né de la fusion de plusieurs méthodes existant auparavant, et devenu désormais la référence en terme de modélisation objet.

2.1. La notion d'objet

La programmation orientée objet consiste à modéliser informatiquement un ensemble d'éléments d'une partie du monde réel (que l'on appelle domaine) en un ensemble d'entités informatiques. Ces entités informatiques sont appelées objets. Il s'agit de données informatiques regroupant les principales caractéristiques des éléments du monde réel (taille, la couleur,…) .

2.2. Les méthodes objets

La modélisation objet consiste à créer une représentation informatique des éléments du monde réel auxquels on s'intéresse, sans se préoccuper de l'implémentation, ce qui signifie indépendamment d'un langage de programmation. Il s'agit donc de déterminer les objets présents et d'isoler leurs données et les fonctions qui les utilisent. Pour cela des méthodes ont été mises au point. Entre 1970 et 1990, de nombreux analystes ont mis au point des approches orientées objets, si bien qu'en 1994 il existait plus de 50 méthodes objet. Toutefois seules 3 méthodes ont véritablement émergé :

- *La méthode OMT de Rumbaugh,*
- *La méthode BOOCH'93 de Booch,*
- *La méthode OOSE de Jacobson (Object Oriented Software Engineering*).

3.L'environnement d'implémentation

Notre choix est basé sur deux critères fondamentaux :

- La programmation sous Windows qui offre beaucoup d'avantages.
- La programmation java en utilisant eclipse.

La programmation sous Windows

Windows est bien plus qu'une interface graphique conviviale, il s'agit d'une révolution pour les programmeurs. Les applications développées sous Windows communiquent entre elles par des événements des envois de messages. Elles se partagent les ressources de l'ordinateur et offrent à l'utilisateur une interface iconique intuitive.

La programmation sous Windows fait bénéficier d'un grand nombre d'avantages :

- Utilisation d'une mémoire par l'exploitation de la mémoire paginée et l'utilisation du disque comme supports d'échange (mémoire virtuelle).
- Une interface utilisateur prédéfinie, elle permet de concevoir des logiciels performants sans se préoccuper des détails de création des composants de l'interface. Elle permet aussi d'obtenir des programmes faciles à manipuler car elle utilise une présentation standard connue par la majorité des utilisateurs, ce qui réduit le temps d'apprentissages du logiciel.
- L'environnement Windows se préoccupe de gérer les détails matériels de bas niveau de la machine ainsi que tous les périphériques (la carte graphique, imprimante…). Les programmes développés sous Windows bénéficient d'une part de l'environnement MS-DOS et d'autre part d'une interface utilisateur graphique cohérente. Cette interface est orientée souris avec menu déroulant, fenêtres et boite de dialogue.

Le langage de programmation java

Java, langage orienté objet, développé par **SUN**(1991). Le but de Java à l'époque était de constituer un langage de programmation pouvant s'intégrer dans les appareils électroménagers afin de pouvoir les contrôler, de les rendre interactifs, et surtout de permettre une communication entre les appareils. La syntaxe est proche de celle du C, mais Java n'est pas une surcouche du C et la syntaxe est beaucoup plus claire que celle du C++. Java présente beaucoup d'avantages : sécurisé, robuste, indépendant du matériel, portable ... Le byte-code, assurant une portabilité complète vers de très nombreux systèmes. L'importance de l'API qui offre tous les services de base, notamment pour la construction des interfaces graphiques.Son adaptabilité dans de nombreux domainesparmis eux les systèmes embarqués.

L'environnement eclipse

Le projet Eclipse trouve son origine au sein de la société IBM, qui a décidé en 2001 de mettre à disposition de la communauté Open Source l'ébauche d'une plate-forme de développement ouverte, entièrement écrite en Java, capable d'intégrer des extensions adaptées à diverses activités (débogage, modélisation, interfaces graphiques...). L'objectif étant de créer un environnement de développement intégré polyvalent, extensible, capable de travailler avec n'importe quel langage de programmation. Et ce grâce à une architecture basée sur la notion de plug-in, module fonctionnel qui se rattache à la plate-forme via un mécanisme appelé point d'extension. La base de cet environnement est constituée par l'Eclipse platform, qui comprend : la Platform runtime, chargée du démarrage de l'environnement et de la gestion des plug-in ; SWT, la librairie graphique de base de l'environnement ; JFace, une librairie graphique de plus haut niveau ; et enfin le Workbench, couche graphique qui permet de manipuler des composants (sous la forme de vues, d'éditeurs, de perspectives, etc.). L'ensemble des outils de développement (Java, C/C++...),

4.Les diagrammes UML :

Parmis les diagrammes UML qu'on a utilisé lors de la conception de notre application sont :

-diagramme de classe
-diagramme d'activité
-diagramme de séquence

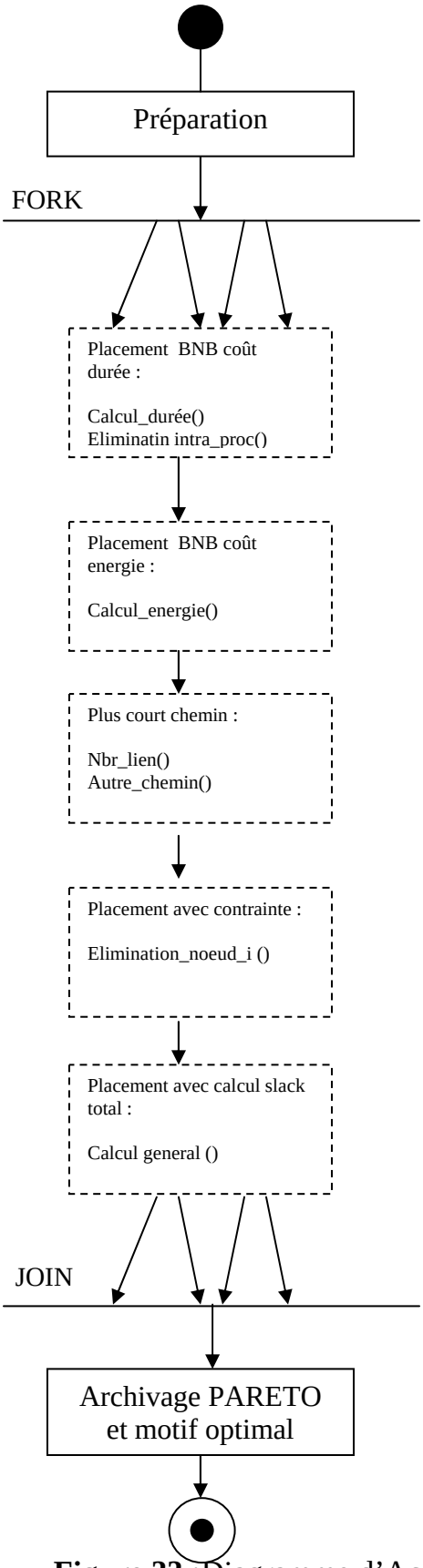


Figure 33 : Diagramme d'Activité**.**

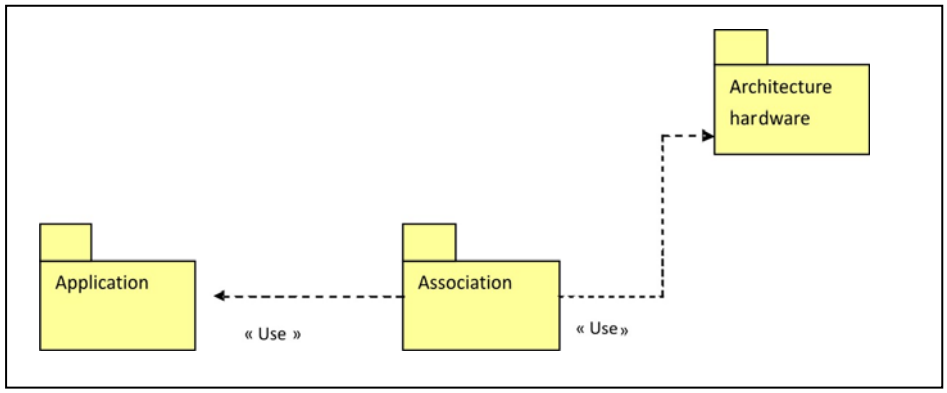


Figure 34 :Diagramme Classe 1 .

