Un algorithme hybride pour la résolution du problème de partitionnement matériel/logiciel

Hybrid algorithm for solving hardware/Software partitioning problem

Mourad Khetatba* & Rachid Boudour

Laboratoire de systèmes embarqués -LASE-, Université Badji Mokhtar-Annaba.
BP 12, Annaba 23000, Algérie.

Soumis le : 16/07/2017 *Révisé le : 19/06/2018* *Accepté le : 20/06/2018*

ملخص

لتطوير نظام مدمج مع نسبة من التكلفة والأداء معقولة، اتجه المصممون نحو التصميم المشترك (عتاد/ برمجيات) أو co-design. تشمل هذه المنهجية سلسلة من المراحل , اول خطوة هي الموصفات القابلة للتنفيذ و آخرها النموذج الافتراضي للنظام. التقسيم عتاد/ برمجيات , خطوة حيوية في عملية التصميم المشترك , تهدف الى تقسيم سلوك النظام الى مجموعتين فرعيتين من الوظائف ، الاولى عتاد والثانية برمجيات. بما أن مسألة التقسيم عتاد / برمجيات هي جد صعبة NP-difficile ,فالمراجع العلمية تعرف وفرة من المناهج التكميلية لمعالجة تقريبية لهذه المشكلة. في هذه الورقة، نقترح خوارزمية تقسيم عتاد / البرمجيات جديدة هجينة وموثوق بها تقوم على الخوارزمية (Late Acceptance Hill Climbing) والخوارزمية الوراثية (Algorithme Génétique: AG). و معروفا أن LAHC وسيلة فعالة للبحث المحلي، يضاف لها الإمكانيات الكبيرة للبحث الشامل الخاصة بالخوارزمية الوراثية .AG تم تطبيق الخوارزمية الهجينة وأعطت نتائج أفضل مقارنة مع تلك الاعمال المنجزة سابقا.

الكلمات المفتاحية: *تقسيم عتاد/برمجيات-الطرق التقريبية العامة-قبول متأخر للمنحدر -خوارزمية وراثية-خوارزمية هجينة*

Résumé

Pour développer un système embarqué avec un rapport coût / performance raisonnable, les concepteurs s'orientent vers la conception conjointe matériel/logiciel (M/L) ou codesign. Cette méthodologie comporte une suite d'étapes dont les résultats de la première est une spécification exécutable et la dernière est un prototype virtuel du système. Le partitionnement M/L, une étape cruciale du processus de Co design, a pour finalité de diviser le comportement d'un système en deux sous-ensembles de fonctions, l'un matériel et l'autre logiciel. Le problème de partitionnement M/L étant NP-difficile, la littérature foisonne d'approches complémentaires pour remédier approximativement à ce problème. Dans ce papier, nous proposons un nouveau algorithme hybride et fiable de partitionnement M/L basé sur le principe de l'algorithme Late Acceptance Hill Climbing(LAHC) et de l'algorithme génétique (AG). LAHC est connue pour sa recherche locale efficace, conjuguée à la potentialité importante de recherche globale de l'AG. L'algorithme hybride, a été appliqué et a donné de meilleurs résultats, comparés à ceux de l'état de l'art.

Mots clés : *Partitionnement M/L-Metaheuristique-Acceptation tardive de Hill Climbing- Algorithme génétique-Algorithme hybride*

Abstract

To develop an embedded system with a reasonable cost / performance ratio, the designers were moving towards the hardware/software co-design. This methodology comprises a sequence of steps whose results from the first is an executable specification and the latter is a virtual prototype of the system. Hardware/software Partitioning, a crucial step in Co-design process, aims to divide the system's behavior in two subsets of functions, one hardware and the other software. The Hw/Sw partitioning problem being NP-hard, the literature abounds with complementary approaches to remedy this problem roughly. In this paper, we propose a new hybrid and reliable Hw/Sw partitioning algorithm based on the principle of the Late Acceptance Hill Climbing (LAHC) technique and the Genetic Algorithm (GA). LAHC is known for its efficient local research, combined with the important potentiality of the AG total research. The hybrid algorithm has been applied and has given better results, compared to those of the state of Art.

Key words: *Hw/Sw partitioning-Metaheuristic-Late Acceptance Hill Climbing- Genetic Algorithm- Hybrid Algorithm*

* *Auteur correspondant :khetatbam@yahoo.fr*

1. INTRODUCTION

Les systèmes embarqués envahissent notre quotidien personnel et professionnel. Ils sont constitués très souvent de composantes logicielles et matérielles. Cependant, la conception de ces systèmes ne peut être implémentée par des méthodes classiques où la conception du matériel précède celle du logiciel.

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 (M/L). La méthodologie la plus récente de conception Matériel/Logiciel (M/L), est connue sous le nom de Co-design [1]. Le Co-design M/L permet de concevoir en même temps le matériel et le logiciel pour une fonctionnalité à implémenter à la recherche des meilleurs compromis entre les différentes solutions. Le but du Co-design est de réduire le temps de mise sur le marché tout en réduisant l'effort de conception et les coûts des produits conçus [2]. Cette méthodologie est structurée en plusieurs étapes partant de la spécification jusqu'à la réalisation physique. Concevoir un système complexe passe obligatoirement par la décomposition de son comportement en un ensemble de fonctions ou modules. L'étape la plus délicate est celle du partitionnement du système en modules matériels ou logiciels [3]. Il est primordial de développer dans les premières étapes du flot de conception des méthodes de partitionnement M/L pour compenser la croissance de la complexité des applications. Les résultats de partitionnement affectent directement le coût et la performance du système applicatif final [4]. En général, plusieurs facteurs (ou métriques) sont pris en compte pour ce problème tels que les contraintes temporelles, la surface de silicium, la consommation. Ces métriques sont combinées pour former une fonction de coût ou d'objectif. Cette fonction exprime donc les critères que le concepteur veut minimiser.

Du fait de tous ces paramètres et des objectifs d'optimisation, le problème du partitionnement est bien connu comme étant un problème NP-difficile [5]. IL assure la transformation des spécifications de la partie du système relevant de l'activité Co-design en une architecture composée d'une partie matérielle et d'une partie logicielle. De nombreuses méthodes ont été développées pour résoudre ce problème et visent à automatiser le processus de partitionnement M/L.

