HAL
open science

Outil de partitionnement hw/sw basé sur l'algorithme Kernighan/Lin amélioré

R. Boudour, M.T. Laskri

▶ **To cite this version:**

R. Boudour, M.T. Laskri. Outil de partitionnement hw/sw basé sur l'algorithme Kernighan/Lin amélioré. Revue Africaine de Recherche en Informatique et Mathématiques Appliquées, 2007, Volume 7, 2007, pp.20-40. 10.46298/arima.1882 . hal-01263496

HAL Id: hal-01263496

https://inria.hal.science/hal-01263496v1

Submitted on 27 Jan 2016

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

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

Outil de partitionnement hw/sw basé sur l'algorithme Kernighan/Lin amélioré

R. Boudour - M.T. Laskri

Département d'informatique
Université Badji-Mokhtar

BP 12 Annaba

ALGERIE

Email : racboudour@yahoo.fr

RÉSUMÉ. Le partitionnement fonctionnel d'un système, en composants matériels et logiciels, acquiert de plus en plus de l'importance en conception conjointe. Plusieurs heuristiques et algorithmes sont utilisés en partitionnement. Dans ce papier, nous présentons l'outil, appelé AutoDec, implémenté en Visual C++ 6.0. Nous vérifions que l'algorithme hierarchical clustering, basé sur des métriques de rapprochement, peut être utilisé pour fusionner des parties fonctionnelles avant l'application de l'algorithme Kernihgan/Lin, entraînant ainsi une réduction notable du temps d'exécution avec souvent une amélioration accrue en qualité. En somme, nous montrons que notre approche, utilisée en partitionnement, permet de réduire le fossé entre les algorithmes rapides et hautement optimaux

ABSTRACT. Partitioning of system functionality for implementation among multiple system components, such as among hardware and software components in codesign, is becoming an increasingly important topic. Various heuristics are used in automatic partitioning. In this paper, we present our tool, called AutoDec, implemented in Visual C++ 6.0. We verified that hierarchical clustering algorithm, based on closeness metrics, can be used to merge pieces of functionality before applying Kernighan/Lin algorithm, resulting in reduced execution time with often improvements in quality. In addition, we show that our approach, when used in partitioning, fills the gap between fast algorithms and highly-optimizing ones.

MOTS-CLÉS : Conception conjointe, partitionnement automatique, métriques de rapprochement, algorithme hierarchical clustering, algorithme Kernighan/Lin.

KEYWORDS : Codesign, automatic partitioning, closeness metrics, hierarchical clustering algorithm, Kernighan/Lin algorithm.

1. Introduction

Pour pallier aux problèmes inhérents à l'approche classique, il a fallu penser à une nouvelle méthodologie qui réexamine les frontières entre le logiciel et le matériel donc qui tient compte dès la phase de spécification des interactions entre les deux parties et qui essaie de réduire le coût et par conséquent le time-to-market du produit final, cette méthodologie est connue sous le vocable d'origine anglosaxonne *codesign* pour conception conjointe [20] [27]. Nos propos se situent à sa deuxième étape, appelée partitionnement matériel/logiciel. Ce partitionnement est qualifié d'automatique lorsque l'algorithme remplace le concepteur dans la prise des décisions [26].

Le partitionnement est bien connu comme étant un problème NP-complet, et pendant les années passées, plusieurs algorithmes basés sur des techniques heuristiques ont été proposés. Kernighan/Lin est l'une de ces techniques prometteuses. Il a fait l'objet de plusieurs extensions particulièrement celle de Fiduccia/Mattheyses pour s'exécuter dans un temps linéaire et pour donner de meilleurs résultats [24]. Mais dans toutes ces améliorations successives, la partition initiale est restée toujours *aléatoire* : souvent tout en logiciel ou tout en matériel. Dans ce contexte, nous essayons de focaliser nos efforts sur cette *partition initiale* en la rendant plus *réfléchie*. Pour l'obtenir, nous allons faire appel à des closeness métrics, condition sine qua none à notre sens pour réduire davantage le temps de calcul d'une part et pour ouvrir la voie au partitionnement de gros problèmes d'autre part. Pour concourir au même objectif, nous considérons plus de métriques globales comme indicateurs de qualité et la spécification est choisie à dessein à grain moyen *le bloc de base* pour renforcer la précision voire la qualité des résultats. Le reste du papier est organisé comme suit : la section 2 aborde des éléments nécessaires à la compréhension du problème de partitionnement logiciel/matériel. La section 3 présente d'une manière assez détaillée, les différentes métriques et contraintes utilisées. La section 4 décrit l'outil *Autodec.* La section 5 résume les résultats expérimentaux et la section 6 termine le papier par une conclusion et de futures directions du travail.

2. Graphes et algorithmes

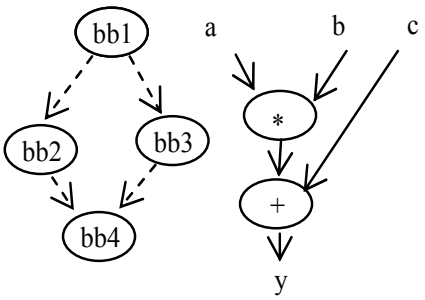
2.1. Graphes et blocs de base

Les spécifications de systèmes mixtes sont décrites en utilisant les langages C/VHDL ou le systemC et le code est traduit ensuite en un graphe de flux de contrôle et de données (cdfg). Le cdfg, une abstraction de la spécification, est utilisé comme

entrée des algorithmes de partitionnement pour mapper les blocs/nœuds du cdfg soit en logiciel, soit en matériel. Nous présentons brièvement ces deux concepts :

Un bloc de base est une séquence d'instructions consécutives dans laquelle le flot de contrôle est activé au début de celle-ci et inhibé à la fin, sans possibilité d'arrêt ou de branchement autre qu'à la fin de la séquence. Ainsi, un bloc de base peut être constitué d'une séquence d'instructions d'affectation, d'une opération non déterministe, ou d'un appel de procédure. Pour plus de détails sur les règles de construction des blocs de base, le lecteur intéressé peut consulter [7] [9].

Un cdfg, G = (V, E), décrit le comportement d'un système, il contient un ensemble de blocs de base, représentés par des nœuds $V = \{v_i / i = 1,2, \ldots m\}$, et un ensemble de dépendances représentées par des arcs $E = \{e_{ij} / e_{ij} = (v_i,v_j), v_i,v_j \in V\}$, en général la partie contrôle (cfg) est distincte de la partie donnée (dfg) [12], comme est illustré en figure 1.



a) Cfg (control flow graph) *b) dfg (data flow graph)*

Figure 1. *cdfg (cfg + dfg) : a) cfg avec blocs de base b) dfg avec opérations et données*

2.2. Algorithmes de partitionnement

On rencontre deux grandes classes d'algorithmes de partitionnement [13] [2] [26] explorant un graphe : *Les algorithmes constructifs* et les algorithmes itératifs. On s'intéresse à Hierarchical clustering et *à* Kernighan/Lin .