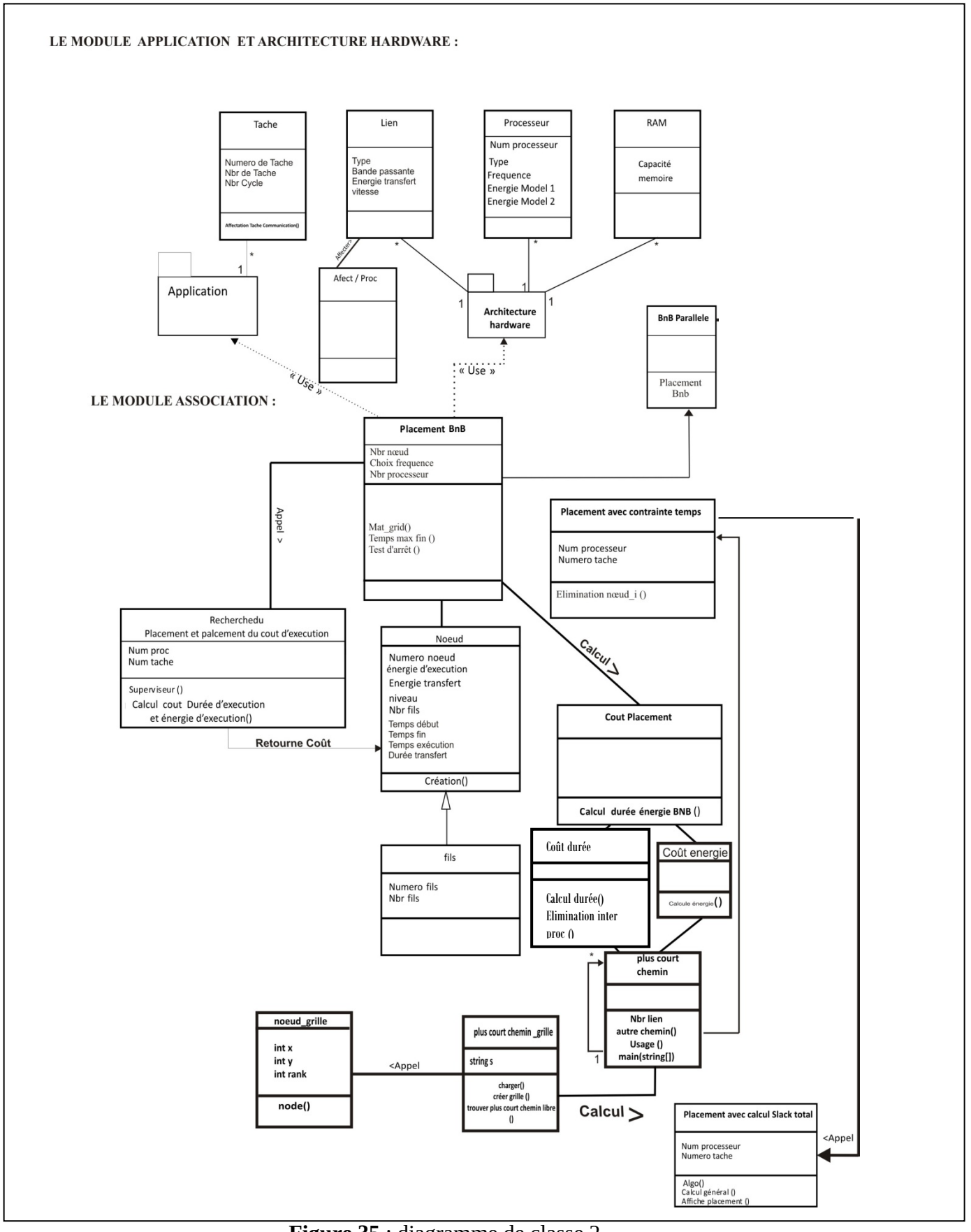


Figure 35 : diagramme de classe 2

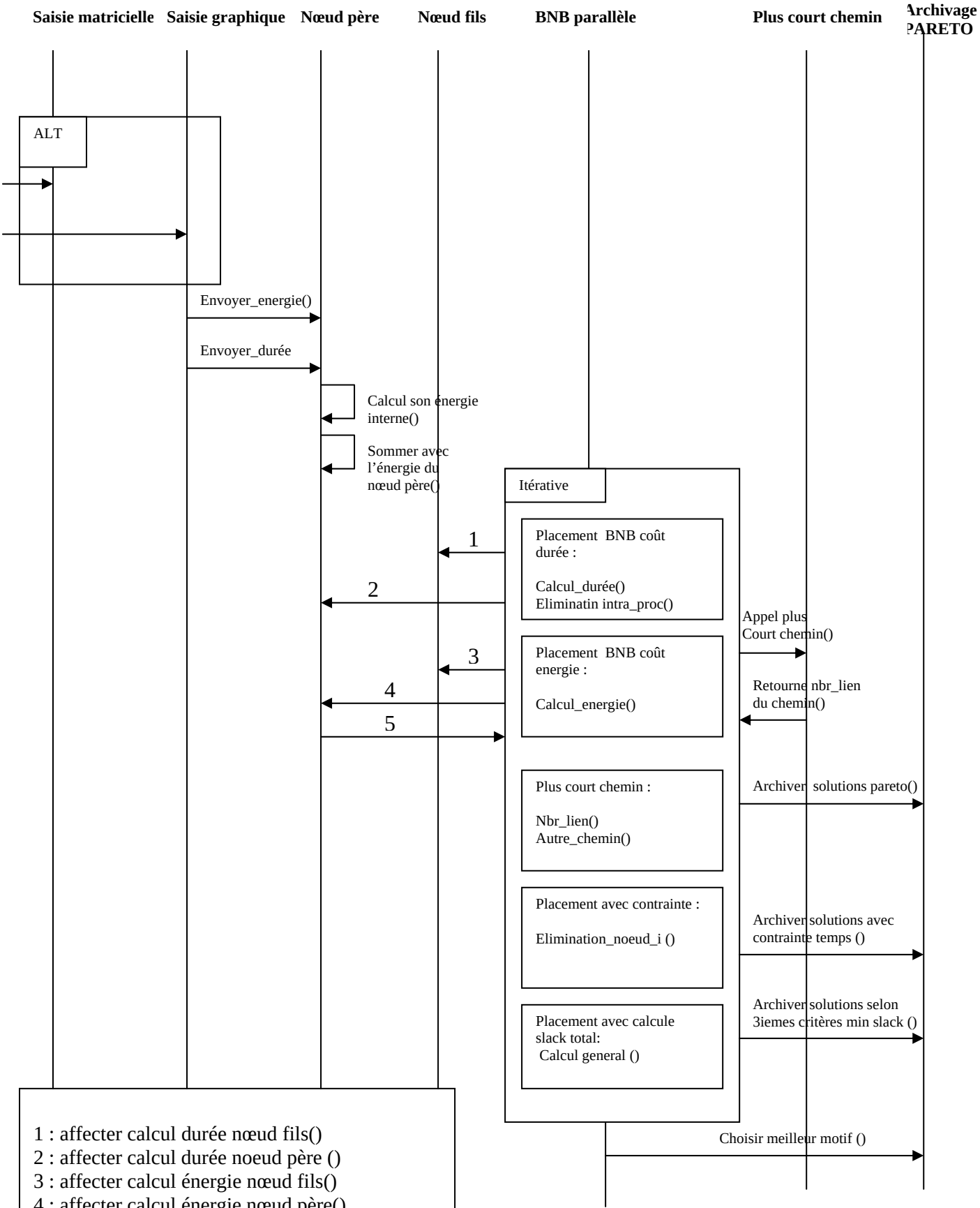


Figure 36 :Diagramme de Sequence de l'application .

5.L'algorithme exact pour les applications DSP (LR) :

Ce qui a était avancé dans le chapitre précédent, le mapping des tâches répétitives LR(Localement Regulier) avec Les problèmes de placement et d'Ordonnancement .

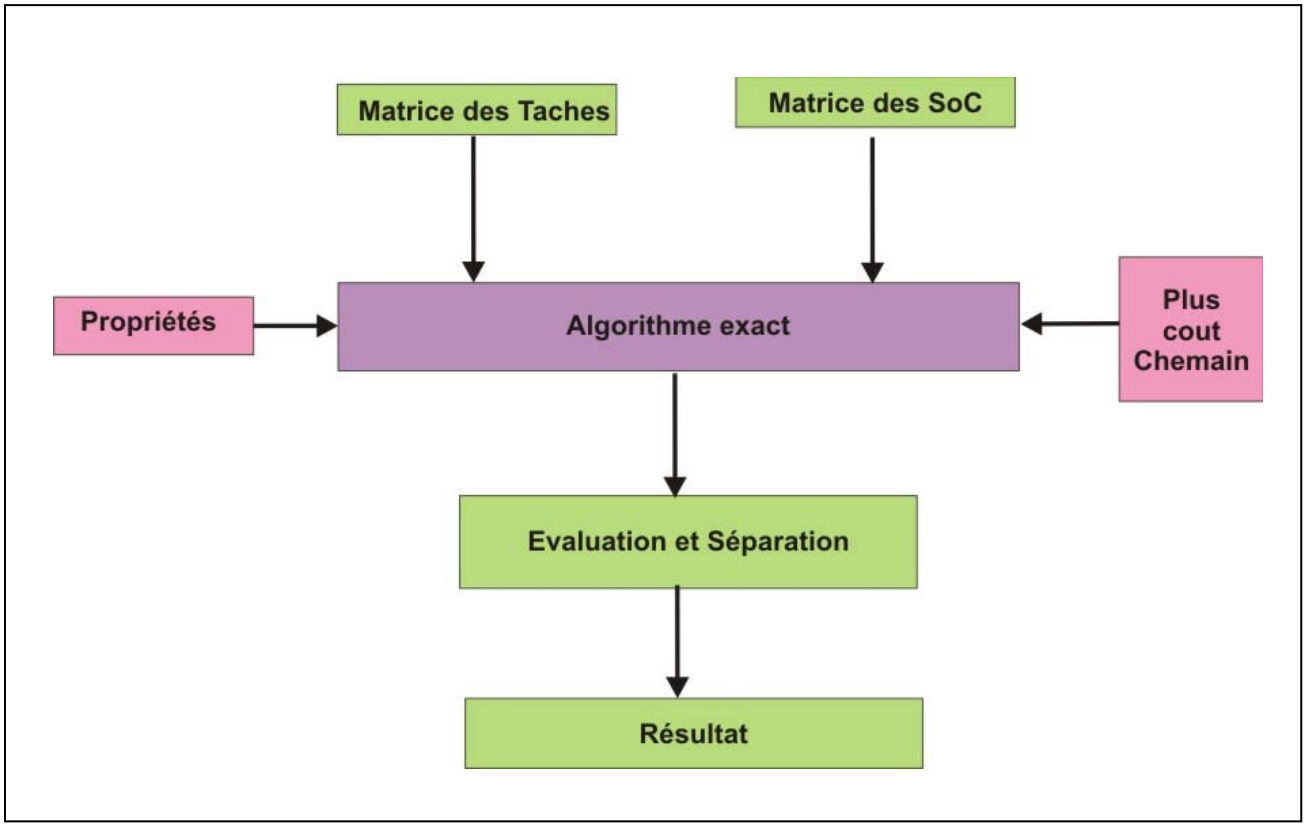


Figure 37 illustrant le lien entre l'algorithme exact et plus court chemin

6.DESCRIPTION DE L'APPLICATION :

6.1/ Saisie Matricielle :

Dans le mode graphique de notre application, on trouve dans la première fenêtre quatre champs textuel ou on peut introduire la taille de la grille torique , le nombre de processeurs ,le nombre des taches et le nombre Totale de taches répétitives de l'application, et afin d'accéder aux Caractéristiques du graphe d'application et aussi caractéristiques de l'Architecture Hardware cible qui est Grille Torique bidirectionnelles Homogène on doit appuyer sur le bouton Suivant.

