République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Faculté des Sciences Exactes et Sciences de la Nature et de la Vie
Département de Mathématiques et Informatique

N° d'ordre : .....
N° série : .....

Mémoire

En vue de l'obtention du diplôme de Magister en informatique

Option: Intelligence Artificielle et Imagerie

Présenté par

Oubadi Soumia

Thème

Optimisation de l'ordonnancement/allocation dans les systèmes embarqués distribués temps réel.

Devant le jury composé de :

Dr. NINI BRAHIM	***Maître de Conférences classe A, Université d'Oum El-bouaghi.***	***Président.***
Dr. MOUKHATI FARID	***Maître de Conférences classe A, Université d'Oum El-bouaghi.***	***Examinateur.***
Dr. BOUZENADA MOURAD	***Maître de Conférences classe A, Université de Constantine2.***	***Examinateur.***
Dr. BOUTEKKOUK FATEH	***Maître de Conférences classe A, Université d'Oum El-bouaghi.***	***Rapporteur.***

- Année Universitaire 2013-2014 -

Dédicace

A toute ma famille

A toutes mes amies

A tous mes collègues

Remerciements

J'adresse ma gratitude sincère à **Allah**, à chaque fois que je fais face à une difficulté, je prie **Allah** de m'aider et il est toujours là pour me protéger et me sauver.

Je voudrais exprimer mes vifs remerciements à mon encadreur ***Dr.Boutekkouk Fateh*** pour son aide, sa patience et son encouragement.

Sans oublier ***Mr.Laboudi*** *et* **Mr.*Marir*** pour m'avoir aidé et expliqué plusieurs sujets que je n'arrive pas à comprendre toute seule.

Je remercie tous les membres du jury qui ont accepté d'évaluer mon travail ainsi que mes collègues du travail et toutes les personnes qui m'ont aidé.

Un grand merci à mes parents pour tous ce qu'ils ont fait pour moi, mon frère et mes sueurs, mes cousines et spécialement à ***Sihem*** qui m'a beaucoup aidé tout au long de la durée de préparation de ce travail,

à mes amies et à toute ma famille.

Je tiens à remercier toute personne ayant contribué de près ou de loin à la réalisation de ce travail.

ملخص

إنجاز الأنظمة المدمجة التي تعمل بمبدأ الوقت الحقيقي، و التي أصبحت حاليا تحتل اهتماما واسعا و تتواجد في عدة مجالات بما في ذلك: السيارات، الطائرات، الأجهزة الآلية... إلخ، يتطلب منا إيجاد الحلول لمشاكل التخصيص و الترتيب في وقت حقيقي للبنية ذات المعالجات المتعددة و الغير متجانسة و الموزعة، وذلك مع مراعاة عوامل الأسبقية، الوقت، الموارد و ما إلى ذلك. في هذه الدراسة، قمنا باستعمال آلية تخصيص و ترتيب في وقت حقيقي للمهام و الرسائل معتمدين على معيار الوقت و الاولويات بين المهام. هذه الآلية يجب أن تضمن تنفيذ جميع المهام الدورية و الغير دورية للنظام على البنية ذات المعالجات المتعددة، مع احترام كل العلاقات بين المهام و كذا الإلتزامات الزمنية.

التصميم الفعال لهذه الأنظمة يتطلب طرق فعالة و دقيقة من أجل تحسين أدائها. ولهذا، قمنا بوضع طريقة جديدة مبنية أساسا على المزج بين الخوارزميات الجينية و مبادئ ميكانيك الكم. هذه الطريقة الجديدة والتي تحمل اسم الخوارزميات الجينية الكمية، ترتكز أساسا على التمثيل الكمي من أجل ترميز فضاء البحث.الخاصية الأساسية لهذه الطريقة الجديدةهي استعمال مجموعة صغيرة من السكان، و معالجة عدد صغير من التكرارات. الحصول على أفضل ترتيب للمهام يتم من خلال العمليات الكمية، التي تعتمد أساسا على مبادئ الإعلام الآلي الكمي مثل تراكب الحالات و التداخل. و قد اظهرت النتائج التجريبية تحسنا بالمقارنة مع الخوارزميات الجينية الكلاسيكية.

الكلمات المفتاحية: أنظمة مدمجة، الترتيب في الوقت الحقيقي، الخوارزميات الجينية، الإعلام الآلي الكمي.

Résumé

La réalisation des systèmes embarqués temps réel qui sont devenus maintenant de grande importance et que l'on trouve dans plusieurs domaines, à savoir l'automobile, l'avionique, la robotique, etc. nous oblige à résoudre les problèmes d'allocation et d'ordonnancement temps réel pour des architectures multiprocesseurs hétérogènes et distribuées, en respectant des contraintes multiples de précédence, de temps, de ressources, etc. Dans ce travail, un mécanisme d'allocation et d'ordonnancement préemptif temps réel de tâches et de messages basé sur le critère du temps et de priorités est employé. Ce mécanisme doit garantir l'exécution de toutes les tâches périodiques et apériodiques du système sur l'architecture multiprocesseurs en respectant les relations de dépendances ainsi que les contraintes temporelles.

La conception efficace de ces systèmes nécessite des méthodes puissantes et précises pour optimiser leurs performances. Pour cela, nous avons développé une nouvelle approche en faisant une hybridation entre les algorithmes génétiques et les principes de la mécanique quantique. Cette approche dite algorithme génétique inspiré-quantique, repose sur une représentation quantique pour le codage de l'espace de recherche. La particularité de cette approche est l'utilisation d'une population de taille nettement inférieure et la manipulation d'un nombre d'itérations raisonnable. L'obtention d'un meilleur ordonnancement de tâches est faite grâce aux opérateurs quantiques, qui sont basées sur des principes de l'informatique quantique comme la superposition d'états, et l'interférence. Les résultats expérimentaux ont montré une amélioration par rapport aux algorithmes génétiques classiques.

Mots-clés: Systèmes Embarqués,Ordonnancement Temps Réel, Algorithme Génétique, Informatique Quantique.

Abstract

The realization of real time embedded systems which have become with great importance and are found in several areas, including automotive, avionics , robotics, etc. requires to solve allocation and real-time scheduling problems for heterogeneous and distributed multiprocessor architectures, respecting multiple criteria such as: precedence constraints, time, resources, etc. In this work, an allocation and preemptive scheduling mechanism of real-time tasks and messages based on the criterion of time and priorities is employed. This mechanism must ensure the execution of both periodic and aperiodic tasks on multiprocessor architecture system respecting the dependency relationships and time constraints.

The effective design of these systems demands powerful and accurate methods to optimize their performance. For this reason, we have developed a new approach by hybridization of genetic algorithms and principles of quantum mechanics. This approach called quantum-inspired genetic algorithm, relies on a quantum representation for encoding the search space. The particularity of this approach is the use of a population of definitely smaller size and the use of a reasonable number of iterations. Obtaining a better tasks scheduling is done through quantum operators, which are based on principles of quantum computing such as states superposition, and interference. The experimental results have shown a remarkable improvement over the conventional genetic algorithms.

Keywords: Embedded Systems, Real Time Scheduling, Genetic Algorithm, Quantum Computing.

Table des matières

CHAPITRE II: ORDONNANCEMENT TEMPS REEL

CHAPITRE III: LES ALGORITHMES GENETIQUES INSPIRES-QUANTIQUES

Liste des figures

Liste des tableaux

Liste des formules

Liste des acronymes

Acronyme	Description
ACET	Average Case Execution Time
AG	Algorithme Génétique
ASIC	Application-Specific Integrated Circuit
AWF	Almost Worst-Fit
BCGA	Bi-Chromosomal Genetic Algorithm
BF	Best-Fit
BS	Background Scheduling
DAG	Directed Acyclic Graph
DM	Deadline Monotonic
DRTES	Distributed Real-Time Embedded Systems
DSP	Digital Signal Processor
EA	Evolutionary Algorithm
EDF	Earliest Deadline First
ES	Embedded System.
FCFS	First Come First Serve
FF	First-Fit
GAO	Genetic Algorithm Optimization
LLF	Least Laxity First
NF	Next-Fit
PFair	Proportionate Fair
PS	Polling Server
QC	Quantum Computing
QIGA	Quantum-Inspired Genetic Algorithm
QIGAO	Quantum-Inspired Genetic Algorithm Optimization
RM	Rate Monotonic
RTOS	Real Time Operating Système.
RTS	Real Time System.
RWS	Roulette Wheel Selection
SJF	Shortest Job First
SoC	System on Chip
SS	Sporadic Server
WCET	Worst Case Execution Time
WF	Worst-Fit

Introduction générale

1. Contexte

Les systèmes embarqués (SEs) (Embedded Systems) sont de plus en plus présents dans notre quotidien. Cette technologie peut être trouvée aussi bien dans les systèmes multimédia que dans les applications de réseau et de télécommunication. L'utilisation de SEs est en train de suivre une courbe exponentielle et on attend même que leurs ventes dépassent en revenus celles des processeurs de PC à usage général. Selon le Dr. Gordon E. Moore, cofondateur d'Intel, la complexité des circuits intégrés suit une loi empirique (la loi de Moore) dans laquelle le nombre et la puissance des transistors doublaient presque tous les deux ans. Ces progrès technologiques ont permis d'intégrer sur une même puce plus de fonctionnalités ce qui a conduit à la définition des systèmes sur puce (SOC : System on Chip).