2.2.1. Algorithme hierarchical clustering

Le groupement hiérarchique ou hierarchical clustering est un algorithme constructif qui utilise, pour trouver la bonne partition, des closeness metrics. La partition résultante va être utilisée dans notre cas par l'algorithme de Kernighan/Lin comme partition initiale. Pour aboutir à une partition complète, l'algorithme opère de la manière suivante : Il parcourt le graphe cdfg issu de l'étape de spécification du codesign, dans le but d'identifier les objets les plus proches, puis il détermine les

valeurs de rapprochement initiales à l'aide d'une fonction de rapprochement, ensuite il groupe les objets trouvés, en recalcule le rapprochement après le groupement. Il répète le processus jusqu'à rencontrer une condition d'arrêt qui peut être par exemple un nombre de groupes à ne pas dépasser ou/et une valeur seuil.

L'algorithme est le suivant :

```
Entrée
        P : partition
Début
                /* Initialiser chaque objet comme étant un groupe */
                Pour (chaque objet o(i) ) faire

            Fin
      Fin
/* Fusionner les objets les plus proches et recalculer le   rapprochement */
        Tant que  ( P≠Ø )  Faire
            p(i,j) = FindClosestObjects(P,C)
            P = P – p(i) –  p(j) ∪ p(ij)
                        Pour (chaque p(k))   faire
                         c(ij, k)= ComputeCloseness(p(ij),p(k) )
                        Fin
         Fin
Fin
```

2.2.2. Algorithme Kernighan/Lin

L'algorithme de Kernighan/Lin, connu aussi sous le nom de *migration de groupe* ou *min cu*t est itératif. Malgré son ancienneté (années 70), beaucoup de systèmes de partitionnement continuent à l'utiliser encore. Il fournit de bons résultats en un temps d'exécution relativement court : Etant donnée une *partition de départ générée aléatoirement*, pour chaque objet (nœud), il calcule la fonction coût en supposant qu'il a été déplacé vers une autre partition (implémentée avec une autre technologie). Puis, il identifie le nœud qui produit la plus grande baisse ou la plus petite augmentation dans le coût. Ensuite, il fait l'échange et répète le processus en utilisant la nouvelle partition comme partition initiale jusqu'à ne plus trouver de partition de coût inférieur (condition d'arrêt). Pour éviter une boucle infinie (pas de cycles dans le graphe), notons que chaque objet ne peut être déplacé qu'une seule fois [6] [14] [24].

L'algorithme décrit ci-dessous, comporte deux parties relativement proches mais distinctes constituant le corps de l'algorithme. Premièrement, la stratégie de contrôle, comportant deux actions : *Select Next Move* est une procédure qui choisit le prochain

déplacement à faire et *Terminate qui* retourne le résultat s'il n'y a plus d'amélioration possible. Deuxièmement, les données concernant le coût : *DS* est une structure de données utilisée pour modéliser les nœuds et à partir de laquelle le coût va être calculé, *UpdateData* initialise DS et la met à jour après chaque mouvement et *CostFct* , la valeur de cette fonction est calculée à partir d'une combinaison des valeurs issues de différentes métriques. Cette valeur est appelée *cost* représentant la qualité de la partition, dans notre cas un coût minimal. Certes, le lien avec le coût pour certaines métriques comme le temps d'exécution n'est pas direct, des conversions préalables devant être effectuées.

```
Entrée :P : partition
 Begin
        //IterationLoop
        Loop    //généralement < 5 itérations
                currP = bestP = P
                /* UnlockedLoop */
                While   (Unlocked Nodes Exist (currP) )   Loop
                        swap = SelectNextMove(currP)
                        currP = MoveAndLockNodes (currP,swap)
                        bestP = GetBetterPartition (bestP;cuurrP)
                End loop
                If not (CostFct (bestP)) < CostFct (P))   Then
                     Return P   //Terminate ; pas d'amélioration
                Else    //Faire une autre itération
                        P = bestP , UnlockAllNodes (P)
                End if
        End loop

 End
//Trouver le meilleur échange en essayant toutes les possibilités.
Procedure SelectNextMove (P)
        //Swap Loop
        For  each (unlocked n_i ϵ p_1 , n_j ϵ p_2)    Loop
                Append (costlog , CostFct (Swap(P, n_i , n_j)
        End loop
        Return (n_i , n_j )
```

Où : N = $\{n_1,\ldots n_n\}$ est l'ensemble des nœuds, P = $\{p_1, p_2, DS\}$ avec $p_1 \cap p_2 = \emptyset$, *MoveAndLockNodes* : fonction qui fait l'échange des deux nœuds et les verrouille, *Append* : fonction qui sauvegarde le coût après l'échange 'supposé' des deux nœuds.

Remarque 1. Cet algorithme est à double sens (2 way exchange), s'utilise beaucoup pour le partitionnement fonctionnel.
Remarque 2. Le nombre d'itérations dépend de la complexité de l'application. D'après les expériences ; pour les petites applications, le nombre d'itérations varie généralement entre 5 et 10 mais pour les applications complexes, il peut facilement atteindre 30 itérations.

Parmi les extensions de Kernighan / Lin, on peut citer : **Fiduccia /Mattheyses,** une extension se différenciant de l'algorithme de base par l'utilisation des hypergraphes plutôt que des graphes. Elle effectue des mouvements sur un seul objet plutôt que de faire un échange entre deux objets. Elle redéfinit aussi la primitive *SelectNextMove* pour trouver une bonne solution en un temps constant. **Lookahead and Multiway**, cette autre extension tente de décroître le coût de la partition finale en remplaçant le choix arbitraire des mouvements par un choix plus réfléchi [25].

3. Métriques et contraintes

Les décisions de partitionnement matériel/logiciel d'un système sont prises par l'algorithme suite à l'évaluation d'une fonction objective de coût, à partir de métriques avec des contraintes. Ces métriques doivent satisfaire les trois propriétés suivantes : précision, fidélité et simplicité. Il existe deux classes de métriques : closeness metrics et métriques globales.