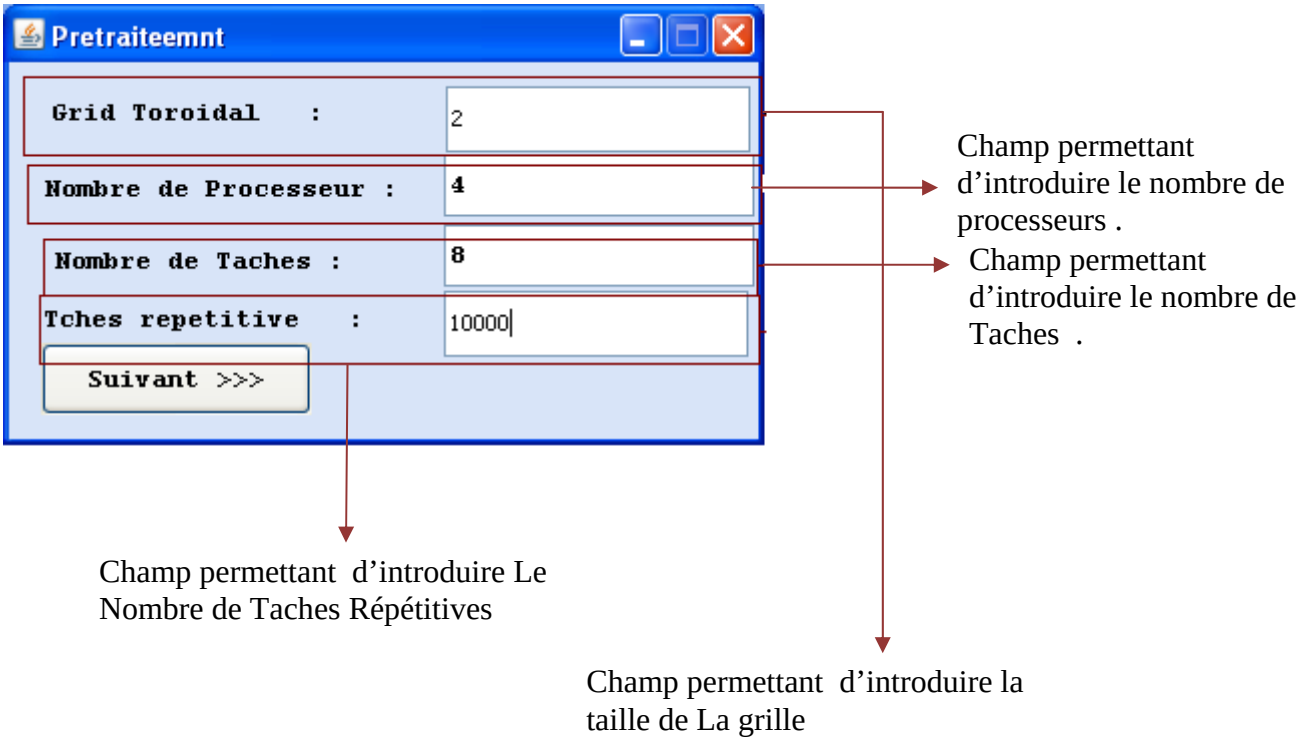
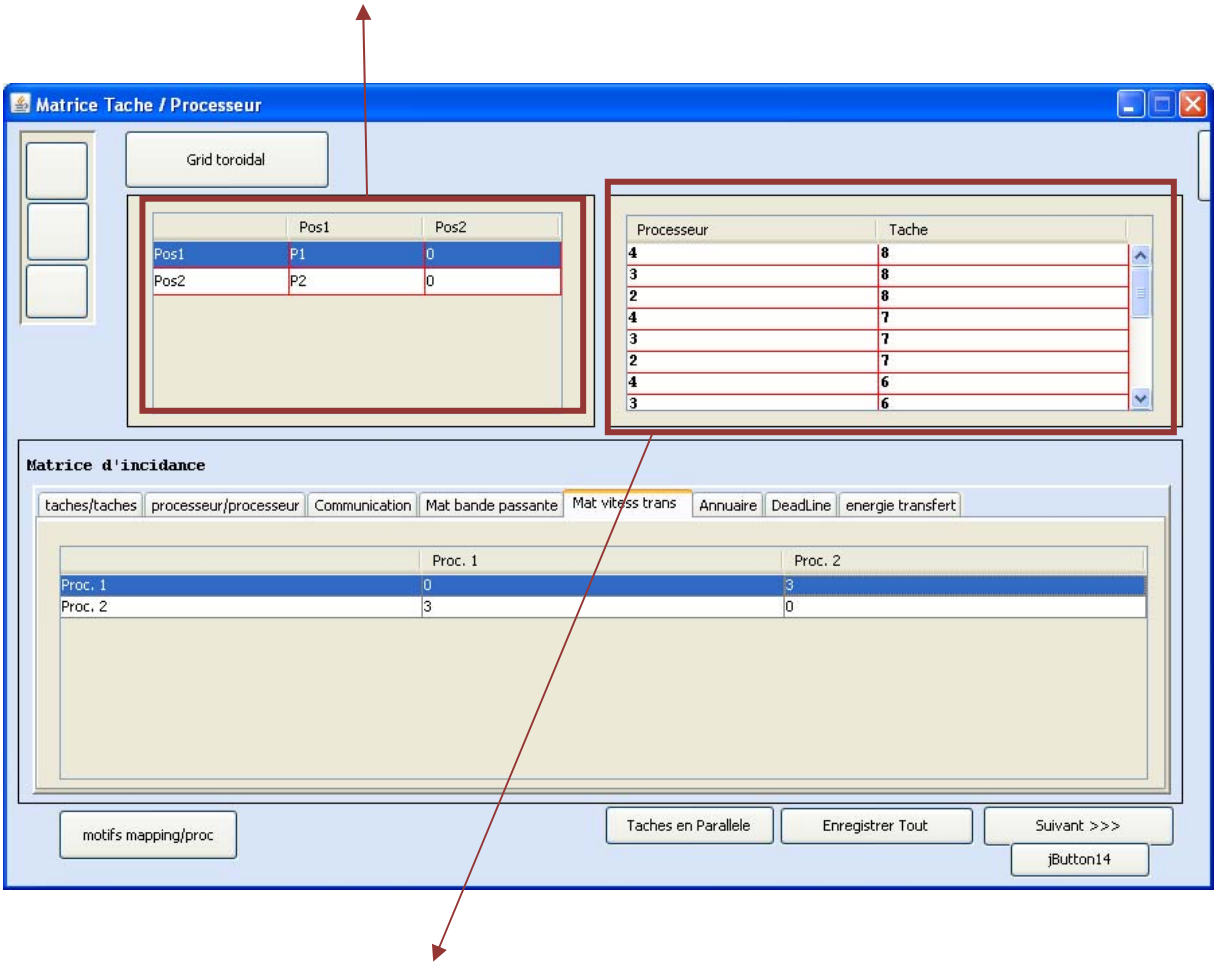


Illustration de La fenêtre Main (introduction).

6.2 **Un exemple du motif 3 Taches sur 2 Processeurs en faisant la saisie des caractéristiques du graphe d'application et du Graphe d'Architecture Hardware :**

La représentation d'une Grille Torique sous forme de Matrice
(Pos $_i$ / Pos $_j$).



Mat Itération avec n iterrations(chacune d'elle est un Motif) permettant de lancer le BnB Parallel .

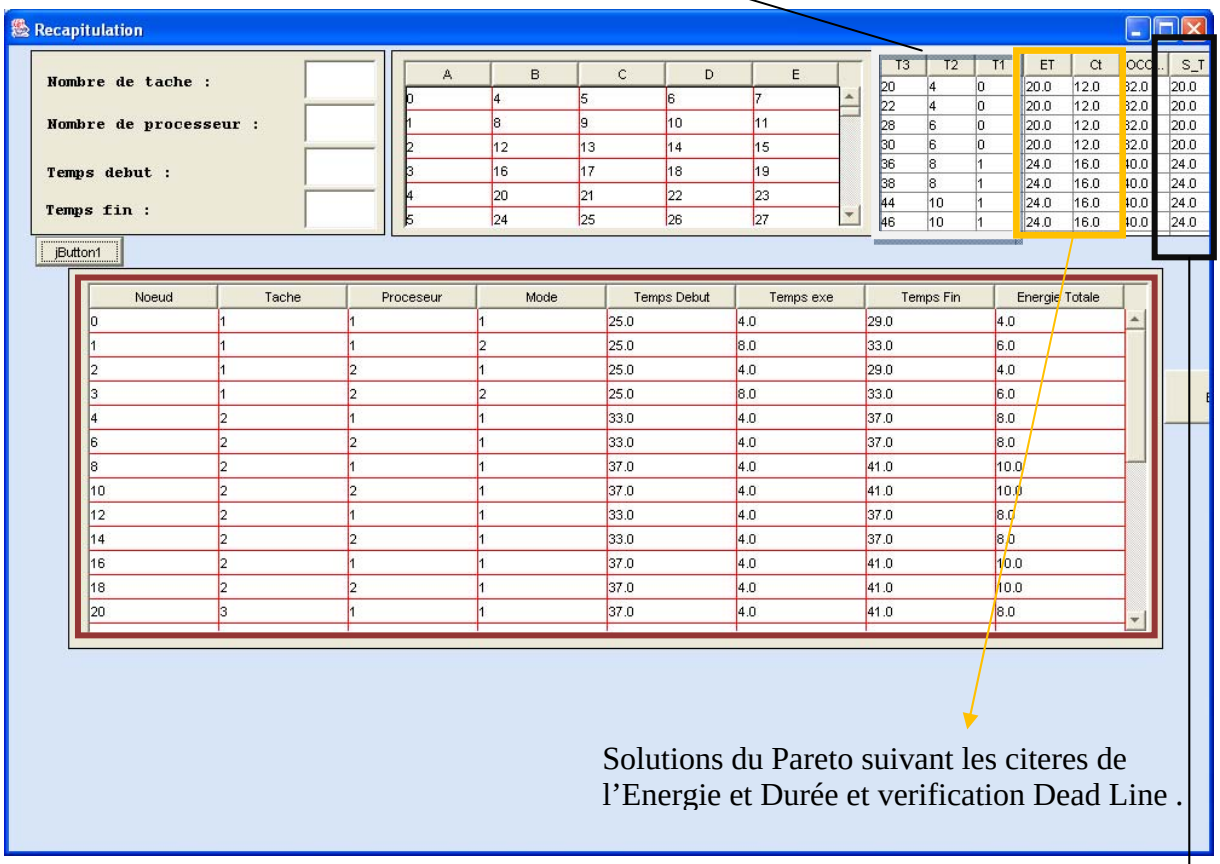
6.3 Placements et Ordonnancement des différents Tableaux de Simulation :

Notre but est de rechercher le meilleur Motif Optimal en appliquant l'algorithme Branch & Bound d'optimisation Parallèle multicritères qui propose un ou des placements intéressants par rapport à ces performances et en vérifiant le critère des contraintes temporelles .

6.3.1 Les Placements BnB Cout avec contraintes :

Les placements en bas de cette fenêtre illustre les solutions réalisables du Pareto avec Optimisation de la fonction coût Energie et coût durée en tenant aussi compte des contraintes temporelles du Dead line de l'application et du minimum Total Slack :

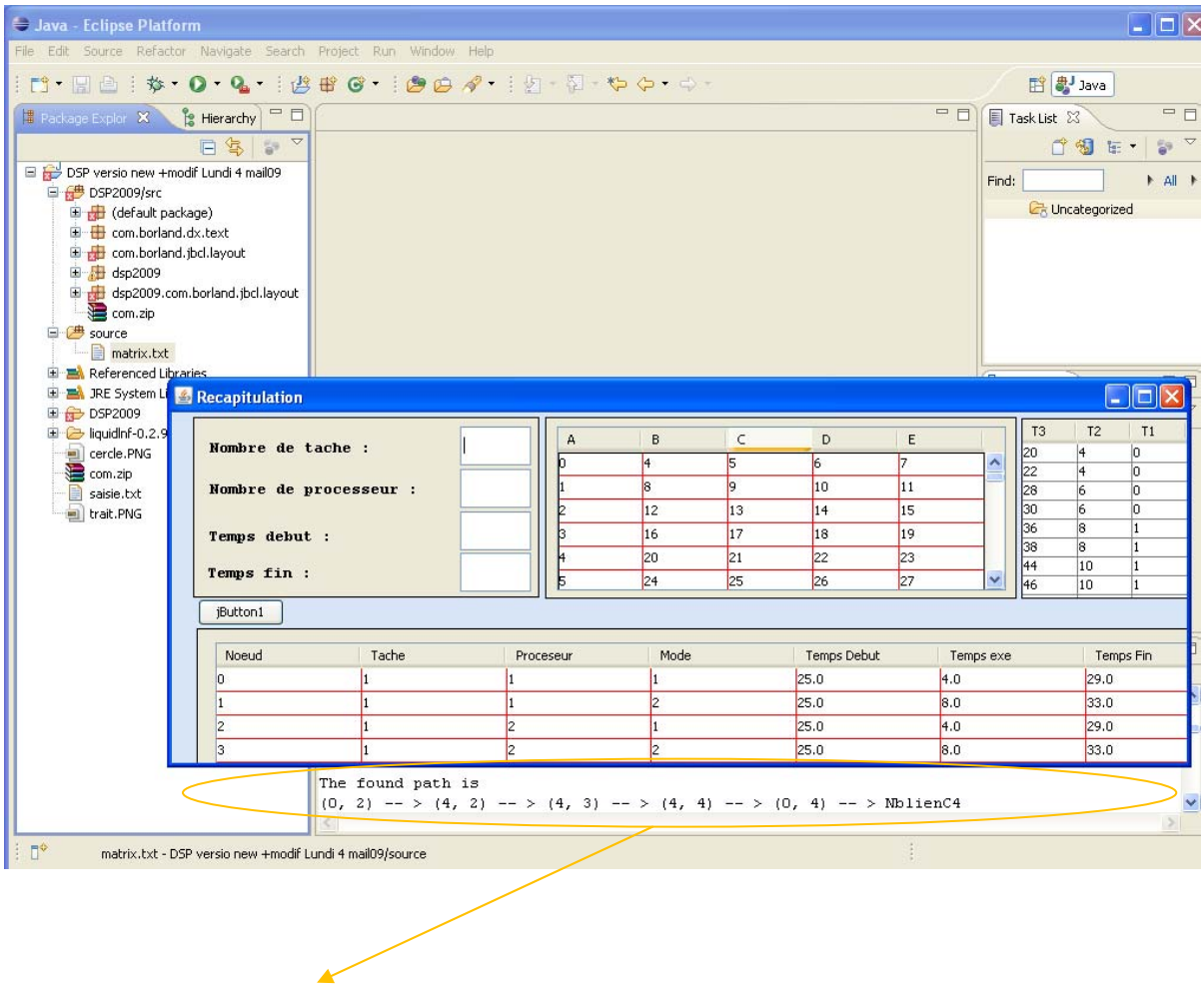
chaque placement est représenter par un triplet (T3 / T2 / T1) .