Mais, il n'existe pas de méthode d'optimisation qui, pour un problème donné, sera meilleure que toutes les autres sur toutes les instances possibles. Toute méthode choisie a des avantages et des inconvénients. Par conséquent, les chercheurs ont combiné ces méthodes et ont conçu des algorithmes hybrides pour avoir une solution optimale au problème. Ces méthodes hybrides gagnent maintenant en popularité, car elles arrivent à fournir de meilleurs résultats. Ainsi, dans ce papier, nous proposons une hybridation séquentielle de deux algorithmes Late Acceptance Hill Climbing(LAHC) [6] et génétique(AG) [7]. Étant donné que la technique LAHC est relativement récente, les variations de cette méthode n'ont pas encore été largement explorées. Par conséquent, la combinaison de LAHC avec d'autres méthodes et stratégies est un champ ouvert à l'expérimentation. LAHC est une méthode de recherche locale à base de solution unique, son but est d'améliorer la qualité (la « fitness ») d'une solution par exploration de son voisinage. La méthode de descente ou Hill-climbing est facilement piégée dans un des optimums locaux, alors que, LAHC essaie d'échapper à ces minima locaux en sortant des sous-espaces de l'espace de recherche. Le premier pas dans l'implantation des algorithmes génétiques est de créer une population d'individus initiaux. Les méta heuristiques basées sur une population de solutions (par exemple AG) sont très sensibles au choix initial de la population et sont sujettes à une détérioration de la diversité si aucune précaution n'est prise par rapport au problème. Cette population sera construite, pour notre étude, à partir des solutions fournies par LAHC. L'algorithme génétique a une forte capacité de recherche globale. L'application des opérateurs génétiques permet à l'AG d'éviter de converger vers des extrema locaux de la fonction et de créer des éléments originaux. L'algorithme proposé sera testé sur des modèles de différentes tailles. Ce papier est organisé comme suit : La section 2 explique le partitionnement M/L en co-conception et des mémoires sur les travaux précédents dans ce domaine, ainsi que la présentation de certains concepts préliminaires sur LAHC et AG. La formulation du problème de partitionnement est donnée dans la section 3. Le modèle combiné proposé pour le problème de partitionnement est décrit dans la Section 4. Les résultats de l'expérience sont montrés à la Section 5. La section 6 conclut le document et discute des travaux futurs.

2. ETAT DE L'ART ET PRELIMINAIRES

2.1 Etat de l'art

Au fil des décennies, plusieurs approches ont été proposées. Elles diffèrent dans la spécification initiale, le niveau de granularité auquel le partitionnement est effectué, le degré d'automatisation du processus de partitionnement, la fonction coût et l'algorithme de partitionnement. Edwards [8] donne l'historique et les premiers travaux concernant le problème de partitionnement M/L. Le découpage en parties matérielles et logicielles du système était effectué manuellement et dépendait des expériences du concepteur [9]. Ces méthodes sont devenues incompatibles avec les contraintes de temps actuelles de mise sur le marché. C'est pourquoi, de nombreuses méthodes ont été développées et visent à automatiser la tâche de partitionnement. Au Début, des approches orientées logiciel [5] ou orientées matériel [10] ont été présentées où les fonctions entières du système sont d'abord implémentées soit en logiciel, soit en matériel. Ensuite, les deux approches faisaient progressivement migrer les partitions du logiciel vers le matériel ou vice-versa pour réduire le coût du système.

Le problème de partitionnement M/L peut être formulé sous forme d'un problème d'optimisation combinatoire. Pour cela, les chercheurs ont proposé deux approches (méthodes exactes et des méthodes approchées) pour résoudre efficacement certains problèmes d'optimisation combinatoire (Fig. 1).

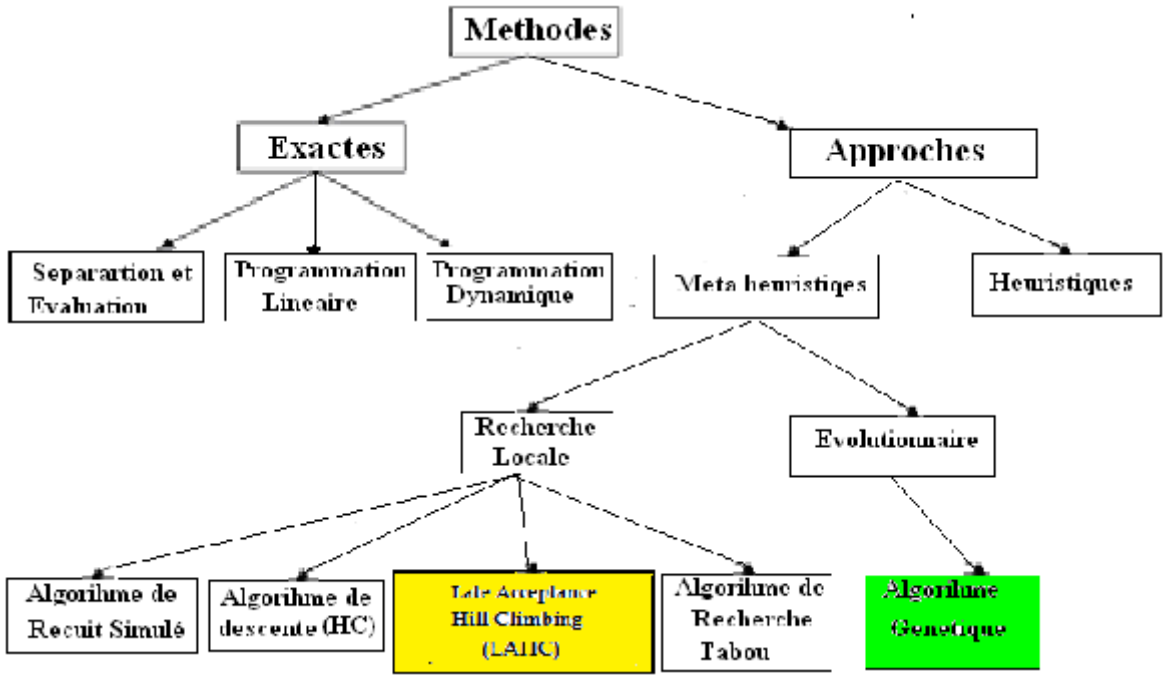

Figure 1 : Principaux algorithmes de partitionnement M/L

Les méthodes (ou algorithmes) exactes garantissent de trouver la solution optimale et de prouver son optimalité pour toutes les instances d'un problème d'optimisation de taille raisonnable, même si c'est un problème NP-difficile [11]. Le principe d'une méthode exacte est généralement d'énumérer souvent implicitement, toutes les solutions de l'espace de recherche. Donc, même l'ordinateur le plus puissant ne pourra pas, dans des temps réalistes, énumérer toutes ces solutions pour trouver la solution optimale. Par conséquent, ces méthodes ont généralement le défaut d'être très coûteuses en temps d'exécution, ainsi elles ne peuvent pas être appliquées à de grands problèmes NP-difficiles. Plusieurs approches de résolution exacte ont été élaborées. Nous pouvons citer La méthode séparation et évaluation [12], La programmation linéaire [13], la programmation dynamique [14].