3.1. Closeness metrics

Le but des closeness matrics est de grouper les objets fonctionnels qui seront implémentés sur un même composant matériel ou logiciel. Les métriques de rapprochement calculent le gain pour que deux objets puissent être implémentés dans la même technologie (matérielle ou logicielle), par exemple si deux fonctions utilisent les mêmes données, s'exécutent séquentiellement et ont les mêmes exigences matérielles, les grouper dans un seul composant améliorera sûrement la conception (en termes de coût et de temps) [23]. Elles peuvent être utilisées de deux façons : Le **Pre-assignment clustering** permet d'améliorer le processus de conception, les objets les plus proches dans la spécification vont être groupés ensemble. Après cette opération, un partitionnement va être opéré en prenant en considération moins d'objets conduisant à une baisse considérable du temps d'exécution et donnant à priori de meilleurs résultats. Contrairement à la première méthode, le **N-way clustering** procédera à un groupement des objets proches jusqu'à arriver à 'n' groupes dont chacun sera affecté soit à une implémentation en matériel ou en logiciel [21]. Avant de présenter les métriques de rapprochement, on suggère de normaliser leurs valeurs sur l'intervalle [0,1] pour deux

raisons principales : d'abord, pour donner une idée exacte de l'importance des valeurs puis pour donner un sens à la combinaison de valeurs utilisant des unités différentes. Il existe deux techniques de normalisation : dans la **normalisation globale**, la valeur de la métrique calculée sera divisée par un nombre représentant cette même métrique calculée pour tout le système et dans la **normalisation locale**, la valeur de la métrique calculée sera divisée par un nombre représentant la même métrique calculée seulement pour les deux objets concernés. Il existe deux situations de groupement d'objets fonctionnels pouvant affecter la taille, la classe ciblée de métriques : Premièrement, les objets groupés sur le même composant personnalisé peuvent partager souvent le même matériel par exemple un seul multiplieur, réduisant ainsi la taille totale du matériel. Deuxièmement, partitionner des objets parmi un nombre de composants identiques, nécessite un équilibre de la taille nécessaire de chaque groupe d'objets. Parmi les métriques de rapprochement existantes en relation avec la taille, on cite *Hardware sharing* et *Balanced size* :

Hardware sharing, cette métrique mesure la totalité du matériel partagé entre deux ensembles d'objets. Par exemple si deux objets utilisent un multiplieur alors dans ce cas ils seront appelés à partager le même multiplieur. On présume qu'il existe une fonction $Size(O_i)$ qui rend la taille matérielle de l'objet O_i. La métrique sera donc calculée par l'équation (1) comme suit :

$$HardwareSharingMetrics(O_j, O_k) = Size_Shared_{j,k} / norm \quad (1)$$

$$Size_Shared_{j,k} = Size_j + Size_k - Size_{j,k} \quad (2)$$

$$Size_{j,k} = Size(O_j, O_k) \quad (3)$$

$$Size_j = Size(O_j) \quad (4)$$

$$Size_k = Size(O_k) \quad (5)$$

$$norm = Size_both_\min \text{ ; pour la norme locale} \quad (6)$$

$$norm = Size_all \text{ ; pour la norme globale} \quad (7)$$

$$Size_both_\min = Min(Size_j, Size_k) \quad (8)$$

$$Size_all = \sum_i Size(O_j) \quad (9)$$

Où : $Size_j$ et $Size_k$: représentent la taille matérielle nécessaire à l'implémentation des objets O_j et O_k respectivement, $Size_{j,k}$: indique la taille matérielle requise pour une seule implémentation des deux objets O_j et O_k, $Size_Shared_{j,k}$: correspond à la taille du matériel partagée entre les objets O_j et O_k, Size_both_min : c'est le minimum entre $Size_j$ et $Size_k$ qui représente la taille maximale partagée entre ces deux objets, Size_all : indique la taille pour une seule implémentation de tous les objets appartenant à la spécification.

Balanced size, le rôle de cette métrique est de grouper les objets de petite taille plutôt que ceux de grande taille afin d'équilibrer la taille des groupes dans la spécification sur des composants similaires. Elle se calcule d'après l'équation (10) comme suit :

$$BalancedSizeMetric(O_i, O_j) = Size_all - Size_{j,k} / norm \quad (10)$$

$$Size_{j,k} = Size(O_j \cup O_k) \quad (11)$$

Où : Size_all, $Size_{j,k}$, et norm représentent respectivement la taille pour une seule implémentation de tous les objets appartenant à la spécification, la taille matérielle requise pour une seule implémentation des deux objets O_j et O_k et la norme globale[23]. Ces métriques ont un sens puisque nous partitionnons parmi des composants hw et sw. Elles permettent d'équilibrer des groupes d'objets de la spécification, ainsi la charge de travail se trouve répartie sur les composants particulièrement similaires de l'architecture.

3.2. Métriques globales

Le conflit existant entre qualité de la performance du partitionnement et la précision du calcul est un facteur critique pour la grande partie des systèmes de conception conjointe existants. Une approche d'estimation qui donne des résultats proches des valeurs réelles permet une meilleure sélection de la réalisation. Par contre, cette précision dégrade le temps de calcul du partitionnement, ce qui restreint beaucoup l'exploration de l'espace des solutions. Ce conflit, oblige les méthodologies à fixer une stratégie ou une heuristique afin de réduire l'espace des solutions exploré [11] [16].

Le but de l'opération d'estimation est de permettre au concepteur de prendre la plus grande partie des décisions dès les premières étapes de la conception. Elle permet aussi la réduction du coût de réalisation, puisqu'il n'est plus nécessaire de dimensionner excessivement l'architecture afin d'être sûr de satisfaire les contraintes.

Notre objectif consiste à trouver un optimum entre coût et temps d'exécution, c'est la raison pour laquelle nous avons proposé deux sortes de métriques globales : métriques de performance comme le temps d'exécution, temps d'accès, latence de circuit, etc. et métriques de coût comme surface du silicium nécessaire, mise en boîtier, etc. Nous avons défini des métriques globales associées à la technologie d'implémentation. Pour le logiciel : temps d'exécution, temps d'accès et espace mémoire occupé et pour le matériel la surface de silicium (ou nombre de portes), la latence du circuit et sa mise en boîtier.

- Métrique temps d'exécution, on peut utiliser l'une des équations pour la calculer :

$$TempsExécution = NombreCycle * TempsCycle \quad (12)$$

$$TempsExécution = NombreInstruction * CPI / FréquenceHor\log e \quad (13)$$

$$TempsExécution = \sum_{i}^{n} NombreInstruction_i * CPI_i * TempsCycle \qquad (14)$$

- Métrique espace mémoire, cette métrique détermine le nombre d'octets requis pour stocker les instructions et les données manipulées par chacun des blocs.