Solutions du Pareto suivant les citeres de l'Energie et Durée et verification Dead Line .

Les Valeurs du Total Slack pour chaque placement selon le 3eme critere Min Slack T.

6.3.2 Placement des Communications sur une Architecture cible Grille Bidirectionnelle torique 5*5 homogène Avec la méthode Nord –West illustrée comme suit :



Le chemin le plus court retourné avec des liens non saturés dans une grille torique bidirectionnelle.

6.3.3 Illustration en Introduisant le BnB parallèle avec 3 Motifs (Tche./Proc.) en parallèle:

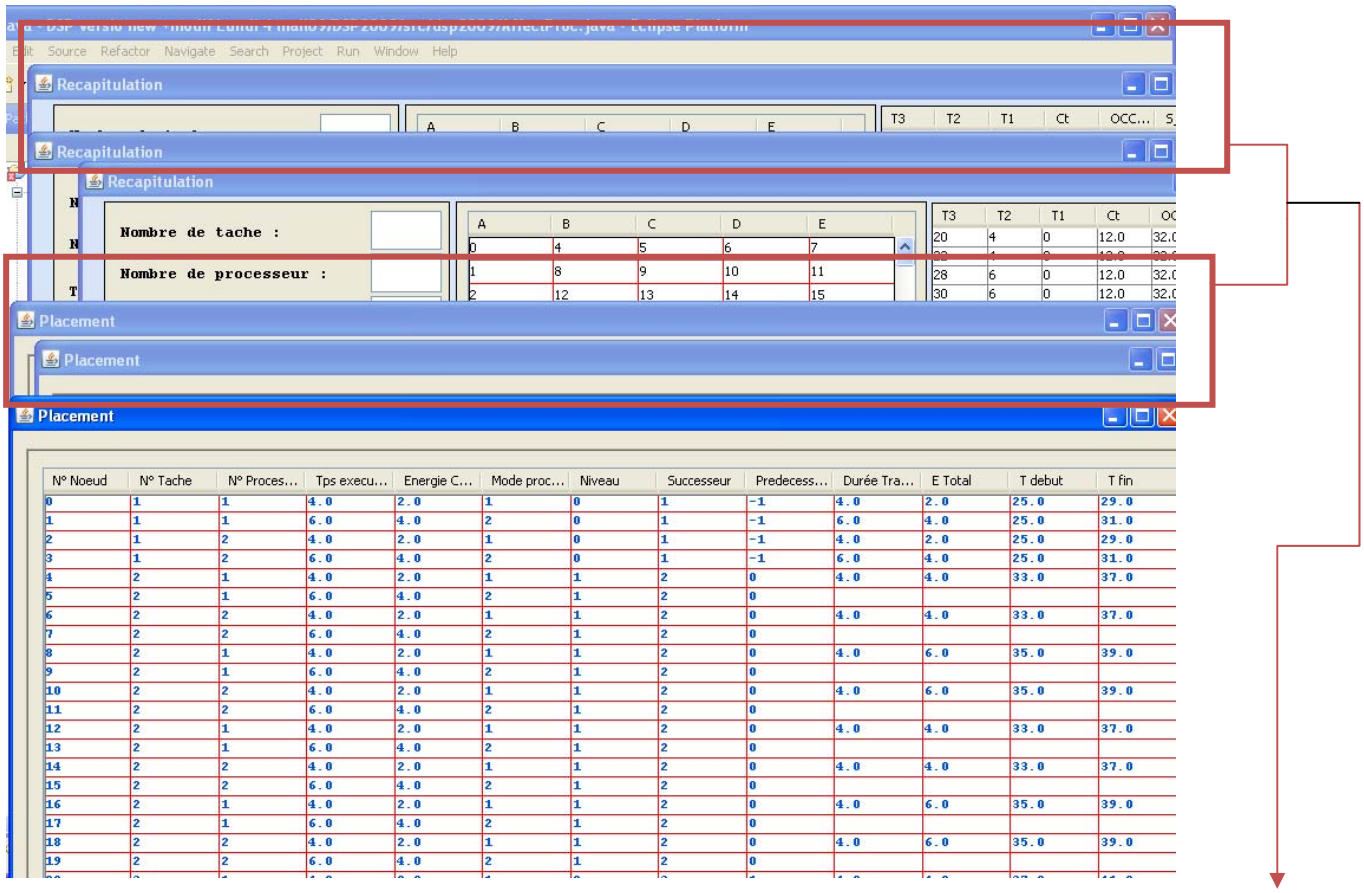


Illustration du lancement de 3 Motifs (Tche./Proc.) en parallèle.

1/ Etape 1:Calcul du Placement Cout BnB Parallele et Placement des Communications.

2/ Etape 2 : Archivage de tous les resultats avec min Slack pour chaque iterration du Motif comme suit :

RESULTAT (ARCHIVAGE PARETO et Choix du Motif Optimal) :

	PROCESSEUR	TACHE	Min Slack S_T
M_1	Nbre P_1	Nbre T_2	Val Min S_T_1
M_2	Nbre P_2	Nbre T_2	Val Min S_T_2
⋮			Val Min S_T_3
M_n	Nbre P_n	Nbre T_n	Val Min S_T_n

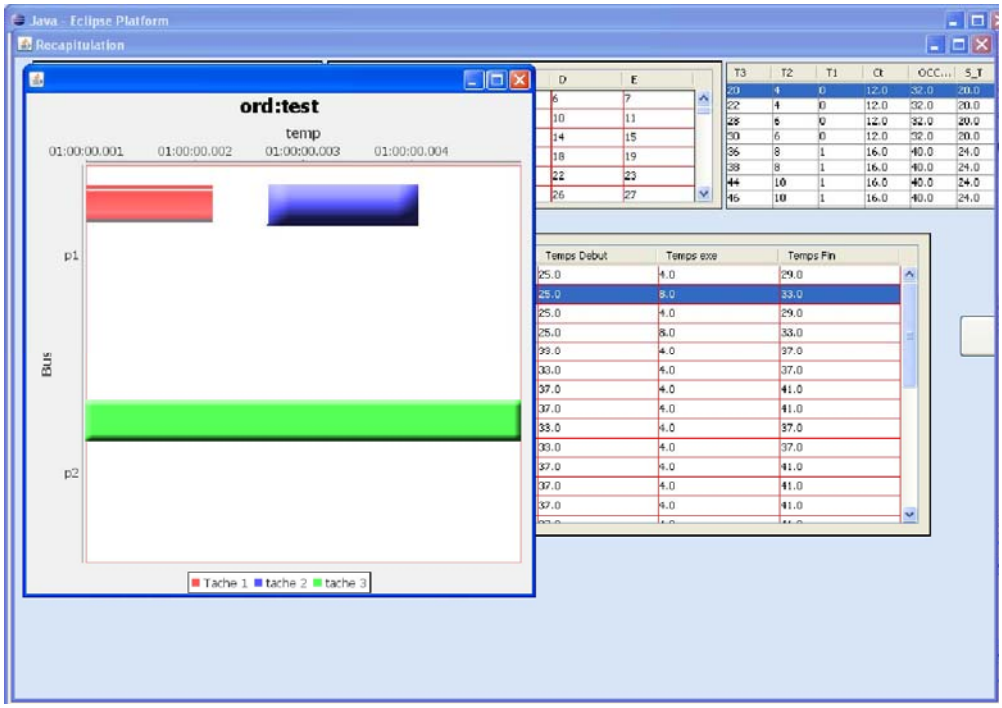
$$\text{le Nbre de répétitions} = \frac{X}{\text{Taille Motif}}$$

(X est le nombres Totales des Taches répétitives).

- Le choix du bon compromis entre Nbre Ti / Nbre Pj est déterminé par l'algorithme d'Optimisation multicritères BnB Parallèle à partir d'Archivage Pareto.

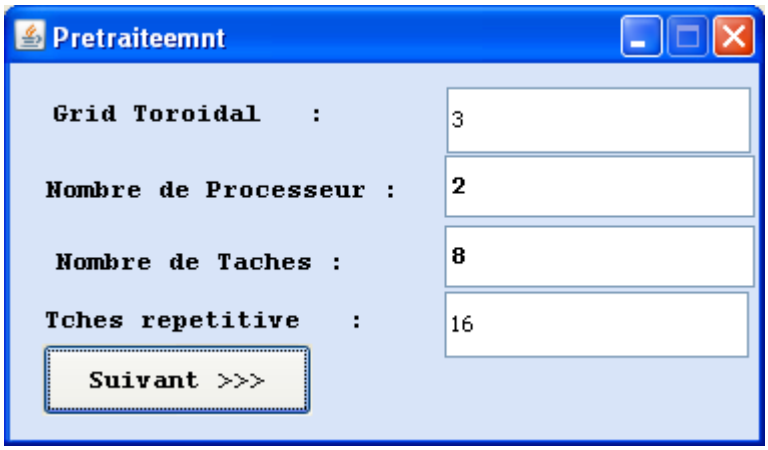
6.4 La Représentation Graphique du diagramme de GANTT :

La Figure ci-dessous montre la représentation du Diagramme Gantt d'un placement (T3/T2/T1) qui vérifie le minimum du total Slack comme suit :



7. Mapping multi-objectif d'une application réel:

Nous avons repris l'exemple d'une application réelle d'un décodeur vidéo avec communication (figure 7.13) **[24]** .