Un SE est un système à application spécifique qui contient du matériel et du logiciel adapté à une tâche particulière et qui fait partie d'un système plus large qui n'est pas forcement informatique (ex. électronique, mécanique, etc.). Un SE interagit avec le monde extérieur via les capteurs/actionneurs et est soumis à des contraintes spatiales, temporelles et énergétiques très strictes. La conception de tels systèmes complexes pose plusieurs défis et nécessite une collaboration entre plusieurs équipes appartenant à des disciplines différentes (ex. logiciel, matériel, système, intégrateur, etc.). Le domaine des systèmes embarqués est donc pluridisciplinaire.

En effet, les SEs sont de nature hétérogène du fait de l'hétérogénéité des applications qu'ils implémentent. Ils combinent typiquement des parties matérielles (des processeurs à usage général, des processeurs à usage spécifique, des processeurs de traitement du signal) et d'autres parties logicielles.

Une implantation logicielle d'une partie de l'application a l'avantage de lui procurer une flexibilité liée à la possibilité de reprogrammation, contrairement à une implantation matérielle figée mais qui a l'avantage de satisfaire plus facilement les contraintes de performances.

Les SEs permettent alors une réduction importante des coûts de production et une augmentation considérable de la fiabilité des systèmes développés. La réalisation des différents composants d'un SE peut être faite par plusieurs équipes de développement, où

chaque équipe utilise son propre environnement de conception pour décrire la fonctionnalité du système étudié. Ce nouveau type de circuit intégré nécessite de nouveaux outils et de nouvelles méthodes de conception, de réalisation et de vérification. La conception des SEs est soumise à un ensemble de contraintes fortes pour assurer leurs bon fonctionnements. Comme ces systèmes regroupent des descriptions logicielles et matérielles, il est important de prendre en considération l'étude conjointe de ces deux concepts et leurs différentes interactions au plus haut niveau d'abstraction pour faciliter et réduire le cycle de développement. Il est alors nécessaire de maîtriser la conception matériel/logiciel des systèmes sur puce tout en respectant les contraintes de mise sur le marché et les critères de qualité.

Un système embarqué distribué (SED) est la version décentralisé d'un système embarqué centralisé avec plusieurs contrôleurs (nœuds) indépendants qui sont reliés entre eux par un bus partagé, une hiérarchie de bus ou un réseau de communication généralement avec trois couches : la couche physique, la couche liaison et la couche application. Les automobiles, les avions, les sous-marins, les réseaux sur puce (NOC : Network On Chip) sont des bons exemples sur des systèmes intégrants des SED.

Un SED est dit temps réel s'il est capable de satisfaire ses contraintes temporelles. En fait, nous pouvons classer les SED selon le type de contrainte de temps en trois classes : les SED durs, souples et fermes.

De manière similaire, les SED peuvent être classés selon le type des processus et/ou les communications en trois familles : dirigés par le temps, dirigés par les événements ou mixtes.

Tout au long de ce mémoire, nous nous sommes intéressés par les SED temps réel mixtes avec des contraintes mixtes (dures et souples).

2. Problématique

Le développement des systèmes embarqués temps réel (SETR) pose des défis importants pour les développeurs. Ceux sont des systèmes réactifs dont la validité est conditionnée non seulement par la correction des résultats mais surtout par les dates auxquelles ceux-ci sont délivrés. Cela implique que le temps de réponse est aussi important que la production des résultats corrects. La validité de résultat de ces systèmes est assurée par le respect des contraintes temporelles lors de l'exécution de l'application.

Les systèmes embarqués temps réel se composent d'un ensemble de tâches exécutées en concurrence pour utiliser le (ou les) processeur et certains ressources partagées, le plus souvent, les exigences temporelles sont reportées sur les échéances d'exécution des tâches.

Ces systèmes sont de plus en plus souvent réalisés avec des architectures multiprocesseur distribuées hétérogènes, c'est-à-dire comportant plusieurs processeurs différents. Ceci est dû au besoin d'augmenter la puissance de calcul.

Le problème d'ordonnancement des tâches mixtes avec des contraintes mixtes dans les systèmes embarqués temps réel réalisés avec des architectures multiprocesseur distribuées hétérogènes, est considéré comme l'un des problèmes les plus importants, et il est qualifié de problème NP-difficile. L'allocation des tâches qui constituent les applications de ces systèmes, et qui sont représentées par un graphe de dépendances (DAG), constitue un défi supplémentaire pour les développeurs, et rend le problème plus complexe. En effet, chacun des processeurs de l'architecture est séquentiel, donc il s'agit de déterminer dans quel ordre et à quels instants on va devoir exécuté les fonctionnalités du système temps réel dans le but d'optimiser la longueur de l'ordonnancement globale du système. Donc, le problème d'ordonnancement temps réel devient de plus en plus complexe.

Pour optimiser les performances de ces systèmes, plusieurs méthodes de résolution ont été proposées et qui fournissent des résultats satisfaisants dans la pratique. Elles peuvent se partager en deux grandes classes, les méthodes complètes dites exactes et les méthodes incomplètes dites approchées. Les méthodes complètes permettent d'effectuer une recherche exhaustive exploitant complètement l'espace de recherche. Malheureusement, ce type de méthode est impraticable dans le cas de grandes instances du fait de la complexité exponentielle de ces méthodes, c'est le cas notamment des méthodes DP, DPLL et de variantes de méthodes plus génériques comme Branch & Bound. Cependant, les méthodes incomplètes n'explorent que certaines zones de l'espace de recherche et s'avèrent particulièrement efficaces pour des instances de grandes tailles bien qu'elles ne donnent pas des solutions exactes.

Donc, afin d'optimiser les performances des systèmes embarqués temps réel, il est souvent nécessaire de trouver des modes de résolution qui fournissent une solution de bonne qualité dans un laps de temps raisonnable, c'est ce que font les heuristiques.

3. Contribution

Le travail présenté dans ce mémoire se focalise sur l'optimisation des performances des systèmes embarqués temps réel, réalisés avec des architectures multiprocesseurs distribués hétérogènes en s'appuyant sur les algorithmes génétiques.

D'après la synthèse faite, nous pouvons constater que la plupart des travaux portant sur l'optimisation des performances par algorithmes génétiques ne prennent en considération qu'un seul type de tâches (périodique ou apériodique) avec un seul type de contraintes (souples ou dures) or les systèmes embarqués temps réel actuels sont mixtes dans le sens où ils peuvent inclure des tâches périodiques et apériodique avec des contraintes temporelles qui peuvent être dures ou souples. Pour remédier à ce problème, nous avons proposé un modèle mixte jugé capable de capturer toutes les spécificités des systèmes temps réel récents.

De l'autre côté, les algorithmes génétiques inspirés-quantiques ont montré leur efficacité dans de nombreux problèmes d'optimisation dépassant celle des algorithmes génétiques conventionnels.

L'avantage des algorithmes génétiques inspirés-quantiques par rapport à celui des algorithmes génétiques classiques réside dans le principe du parallélisme causé par la représentation du qubit et la superposition d'état. Cela signifie que le résultat effectué par ces algorithmes est plus rapide que celui effectué par les algorithmes génétiques classiques.

Nous avons décidé d'adopter ces algorithmes pour la simple raison que l'analyse des travaux effectués sur ce domaine, a montré qu'aucun chercheur n'a abordé l'optimisation des performances dans les systèmes embarqués distribués temps réel mixtes par des algorithmes génétiques quantiques. Ce manque nous a poussés à s'engager dans cette voie.