- Métrique temps d'accès mémoire, pour calculer cette métrique, on procède en quatre étapes : 1) Déterminer les probabilités de branchement en utilisant l'une des méthodes décrites ci-dessus, 2) Un nœud de début *S*, précédant le premier nœud du graphe, est ajouté. Sa fréquence d'exécution, *F(S)* est égale à 1 puisque ce nœud est exécuté exactement une seule fois durant l'exécution du graphe, 3) La fréquence d'exécution $F(N_j)$ pour un nœud N_j dépend des fréquences d'exécution de tous ces nœuds prédécesseurs. La fréquence d'exécution pour chaque nœud prédécesseur N_i est multipliée par la probabilité de branchement $P(e_{ij})$ de l'arc entre N_i et N_j. Pour chaque nœud du graphe, l'équation de fréquence de branchement (15) est :

$$F(N_j) = \sum_{i} N_i * P(e_{ij}) \text{ ; pour tout nœud } N_i \text{ prédécesseur de Ni} \qquad (15)$$

4) Au cours de cette étape, on détermine le *nombre d'accès mémoire d'un noeud.* Il est calculé en multipliant le nombre de variables utilisées dans ce nœud par sa fréquence d'exécution calculée auparavant. Et enfin, en dernière étape, on évalue le *temps d'accès mémoire* d'un bloc d'instructions T_{ab}. Il est décrit par l'équation (16) :

T_{ab} = *Le nombre d'accès mémoire * le temps d'un seul accès e.* (16)

- Métrique surface du circuit, la surface des circuits combinatoires se décompose en surface active et surface des interconnexions. La surface active représente la surface des portes combinatoires effectuant une opération booléenne. La surface d'interconnexion ne représente qu'un petit pourcentage de la surface totale. On en déduit donc que la surface active est une bonne estimation de la surface des circuits combinatoires et que la surface des interconnexions correspondantes peut être négligée [14]. Cette surface résulte principalement de l'architecture choisie.

- Métrique latence, la latence d'un circuit correspond au temps écoulé entre l'entrée des données et leur sortie. Pour un circuit purement combinatoire, il s'agit du chemin critique. Mais pour un circuit séquentiel, elle se calcule en nombre de cycles d'horloge. Souvent, elle s'exprime en secondes et non en cycles d'horloge.

- Métrique coût de mise en boîtier, la mise en boîtier d'un circuit permet de déterminer le coût matériel nécessaire à la réalisation d'une application, ajouté bien sûr à la surface et à d'autres critères.

3.3. Contraintes

Il est connu lors de la conception des systèmes mixtes que l'implémentation matérielle donne de meilleures performances tandis que l'implémentation logicielle baisse considérablement le coût de la réalisation. Donc, il serait intéressant que lors de la conception d'outils, de prendre en considération les désirs de l'utilisateur en soumettant l'application à des contraintes en tenant compte de l'aspect matériel et logiciel comme indiqué en figure 2.

3.3.1. Contraintes de performance

La méthode utilisée permet d'estimer les limites supérieures et inférieures *upper bound* et *lower bound* du temps d'exécution d'un programme source sur une architecture cible. En supposant que L_S et L_i sont respectivement les limites supérieures et inférieures de la performance calculée pour un cdfg d'une application donnée, il faudrait que la performance résultat de l'utilisation de l'algorithme choisi soit incluse dans l'intervalle $[L_S,L_i]$. Cette méthode permet non seulement d'avoir une bonne estimation de la performance mais aussi de vérifier si les exigences ou les besoins de l'utilisateur sont satisfaits par l'architecture cible. Si le résultat est supérieur ou égal à L_S cela signifie que la contrainte est satisfaite mais si le résultat est inférieur ou égal à L_i on dit que l'architecture choisie ne satisfait pas les contraintes [5] [8] [17].

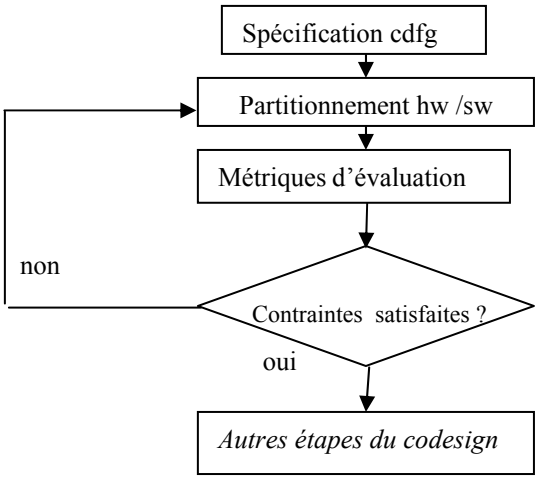


Figure 2. *Influence des contraintes de conception sur le partitionnement*

La détermination de la limite supérieure de la performance pour un *cdfg* donné, comporte deux étapes : On détermine d'abord la limite supérieure pour chaque bloc de

base du *cdfg* puis on déduit la limite supérieure pour tout le *cdfg*. La limite supérieure de la performance pour un bloc de base, correspond à son temps d'exécution qui est équivalent au temps nécessaire à l'exécution des opérations se trouvant dans le *dfg* correspondant. La limite supérieure de la performance pour le *cdfg*, tout d'abord on doit définir la longueur des chemins dans le *cfg* correspondant. Cette longueur se calcule en sommant les temps d'exécution des blocs de base qui forment ce chemin. Pour les boucles, cette longueur sera multipliée par le nombre d'itérations. La limite supérieure de la performance prendra seulement en considération le plus long chemin.

L'algorithme ci-dessous résume la méthode :

Entrées : cdfg
La limite supérieure de la performance pour chaque bloc de base.
Nombre maximum d'itérations pour chaque boucle.
Sorties : La limite supérieure de la performance pour tout le cdfg.
Début
- Associer à chaque nœud du graphe un poids équivalent à sa limite supérieure.

Répéter
Pour chaque boucle interne **faire**
- Déterminer la longueur '*L*' du plus long chemin dans le corps de

L'estimation de la limite inférieure de performance se calcule de la même façon que l'estimation de la limite supérieure de performance sauf que dans ce cas on prend en considération les chemins les plus courts et le nombre minimal d'itérations.