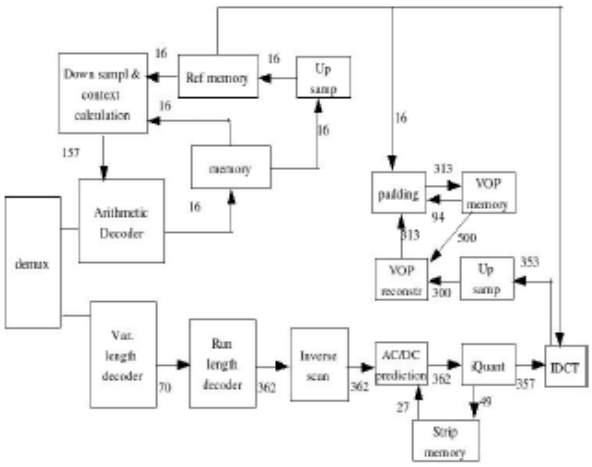


Figure 7.13 : diagramme de bloc de l'application

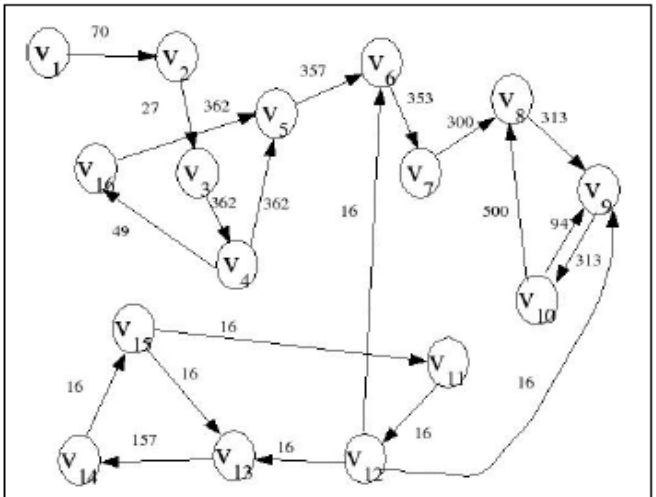
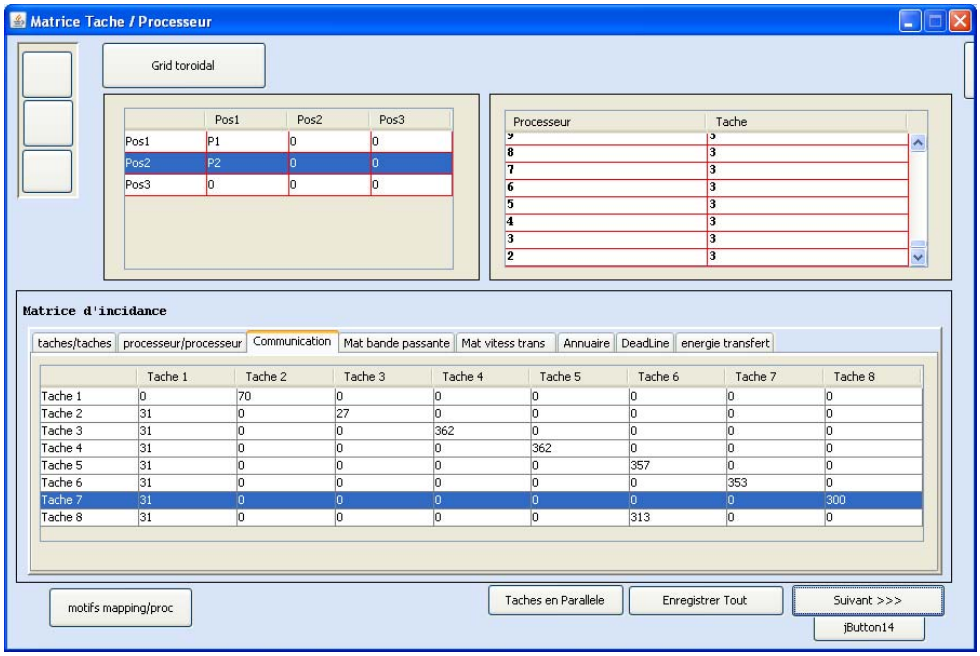
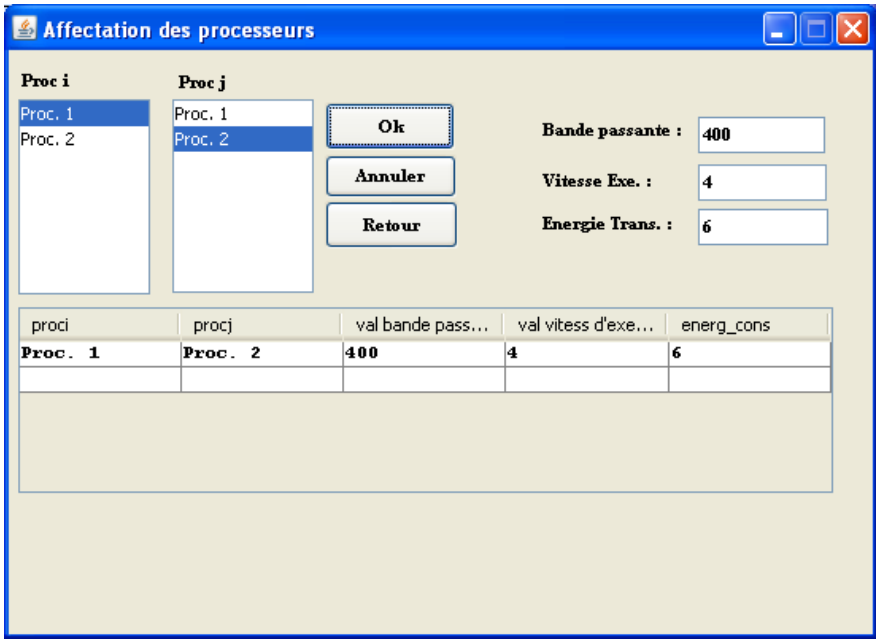


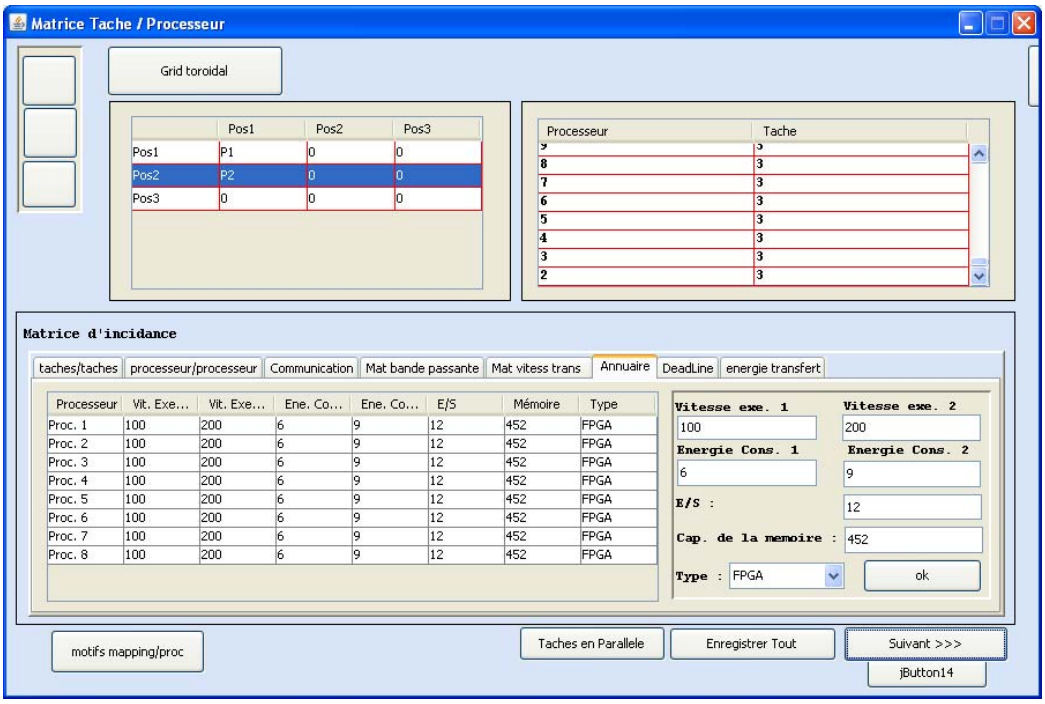
Figure 7.14 : le graphe de tâche de l'application figure 7.13

Les communications ainsi que les volumes d'échanges sont donnés par l'Onglet Communication ci-dessous :



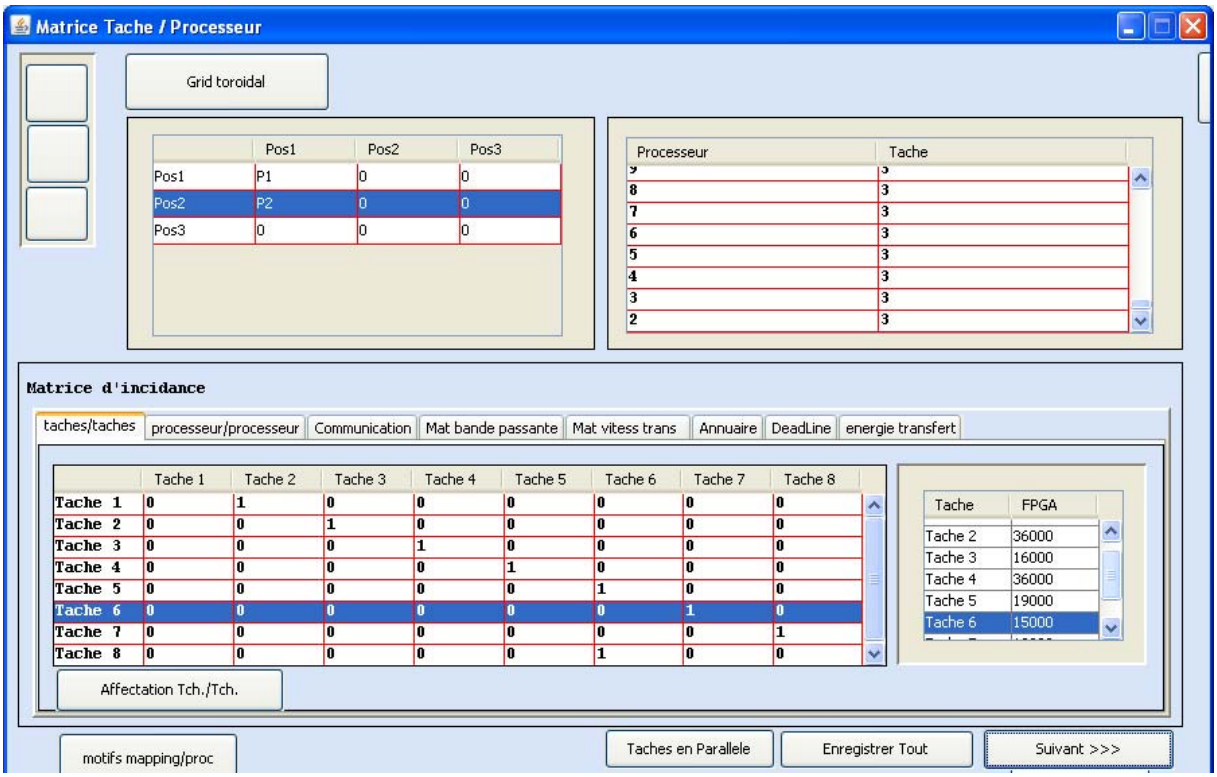
L'architecture cible est un Mesh-3D Torique constitué de 9 processeurs de type identique qui est un FPGA. La topologie est illustré par [24] comme suit :





Dans la figure ci-dessus la **Vitesse exe . 1** des processeurs au mode minimum c'est-à-dire lent, elle correspond au nombre de cycles exécutés par unité de temps, **Energie Cons.1** étant l'énergie consommée en unité voltage pour l'exécution d'un cycle à cette vitesse.

la **Vitesse exe . 2** et **Energie Cons.2** ont les mêmes définitions mais au mode maximum des processeurs c'est-à-dire en mode rapide. Les tailles des tâches sont données dans la figure ci dessous. Ces tailles sont mesurées en nombre de cycles selon le type de processeur qui est un **FPGA** .