Nous pouvons faire le résumé de nos contributions principales sur le sujet proposé comme suit :

- Notre travail consiste à faire, dans un premier temps l'allocation de l'ensemble de tâches constituant les fonctionnalités du système embarqué temps réel modélisées par un ensemble de graphes de tâches périodiques et apériodiques avec précédences et des contraintes temporelles mixtes (dures et souples) , sur une architecture hétérogène composée d'un ensemble de processeurs hétérogènes, un ensemble de bus reliés par des ponts, et un ensemble de liens bipoints.

- Dans un deuxième temps, faire l'ordonnancement global de l'ensemble de tâches temps réel dépendantes de type périodique ou apériodiques, de contrainte souple ou dure et l'ensemble de messages, sur l'architecture cible en utilisant des politiques d'ordonnancement temps réel (RM, DM, BS).
- Optimisation des performances de système embarqué temps réel à savoir le temps de réponse moyen du système et le nombre de tâches respectant leurs échéances, en utilisant la méta-heuristique : les algorithmes génétiques classiques.
- A la fin, faire une hybridation entre l'informatique quantique et les algorithmes génétiques classiques, le résultat de cette hybridation nous donne les algorithmes génétiques inspirés-quantiques, qui sont utilisés enfin pour l'optimisation des performances de système embarqué distribué temps réel mixte.

4. Organisation du mémoire

Ce mémoire est scindé en quatre chapitres :

- Le premier chapitre est consacré à une introduction aux systèmes embarqués distribués temps réel, nous y introduirons l'ensemble des concepts nécessaires à la compréhension des chapitres suivants.
- Le deuxième chapitre présente tous ce qui concerne l'ordonnancement des tâches temps réel sur une architecture monoprocesseurs puis multiprocesseurs ainsi que les principales classes des algorithmes d'ordonnancement.
- Le troisième chapitre met la lumière sur les algorithmes génétiques classiques ainsi que leurs principales étapes, puis donner un état de l'art sur les travaux concernant l'application de ces derniers aux problèmes d'ordonnancement classiques et temps réel, en fin le concept des algorithmes génétiques inspirés quantiques est présenté.
- La modélisation, la conception, les tests et les résultats de simulation de notre approche font l'objet du quatrième chapitre.
- Enfin, nous terminerons par une conclusion générale tout en dégageant les perspectives envisageables qui font l'objet des futurs travaux.

Chapitre 1 Systèmes embarqués distribués temps réel

Dans ce chapitre :

1. Introduction
2. Systèmes temps réel (STR)
   - Définition des systèmes temps réel.
   - Modèle de base d'un système temps réel.
   - Caractéristiques des systèmes temps réel.
   - Domaine d'application des systèmes temps réel.
   - Classification des systèmes temps réel.
3. Systèmes embarqués (SE)
   - Définition des systèmes embarqués.
   - Brève histoire sur les systèmes embarqués.
   - Domaine d'application des systèmes embarqués.
   - Caractéristiques des systèmes embarqués.
   - Classification des systèmes embarqués.
4. Système embarqués distribués temps réel (SEDTR)
   - Définition des systèmes embarqués temps réel.
   - Flot de conception de systèmes embarqués réel temps.
   - Contraintes des systèmes embarqués temps réel.
   - Spécification des systèmes embarqués temps réel.
   - Les architectures des systèmes embarqués distribués temps réel.
5. Conclusion

Résumé

Ce chapitre est une introduction aux systèmes embarqués distribués temps réel, nous y introduirons l'ensemble des notions nécessaires à la compréhension des chapitres suivants. Ce chapitre est composé d'une première partie décrivant les systèmes temps réel, puis une deuxième partie décrivant les systèmes embarqués et en fin une partie pour décrire les systèmes embarqués distribués temps réel.

1. Introduction

Nous vivons aujourd'hui dans un monde dans lequel le logiciel joue un rôle critique. La plupart de ces logiciels ne s'exécutent pas dans plusieurs systèmes et ordinateurs mais par contre, ils s'exécutent à l'intérieure des infrastructures et dans des appareils que nous utilisons tous les jours à savoir; les produits électroniques, le transport, l'automobile la télécommunication, …etc. Ces logiciels, qui envahissent notre vie quotidienne, sont conçus principalement pour faciliter la vie humaine en évitant les travaux manuels, complexes et fastidieux.

La majorité de ces logiciels ont conscience de l'écoulement du temps, et prennent des décisions en fonction de celui-ci, On parle alors de « ***système temps réel ».*** Dans ce dernier, le respect de toutes les contraintes temporelles spécifiées avant toute exécution réelle du système est aussi important que l'exactitude du résultat.

Lorsque le système temps réel est intégré à un environnement et soumis aux mêmes contraintes physiques externes à cet environnement, il est dit « ***système embarqué temps réel »***.

2. Systèmes temps réel (STR)

2.1 Définition des systèmes temps réel

Avant d'entamer la définition de système temps réel, il faut d'abord expliquer ce que signifie le terme *temps réel*. Ce terme « qui signifie sans attente ou immédiat » est employé pour désigner le temps physique. Il est largement utilisé dans de nombreux contextes techniques et conventionnels. Il est aussi utilisé pour décrire un certain nombre de caractéristiques de l'ordinateur. Par exemple, les systèmes d'exploitation en temps réel sont des systèmes qui répondent à l'entrée immédiatement. Ils sont utilisés pour des tâches telles que la navigation, dans lequel l'ordinateur doit réagir à un flux constant de nouvelles informations sans interruption. La plupart des systèmes d'exploitation d'usage général ne sont pas en temps réel, car ils peuvent prendre quelques secondes, voire quelques minutes, avant de réagir.

Temps réel, peut également se référer à des événements simulés par ordinateur à la même vitesse qu'ils se produisent dans la vie réelle. Dans l'animation graphique, par exemple, un programme en temps réel va afficher des objets en mouvement à travers l'écran à la même vitesse que dans la réalité **[102]**.

2.1.1 Les systèmes réactifs

Un système réactif est un système qui répond (réagit) constamment à des événements (stimuli) provenant de son environnement en produisant les actions de sortie adéquates sur celui-ci. Donc son comportement dépend des évènements et de l'ordre d'arrivée de ces derniers **[1]**.

2.1.2 Les systèmes temps réel

Dans certains systèmes réactifs, il est nécessaire que les réactions du système aux événements doivent être effectuées dans un délai inférieur à un temps limite. Les applications soumises à ce type de contraintes sont appelées « applications temps réels » **[2]**.

D'après la définition citée dans **[3]**, « *Un système temps réel est un système dont la correction des résultats ne dépend pas seulement de l'exactitude des traitements effectués mais également de la date à laquelle les résultats de ces traitements sont produits* ».

Mais, il faut différencier entre un système temps réel et un système rapide, en effet, le premier système doit atteindre le résultat désiré en respectant les contraintes temporelles, ces dernières dépendent de l'application et de l'environnement, par contre dans le deuxième système la rapidité dépend de techniques et outils utilisés **[4]**.

Un retard est donc considéré comme une erreur qui entraîne la défaillance du système et peut mener à des conséquences plus ou moins catastrophiques.

L'idée générale du fonctionnement en temps réel n'est pas nouvelle, et peut être illustrée en donnant des exemples de la vie quotidienne ; à titre d'exemple, une femme au foyer qui effectue ses tâches quotidiennes doit surveiller une machine à laver, préparer un repas pour le dîner, et prendre soin de ses enfants. Les signaux provenant de son environnement la forcent à basculer entre les tâches et réagir selon la vitesse déterminée par le type de signal. En cas d'urgence, quand un enfant est en danger, elle doit réagir immédiatement **[5]**.

2.2 Modèle de base d'un système temps réel

Le système temps réel peut être modélisé par un simple modèle (En termes des blocs fonctionnels importants) comme le montre la figure suivante :

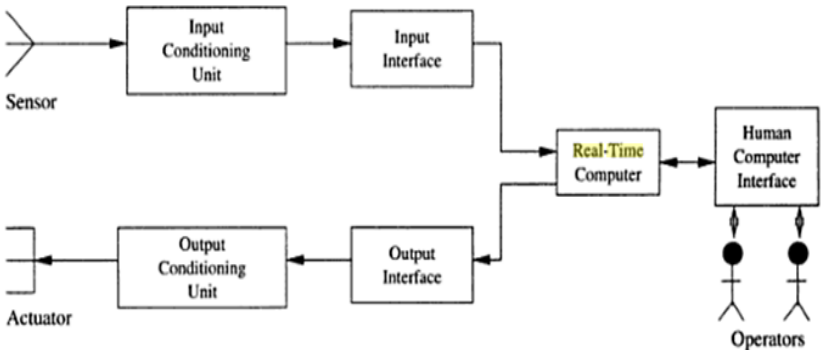


Figure 1.1: Un modèle de système temps réel.