Au contraire, les méthodes approchées visent à générer des solutions de haute qualité en un temps de calcul raisonnable, mais il n'existe aucune garantie de trouver la solution optimale. Leur utilisation est fortement recommandée pour trouver des solutions qui se rapprochent de l'optimum là où les méthodes exactes échouent. Mais, Il est déconseillé de les utiliser pour résoudre des problèmes où des

algorithmes exacts et efficaces sont disponibles. Dans cette catégorie de méthodes, les méta heuristiques [15,16] se sont distinguées pendant les dernières décades et ont montré leur efficacité dans de vastes domaines d'application en résolvant de nombreux problèmes d'optimisation pour ne citer que le problème de partitionnement. Plusieurs définitions d'un méta heuristique ont été proposées dans la littérature. Les métas heuristiques sont des stratégies permettant de guider la recherche d'une solution optimale. Elles sont en général non déterministes (pas de garantie d'optimalité). Généralement, les métaheuristiques contiennent des techniques pour pouvoir échapper à des minima locaux. Boussaid [17] donnent un bon aperçu sur ces méthodes. Les métaheuristiques peuvent être classées en deux grandes familles [18], celles de voisinage ou à base de solution unique et celles à base de population. Les métaheuristiques de voisinage (telles que les algorithmes de recherche tabou(RT) [19], de recuit simulé (RS) [20], algorithme LAHC pour Late Acceptance Hill Climbing [6], etc.) ont en commun d'avoir un mécanisme d'acceptation de solutions dégradées leur permettant de s'extraire des minima locaux. Celles à base de population (Les algorithmes génétiques (AG) [7], algorithmes à optimisation d'essaim de particules(OEP) [21], etc.) disposent d'un mécanisme collectif pour sortir des minima locaux. Mais, il n'existe pas de méthode d'optimisation qui, pour un problème donné, sera meilleure que toutes les autres sur toutes les instances possibles. Ainsi, les chercheurs ont combiné les avantages de deux ou plusieurs algorithmes pour en faire un seul, appelé algorithme hybride. Ces avantages ou points forts devraient être complémentaires entre eux de sorte que l'algorithme hybride résultant puisse en bénéficier deux [11,22]. Les méthodes hybrides peuvent être divisées en deux groupes [18] : les métaheuristiques hybrides qui impliquent une combinaison de plusieurs métaheuristiques et les méthodes hybrides impliquant une combinaison d'une méthode exacte et d'un métaheuristique. Un état de l'art de ces coopérations Meta/exactes a d'ailleurs été proposé récemment par Stützle et Dumitrescu [23]. La recherche locale, le recuit simulé, la recherche avec tabous et les algorithmes évolutifs ont déjà été hybridés avec succès dans plusieurs applications [24,25, 26].

2.2 Concepts Préliminaires

2.2.1. L'algorithme Late Acceptance Hill-Climbing

Late acceptance Hill-Climbing [6], comme son nom l'indique, LAHC est dérivé de l'algorithme de Hill Climbing (HC). C'est l'une des nouvelles méthodes de recherche locale qui modifie une solution et accepte des changements si la solution courante modifiée est meilleure que la solution courante déjà plusieurs itérations. Lorsqu'une fonction objective contient plusieurs optimums locaux concernant le voisinage étudié par la stratégie de recherche, le Hill-Climbing est facilement emprisonné dans un des optimums locaux, alors que LAHC essaie d'échapper de ces minima locaux. Il commence à partir d'une simple solution initiale (habituellement aléatoire), et à chaque itération modifie de manière aléatoire la solution courante afin de produire une solution candidate dans le voisinage. Alors l'idée fondamentale est de retarder la comparaison entre les solutions du voisinage et de comparer les nouvelles solutions candidates à une solution courante qui existait depuis plusieurs étapes passées déjà. La recherche est généralement terminée lorsqu'elle converge (aucune amélioration n'est détectée pendant un certain nombre d'itérations). Le LAHC a deux avantages majeurs par rapport à ses concurrents. Tout d'abord, il est nécessaire de configurer la longueur du vecteur d'adaptation (Lfa : List fitness array) plutôt que d'un programme de refroidissement tel que le recuit simulé, puis, l'absence d'un calendrier de refroidissement rend l'algorithme plus fiable. Le fonctionnement du LAHC est illustré par la figure 2.

(1) Initialiser les paramètres de LAHC ; // Nb_iter(pour Nombre d'itérations)= 0 ;
//Lfa(pour la taille du vecteur d'adaptation)=20 ;
(2) Générer une solution aléatoire X0 ;
(3) Calculer la fonction cout f(X0)
(4) Meil_sol=X0
(5) Initialiser le vecteur d'adaptation HCL à f(X0) ;
(6) Répéter
(7) V= Nb_iter mod Lfa
(8) Générer Voisinage de X0 // $\mathcal{N}$(X0)
(9) Sélectionner une solution X ϵ $\mathcal{N}$(X0)
(10) Si f(Meil_sol)≤fv or f(X)≤f(X0) alors
(11) Accepter la solution ; // X0=X
(12) Si f(X)≤f(Meil_sol) Alors Meil_sol=X
(13) Sinon Rejeter la solution ; // X0=X0
(14) Insérer la solution dans le vecteur d'adaptation ; // HCL[v]=X
(15) Incrémenter le nombre d'itérations ; // Nb_iter=Nb_iter+1
(16) Jusqu'à (Condition d'arrêt)
(17) Retourner (Meil_sol) ;

Figure 2 : Fonctionnement de l'algorithme LAHC

2.2.2. L'algorithme Génétique

Les algorithmes génétiques [7] sont, à l'heure actuelle, l'approche la plus utilisée parmi les méthodes évolutives. Ils sont des algorithmes d'optimisation s'appuyant sur des techniques dérivées de l'évolution naturelle : croisements, mutations et sélections basés sur la survie du meilleur. Ils travaillent sur une population de solutions (individus), plusieurs solutions en parallèle, et non pas sur une solution unique. Travailler sur plusieurs solutions en parallèle avec des opérateurs aléatoires réduits la possibilité de tomber sur un optimum local.
La population d'individus évolue en même temps comme dans l'évolution naturelle en biologie. Pour chacun des individus, on mesure sa faculté d'adaptation à l'environnement par une fonction d'adaptation (fitness). Ils utilisent les opérateurs de sélection et de reproduction. La sélection est le choix de pairs d'individus qui vont participer à la reproduction de la génération suivante. La reproduction, quant à elle, s'effectue à l'aide d'un opérateur de croisement et/ou un opérateur de mutation. Les meilleurs individus sont autorisés à se reproduire. L'algorithme génétique termine si la solution satisfait le critère d'arrêt, c'est-à-dire que la population convergera lorsque plus de 95% des individus de la population partagent la même valeur de la fonction d'adaptation "fitness". Le fonctionnement général de l'algorithme génétique est illustré en figure 3.