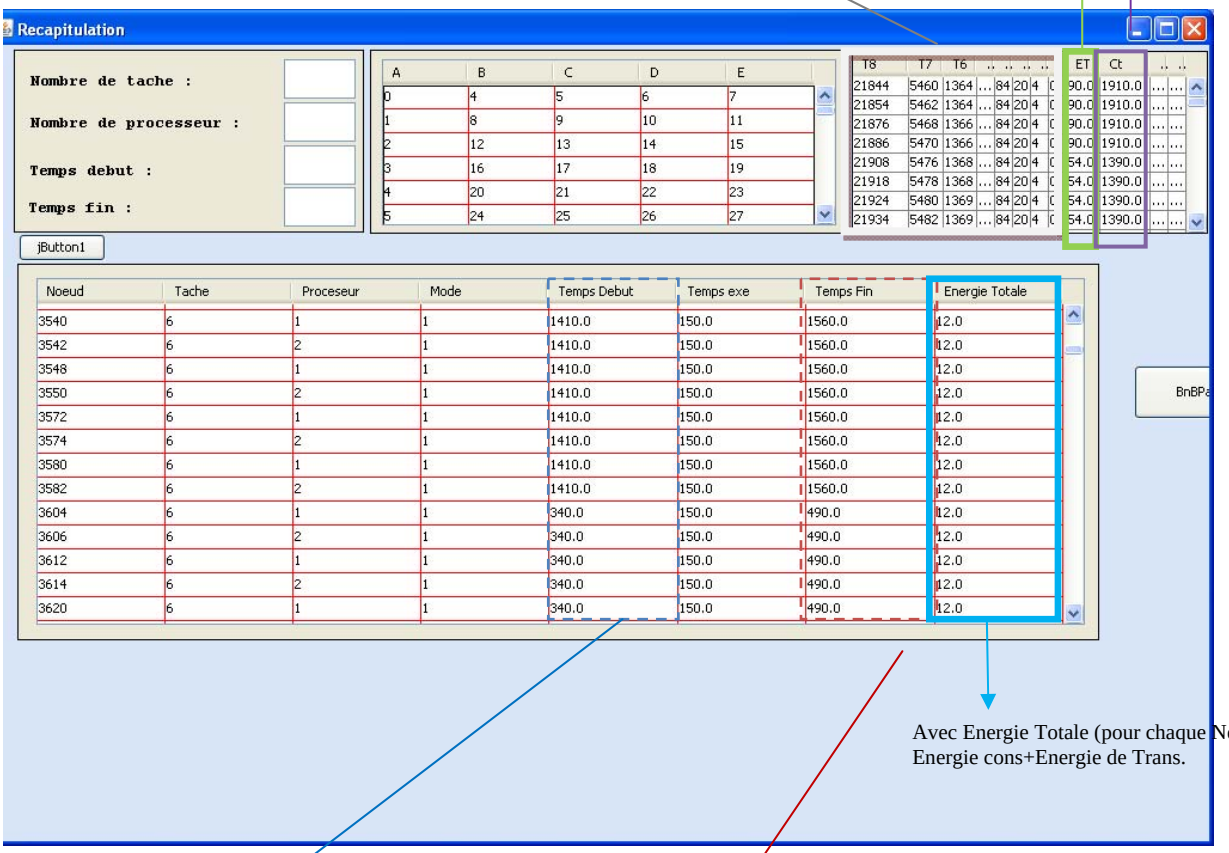


On obtient toutes les bonnes solutions en vérifiant le deadline d'application .Ces placements sont donnés dans la figure ci-dessous.

Avec CT (Cout Total d'Execution pour pour chaque Placement T8/T7/T6/T5/T4/T3/T2/T1)

Avec Energie Totale (pour chaque Nœud$_i$)= Energie cons+Energie de Trans.

Avec ET (Energie Totale pour chaque Placement T8/T7/T6/T5/T4/T3/T2/T1)



Avec Energie Totale (pour chaque Nœud$_i$)= Energie cons+Energie de Trans.

Avec Temps Début (pour chaque T_i Dans placement T8/T7/T6/T5/T4/T3/T2/T1) est représenter dans Diagramme de Gantt.

Avec Temps Fin (pour chaque T_i Dans placement T8/T7/T6/T5/T4/T3/T2/T1) en Verifiant Contrainte du Total Slack minimum est représenter dans Diagramme de Gantt. . minimum

Illustration des Meilleurs Solutions trouvées en vérifiant les contraintes temporelles du Dead Line de l'application.

L'Etude d'expérimentation établie dans la Thèse [24] En utilisant le **GI** (Globalement Irrégulier) ont obtenu toutes les bonnes solutions vérifiant les deadlines obtenues au mode maximum. Ces placements sont donnés dans le tableau suivant :

Tâche	Pla1	Pla2	Pla3	Pla4	Pla5	Pla6	Pla7	Pla8	Pla9	Pla10	Pla11	Pla12
t_1	P9	P6	P8	P8	P4	P1	P6	P3	P3	P8	P1	P1
t_2	P2	P2	P4	P1	P2	P2	P4	P4	P6	P7	P7	P2
t_3	P9	P3	P3	P2	P2	P3	P3	P3	P9	P9	P9	P9
t_4	P5	P1	P1	P1	P1	P1	P9	P6	P6	P6	P6	P6
t_5	P3	P3	P3	P3	P6	P6	P6	P9	P5	P5	P5	P5
t_6	P8	P8	P8	P8	P5	P5	P5	P1	P1	P1	P4	P4
t_7	P6	P6	P4	P4	P4	P8	P1	P1	P4	P4	P4	P2
t_8	P4	P5	P6	P7	P7	P7	P7	P2	P2	P3	P3	P3
t_9	P4	P4	P2	P1	P3	P3	P2	P5	P5	P4	P4	P4
t_{10}	P2	P2	P2	P2	P2	P3	P3	P8	P8	P8	P8	P8
t_{11}	P1	P1	P5	P5	P5	P5	P5	P4	P7	P7	P7	P7
t_{12}	P1	P7	P7	P6	P6	P6	P6	P3	P3	P6	P4	P4
t_{13}	P2	P2	P9	P3	P9	P2	P2	P1	P4	P4	P4	P3
t_{14}	P1	P1	P9	P9	P9	P9	P7	P7	P7	P7	P7	P7
t_{15}	P4	P4	P4	P4	P4	P4	P4	P9	P9	P9	P9	P9
t_{16}	P8	P8	P8	P2	P2	P1	P2	P2	P82	P4	P1	P1
Coût d'exéc -ution	1913,9	1775,4	1693,29	1689	1843,70	1759,64	1764,06	1876,68	1763,15	1583,99	1715	1751

Tableau des meilleurs placements de [24]

Etude Comparative avec l'expérimentation établie dans [24] :

Partie 1 : Branch &bound Hybride Multi-criteres:

1/ Pour la Contrainte du Coût d'exécution :

d'après Nos Résultats approximatifs obtenus nous avons constaté que le Coût d'exécution est 2 fois plus grand que le Coût d'exécution réalisé dans l'expérimentation du [24] ,

par contre concernant le nombre de ressources des processeurs où nous avons même pour la deuxième répétition uniquement deux processeurs actifs.

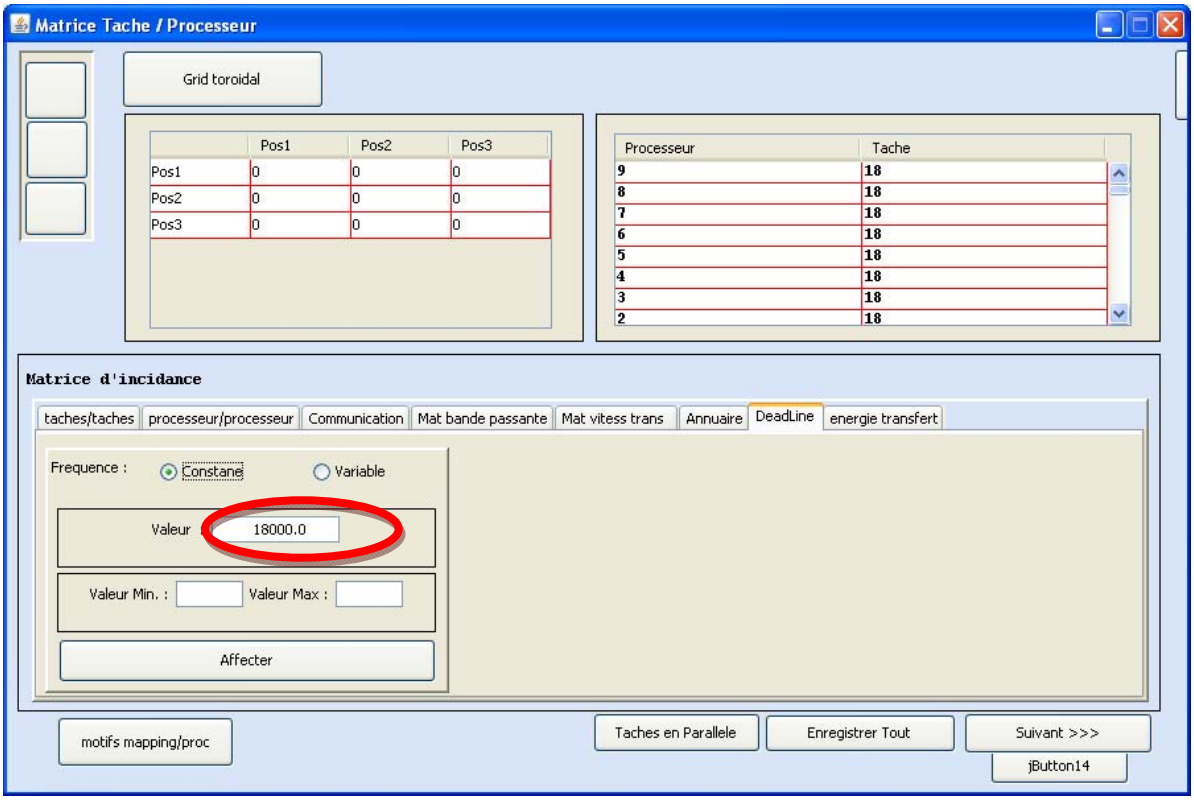
2/ Pour la Contrainte d'Energie Totale :

D'après nos Résultats pour Energie Totale des deux Motifs =2*Energie du motif (8Tche./ 2 Proc.) est approximativement inferieure à celle de la Consommation d'Energie vue en [24] comme suit :

Placement	Pla1	Pla2	Pla3	Pla4	Pla5	Pla6	Pla7	Pla8	Pla9	Pla10	Pla11	Pla12
Consommation d'énergie en 10^{-3} mv	2926	3164	3190	2937	2793	2998	2920	3106	3198	3465	3238	2953

Les consommations d'énergie des placements trouvés

Le deadline de cette application est de 18000 **ut** est donné par l'onglet DeadLine :



3/ Pour la Contrainte du Minimum Slack_Total :

D'après l'expérimentation de [24] leurs meilleures solutions données dans le tableau précédent ,ils ont dégagés l'ensemble S' des solutions vérifiant le deadline. S'= {Pla2, Pla3, Pla4, Pla6, Pla7, Pla9, Pla10, Pla11, Pla12} et ils ont constaté que pour le meilleur en temps d'exécution c'est le placement 10, il est aussi le plus coûteux en terme de consommation d'énergie. Le tableau suivant illustre le placement et l'ordonnancement obtenu pour le Placement 10. Le scheduling des autres placements s'obtient de la même manière.