3.3.2. Contrainte du taux d'utilisation du processeur

Pour que cette contrainte soit satisfaite, il faudrait que le taux d'utilisation calculé du processeur soit supérieur au taux estimé [7]. Le taux est calculé en utilisant la formule (17) suivante :

$$Taux(clk) = 1 - ave_T\ slack(clk)/clk \quad (17)$$
$$ave_slack(clk) = \sum_T [occur(t_i) * slack(clk, t_i) / \sum_i occur(t_i) \quad (18)$$
$$slack(clk, t_i) = N * clk - delay(t_i) \quad (19)$$

Les abréviations suivantes clk, N, delay(t_i), slack(clk, t_i), ave_slack et occur(t_i) désignent respectivement : le temps du cycle horloge, le nombre de cycles nécessaires pour l'exécution t_i, le temps effectif de l'opération t_i, le temps perdu, le temps perdu moyen et le nombre d'occurrences de l'opération t_i dans un nœud [9].
L'algorithme ci-dessous décrit le calcul de la contrainte :

Taux =0 ;
Début
Pour chaque nœud logiciel dans le graphe **faire**
Pour chaque opération $\mathbf{P_i}$ constituant le nœud **faire**
Calculer le temps perdu ***slack*** pour chaque $\mathbf{P_i}$
Finpour

- Calculer le temps perdu moyen pour chaque nœud **ave_slack (clk)**
- Calculer **Taux (clk)**

Finpour

Fin

3.3.3. Contraintes du coût global

Cette contrainte sera satisfaite si le coût trouvé comme solution au problème est inférieur au coût souhaité par l'utilisateur. Les métriques globales sont appliquées pour le calcul de la fonction Objectif de cet algorithme dans le but d'atteindre le compromis entre une implantation en matériel coûteuse mais rapide et une autre en logiciel moins chère mais gourmande en temps d'exécution.

4. Le système Autodec

Le choix des caractéristiques suivantes est important : le modèle en entrée, la granularité de la spécification, les métriques d'estimation et l'algorithme de découpage.

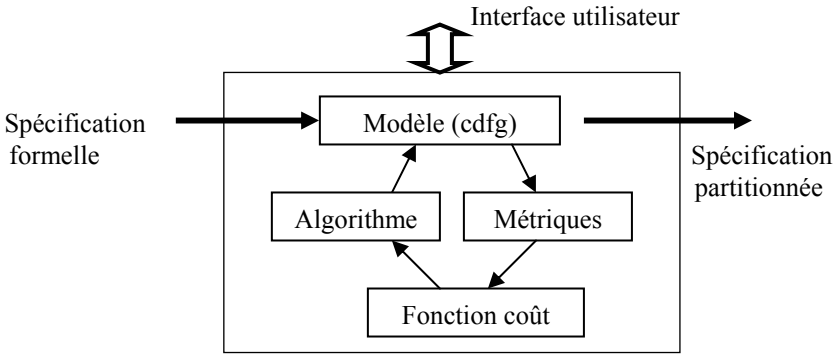


Figure 3. *Modèle de conception d'AutoDec*

4.1. Principe

Le but de notre outil, est de fournir à partir d'une spécification formelle une description partitionnée en utilisant des métriques de rapprochement et globales. Les

métriques de rapprochement confectionnent une partition initiale et les métriques globales l'améliorent par itérations successives. La figure 3 montre ce processus.

4.2. Modèles en entrée et architecture

Quand les modèles utilisés pour la spécification et pour l'architecture cible sont simples, le partitionnement automatique est alors possible [15]. Pour cette raison, on a choisi comme spécification d'entrée un cdfg puisque c'est un modèle simple à comprendre et en même temps puissant pour la modélisation des systèmes complexes car il permet la représentation des dépendances entre les données ainsi que les séquences d'exécution. Comme granularité pour cette spécification, on a considéré un bloc de base. L'architecture cible adoptée est monoprocesseur, composée d'un processeur central agissant comme maître, d'une mémoire et d'un ensemble de coprocesseurs arithmétiques se comportant comme esclaves (*figure 4*).

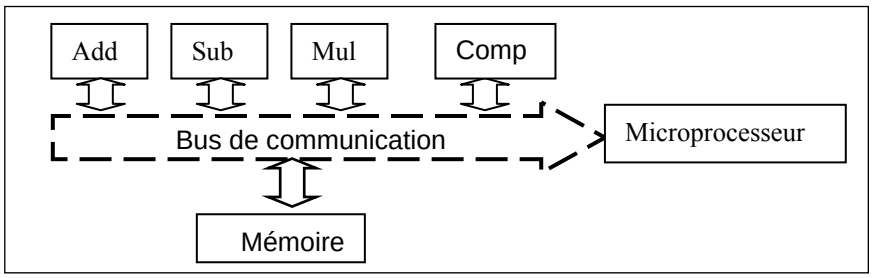


Figure 4. *Architecture cible*

Add : Additionneur, Mul : Multiplicateur, Sub : Soustracteur et Comp : Comparateur. Les caractéristiques de cette architecture sont données dans le tableau 1 suivant :

Métrique/copr.	Additionneur	Multiplieur	Soustract	Comparateur
Surface	120	100	110	120
Boîtier	10	20	30	40
Temps d'exéc.	80	150	90	100

Tableau 1. *Caractéristiques de l'architecture cible*

Lors des calculs, on a considéré les valeurs des types de base telles qu'un ***entier*** occupe 2 octets, un ***réel*** 4 octets et un ***caractère*** 1 octet en mémoire. De même, le

temps de cycle du microprocesseur est de 2 ns et que le temps d'un accès mémoire est de 120 ns [1] [4] [6] [19].

4.3. Métriques

L'un des défis majeurs du partitionnement automatique est l'obtention d'un coût minimal de la solution finale. Les critères choisis pour l'estimer varient d'un outil à un autre. A chaque critère est associé une valeur métrique de même qu'un facteur qui sert à les pondérer et qui dépend de la technologie utilisée et du domaine d'application.
Il est donc difficile de définir une fonction d'estimation qui soit réaliste même pour un domaine d'application spécifique. *AutoDec*, notre outil réalisé, fournit une estimation du coût de la partition résultat en utilisant comme métriques d'évaluation : La surface en silicium nécessaire pour l'implémentation, la mise en boîtier des circuits utilisés, le temps d'exécution requis, l'espace mémoire utilisé et le temps d'accès mémoire. Notre but étant de minimiser cette fonction coût tout en cherchant un compromis coût / performance. La fonction objective se formalise comme suit :

$$Min(Fct) = k1 * surf + k2 * meb + k3 * tpsexe + k4 * espace + k5 * acces$$

Sous les contraintes : $CTotal \leq coûtglobal,\ PerfTotal \geq performance\ \ et\ \ Taux \geq taux$

Avec : *Fct* représentant un coût global; *Surf* indiquant une surface en silicium du circuit (mm²), *meb* une mise en boîtier du circuit, *tpsexe* un temps d'exécution, *espace* un espace mémoire utilisé, *acces* un temps accès mémoire et *Ki* des facteurs choisis en fonction de l'importance de la métrique dans le calcul de la fonction Objectif.