D'après la figure 1.1, nous pouvons décrire brièvement les rôles des différents blocs fonctionnels d'un système temps réel

✓ *Les capteurs:* convertissent certaines caractéristiques physiques de son environnement en signaux électriques.

✓ *Les actionneurs:* Un actionneur est un dispositif qui prend ses entrées de l'interface de sortie de l'ordinateur et convertit ces signaux électriques en certaines actions physiques appliquées ensuite sur son environnement. les actions physiques peuvent être sous forme de mouvement, changement d'énergie thermique, électrique, ou pneumatique de certains objets.

✓ *Les unités de conditionnement:* les signaux électriques qui sont produits par un ordinateur peuvent rarement être utilisés pour provoquer directement un actionneur. Les signaux d'ordinateur ont généralement besoin de conditionnement avant qu'ils ne puissent être utilisés par l'actionneur. C'est ce qu'on appelle « le conditionnement des sorties ». De même, le conditionnement des entrées doit être effectué sur les signaux du capteur avant qu'ils ne puissent être acceptés par l'ordinateur.

✓ *Les unités d'interface:* normalement, les commandes du processeur sont livrés à l'actionneur à travers une interface de sortie. Cette dernière convertit la tension stockée en une forme analogique, puis la délivre en sortie à l'actionneur. Conversion analogique-numérique est souvent déployée dans une interface d'entrée **[6]**.

2.3 Caractéristiques des systèmes temps réel

Le fonctionnement des systèmes temps réel se distingue principalement des autres systèmes par l'implication explicite du facteur temps. Lorsque nous voulons concevoir un système temps réel, nous devons nous assurer qu'il répond à ces caractéristiques :

✓ *Contraintes de temps*: Chaque tâche temps réel est associée à des contraintes temporelles, parmi ces contraintes on cite ce qu'on appelle l'échéance ou « deadline ». le deadline de tâche sert à spécifier le temps avant lequel la tâche doit terminer son exécution et produire les résultats. C'est le système d'exploitation temps réel (RTOS) qui doit assurer que toutes les tâches respectent leurs deadlines à travers des stratégies d'ordonnancement des tâches appropriées.

✓ *Exactitude de fonctionnement:* La notion de l'exactitude dans les systèmes temps-réel est différente de celui utilisé dans des systèmes traditionnels. En effet, dans les systèmes temps réel, l'exactitude implique non seulement la production des résultats corrects, mais aussi la date à laquelle ces résultats sont produits. Un résultat logique correct produit après la date limite sera considérée comme un résultat incorrect.

✓ *Sécurité et fiabilité:* Pour les systèmes non-temps réel, la sécurité et la fiabilité sont des facteurs indépendants. Cependant, dans de nombreux systèmes temps réel ces deux facteurs sont liés entre eux ce qui rend la sécurité « critique ».Un système de sécurité critique est nécessaire pour être très fiable puisque toute défaillance du système peut causer des dommages importants.

✓ *Prévisibilité :* Permet de déterminer à l'avance si un système va respecter ses contraintes temporelles, et ça nécessite une bonne connaissance des paramètres liés aux calculs des activités = déterministe.

✓ *Simultanéité:* un système temps réel doit généralement répondre à plusieurs événements indépendants dans un temps très court et strict.

✓ *Distribué:* dans beaucoup de systèmes temps réel, les différentes composantes du système sont réparties naturellement dans des endroits géographiques éloignés.

✓ *Matériel personnalisé:* un système temps réel est souvent mis en œuvre sur un matériel personnalisé qui est spécifiquement conçu et développé à cet effet **[6] [7]**.

2.4 Domaine d'application des systèmes temps réel

Les systèmes temps réel sont présentés aujourd'hui dans des domaines très variés. Parmi les exemples usuels de systèmes temps réel, nous pouvons citer les domaines d'application suivants :

✓ *Applications industrielles*: elles constituent un domaine d'utilisation très important pour les systèmes temps réel, contrôle/commande de procédés manufacturiers ou continus.

✓ *Aérospatial :* simulation de vol, suivi de satellites, pilotage automatique,

✓ *Médical:* scanners IRM, matériel de radiothérapie, assistance et supervision de malades.

✓ *domaine militaire :* suivi de trajectoires de missiles, …

✓ *Équipement périphérique:* les imprimantes laser, photocopieurs numériques, télécopieurs, appareils photo numériques et les scanners.

✓ *Automobile et Transport:* systèmes automobiles de moteur de contrôle, le contrôle du trafic aérien, le contrôle du train à grande vitesse, guidage de véhicules, contrôle et régulation de trafic en milieu urbain

✓ *Applications de télécommunications:* transport de la parole, et systèmes de commutation.

✓ *Internet et applications multimédias:* la réalité virtuelle, téléconférences, consoles de jeu.

✓ *Electronique grand public:* les équipements audio, la téléphonie sur Internet, les fours à micro-ondes, machines à laver intelligentes, systèmes de sécurité à domicile, la climatisation et la réfrigération **[1] [6] [8]**.

2.5 Classification des systèmes temps réel

Suivant les contraintes temporelles, on peut distinguer Trois catégories de systèmes temps réel:

✓ *Systèmes temps réel à contraintes dures (hard real time constraints) :* ce sont des systèmes dont le fonctionnement est dégradé si les résultats ne se déroulent pas selon les contraintes temporelles spécifiées. Toutes les contraintes temporelles doivent être impérativement respectées, et par conséquent, le non-respect d'une contrainte temporelle peut avoir des conséquences catastrophiques.

Exemple : Le système d'interception de missiles est considéré comme étant un système à contraintes temporelles dures, car la non interception de missile à temps va conduire à des conséquences graves.

- ✓ *Systèmes temps réel à contraintes souples (soft real time constraints) :* à l'inverse de premier type de systèmes, ici, le non-respect des contraintes temporelles est toléré par le système, et ne conduit pas à des conséquences catastrophiques. le résultat peut être exploitable même s'il est fourni après l'échéance.

  Exemple : Le système de téléconférences est considéré comme étant un système à contraintes temporelles souples. Le décalage entre le son et l'image est souvent toléré.

- ✓ *Systèmes temps réel à contraintes fermes (firm real-time systems) :* le dépassement occasionnel des échéances est toléré.

  Exemple : projection vidéo (perte de quelques trames d'images).

Enfin, un seul système peut avoir un ensemble de sous-systèmes temps réel durs et souples à la fois **[7] [9] [10]**.

La plupart des systèmes temps réel sont par nature embarqués, c.à.d. ils sont intégrés dans un environnement et soumis aux mêmes contraintes physiques de cet environnement. Dans ce qui suit, on va essayer de détailler un peu ce qu'on appelle « les systèmes embarqués ».

3. Systèmes embarqués (SE)

Jour après jour, Notre vie devient de plus en plus dépendante des « systèmes embarqués ».

Les technologies embarquées sont collées dans nos activités quotidiennes, sans même qu'on ne s'en rende compte. Connaissons-nous le fait que le réfrigérateur, la machine à laver, le four à micro-ondes, l'air conditionné, la télévision, le lecteur de DVD et les systèmes de musique que nous utilisons dans notre maison sont construits autour d'un système embarqué? Nous pouvons voyager dans un véhicule "Honda", "Toyota" ou "Ford" mais avons-nous déjà pensé aux personnes génies qui travaillent derrière ses systèmes de sécurité offerts par le véhicule pour nous? Ce n'est rien mais un système embarqué intelligent **[11]**.

Le domaine des systèmes embarqués est très vaste ce qui rend difficile de donner une définition ou une description exacte de ces systèmes. La première question qui doit être posée est "Qu'est-ce que exactement un système embarqué ?

3.1 Définition des systèmes embarqués

Pour répondre à cette question, il est très facile de comprendre ce qui n'est pas un système embarqué, plutôt que d'essayer de décrire toutes les choses que le système embarqué peut-être. En effet, Un simple ordinateur fait habituellement un certain nombre d'objectifs:

consulter les emails, naviguer sur Internet, écouter de la musique, faire des traitements de texte, etc... Cependant, les systèmes embarqués « *embedded systems* » exécutent généralement une seule tâche, ou un très petit nombre de tâches en relations pour effectuer un traitement. On peut donc dire qu'un système embarqué est souvent un système qui est conçu pour un traitement particulier.