(18) Générer une population initiale P0.
(19) Evaluer P0.
(20) Tant Que (Non (condition d'arrêt)) faire :
(21) Répéter
(22) Sélectionner des Parents dans P0
(23) Appliquer Opérateur de Croisement sur les parents sélectionnés
(24) Appliquer Opérateur de Mutation sur les individus issus du croisement
(25) Placer ces individus mutants dans une nouvelle population P
(26) Jusque (nouvelle population P soit pleine)
(27) Evaluer P
(28) Remplacer les Anciens de P0 par leurs Descendants de P
(29) FinTantQue
(30) Retourner la meilleure solution

Figure 3 : Fonctionnement de l'algorithme génétique

3. FORMULATION DU PROBLEME

Face à un problème d'optimisation, l'étape la plus importante est sans doute l'étape de la modélisation (c'est-à-dire passer d'un problème réel à une formulation mathématique de ce problème). Le choix du modèle est souvent guidé par le type d'application. Le graphe de flot de données et de contrôle (CDFG : Control Data Flow graph), ou un graphe orienté acyclique tout simplement (DAG : Directed Acyclic Graph), une abstraction de la spécification, est utilisée dans notre étude comme entrée des algorithmes de partitionnement pour mapper les blocs/nœuds du CDFG soit en logiciel, soit en matériel [27]. L'extraction du graphe de la spécification n'est pas prise en considération dans notre étude. Le CDFG ou DAG G = (V,E), qui est utilisé pour décrire le comportement du système, consiste en un ensemble de fonctions (modules) représentées par les sommets $V = \{v_i \mid i = 1,2, \ldots n\}$, et un ensemble de dépendances et de contrôles de données qui sont représentées par les arcs $E = \{e_{ij} \mid e_{ij} = (v_i, v_j), v_i, v_j \in V$. Le problème du partitionnement de V dans deux ou plusieurs blocs interactifs peut être exprimé en $P = \{P_i \mid i = 1,2, \ldots N\}$, où $P_i = (V_i, E_i)$, $U\ V_i = V$ et $V_i \cap V_j = \emptyset$ si $i \neq j$, avec certaines contraintes. Supposons que n indique le nombre de nœuds (modules), x_i indique comment v_i (module i) est réalisé où $x_i = 1$ ($x_i = 0$) signifie que v_i est réalisé en matériel (en logiciel), m1 nœuds seront implémentés en matériel et m2 autres nœuds seront développés en logiciel ($n = m1 + m2$, $V_{Hw} = \{v_i \mid i = 1,2, \ldots m1\}$, $V_{Sw} = \{v_i \mid i = 1,2, \ldots m2\}$). Chaque noeud v_i du graphe est marqué par les attributs suivants : A_i^{Hw} est la surface occupée par le ième module réalisé en matériel ; T_i^{Hw} indique le temps d'exécution matériel, T_i^{Sw} représente le temps requis pour l'exécuter sur le processeur; DM_i^{Sw} et CM_i^{Sw} sont les capacités mémoire pour les données et instructions (codes) respectivement; S_i^{Sw} est le coût logiciel , S_i^{Hw} est le coût matériel toujours du ième module, les arcs sont également associés à une valeur de communication ($Ccom_{ij}$). Ccomij indique le coût de communication entre v_i et v_j. Une hypothèse importante est que le coût de communication logicielle-logicielle ou materielle-materielle est négligeable, c'est-à-dire il peut être considéré comme étant 0 à des fins pratiques.

A, C, G, Mb et T représentent respectivement les contraintes de surface, de communication, de coût global, de taille mémoire et la contrainte de temps d'exécution. Texe est le temps d'exécution qui est défini comme étant la somme du temps de traitement de tous les modules. Ccom est la somme des coûts de communication de tous les modules. Gc est la somme des coûts du logiciel et des coûts du matériel de tous les modules. TM est la somme de la taille de la mémoire de données et de codes de tous les modules logiciels. A^{Hw} est la somme de la surface de tous les modules matériels. T^{Sw} est le temps requis pour exécuter tous les modules logiciels sur le processeur ; T^{Hw} représente la somme du temps d'exécution des modules matériels.

$$Texe = \sum_{i=1}^{n} \left((x_i + T_i^{Hw}) + (1 - x_i) * T_i^{Sw} \right) \quad (1)$$

$$T^{Sw} = \sum_{i=1}^{n} \left((1 - x_i) * T_i^{Sw} \right) \quad (2)$$

$$T^{Hw} = \sum_{i=1}^{n} \left(x_i * T_i^{Hw} \right) \quad (3)$$

$$TM = \sum_{i=1}^{n} \left((1 - x_i) * (DM_i^{Sw} + CM_i^{Sw}) \right) \quad (4)$$

$$A^{Hw} = \sum_{i=1}^{n} (x_i * A_i^{Hw}) \quad (5)$$

$$Ccom = \sum_{i=1}^{n} \left(\sum_{j \in V_{Sw}} (x_i * Ccom_{ij}) + \sum_{j \in V_{Hw}} (1 - x_i) * Ccom_{ij} \right) \quad (6)$$

$$Gc = \sum_{i=1}^{n} \left((x_i * S_i^{Hw}) + (1 - x_i) * S_i^{Sw} \right) + Ccom \quad (7)$$

Avant de commencer le partitionnement, nous devons définir la fonction objective et les contraintes. Pour cela, chaque métrique devra être accompagnée d'un poids ki, qui est une valeur numérique indiquant l'importance de cette métrique aux yeux du concepteur. Autrement, le concepteur accordera la valeur ki positive ou nulle aux métriques selon leur importance relative.

$$Objfct = k1 * T^{Sw} + k2 * T^{Hw} + k3 * TM + k4 * Gc + k5 * A^{Hw} \quad (8)$$

La fonction objective, donnée par la formule (8), est une fonction paramétrable puisque nous donnons à l'utilisateur de notre outil de partitionnement M/L le choix de sélectionner l'ensemble des métriques et leurs poids. Si une métrique n'est pas sélectionnée, son poids aura une valeur zéro. Nous avons volontairement séparé les temps d'exécution logiciel et matériel pour donner plus de liberté au concepteur s'il veut avantager l'implémentation logicielle ou vice-versa. Afin d'optimiser la fonction coût, nous avons besoin des conditions suivantes :