4.4. Méthode de partitionnement

Pour réaliser AutoDec, on a opté lors du choix d'algorithmes pour *Kernighan /Lin*. Pour remédier à ses inconvénients, notre outil tâchera de fournir une solution initiale *non aléatoire*. Cette solution est calculée à partir de *métriques de rapprochement*. Rappelons que lors de la spécification, le grain a été pris égal à un bloc de base, les métriques de rapprochement appropriées, décrites précédemment sont : *Hardware Sharing* et *Balanced Size*. Le seuil utilisé ou condition d'arrêt pour la partition initiale est de 0.4 c'est-à-dire qu'on arrête le processus de regroupement si on ne trouve plus deux noeuds avec une closeness metric supérieure à 0.4. Les autres métriques de rapprochement sont beaucoup plus intéressantes à appliquer quand les grains choisis sont des procédures ou des fonctions. Au moyen de métriques globales, l'algorithme Kernighan/Lin modifie une fonction coût. Lorsque la condition d'arrêt est vérifiée c'est-à-dire que les valeurs successives de la fonction coût ne diffèrent plus que d'une

précision de l'ordre de 5% sachant que cette dernière est une fonction monotone et décroissante, alors la solution optimale est atteinte. La partition finale est exprimée ensuite sous forme d'un cdfg bicoloré où chacune des deux couleurs indique une technologie d'implémentation logicielle/matérielle. Les principales étapes de conception de l'outil *Autodec* sont résumées à la figure 5.

4.5. Schéma général d'AutoDec

Nous avons recensé les principales fonctionnalités de notre démarche ainsi que les liens entre-elles ci-dessous :

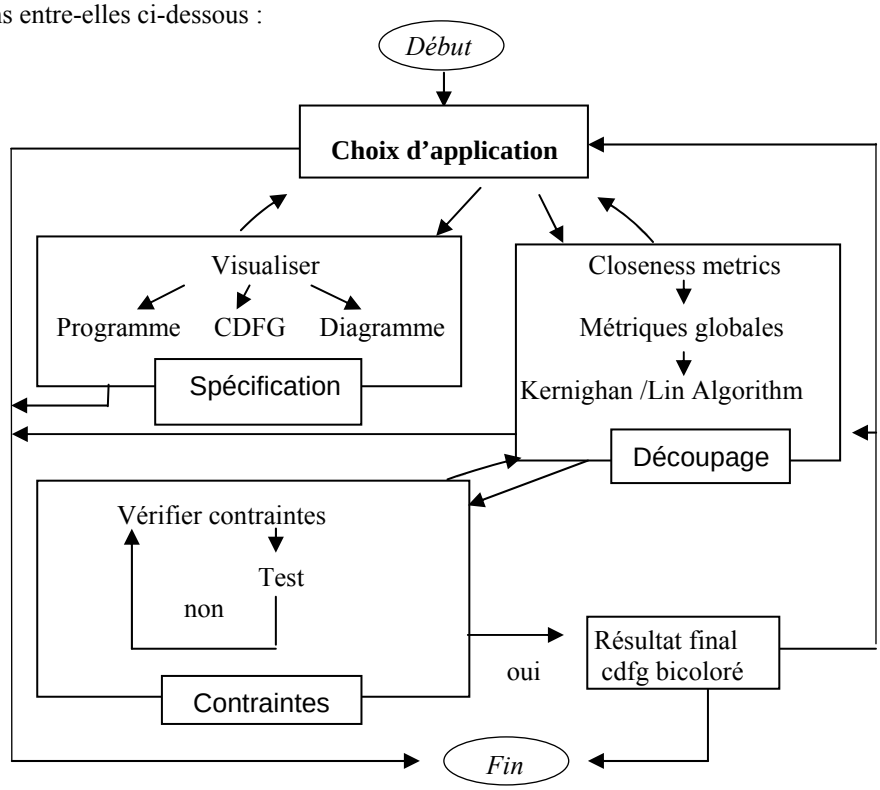


Figure 5. *Schéma des liens entre les fonctionnalités d'AutoDec*

5. Expérimentation et tests

Pour valider cet outil, trois applications sont considérées :

- Une application réelle : un distributeur automatique, cas du sous-système de rendu de monnaie.
- PGCD (Plus Grand Commun Diviseur) : Il s'agit d'un algorithme de calcul du plus grand diviseur commun de deux nombres.
- Un algorithme standard, écrit en langage VHDL sans fonction sémantique déterminée mais dont le rôle principal est de servir dans la comparaison de résultats.

5.1. Distributeur automatique

Etant donné un distributeur qui peut être de friandises ou de boissons, n'acceptant que des pièces de 5, 10 ou 20 um (unités monétaires), on va prendre en considération le sous-système de rendu de monnaies. La machine rend de préférence des pièces de 10 um, ensuite des pièces de 5 um. Par exemple, sur 20 um, la machine rend de préférence (si le stock de pièces le permet) 10 um + 5 um plutôt que trois fois 5 um. Pour implémenter ce système, on doit disposer : d'une information N concernant le type de pièce introduite ($N = 5 \rightarrow$ Pièce de 5 um, $N = 10 \rightarrow$ Pièce de 10 um et $N = 20 \rightarrow$ Pièce de 20 um), d'une information S_i donnant l'état de stock des pièces ($S_5 \rightarrow$ Stock de pièces de 5 um, $S_{10} \rightarrow$ Stock de pièces de 10 um et $S_{20} \rightarrow$ Stock de pièces de 20 um) ainsi que des informations donnant l'état du stock du produit et l'état du distributeur.

5.2. PGCD

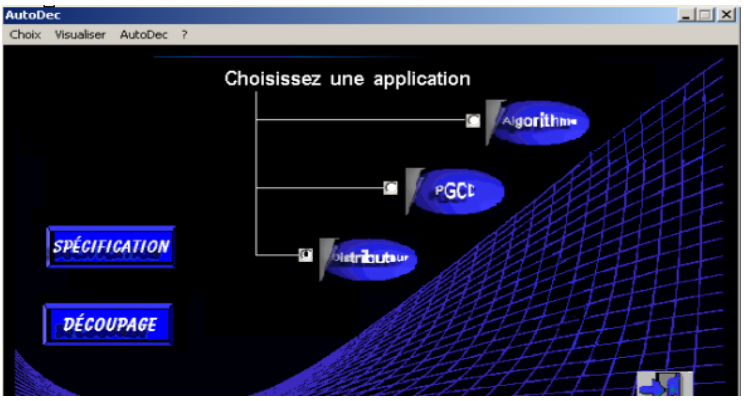
Cet exemple calcule le plus grand diviseur commun pour deux entiers quelconques.