Placement		Coût exécution	Coût comm	Coût total	ordonnancement	
tâche	processeur				T_{debut}	T_{fin}
t_1	P8	23,75	17,50	41,25	0,00	41,25
t_2	P7	55,38	0,00	55,38	41,25	96,63
t_3	P9	80,00	181,00	261,00	96,63	357,63
t_4	P6	37,89	120,67	158,56	357,63	516,20
t_5	P5	56,00	119,00	175,00	516,20	691,20
t_6	P1	62,50	117,67	180,17	691,20	871,36
t_7	P4	71,11	117,67	188,78	871,36	1060,14
t_8	P3	37,65	100,00	137,65	1060,14	1197,79
t_9	P4	37,78	52,17	89,94	1197,79	1287,73
t_{10}	P8	23,75	99,00	122,75	1287,73	1410,48
t_{11}	P7	23,08	88,25	111,33	1472,66	1583,99
t_{12}	P6	20,00	3,20	23,20	1287,73	1310,93
t_{13}	P4	71,11	0,00	71,11	1310,93	1382,04
t_{14}	P7	24,62	4,00	28,62	1382,04	1410,66
t_{15}	P9	56,67	5,33	62,00	1410,66	1472,66
t_{16}	P4	66,67	120,67	187,33	516,20	703,53

Placement et ordonnancement

D'après nos Résultats d'Expérimentation illustré dans la figure qui suit, on a plusieurs solutions qui sont meilleurs en vérifiant la contrainte Dead Line de l'application d'où on est amené à prendre le Placement (T8/T7/T6/T5/T4/T3/T2/T1) qui vérifie le plus Minimum Slack_Total et chaque Ti a ses propres Paramètres Temps début et fin :

Noeud	Tache	Proceseur	Mode	Temps Debut	Temps exe	Temps Fin	Energie Totale
3540	6	1	1	1410.0	150.0	1560.0	12.0
3542	6	2	1	1410.0	150.0	1560.0	12.0
3548	6	1	1	1410.0	150.0	1560.0	12.0
3550	6	2	1	1410.0	150.0	1560.0	12.0
3572	6	1	1	1410.0	150.0	1560.0	12.0
3574	6	2	1	1410.0	150.0	1560.0	12.0
3580	6	1	1	1410.0	150.0	1560.0	12.0
3582	6	2	1	1410.0	150.0	1560.0	12.0
3604	6	1	1	340.0	150.0	490.0	12.0
3606	6	2	1	340.0	150.0	490.0	12.0
3612	6	1	1	340.0	150.0	490.0	12.0
3614	6	2	1	340.0	150.0	490.0	12.0
3620	6	1	1	340.0	150.0	490.0	12.0

Illustrations des meilleurs solutions en verifiant les contraintes Temporelles et le temps début et temps fin de chaque Ti du placement trouvé.

Partie 2: Branch & Bound hybride parallèle multicritères avec 3 Motifs (Tache i /Processeur j) avec l'exemple de l'application réel de l'encodeur H263 :

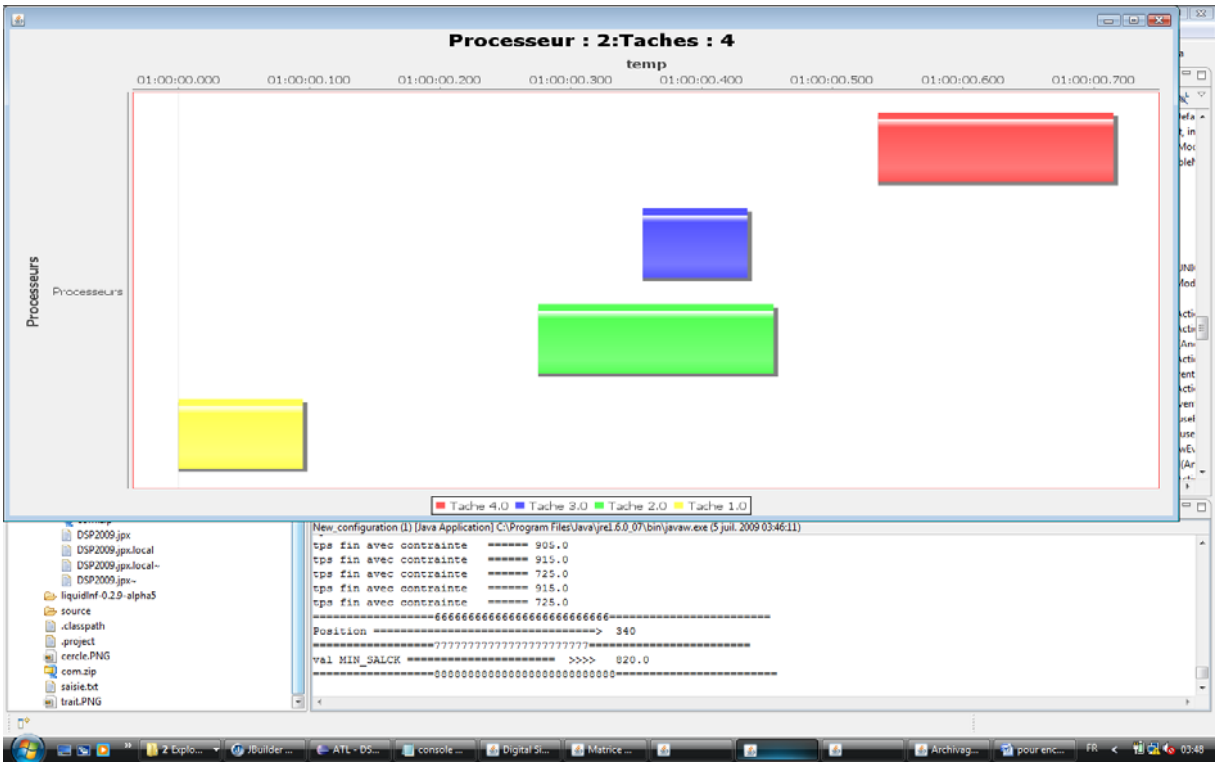
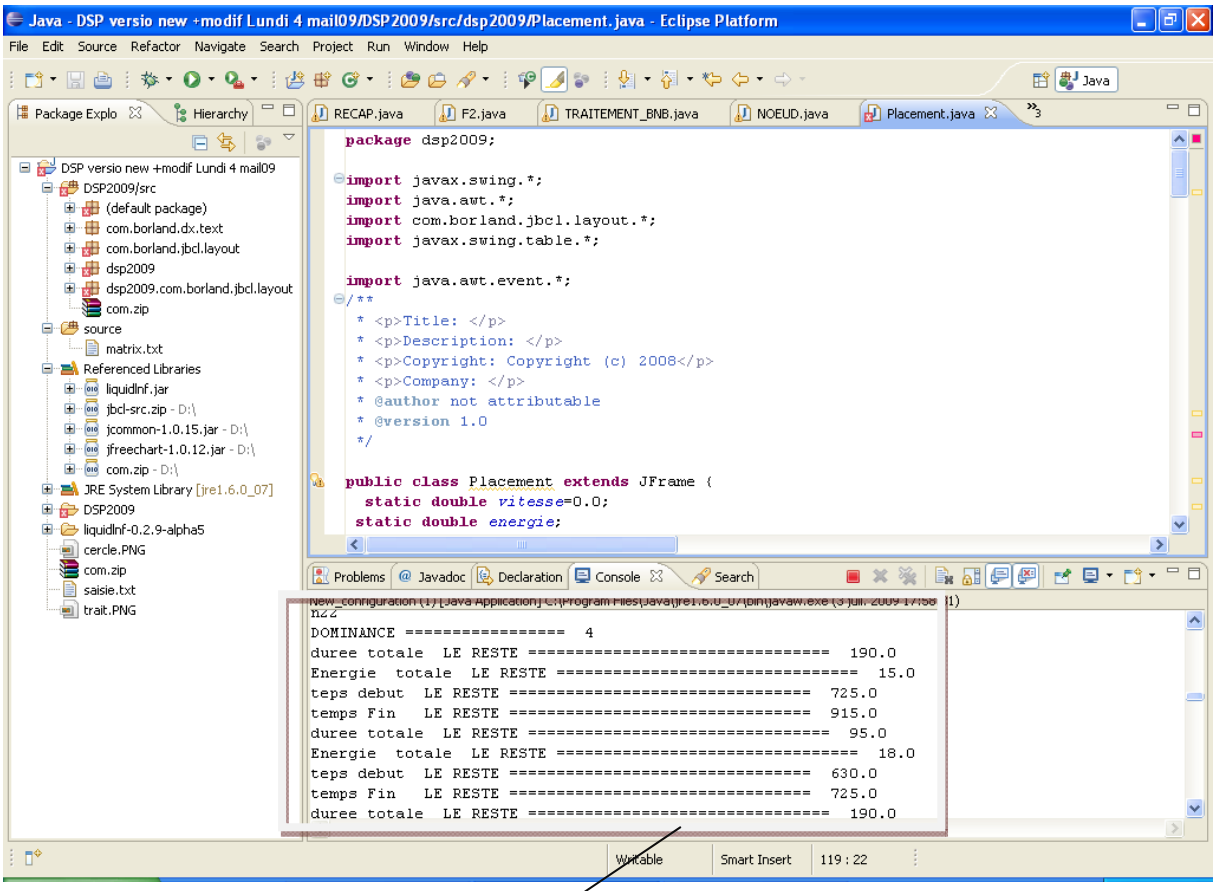
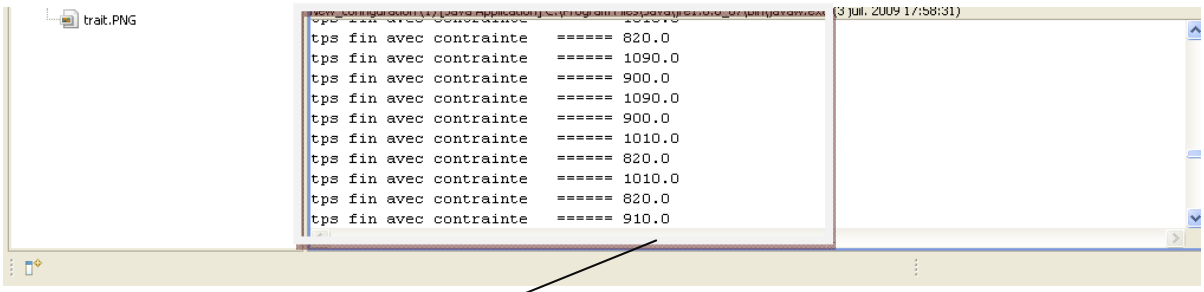


Illustration diagramme de GANTT du Motif de 4 Taches .