Chaque maison dispose de plusieurs exemples de systèmes embarqués. Tout appareil qui dispose d'une horloge numérique, par exemple, possède un petit microcontrôleur embarqué qui n'effectue aucune autre tâche que d'afficher l'horloge **[12]**.

De manière générale, les systèmes embarqués « *sont des dispositifs utilisés pour contrôler, surveiller ou aider le fonctionnement des équipements, des appareils ou des machines. "Embarqués" veut dire le fait qu'ils font partie intégrante du système. Dans de nombreux cas, leur «embarquabilité» peut être dans le sens où leur présence est loin d'être évidente à un simple observateur* » **[13]**.

De manière précise, un système embarqué peut être défini comme un système électronique et informatique autonome, possédant des ressources d'ordre spatial (taille limitée) et énergétique (consommation restreinte) limitées. Dans ce dernier, le processeur est intégré dans un système mécanique ou électrique plus grand et doté d'une autre fonction, pour résoudre un problème ou une tache spécifique. Le terme indique aussi la fusion de deux notions logiciel et matériel, c'est-à-dire le matériel et l'application sont intimement liés et noyés dans le matériel, et ne sont pas facilement discernables **[103]**.

3.2 Brève histoire sur les systèmes embarqués

Dans les premières années de l'informatique en 1930, des ordinateurs sont parfois dédiés à exécuter une seule tâche.

L'un des premiers systèmes embarqués moderne était « Apollo Guidance Computer », qui sert à collecter et fournir des informations de vol et de contrôler automatiquement toutes les fonctions de navigation des vaisseaux spatiaux du programme Apollo. Ce système a été développé au début des années 1960 par Charles Stark Draper au Laboratoire d'instrumentation MIT. Ensuite, un nouveau système embarqué a été produit en série était le D-17 d'Autonetics. Il servait de système de contrôle aux missiles Minuteman, publié en 1961**[11]**. Lorsque le Minuteman II est entré en production en 1966, le D-17 est remplacé par un nouvel ordinateur qui utilise des circuits intégrés de gros volumes.

Depuis ces premières applications, le prix des systèmes embarqués a diminué et leur puissance de traitement a augmenté. Le premier microprocesseur par exemple, Intel 4004 a été conçu pour les calculatrices et autres petits systèmes, mais il a encore besoin de mémoire externe.

Au milieu des années 1980, la plupart des composants communs des systèmes externes ont été intégrés dans la même puce que le processeur. Cette forme moderne de microcontrôleur a permis une utilisation plus large **[104]**. Maintenant les systèmes embarqués sont devenu le cerveau de la plupart des systèmes électroniques et ils ont existés presque dans toutes nos activités quotidiennes et dans toutes les choses qui nous entourent.

3.3 Domaine d'application des systèmes embarqués

Comme nous avons dit précédemment, nous vivons dans un monde où les systèmes embarqués jouent un rôle vital dans notre vie quotidienne. Ces dernières existent dans de nombreux objets et/ou systèmes. Et leur portée ne cesse de croître.

Quelques exemples des systèmes embarqués sont les suivantes:

- ✓ *Électroménager :* télévision, four à micro-ondes, machine à laver, réfrigérateur ;
- ✓ *Transport* : Automobile, Aéronautique (avionique) ;
- ✓ *Astronomie :* fusée, satellite artificiel, sonde spatiale ;
- ✓ *Télécommunication:* Set Top box, téléphonie, routeur, pare-feu ;
- ✓ *Impression :* imprimante multifonctions, photocopieur, scanner ;
- ✓ *Informatique:* disque dur, Lecteur de disquette ;
- ✓ *Multimédia:* console de jeux vidéo ;
- ✓ *Guichet automatique bancaire*(GAB) ;
- ✓ *Équipement médical* : scanners IRM.

Ces systèmes font maintenant partie intégrante dans beaucoup d'applications plus intéressantes telles que l'agriculture, la communication...etc. La conception de l'interface graphique d'utilisateur varie d'une application à une autre. Certaines applications n'ont pas besoin d'interface graphique du tous, alors que d'autres applications ont besoin d'au moins une interface audio.

La figure ci-dessous montre l'application des systèmes embarqués dans plusieurs domaines :

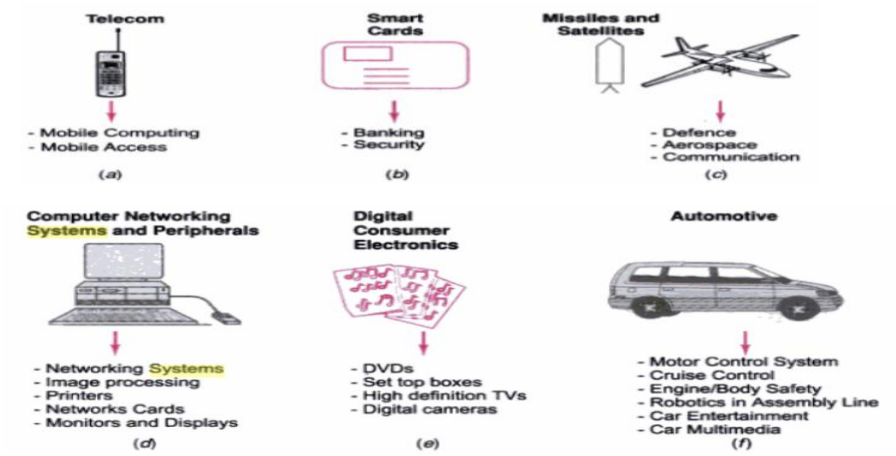


Figure 1.2: L'application de systèmes embarqués dans différents domaines [14].

3.4 Caractéristiques des systèmes embarqués

A la différence des systèmes universels qui effectuent plusieurs tâches, les systèmes embarqués sont conçus pour gérer un nombre limité de tâches. Les principales caractéristiques d'un système embarqué sont les suivantes :

✓ *Fonction unique:* C'est un système principalement numérique et exécute généralement un seul programme spécifique de manière répétitive.

✓ *Contraintes strictes*: tous les systèmes informatiques ont des contraintes sur les paramètres de conception, mais ces contraintes peuvent être plus étroites dans le cas des systèmes embarqués.

✓ *Réaction et temps réel:* un système embarqué doit constamment réagir aux évènements provenant de son environnement et doit calculer certains résultats en temps réel.

✓ *Traitement en parallèle:* les systèmes embarqués peuvent gérer plusieurs activités en temps réel en même temps. Ils peuvent contrôler simultanément différentes opérations.

✓ *consommation d'énergie faible.*

✓ *coût de fabrication faible :* ce qui signifie souvent la taille du code qui doit être limité **[15]**.

3.5 classification des systèmes embarqués

Il est possible d'avoir de multiples classifications pour les systèmes embarqués, en fonction de différents critères. Voici quelques critères utilisés dans la classification des systèmes embarqués, selon:

- la génération ;
- les conditions de complexité et de performance ;
- le comportement déterministe ;
- le déclenchement.

3.5.1 Classification selon la génération

Cette classification est basée sur l'ordre dans lequel les systèmes embarqués ont évolués depuis la première version jusqu'aujourd'hui. Donc ils peuvent être classés en:

✓ *Première génération:* les premiers systèmes embarqués ont été construits avec des circuits simples et de logiciel développés en code Assembleur. Le téléphone numérique avec clavier est un exemple de cette génération.

✓ *Deuxième génération:* le jeu d'instructions pour la deuxième génération de processeurs/contrôleurs est beaucoup plus complexe et plus puissant que la première génération de processeurs/contrôleurs. Certains systèmes embarqués de la deuxième génération contiennent des systèmes d'exploitation embarqués pour leur fonctionnement. Les systèmes d'acquisition de données, est un exemple de systèmes embarqués de la deuxième génération.

✓ *Troisième génération:* avec les progrès dans la technologie de processeur, les développeurs de systèmes embarqués commencent à utiliser des processeurs et des microcontrôleurs puissants dans leur conception. Un nouveau concept d'application et un domaine spécifique des processeurs/contrôleurs comme les processeurs de signaux numériques (DSP) et les circuits intégrés faisant des applications spécifiques (ASIC) ont fait leur apparition. le jeu d'instructions de processeurs est devenu plus complexe et plus puissant et le concept d'instruction pipeline a aussi évolué. Les systèmes embarqués sont déployés dans de nombreux domaines tels que la robotique, les médias, le contrôle des processus industriels, les réseaux ...