A^{Hw} <= A ; TM <= Mb ; Ccom <= C ; Gc <= G ; Texe< = T

4. ALGORITHME HYBRIDE PROPOSE

Notre application est composée de n modules SystemC (M_1, M_2 ..., M_n). Dans notre modèle, une solution au partitionnement est exprimée sous la forme d'un ensemble de bits X = {x_1, x_2, ..., x_n}, où le ième élément représente le type d'implémentation assigné au Module M_i ; Si x_i = 1, cela implique que la réalisation de cet ieme module sera en matériel, dans le cas contraire la réalisation sera en logiciel. L'efficacité d'exploration, la qualité des résultats et la robustesse sont les principales préoccupations d'un algorithme de partitionnement. Nous proposons une stratégie combinant algorithmes complémentaires LAHC et AG, et l'appliquer à un problème de partitionnement M/L. L'idée de l'hybridation est simple : utiliser les points forts de chaque méthode pour avoir des solutions plus satisfaisantes, combiner la bonne exploitation des méthodes de recherche locale LAHC avec la puissance d'exploration d'espace de recherche des méthodes basées population (AG). Cette approche hybride tire profit de la complémentarité des algorithmes Génétiques (AGs) possédant une connaissance globale à travers les populations et explorant l'espace de recherche tandis que l'algorithme LAHC détenant une connaissance locale et exploitant le voisinage. La motivation derrière une telle hybridation des différents métaheuristiques est d'avoir des méthodes plus efficaces. Le but de l'algorithme combiné est de déterminer la nature d'implémentation (logiciel ou matériel) de chaque module. Avant d'opérer, nous avons les paramètres initiaux suivants (la longueur du vecteur d'adaptation Lfa, nombre maximal de générations, taille de la population, taux de croisement et taux de mutation). LAHC-AG a comme point de départ les entrées de LAHC. Nous générons au hasard une solution initiale X0, ensuite nous calculons fitX0, la fonction d'adaptation (Fitness) de X0, et

finalement, nous initialisons le vecteur d'adaptation HCL avec fitX0. Nous répétons le processus plusieurs fois pour ne prendre que les meilleures solutions qui vont former la population initiale de l'AG (Fig. 5). Après avoir construit la population initiale, nous passons à l'évaluation de la population en calculant les fonctions d'adaptation de chaque individu en utilisant la fonction objective. Une fois les coûts triés, nous passons aux opérateurs de l'AG, tels que la sélection, le croisement et la mutation. Nous avons choisi la méthode de la roulette pour la sélection des parents (Fig. 4).

```
Void selection_roulette(int p)
{   // Selects a chromosome from the population via roulette wheel selection
     int totfitness=0;
     int i;
    for ( i=p;i<Form1->Pop_size;i++)
    {     totfitness=totfitness+Form1->HCL[i].fitness;    }
    //generate a random number between 0 & total fitness count
    float random_num=random(totfitness);
    float Slice = (float)(random_num / totfitness);
    //go through the chromosones adding up the fitness so far
    float FitnessSoFar = 0.0f;
    int n;    i=p;n=0;
    while (i<=Form1->Pop_size && n==0)
   {    float x=Form1->HCL[i].fitness;
         FitnessSoFar  += x;
         float y=(FitnessSoFar/Form1->totfitness);
         //if the fitness so far > random number return the chromosome at this point
         if (y >= Slice)
         { Form1->individu_select=Form1->HCL[i].comb; n=1;         }
        i++;
    }
}
```

Figure 4 : Fonction de Sélection

Comme le croisement n'est généralement pas appliqué à toutes les paires d'individus sélectionnés pour la reproduction. Une probabilité est générée de manière aléatoire et dont la valeur doit être inférieure ou égale au taux de croisement. Si le crossover n'est pas appliqué, les descendants sont produits simplement en reproduisant les parents. Dans notre travail, nous avons choisi d'utiliser le croisement à deux points, parce que dans plusieurs articles, il est cité que le croisement à 2 points donne de bons résultats pour les grandes populations. La mutation est appliquée à chaque enfant. Elle modifie de façon aléatoire chaque gène avec une faible probabilité (moins que le taux de mutation). La méthode de mutation que nous avons utilisée dans cette étude est la mutation uniforme. Nous procédons ensuite à la réévaluation de la nouvelle population. Les nouveaux coûts sont triés, les coûts précédents et suivants sont comparés. Les meilleurs individus sont ajoutés pour former la nouvelle génération. Cette méthode permet de s'assurer que les individus performants seront conservés, alors que les individus peu adaptés seront progressivement éliminés de la population. La condition d'arrêt de tous les algorithmes est effective s'il y a convergence vers une meilleure solution ou une atteinte des itérations maximales. Sinon, nous devons répéter les principales opérations de l'AG. Le fonctionnement global de l'approche est incarné par l'organigramme de la figure 5.