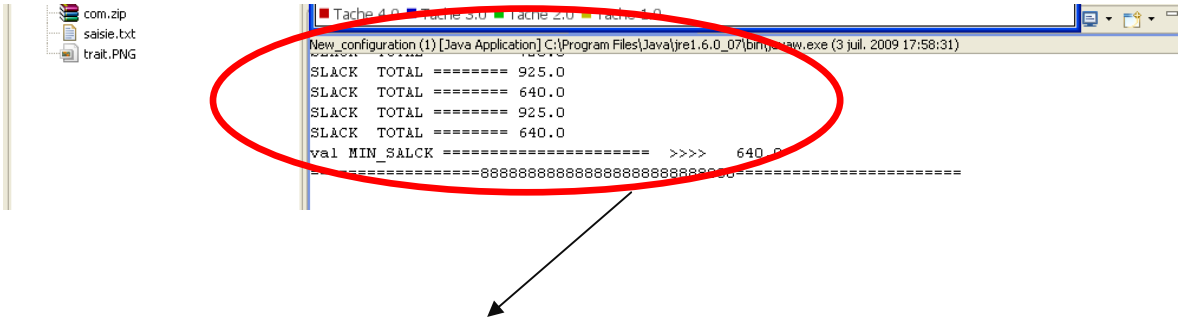
Comme Remarque préliminaire nous avons constaté que le temps de recherche du Branch and Bound Parallèle est largement plus rapide que le Branch and Bound Séquentiel.



Les solutions de placement non dominés du PARETO



Les solutions de placement qui vérifient les contraintes temporelles de Dead Line de nos résultats Expérimentaux avec le Branch and Bound Parallèle du Motif de 4 Taches.



Affichage de la valeur minimale du total slack parmi l'ensemble du total slack

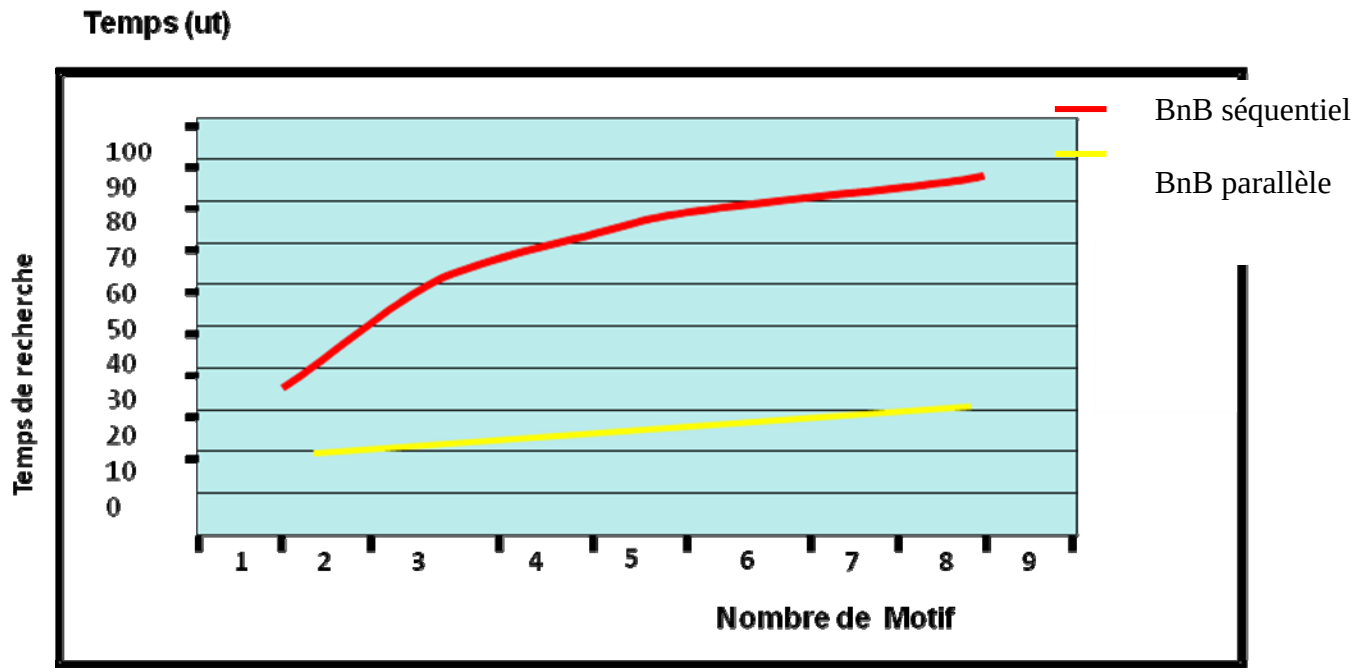


Illustration pour une comparaison du Temps de recherche entre Branch and Bound et le Branch & Bound Parallèle.

Conclusion :

Ce chapitre illustre la mise en Œuvre de notre Contribution qui élabore la Méthode Hybride Multicritères avec le Branch and Bound et aussi Branch and Bound parallèle . Notre souci est d'expérimenté cette approche dans un contexte spécifique qui est dans notre cas les NOC, par des Simulations et par cette brève étude d'expérimentation pour notre travail et d'essayer de faire des expérimentations plus larges pour des problèmes réels divers dans le domaine des Systémes embarqués .

CONCLUSION GENERALE

Par cette modeste étude nous avons montré que pour le domaine DSP il existe plusieurs langages qui nous permettent d'expliquer le parallélisme des taches répétitives sur une architecture régulière embarquée. Par le mapping on parle d'assignation, d'affectation et d'ordonnancement d'éléments d'application sur des éléments d'architectures.

Notre première préoccupation a été de trouver une solution pour le mapping des applications DSP sur des architectures régulières. Nous avons opté pour le langage ARRAY-Ol pour présenter ces applications DSP qui traitent de grandes quantités de données avec une répétition des calculs. Avec cette méthode exacte nous permet d'atteindre plusieurs objectifs à optimiser.

Le dernier volet de notre travail a été une contribution pour construire un modèle comprenant les valeurs de temps d'exécution de chaque tâche sur chaque processeur en fonction du voltage ainsi que les performances des réseaux d'interconnexion en fonction du volume de données transportées. Dans le cadre de conception et développement du logiciel où ces valeurs sont fournies à l'algorithme Branch & Bound d'optimisation Parallèle multicritères qui propose un placement qui représente l'optimum théorique « exact » ou plusieurs placements intéressants par rapport à ces performances et aussi en vérifiant le critère des contraintes temporelles pour le choix du motif Optimal.

LES REFERENCES

[1]: http://fr.wikipedia.org/wiki/ Systèmes Embarqués 2009

[2]: Systèmes Embarqués et Grandes Infrastructures –Bertrand BRAUNSCHWEIG Edition 2008 -

[3]: WESTTeam LIFL, LIlle, France. ―Graphical Array Specification for Parallel and Distributed Computing (GASPARD-2)‖. http://www.lfl.fr/west/ gaspard/,2005.

[4] : Christophe Manas. « Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrone ». PHD thesis, Université de Rennes, Décembre 1989.

[5] : Praveen K.Murthy and Edward A.lee. « Multidimensional synchronous data flow ». IEEE Transaction on signal Processing, juillet 2002.

[6]: P.K.Murthy and Edward A.lee. ―A generalization of multidimensional synchronous dataflow to arbitrary sampling Lattices‖. Technical Report UCB/ERL M95/59, EECS Department of University of California, Berkeley, mais 1995.

[7]: AH.Benyamina and P.Boulet.‖ Multi-objective Mapping for NoC Architectures‖. Journal of Digital Information Management (JDIM). Volume 5 Numéro 6. Pages 378—384. Décembre 2007.

[8]: P.K.Murthy and Edward A.lee. ―A extension of multidimensional synchronous dataflow to arbitrary sampling Lattices‖. Technical Report UCB/ERL M95/59, EECS Department of University of California, Berkeley, mais 1995.

[**9**]: A.Amar, P.Boulet, and P.Dumond. ―Projection of the Array-OL specification langage onto the Kahn process network computation model‖. In international symposium on parallel Architecture, Algorithms, and NetWorks, Las Vegas, Nevada, USA, December 2005.

[10] : S.Le Beux. « Un flot de conception pour application de traitement de signal ». Thèse de doctorat. USTL. France 2007.

[11]: S.Sayin and S.Karabati. A bicriteria approach to the two-machine flow shop scheduling problem. Eurpean Journal of operational Research, 113: 435-449,1999.

[12]: E.L.Ulungu and J.Teghem. The two phase method: An efficient procedure to solve bi-objective combinatorial optimisation problems. In Foundations of Computing and Decision Sciences 1995.

[13]:. http://fr.wikipedia.org/wiki/S%C3%A9paration_et_%C3%A9valuation

[14] Gilles Kahn. < The semantics of a simple language for parallel programming Dans Jack L. Rosenfeld, _editeur, Information Processing 74: Proceedings of the IFIP Congress 74, pages 471{475. IFIP, North-Holland Publishing , aout 1974.

[15] :Gilles Kahn et David B. MacQueen. Coroutines and networks of parallel processes.Dans B. Gilchrist, _editeur, Information Processing 77, pages 993{998. North-Holland,1977. Proc.IFIP Congress.

[16] : Pierre BouletContributions aux environnements de programmation pour le calcul intensif université LILLE 2 décembre 2002

[17] :TOBIAS BJERREGAARD AND SHANKAR MAHADEVAN "A Survey of Research and Practices of Network-on-Chip" Technical University of Denmark March 2006.

[18] :JERRAYA A . « Conception logique et physique des systèmes mono puce », 2002

[19] : Mickael SAMYN « Une simulation fonctionnelle d'un système monopuce dédié au traitement de signal intensif » Thése présentée pour obtenir le titre de Docteur de l'Université des Sciences et Technologies de Lille 14 décembre 2005

[20]: "Visual Data-parallel Programming for Signal Processing Application" , PDP 2001

[21] : Pierre Boulet "Array-OL Revisited, Multidimensional Intensive Signal Processing Specification » institut national de recherche en informatique et en automatique de LILLE

[22] : Julien SOULA « Principe de compilation d'un langage de traitement de signal »Thèse présentée pour obtenir le titre de Docteur de l'Université des Sciences et Technologies de Lille, 20 décembre 2001

[23] : Yann Collette , Patrick Siarry - Optimisation multiobjectif -Septembre 2002.

[24]: BENYAMINA Abou El Hassan **«** ordonnancement hiérarchique multi-objectifs d'applications embarquées intensive »thèse de doctorat d'état 2008

[25]: R. E. Kalman Mehul Motani **«**On Kalman Filtering A study of "A New Approach to Linear Filtering and Prediction Problems" » February 11, 2000

[26]: Michael Weeks **«**digital signal processing using MATLAB and Wavelets electrical engineering series»

[27] :Thierry Grand pierre, Christophe Lavarenne **«**modèle d'exécutif distribué temps réel pour Syndex» 1998

[28] :ashich Meena **«**Allocation,Assignation et Ordonnancement pour les systèmes sur puce multiprocesseur » 15 décembre 2006