✓ *Quatrième génération:* L'arrivée de système sur puce (SoC), et les processeurs reconfigurables apporte des performances élevées, une intégration étroite et une miniaturisation dans le marché des dispositifs embarqués. Dans cette génération, les systèmes embarqués utilisent des systèmes d'exploitation en temps réel embarqués de haute performance pour leur fonctionnement. les téléphones intelligents, et les appareils d'internet mobiles (MID), sont des exemples de systèmes embarqués de quatrième génération.

3.5.2 Classification selon la complexité et la performance

Selon cette classification, les systèmes embarqués peuvent être regroupés en:

- ✓ *Systèmes Embarqués à petite échelle :* ils possèdent peu de matériel, et les logiciels aussi sont un peu complexes. Les systèmes embarqués à petite échelle sont généralement construits avec une faible performance et un faible coût de 8 ou 16 bits de microprocesseurs/microcontrôleurs. ils peuvent contenir ou non un système d'exploitation pour le fonctionnement.
- ✓ *Systèmes embarqués à moyenne échelle :* Les systèmes embarqués qui impliquent des exigences matérielles et logicielles un peu complexe entrent dans cette catégorie. Les systèmes embarqués à moyenne échelle sont généralement construits avec une performance moyenne et un faible coût de 16 ou 32 bits de microprocesseurs /microcontrôleurs ou des processeurs de signaux numériques. Ils contiennent un système d'exploitation pour le fonctionnement.
- ✓ S*ystèmes embarqués à grande échelle:* ils impliquent des exigences matérielles et logicielles très complexe, et ils sont employés dans des applications exigeant une haute performance. La mise en œuvre de la fonction cryptographique est un exemple de cette catégorie **[11]**.

3.5.3 Classification selon le comportement déterministe

✓ *Système Transformationnel :* système qui lit ses données et ses entrées lors de son démarrage, fournit ses sorties puis meurt.

✓ *Système Interactif :* système en interaction quasi permanente avec son environnement, y compris après l'initialisation du système; la réaction du système est déterminée par les événements reçus et par l'état courant (fonction des événements et des réactions passés); le rythme de l'interaction est déterminé par le système et non par l'environnement.

✓ *Système Réactif ou Temps Réel :* système en interaction permanente avec son environnement, y compris après l'initialisation du système; la réaction du système est déterminée par les événements reçus et par l'état courant (fonction des événements et des réactions passées); mais le rythme de l'interaction est déterminé par l'environnement et non par le système **[14]**.

4. Systèmes embarqués distribués temps réel (SEDTR)

4.1 Définition des systèmes embarqués temps réel :

Aujourd'hui, les systèmes embarqués trouvent leur place dans de nombreuses applications autour de nous, commençant par les produits électroniques, les appareils et se terminant avec les systèmes critiques pour la sécurité dans des applications telles que l'avionique, l'automobile, les équipements médicaux, etc…Très souvent, la réponse de ces systèmes informatiques doit être en temps réel, donc ce sont des systèmes embarqués temps réel.

4.2 Flot de conception de systèmes embarqués temps réel

La conception d'un système embarqué temps réel nécessite quelques procédures afin de garantir que tous les délais seront respectés (dans le cas des systèmes durs). Si ces contraintes temporelles ne peuvent pas être fournies, le système est considéré non ordonnançable et plus probablement, sa conception ne répondra pas aux exigences en termes de rapidité d'exécution.

Dans la figure 1.3, nous présentons un flot de conception au niveau système qui commence par une spécification de haut niveau (informelle), qui peut être exprimé dans plusieurs langues, y compris le langage naturel. La spécification du système est ensuite raffinée dans un modèle formel et abstrait (qui peut être capturé dans un ou plusieurs langages de modélisation). A partir du modèle de système, une méthodologie suit une phase d'exploration de la conception dans laquelle plusieurs architectures de système sont sélectionnés, différents moyens pour allouer les fonctionnalités aux ressources disponibles sont évalués, et plusieurs alternatives pour l'ordonnancement et la synthèse des paramètres de communication sont examinés, pour garantir à la fin que le modèle résultant du système répond aux exigences imposées par la conception actuelle **[16].**

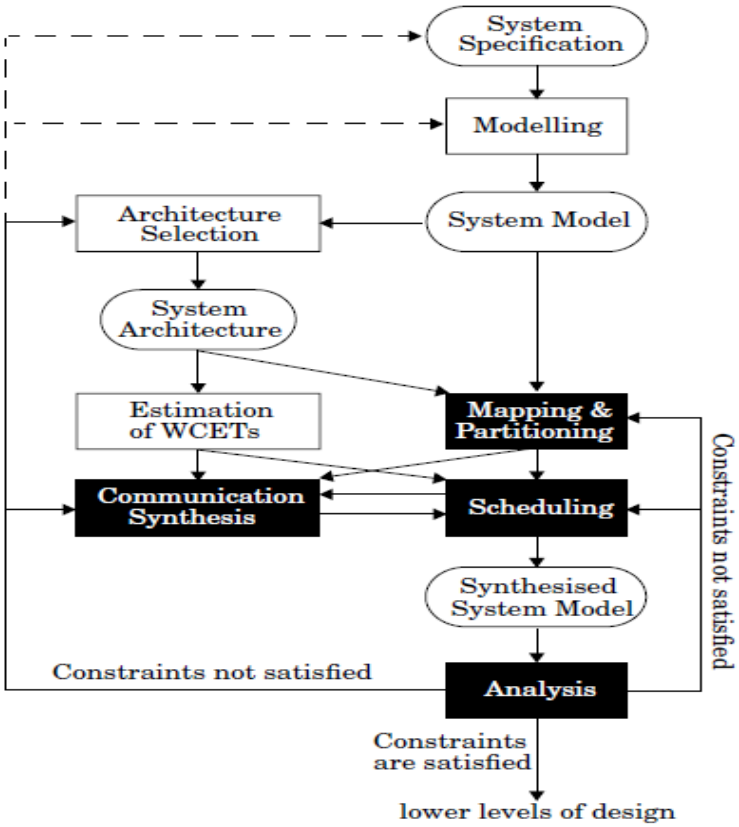


Figure 1.3: Flot de conception au niveau système [16].

4.3 Contraintes des systèmes embarqués temps réel

Nous distinguons deux types de contraintes : les contraintes temporelles et les contraintes matérielles.

✓ *contraintes temporelles :* les contraintes temporelles dans les systèmes embarqués temps réel peuvent être strictes (dures) ou souples. Dans le premier cas (contraintes dures), le non-respect d'une contrainte (échéance, arrivée d'un message après les délais, …) peut mener à des conséquences graves. Par contre dans le deuxième cas (contraintes souples), le non-respect de contraintes temporelles est toléré.

✓ *Contraintes matérielles :* généralement, les systèmes embarqués temps réel disposent d'une puissance de calcul faible et d'une mémoire limitée, cela est dû aux restrictions sur le poids, le volume, la consommation d'énergie, et le prix. La gestion de la mémoire et la consommation d'énergie sont les contraintes les plus importantes **[17]**.

4.4 Spécification des systèmes embarqués temps réel

La spécification de ces systèmes se fait en trois étapes complémentaires :

✓ *Spécification fonctionnelle :* dans cette étape, les fonctions spécifiques nécessaires au contrôle et à la commande de l'environnement doivent être présentées, chacune de ces fonctions est assurée par un composant logiciel ou bien ce qu'on appelle la tâche, l'ensemble des tâches et leurs relations de dépendance sont modélisés par un graphe de dépendance.

✓ *Spécification architecturale :* dans cette étape, les différents composants de l'architecture doivent être spécifiés ainsi que le type de réseau de communication :

- Le processeur : il faut bien spécifier les types des processeurs utilisés dans l'architecture matérielle selon les fonctions du système.
- Les capteurs et les actionneurs : les systèmes embarqués temps réel doivent doter des capteurs/actionneurs pour faire l'interaction avec l'environnement, ces derniers doivent être placés dans l'architecture dans le bon endroit.
- Moyen et type de réseau de communication : la communication inter-processeurs se fait soit par envoie de message ou par partage de données, pour gérer cette communication une connaissance de la topologie de réseau et des moyens de communication (mémoire partagée, bus, lien point-à-point, pont…) est nécessaire.