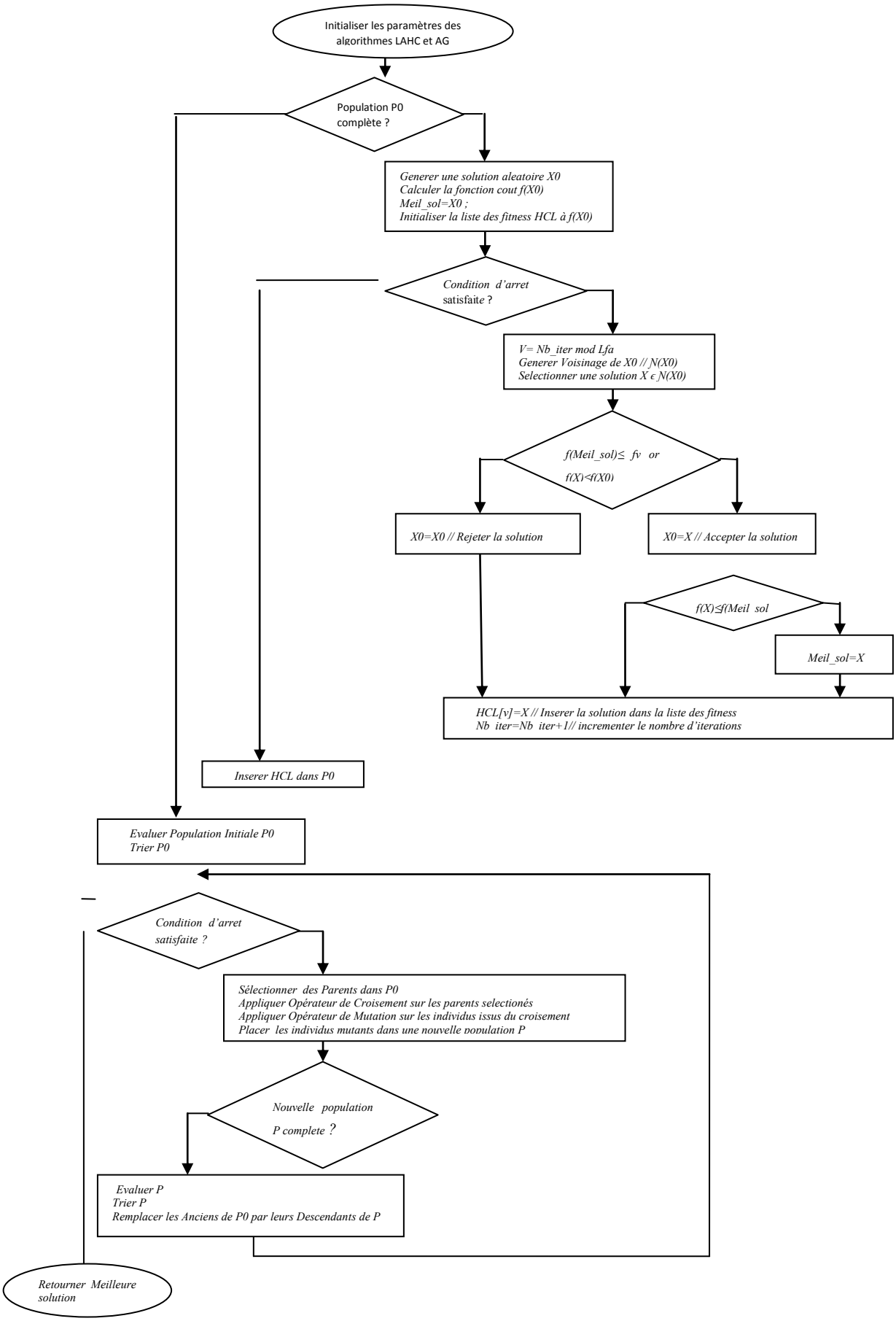

Figure 5 : Fonctionnement global de l'approche

5- RESULTATS ET DISCUSSIONS

A l'heure actuelle, il n'y a pas de jeu de tests largement partagé pour le problème du partitionnement M/L pour comparer l'exécution de différents algorithmes dans la littérature en raison des grandes différences dans les environnements Co-design. Aucune donnée originale n'a pu être obtenue, ainsi l'évaluation des algorithmes est basée en grande partie sur des résultats expérimentaux. Un DAG généré de manière aléatoire est une méthode commune, et les attributs sont assignés aux sommets et aux arcs. Pour tester leurs travaux, de nombreux auteurs Zhao[24], Li[25], Govil [28] , Yahyaoui [29] ,etc, ont utilisé plusieurs instances aléatoires avec différents noeuds et métriques, en fixant au départ certaines règles pour générer les paramètres du partitionnement, par exemple, le temps matériel d'un module T_i^{Hw} soit généré de manière aléatoire dans un intervalle bien déterminé ([1,100] ou [0,60]). Compte tenu de la rationalité des paramètres, ils ont proposé que le T_i^{Hw} créé au hasard, soit de l'ordre de 2 à 5 fois plus rapide que le T_i^{Sw}.

Pour nos expériences, nous avons utilisés l'outil TGFF pour créer plusieurs DAG possédant un nombre de nœuds spécifié, où chaque nœud est, associé à un module. Puis, nous avons défini un ensemble de métriques (T_i^{Sw} , T_i^{Hw} ,…) pour les différents modules (tableau 1). Ensuite, nous avons généré 10 DAG avec 10,20 30,40,50,60,70,80,90,100,120,150,180, et 200 nœuds respectivement. Enfin, nous précisons que nos algorithmes sont décrits dans le langage Borland c++ Builder 6.0 et exécutés sous le système d'exploitation Windows 7 sur Un PC – Lenovo de configuration (Intel(R) Core™ i5-3230M, 2.60 GHz, 2 cœur(s), 4 Go de RAM).

Tableau 1 : Paramètres de Partitionnement

Paramètres	Valeurs
Temps d'exécution matériel (T_i^{Hw})	[1..100]
Temps d'exécution logiciel (T_i^{Sw})	3* T_i^{Hw}
Surface ($\mathbf{A_i^{Hw}}$)	[1..100]
Capacité Mémoire de données (DM_i^{Sw})	[1..20]
Capacité Mémoire d'instructions (CM_i^{Sw})	100+ DM_i^{Sw}
Coût du logiciel(S_i^{Sw})	300+ DM_i^{Sw}
Coût du materiel(S_i^{Sw})	300+ $\mathbf{A_i^{Hw}}$
Coût de Communication ($Ccom_{ij}$)	[0..100]

Afin de démontrer l'efficacité de l'algorithme proposé, trois algorithmes heuristiques : AG, LAHC et hybride LAHC-AG ont été développés et confrontés pour la comparaison. Nous avons testé le bon fonctionnement de l'algorithme combiné LAHC-AG afin d'optimiser la fonction objective du problème de partitionnement. En raison de l'effet des paramètres de contrôle sur le résultat, nous avons utilisés les mêmes paramètres de contrôle dans le processus de comparaison pour garantir la cohérence. La configuration de tous les algorithmes a été effectuée en tenant compte des valeurs de paramètres suivantes : Lfa = 20 ; le nombre d'itérations = 20 pour LAHC ; les générations maximales = 500 ; la taille de la population = 50 ; le taux de croisement = 0.5 ; le taux de mutation = 0,15 pour AG. Tous les algorithmes se terminent s'ils convergent vers une meilleure solution ou qu'ils atteignent les itérations maximales. Nous avons exécuté les programmes LAHC, AG et LAHC-AG séparément avec les mêmes paramètres de contrôle. Pour les tester, nous avons pris les pondérations k1 = 1, k2 = 1, k3 = 0, k4 = 0 et k5 = 1 (voir fig. 5) ; c'est-à-dire :

$$Objfct = T^{Sw} + T^{Hw} + A^{Hw} \quad (9)$$

Il est à noter que nous avons subdivisé à dessein la métrique du temps d'exécution en temps d'exécution du matériel et temps d'exécution du logiciel dans le but d'introduire une certaine flexibilité chez le concepteur : s'il souhaite favoriser l'implémentation du logiciel ou du matériel en prenant en charge la pondération de chaque type de mise en œuvre de module. Pour un DAG = 10, les trois algorithmes donnent le même résultat (configuration= « 1111101001 » de fitness=811, c'est-à-dire trois modules seront réalisés en logiciel, et les sept autres modules seront implémentés en matériel, la seule différence réside dans le temps de recherche. Le tableau 2 affiche des détails sur le déroulement des 3 algorithmes.

Tableau 2 : Résultats du partitionnement pour 10 modules

Méthodes / Itérations	LAHC		AG		Algorithme Hybride	
	Solution	Fitness	Solution	Fitness	Solution	Fitness
Solution Initiale	1010101010	1248	1010101010	1248	1010101010	1248
Iteration1	1110101000	1073	1101101000	930	1111101001	811
Iteration2	1111101000	908	1111101011	814	1111101001	811
Iteration3	1111101001	811	1111101011	814	1111101001	811
Iteration4	1111101001	811	1111101001	811	1111101001	811
Iteration5	1111101001	811	1111101001	811	1111101001	811
Iteration6	1111101001	811	1111101001	811	1111101001	811
Iteration7	1111101001	811	1111101001	811	1111101001	811

La colonne de LAHC montre l'évolution des solutions insérées dans le vecteur d'adaptation. Par contre, pour les Algorithmes AG et Hybride, leurs colonnes respectives affichent la meilleure solution pour chaque nouvelle génération. Ainsi, les trois algorithmes sont initialisés avec la même solution initiale X0= « 1010101010 » générée aléatoirement (fitness =1248), et nous sommes limités seulement à la 7eme itération car les 3 algorithmes ont atteint la solution optimale citée précédemment. Nous signalons que cette solution est le résultat optimal (parmi les 1024 configurations possibles) obtenu par la méthode exacte que nous avons aussi développé pour comparer ces résultats. LAHC, après deux itérations, trouve la solution. Trois générations ont suffi pour AG pour qu'il trouve la même solution. Tandis que pour l'algorithme hybride, la solution est déjà trouvée par LAHC, donc, elle fait partie de la population initiale P0 de AG, c'est pourquoi, la convergence est directe vers cette solution.