5.3. Algorithme standard

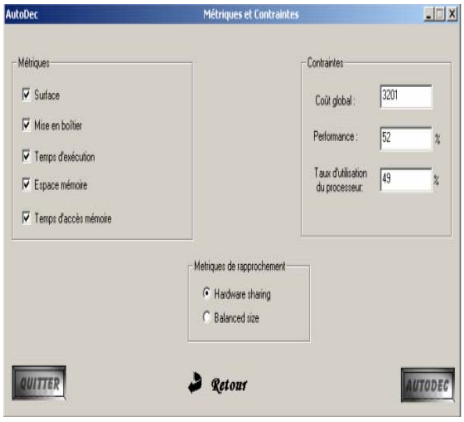
Cet exemple, tiré du guide utilisateur VHDL, est un algorithme standard dont les résultats optimaux sont connus, utilisé à des fins de comparaison de résultats.

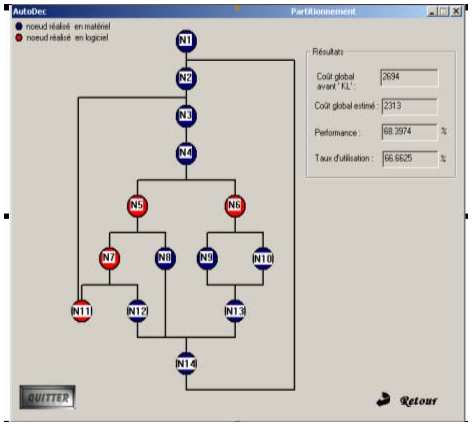
5.4. Quelques écrans du système Autodec

Nous présentons dans cette sous-section quelques écrans pour donner une idée sur les possibilités de notre outil. La figure 6.a montre l'interface principale, la figure 6.b visualise les choix retenus par le concepteur des différentes métriques et contraintes et en figure 6.c, est affiché le résultat du partitionnement M/L du distributeur automatique représenté par un cdfg bicoloré.



(6.a)





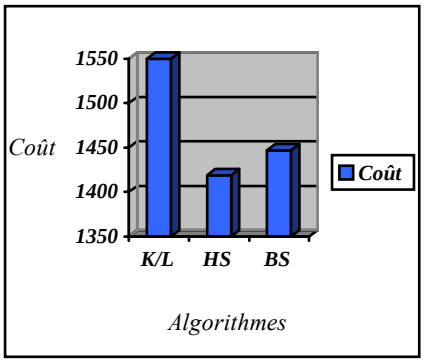
(6.b) **(6.c**

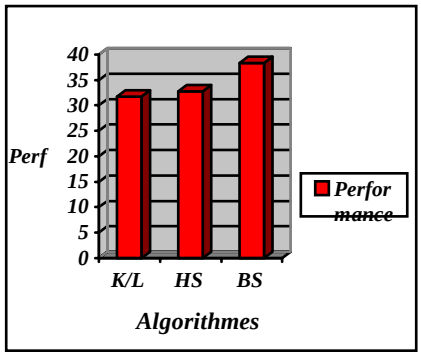
Figure 6. (6.a) *interface utilisateur*, **(6.b)** *choix de métriques et de contraintes par l'utilisateur et* **(6.c)** *CDFG distributeur automatique partitionné*

Une double comparaison a été imposée à AutoDec : d'abord par rapport à Kernighan/Lin seul puis par rapport à deux approches, l'une utilisant le recuit simulé et l'autre un algorithme génétique sur des exemples similaires et sous les mêmes conditions [18] [28].

Les graphiques ci-dessous montrent les résultats de simulation obtenus sur l'application du distributeur automatique en utilisant : seulement l'algorithme de Kernighan/Lin, Kernighan/Lin en combinaison avec hierarchical clustering en

appliquant *Hadware Sharing* (HS) puis *Balanced Size* (BS) comme closeness metrics. On constate que la closeness metric HS a une influence plus importante que BS sur le coût, elle le réduit de 5% (figure 7.a) tandis que BS agit beaucoup plus sur le temps d'exécution et par conséquent sur la performance donnant respectivement une réduction de 20 % du temps d'exécution et une amélioration de performance de 25 %.approximativement (figure 7.b).





(7.a) **(7.b)**

. **Figure 7. (7.a)** *diagramme montrant la réduction de coût Kernighan/Lin combiné*, **(7.b)** *diagramme illustrant le gain en performance de K/L combiné*

Le tableau 2, résume des résultats obtenus par Kernighan/Lin combinée avec hierarchical clustering utilisant la métrique HS et les algorithmes de recuit simulé et génétique. Ces résultats sont des moyennes arithmétiques de 3 essais successifs. Le choix de paramètres se trouve dans [3], pour le recuit simulé la température initiale T est prise égale à 7000, loi de décroissance T=T-10 et critère d'arrêt à T = 0 ou après 50 déplacements sans changements de valeur de la fonction coût. Pour l'algorithme génétique, la taille de la population est égale à 20 individus, le nombre de générations est de 100, l'opérateur de mutation est appliqué avec une probabilité de 0.001 et le critère d'arrêt est soit le nombre de générations ou une convergence de la population vers une même solution. Pour ces deux algorithmes, la partition initiale est aléatoire. L'expérience consiste en 3x3 simulations faites sur le troisième exemple pour chacun des algorithmes avec des contraintes et des métriques différentes.

Algorithmes de partitionnement						***RS***	***AG***	***KL_HS***
Contraintes					Coût global	PS	740	707
Taux CPU ≥ 50,33 % Coût matériel ≤ 750					Taux	PS	59.35	59.35
Métriques					Coût matériel	PS	240	240
C	S	E	T	A				
Contraintes					Coût global	244	208	200
Taux CPU ≥ 0% Coût matériel ≤ 900					Taux	60,82	60,68	60.7
Métriques					Coût matériel	130	86.66	80.5
C	S	E	-	A				
Contraintes					Coût global	372	284	270
Taux CPU ≥ 0 Coût matériel ≤ 300								
Métriques					Taux	60,4	60.4	59
-	S	-	T	A	Coût matériel	60	76.66	70

Tableau 2. *Résultats comparatifs KL combiné à Hierarchical clustering avec recuit simulé et algorithme génétique*

Avec C : coût global, S : surface du matériel, E : espace mémoire occupé par les données,.T : temps d'exécution et A : accès mémoire. Les abréviations utilisées dans le tableau concernant les algorithmes sont RS pour Recuit Simulé, AG pour algorithme génétique et KL_HS pour Kernighan/Lin combiné avec Pre-assignment clustering utilisant la closeness metric Hardware sharing.

Le tableau 2 montre clairement que les résultats obtenus par l'algorithme Kernighan/Lin sont satisfaisants comparés à ceux obtenus par les algorithmes de recuit simulé et génétique. D'après ces résultats, le clustering pour l'algorithme de KL est d'un apport important car il le fait converger vers la solution optimale en une seule itération tout en améliorant la performance sans un surcoût supplémentaire.