✓ *Spécification des contraintes :* pour exécuter les différentes fonctions du système sur l'architecture spécifiée, quelques contraintes temporelles et matérielles sont nécessaires. Chaque tâche est associée par une date de début d'exécution, une période et une date limite avant laquelle la tâche doit terminer son exécution. Aussi, il est nécessaire de déterminer les contraintes matérielles pour chaque tâche c'est-à-dire allouer la tâche à un ou plusieurs processeurs pour l'exécuter **[18].**

4.5 Architectures des systèmes embarqués distribués temps réel

4.5.1 Architecture faiblement distribuée

Au début des années 80, des bus de terrain ont été développés par les constructeurs automobiles et équipementiers : bus CAN par Bosh, Van par PSA et Renault, J1850 par les constructeurs américains, A-Bus par Volkswagen et K-Bus par BMW. Ces bus bifilaires dédiés aux environnements perturbés tels que les automobiles permettent de relier entre eux les calculateurs. Le bus le plus utilisé actuellement est le bus CAN, il est devenu un standard dans les applications automobiles. Grâce à ces nouveaux bus, ils sont apparues dans les applications temps réel embarquées des architectures que l'on qualifie ici de « faiblement

distribuées ». Elles sont constituées d'un ensemble de calculateurs reliés entre eux par un bus. Il est possible de contrôler le comportement global de l'ensemble de tous les calculateurs et il est possible de partager des capteurs entre les calculateurs.

4.5.2 Architecture fortement distribuée

Cette architecture est la plus utilisée. Les capteurs et actionneurs deviennent intelligents, ils peuvent directement être connectés sur le bus. Cette approche permet de réduire considérablement tous les câblages, car tous les organes électriques du véhicule peuvent être reliés au bus. C'est aussi l'approche la plus complexe à mettre en œuvre au niveau logiciel. Toute la difficulté consiste à gérer efficacement le multiplexage des données issues des capteurs et des calculateurs sur le bus de telle sorte que les contraintes temporelles de chacun des signaux soient satisfaites.

4.5.3 Hétérogénéité

Les architectures temps réel embarqués sont actuellement non seulement fortement distribuées, mais elles sont aussi hétérogènes. Une même application peut intégrer une dizaine de calculateurs. Pour garantir une exécution temps réel des algorithmes il est parfois nécessaire de disposer d'une grande puissance de calcul fournie par exemple par des DSP1 ou des microprocesseurs performants. Certaines fonctions peuvent aussi être programmées sur des FPGA ou des ASIC. Des microcontrôleurs sont aussi utilisés pour leur capacité à gérer un grand nombre d'entrées et de sorties avec un minimum de composants périphériques. Enfin, une application peut aussi intégrer plusieurs médias de communication différents.

5. Conclusion

Les systèmes embarqués temps réel rencontrent plusieurs problèmes lors de leur réalisation afin de garantir certaines propriétés à savoir : l'efficacité, la rapidité, la fiabilité,…

Le problème d'optimisation des performances est un facteur qui a un grand impact sur ces systèmes. Plusieurs techniques d'optimisation ont été développés, la résolution du problème d'allocation et d'ordonnancement des tâches est la première étape avant d'entamer la plupart de ces techniques, c'est ce qu'on va détailler dans le prochain chapitre.

Chapitre 2 Ordonnancement temps réel

Dans ce chapitre :

6. Introduction.
7. Ordonnancement temps réel.
   - Ordonnancement temps réel monoprocesseur.
   - Classification des algorithmes d'ordonnancement.
   - Ordonnancement temps réel multiprocesseurs.
8. Conclusion.

Résumé

Le début de ce chapitre décrit tous ce qui concerne l'ordonnancement des tâches temps réel sur une architecture monoprocesseurs, puis sur une architecture multiprocesseurs. Les principales classes des algorithmes d'ordonnancement temps réel (pour les tâches périodiques et apériodiques) sont aussi présentées.

1. Introduction

Le problème d'ordonnancements peut être rencontré dans plusieurs situations de la vie quotidienne et il doit être résolu dans de nombreux domaines. Par exemple : Organiser des activités ou bien planifier un itinéraire de voyage sont autant d'exemples de petits problèmes d'ordonnancement que nous tentons de résoudre tous les jours sans même qu'on ne s'en rende compte.

L'ordonnancement est aussi lié à plusieurs secteurs de recherches et d'activités. En informatique, l'allocation des tâches aux différents processeurs est considérée comme un problème d'ordonnancement. En production, l'ordonnancement consiste à déterminer l'ordre des opérations à réaliser sur les différentes machines de l'atelier. En gestion de projet, ordonnancer, c'est planifier l'ensemble des activités constituant le projet. Chacun de ces contextes nécessite la détermination des caractéristiques propres des tâches et des ressources.

Les systèmes embarqués temps-réel sont des systèmes dans lesquels l'élément clé est d'exécuter les différentes tâches dans des délais bien précis. Un exemple classique est celui de l'airbag d'une voiture, dans lequel le sac doit se gonfler exactement dans un temps bien précis ni trop tôt ni trop tard ce qui implique que l'ordre d'exécution des tâches est très important pour ce type de systèmes.

En plus, les systèmes embarqués temps réel sont souvent caractérisés par le besoin d'exécuter plusieurs tâches sur un nombre limité de processeurs. L'ordonnancement de ces tâches sur les processeurs avec la garantie a priori des diverses exigences de ces systèmes (des contraintes de temps, de ressources, d'énergie consommé,…) est considéré comme un problème difficile.

2. Ordonnancement temps réel embarqués

Les premiers travaux qui appliquent les politiques d'ordonnancements ont commencé depuis le début de l'informatique. En 1973, Liu et Layland publient un papier sur l'ordonnancement des tâches temps réel indépendantes en utilisant un seul processeur. Depuis, des questions sur le sujet d'ordonnancement ont donné lieu à de nombreux travaux, avec récemment un nouvel engouement pour les architectures multiprocesseurs **[19] [20] [21] [22] [23] [24] [25] [26] [27] [28]**.

2.1 Définition d'ordonnancement temps réel embarqué

De façon générale, l'ordonnancement consiste à donner l'ordre d'exécution d'un ensemble de tâches ayant certaines caractéristiques connues sur un ensemble limité d'unités de traitement d'une architecture, afin d'optimiser un ou plusieurs objectifs.

Pour une architecture donnée, il existe un grand nombre de possibilités d'ordonnancement mais seulement un certain nombre parmi ces possibilités permet de respecter les contraintes de dépendances de données, les contraintes temporelles ainsi que les contraintes de l'architecture. Cet ensemble de solutions valides est d'autant plus restreint que les contraintes sont fortes. C'est le cas des systèmes embarqués temps réel qui sont des applications souvent complexes et dont les contraintes de coût se traduisent généralement par de fortes contraintes matérielles **[18]**.

Dans les systèmes classiques (non-temps réel), l'objectif typique de l'ordonnancement est de maximiser le nombre de tâches qui terminent leur temps d'exécution par unité de temps et/ou de minimiser leur temps d'attente moyen. Dans le cas de l'ordonnancement temps réel, l'objectif est de respecter l'échéance de chaque tâche en assurant que chaque tâche doit terminer son exécution dans un délai spécifié.

De manière formelle, L'ordonnancement désigne le mécanisme permettant de choisir quel processus doit être exécuté par quel processeur, de manière à optimiser un ou plusieurs critères. La particularité de l'ordonnancement temps réel est qu'il nécessite de respecter des contraintes de temps réel. Le processus qui définit l'ordre d'exécution entre les composants de l'algorithme (les tâches), est appelé algorithme d'ordonnancement **[29].**

2.2 Ordonnancement temps réel monoprocesseur

Comme nous avons dit précédemment, pour un ensemble donné de tâches, le problème d'ordonnancement réside dans l'ordre sur lequel les tâches doivent être exécutées de telle sorte que toutes les contraintes sont satisfaites. Une tâche est caractérisée par un temps d'exécution, une date de début d'exécution, une date limite, et un ensemble de ressources. L'ordre d'exécution des tâches dépend fortement des relations de précédence entre eux. C'est à dire que l'exécution d'une tâche ne peut pas commencer avant que tous ses prédécesseurs sont terminés. Lorsque l'ensemble des tâches doit s'exécuter sur une architecture composée d'un seul processeur, l'ordonnancement est alors monoprocesseur.

Le problème d'ordonnancement monoprocesseur consiste alors à déterminer pour chaque instant quelle tâche doit bénéficier de la ressource (processeur).