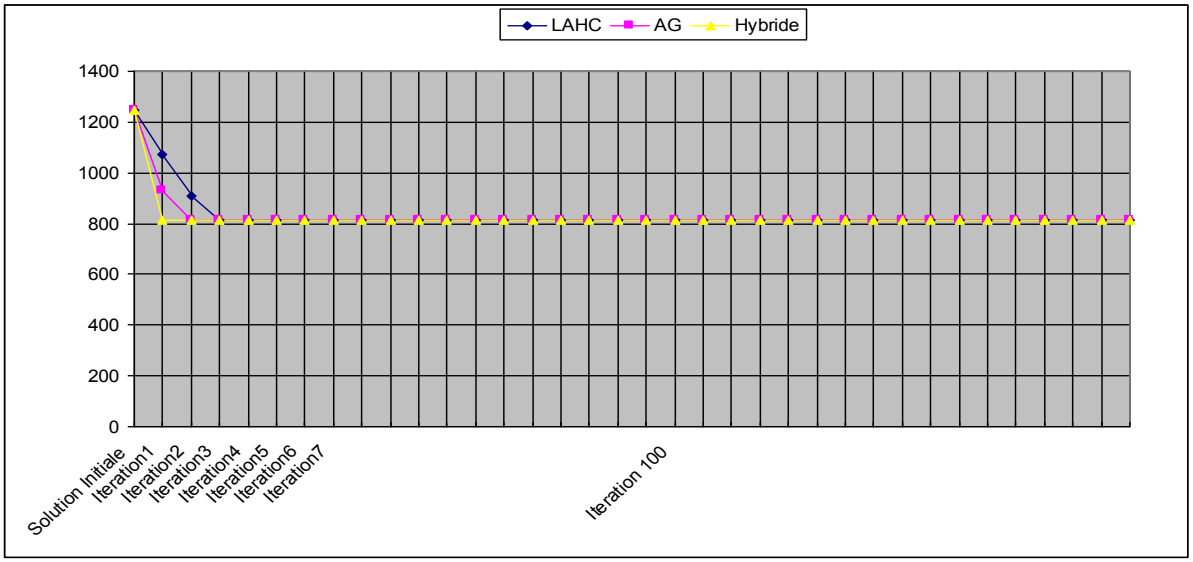

Figure 6 : Convergence des 3 algorithmes vers la même configuration

La figure 6 montre clairement la convergence rapide de l'algorithme hybride pour le cas de 10 modules. Nous généralisons l'expérimentation à des nombres de modules quelconques inclus dans [10,200]. Le tableau 3 synthétise les résultats générés par les trois algorithmes. D'après le tableau 3 et la figure 7, nous pouvons déduire que la fiabilité de notre algorithme combiné LAHC-GA est supérieure aux algorithmes GA et LAHC pris séparément.

Tableau 3 : Résultats de partitionnement M/L

Modules	LAHC	AG	LAHC-AG
10	**811**	**811**	**811**
20	**1458**	**1458**	**1458**
30	**2159**	**2159**	**2159**
40	**3081**	3083	**3081**

50	**3940**	3946	**3940**
60	**4995**	5041	4995
70	5852	5860	**5813**
80	6411	6542	**6394**
90	7920	7813	**7658**
100	9420	9152	**8926**
120	11092	11209	**10920**
150	14050	14171	**13987**
180	18490	17435	**17068**
200	20836	20263	**19418**

Lorsque la taille de la population est supérieure à 70, il est possible de noter que le LAHC-GA était légèrement supérieur à l'AG et LAHC séparément(les meilleures solutions sont en gras dans le tableau 3). C'est un résultat intéressant, puisque LAHC est un nouveau méta heuristique et c'est la première fois qu'elle est combinée avec l'algorithme GA, du moins à notre connaissance. L'algorithme combiné a fourni une solution améliorée et presque optimale.

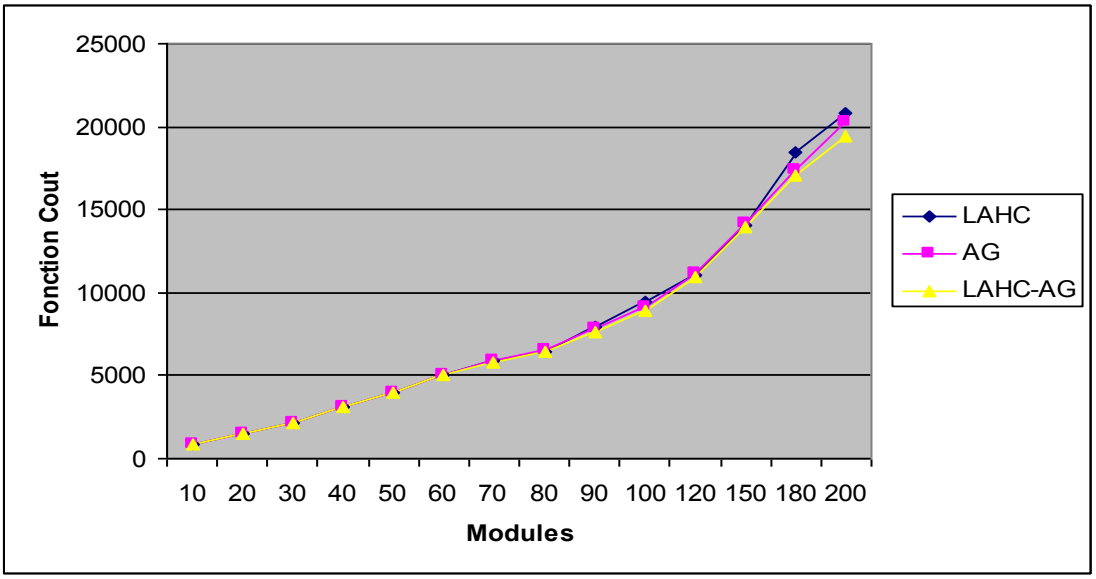

Figure 7 : Comparaison des 3 algorithmes

6. CONCLUSION

De nombreuses méthodes et algorithmes ont abordé le problème du partitionnement M/L sans fournir de solutions satisfaisantes ; ceci est du à plusieurs facteurs : l'évolution de la conception, les caractéristiques des composants et la complexité des applications et des architectures. Dans ce contexte, nous avons proposé un algorithme de partitionnement M/L combiné basé sur LAHC-AG pour déterminer le partitionnement optimal. L'algorithme LAHC-AG combine la capacité de recherche globale de l'algorithme génétique et la capacité de recherche partielle de l'algorithme LAHC. Par rapport au LAHC et à l'AG pris séparément, l'algorithme proposé fournit de meilleurs résultats. Au mieux de nos connaissances, la combinaison LAHC-AG est nouvelle et relance l'optimisation du problème de partitionnement M/L, pierre angulaire du développement embarqué.