En conséquence, nous pouvons avancer que les closeness metrics utilisés par l'algorithme herarchical clustering réduisent le temps d'exécution sans perte de la qualité de la partition (pas d'augmentation du coût) pour l'algorithme Kernighan/Lin. Cette réduction du temps d'exécution est intéressante à plus d'un titre, d'une part elle encourage le concepteur à traiter de gros problèmes et d'autre part elle éveille un vif intérêt envers les algorithmes de partitionnement automatique. Les closeness metrics

utilisées par l'algorithme constructif peuvent être considérées comme un complément des métriques globales, puisque l'ensemble de ces métriques concourt à l'amélioration des performances et à la réduction du coût. Autres caractéristiques de notre travail : la fonction objective avec plus de critères contraints et pondérés et également l'interactivité entre l'outil et le concepteur qui est importante à nos yeux : le concepteur est appelé à prendre diverses décisions sur le choix des métriques, des contraintes par exemple pour partitionner son système avec une assistance aisée de Autodec dans la sélection et la correction.

6. Conclusion

Notre travail peut être perçu sous plusieurs aspects : la combinaison de deux algorithmes constructif (Hierarchical Clustering) et itératif (Kernighan / Lin), basés sur l'utilisation des métriques, l'extension de la fonction Objectif à plus de critères et l'interactivité concepteur-outil. Les résultats obtenus sont satisfaisants comparés à ceux obtenus par les algorithmes génétiques et de recuit simulé avec une partition initiale aléatoire. Nous estimons avoir amélioré l'algorithme kernighan/Lin en le dotant d'une partition initiale réfléchie et nous pouvons dire sans réserves que le clustering avec un choix judicieux de métriques, améliore les algorithmes de partitionnement itératifs, et par conséquent réduit le gap entre les algorithmes rapides et les algorithmes hautement optimaux.

Ce travail peut être étendu à d'autres métriques de rapprochement comme celles de connectivité ou de performance voire les combiner éventuellement et en diversifiant autant que possible les domaines d'applications. Nous sommes actuellement en train de refaire la même expérience avec les algorithmes de recuit simulé et génétique pour les doter d'une solution initiale réfléchie afin de consolider nos propos sur le clustering, qu'il est un facteur améliorant de la qualité de solution et des performances des algorithmes de partitionnement itératifs. Une autre direction améliorante du travail porte sur la combinaison du partitionnement avec l'ordonnancement des blocs de base sur les composants de l'architecture.

Enfin, mentionnons que le partitionnement automatique, avec la jeunesse de ses outils et méthodes actuelles, ne peut prétendre concurrencer le partitionnement interactif sur les petites applications, dans lequel l'intervention du concepteur expert demeure incontournable. Toutefois, pour les applications assez complexes, un partitionnement automatique est nécessaire.

7. Bibliographie

[1] Azizi Mostafa, "Covérification des Systèmes Intégrés", Phd de l'université de Montréal, 2000.

[2] Ben Ismai*l* Tarek, *"*Synthèse au niveau système et conception de systèmes mixtes Logiciels/Matériels"*,* Thèse INPG, Grenoble 1996.

[3] Chaffai M. N., Bouzbid H. et Boudour R., « Partitionnement automatique au niveau de la méthodologie codesign », RR 10-2002, département d'informatique, université d'Annaba, Algérie, 2002

[4] Coste Pascal, "Conception des systèmes hétérogènes multilangages", Thèse de l'université Joseph Fourier, 2001.

[5] Diguet J. P., Gogniat G. et Martin E., *«*Codesign ou Conception Conjointe Logiciel/Matériel», Lester, UBS.

[6] Eles P., Kuchcinski K. et Peng Z., "System synthesis with VHDL", Kluwer Academic Publisher, 1998.

[7] Gajski D.D., Vahid F., Narayan S. et Gong J., "Estimation", Université d'Irvine, 1994.

[8] Gajski D.D. et al., " Specification and design of Embedded Systems", *PTR Prentice Hall*, 1994, pp. 171-231.

[9] Gong J., Gajski D. D., Narayan S., *"Software Estimation from Executable Specifications", T* echnical Report ICS-93-5, 1993.

[10] Kahlo Alexander, "Functional verification of an ASIC design on register transfer level with Celaro/ModelSim co-simulation" , Thèse University of Applied Sciences Braunschweig, 2000.

[11] Kajonen Jarkko, "*Electronics and signal processing",* 2001

[12] Knerr B., Holzer M. et Rupp M., « Hw/Sw Partionin Using High level Metrics », proceedings conférence CCCT, p. 33-38, Austin, 2004.

[13] Kuchcinski Kris, " *System Partitioning"*, 2002.

[14] Laurent Bernard, "conception des blocs réutilisables et réflexion sur la méthodologie", Thèse INPG, 1999.

[15] Marchioro Gilberto Fernandes, *"Découpage Transformationnel pour la Conception de Systèmes Mixtes Logiciel/Matériel",* Thèse INPG, 1998.

[16] Martin Eric, "Méthodes de développement logiciel/matériel : le CODESIGN", Brest, 1998.

[17] MCC/OMI, "Hardware software codesign study report", MCC/OMI, 1997.

[18] Serra M. et Gardner W. B.*,* *"* Hardware/Software Codesign – introducing an interdisciplinary course*", Conférence WCCCE Vancouver, 1998*.

[19] Roux Sébastien*, «Adéquation Algorithme – Architecture pour le traitement multimédia embarqué »*, Thèse INPG, 2002.

[20] Suger Zoltan, *" Synthèse comportementale basée sur l'ordonnancement*", Thèse INPG, 2000.

[21] Thomas Hervé, Diguet J-P. et Philippe J-L., "Estimation et métriques au niveau système pour la conception conjointe logicielle/matérielle", Lester, UBS

[22] Vahid F. et Gajski D.D., *" Clusetring for improved system-level functional partitioning", 1997.*

[23] Vahid F. et Gajski D. D., *"Closeness metrics for system-level functional partitioning*", *EURODAC* 1995, p. 328-333.

[24] Vahid F. et Thuy Dm LE, *"Extending the Kernighan / Lin heuristic for hardware and software functional partitioning*", Design automation for embedded systems, Vol.. 2, p. 237-261, 1997.

[25] Vahid et Le, "*HW/SW Partitioning based on Kernighan and Lin*", Version 2, October 2000.

[26] Valderrama Carlos Alberto, *"* Prototype virtuel pour la génération des architectures mixtes Logicielles/Matérielles"*,* Thèse INPG, 1998.

[27] Wander Oliveira Cesărio, *"Synthèse architecturale flexible*", Thèse INPG, 1999.

[28] Williams Mickey, "*VISUAL C++ 6"*, Edition *CampusPress*, 2001