REFERENCES

[1] S. Ha, J. Teich, C. Haubelt, M. Glaß, T. Mitra, R. Dömer, P. Eles, A. Shrivastava, A. Gerstlauer, and S. S. Bhattachary, 2017, Introduction to Hardware/Software Co design, Springer Science+Business Media, S. Ha, J. Teich (eds.), Handbook of Hardware/Software Codesign.

[2] G. Bosman, 2005, A Survey of Co-Design Ideas and Méthodologies, Master's Thesis. Vriji University, Amsterdam.

[3] I. Mhadhbi, S. Ben Othman, and S. Ben Saoud, 2016, An Efficient Technique for Hardware/Software Partitioning Process in Co design, Hindawi Publishing Corporation Scientific Programming, Article ID 682765.

[4] S. H. Baranga and A. Szekeres, 2009, Software/Hardware Partitioner, Roedunet International Conference, pp. 252-257 (ISBN 978-1-4244-7335-9)

[5] W. Jigang, T. Srikanthan, and G. Chen, 2010, Algorithmic aspects of hardware/software partitioning: 1D search algorithms, Institute of Electrical and Electronics Engineers, Transactions. on Computers, vol. 59, no. 4, pp. 532–544.

[6] E. K. Burke and Y. Bykov, 2012, The late acceptance hill-climbing heuristic, Technical Report CSM-192, University of Stirling.

[7] M. Riabi, Y. Manai,and J. Haggege, 2015, Hardware/Software partitioning approach for embedded system design based on genetic algorithm, International Journal of Scientific Research & Engineering Technology, vol. 3, issue 3, pp. 20-25.

[8] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, 1997, Design of embedded systems: Formal models, validation, and synthesis, Proceeding of the institute of Electrical and Electronics Engineers, vol. 85, n3, pp. 366-390.

[9] B. Mei, P. Schaumont, and S. Vernalde, 2000, A hardware-software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems, In Proceedings of ProRISC, Citeseer, pp. 405–411.

[10] J. Wu, T. Srikanthan, and C. Yan, 2008, Algorithmic aspects for power-efficient hardware/software partitioning, Math Comput Simul 79(4), pp. 1204–1215

[11] C. Blum, J. Puchinger, G. Raidl, and A. Roli, 2011, Hybrid metaheuristics in combinatorial optimization: A survey, Applied Soft Computing, vol: 11, pp. 4135–4151.

[12] W. Jigang, B. Chang, and T. Srikanthan, 2009, A hybrid branch and- bound strategy for hardware/software partitioning, In Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science (ICIS '09), pp. 641–644.

[13] S. Banerjee, E. Bozorgzadeh, and N. D. Dutt, 2006, Integration physical constraints in hw-sw partitioning for architectures with partial dynamic reconfiguration, IEEE Transactions on Very Large Scale Integration (VLSI)Systems, vol. 14, no. 11, pp. 1189 -1202.

[14] J. Wu and T. Srikanthan, 2006, Low-complex dynamic programming algorithm for hardware/software partitioning, Information Processing Letters, vol. 98, no. 2, pp. 41–46.

[15] S. Dimassi, M. Jemai, B. Ouni and A. Mtibaa, 2015, Meta-heuristics: for the problem of partitioning Hardware/Software, Conference Paper, DOI : 10.1109/WSWAN.2015.7210339

[16] W. Shi, J. Wu, S.K. Lam, and T. Srikanthan, 2016, Algorithms for bi-objective multiple-choice hardware/software partitioning, Computers and Electrical Engineering, pp. 1 -16.

[17] I. Boussaïd, J. Lepagnot, and P. Siarry, 2013, A Survey on Optimization Metaheuristics, Information Sciences, Vol.237, pp.82-117.

[18] E.-G.Talbi, 2009, Metaheuristics : From Design to Implementation, John Wiley & Sons, Inc., Hoboken, New Jersey.

[19] M. Jemai , S. Dimassi, B. Ouni, and A. Mtibaa, 2017, A Metaheuristic based on the tabu search for hardware-software partitioning, Turkish Journal of Electrical Engineering & Computer Sciences, http://journals.tubitak.gov.tr/elektrik/

[20] R. Ernst, J. Henkel, and T. Benner, 2002, Hardware-software cosynthesis for microcontrollers, IEEE Design and Test of Computers, vol. 10, no. 4, pp. 425–449.

[21] J. Dai, H. Han, Q. Hu, and M. Liu, 2016, Discrete particle swarm optimization approach for cost sensitive attribute reduction, Knowldge Based Sysem, vol.102. pp.116-126

[22] G. Raidl, 2015, Decomposition based hybrid Metaheuristics, European Journal of Operational Research, vol. 244. pp. 66–76

[23] I. Dumitrescu and T. Stützle, 2003, Combinations of local search and exact algorithms, In EvoWorkshops 2003, pages 211223.

[24] X. Zhao, H. Zhang, Y. Jiang, S. Song, X. Jiao, and M. Gu, 2013, An Effective Heuristic-Based Approach for Partitioning, Journal of Applied Mathematics, Volume 2013, Article ID 138037, 8 pages

[25] W. Li, L. Li. J. Sun, Z. Lv and F. Guan, 2014, Hardware/software partitioning of combination of clustering algorithm and genetic algorithm, International journal of control and automation, vol. 7, no. 1, pp. 347-356, http://dx.doi.org/10.14257/ijca.2014.7.1.31

[26] M. Jemai, S. Dimassi, B. Ouni, and A. Mtibaa, 2015, Combined Partitioning Hardware-Software Algorithms, International Journal of Computer Applications (0975 – 8887) Volume 119 – No.4.

[27] Y. Peng, M. Lin, and J. Yang, 2005, Hardware-software partitioning research based on resource constraint, Journal of Circuits and Systems, vol. 10, no. 3, pp. 80–84, Chinese.

[28] N. Govil, and S. R. Chowdhury, 2015, A High Speed Metaheuristic Algorithmic Approach to Hardware Software Partitioning for Low-cost SoCs, International Symposium on Rapid System Prototyping (RSP).

[29] K. Yahyaoui, 2017, Partitioning And Scheduling Resolution Problems By Bees Mating Strategy In Dres' Systems, International Journal of Computing, 16(2), pp. 97-105.