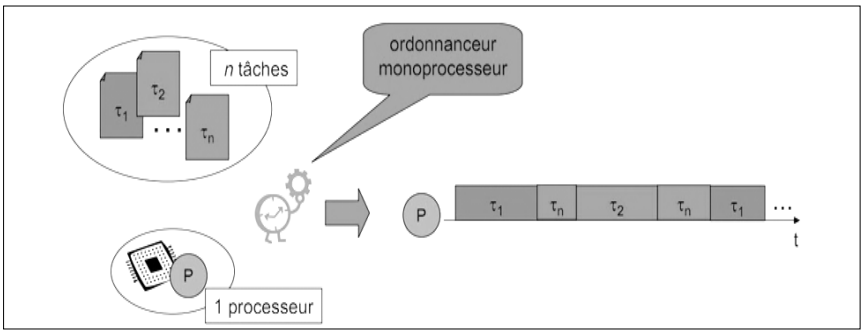


Figure 2.1 : Ordonnancement monoprocesseur [30].

Un système temps réel monoprocesseur est défini par:

- ✓ Un modèle de tâche.
- ✓ Un modèle de contraintes.
- ✓ Un modèle d'ordonnancement.

2.2.1 Modèle canonique des tâches temps réel

Pour analyser le comportement temporel d'un système temps réel, toutes les activités qui vont être exécutées dans le processeur sont modélisés sous forme d'un ensemble T de n tâches temps réel, T = {τ1, τ2, ..., τn}, où chaque tâche τi est une séquence ordonnée indivisible d'instructions **[43]**. Chaque tâche est dotée d'un ensemble de caractéristiques :

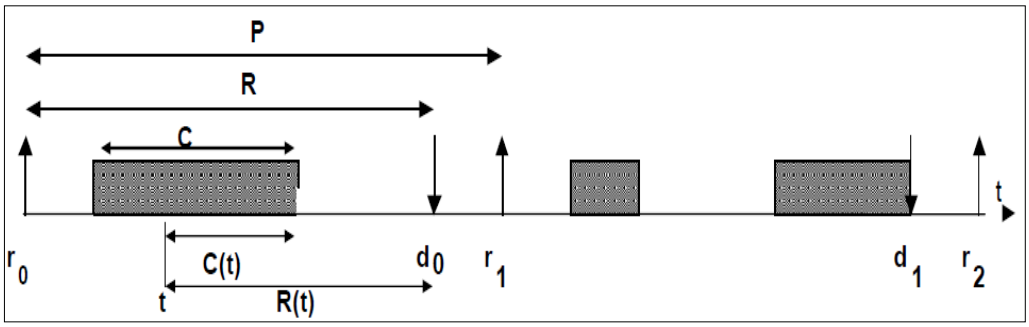


Figure 2.2 : Le modèle canonique des taches temps réel.

- P : période d'exécution.

- C: durée d'exécution maximale, C(t): temps d'exécution restant à t.

- R: délai critique (deadline), R(t): délai critique dynamique (temps restant à t jusqu'à d).

- r_k: date de réveil de la k^{ieme} instance de la tâche, $r_k = r_0 + kP$.

- $d_k = r_k + R$: date d'échéance.

- Quand R=P : tâche à échéance sur requête.

Pour simplifier, toutes les instances de la même tâche τi sont supposés avoir la même durée d'exécution maximale Ci, le même deadline Di et la même période Ti.

2.2.1.1 Caractéristiques d'une tâche temps réel

Une tâche est composée d'un ensemble d'instructions qui vont s'exécuter sur un processeur. La caractérisation d'une tâche peut varier d'un modèle d'ordonnancement à l'autre et selon la nature de celle-ci.

L'instance d'une tâche temps réel est définit comme suit : chaque fois qu'un événement se produit, il déclenche la tâche qui gère cet événement. En d'autres termes, une tâche est générée lorsqu'un événement particulier survient, ce qui implique que cette dernière se reproduit plusieurs fois à des instants différents en fonction de l'arrivée des événements. Il est possible que les tâches temps réel reviennent à des instants aléatoires. Cependant, la plupart des tâches temps réel se reproduisent à des périodes fixes. Par exemple, dans un système de détection de la température dans une usine chimique, la tâche qui est dédiée à cette fonction peut se reproduire indéfiniment avec une période fixe, parce que la température est prélevée périodiquement, alors qu'une tâche qui manipule par exemple l'interruption d'un appareil peut se produire à des instants aléatoires. Chaque fois qu'une tâche se produit, elle est appelée « une instance de la tâche ». La première fois qu'une tâche se produit est appelé la

première instance de la tâche. La prochaine apparition de la tâche est appelée la deuxième instance, et ainsi de suite **[105]**.

Parmi les caractéristiques des tâches temps réel les plus utilisées on peut citer : (tous ces paramètres sont illustrés sur la Figure 2.2) :

✓ *Ci (Computing time) :* c'est la durée d'exécution d'une tâche ti. Ce paramètre est considéré dans la majorité des travaux sur l'ordonnancement temps réel comme le pire cas des temps d'exécution (WCET pour Worst Case Execution Time) sur le processeur où elle va être exécutée. Le WCET représente une borne supérieure du temps d'exécution c'est à dire que la tâche peut se terminer plus tôt. Pour être valable, la valeur de ce paramètre ne doit pas être trop surestimée et elle doit être sûre (jamais dépassée).

✓ *Di (Deadline) :* c'est l'échéance ou la date au plus tard. il représente l'instant auquel la tâche doit terminer son exécution et produire des résultats et dont le dépassement provoque une violation de contrainte temporelle. Deux types de deadlines existent :

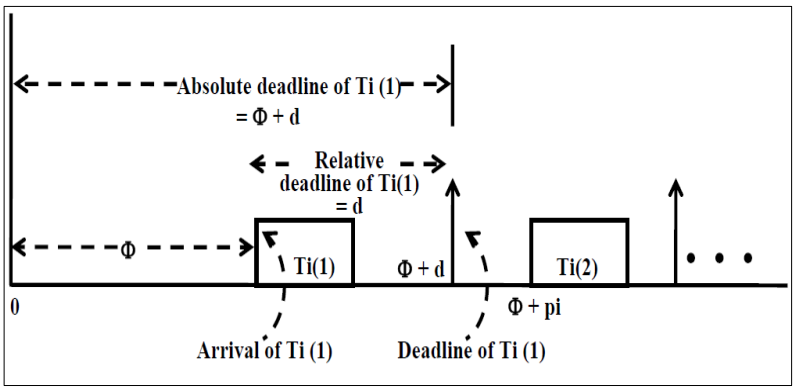


Figure 2.3 : Le deadline absolu et le deadline relative [105].

- *Deadline absolu D (ti) +R (ti) :* est la valeur de temps absolue (à compter de la date 0), cette valeur est égale à l'intervalle de temps entre l'instant 0 et l'instant réel auquel le deadline se produit.
- *Deadline relatif D (ti)* : est l'intervalle de temps entre la date d'arrivée de la tâche et l'instant où l'échéance correspondante se produit.

On peut observer dans la figure **2.3** que le deadline relatif de la tâche Ti(1) est **d**, alors que son deadline absolu est **φ + d**.

✓ *Ti (Task period):* période d'exécution d'une tâche, ou la séparation temporelle entre les temps d'arrivée successifs générés par la tâche, et elle n'est valide que pour les tâches périodiques.En fonction de la valeur associée à Ti et Di, nous pouvons distinguer trois classes de systèmes :

- les systèmes à échéances sur requête : lorsque Di = Ti $\forall$i ;
- les systèmes à échéances contraintes : lorsque Di $\leq$Ti $\forall$i ;
- les systèmes à échéances arbitraire : lorsqu'il n'existe pas de contrainte entre l'échéance et la période des tâches.

✓ *Ri (Ready time ou Release time) :* c'est la date à laquelle la tâche Ti peut commencer son exécution (prête à être exécutée), elle est appelée aussi date de demande d'activation. si toutes les tâches démarrent au même instant, on dit que les tâches sont à départ simultané.

✓ *Si (Start time), Ei (End time) :* sont respectivement la date à laquelle la tâche Ti est exécutée sur le processeur, elle est appelée aussi la date de début d'exécution, et la date à laquelle la tâche Ti finit son exécution, elle est appelée aussi la date de fin d'exécution.

✓ *li (Laxity):* c'est la laxité d'une tâche Ti, qui représente le temps maximum pendant lequel la tâche peut ne pas disposer du processeur sans manquer son échéance.

On plus de ces paramètres, une tâche temps réel peut également posséder les contraintes et les exigences suivantes :

✓ *Temps de réponse:* le temps de réponse d'une tâche est la durée de temps qui s'écoule entre l'activation de la tâche et le temps où la tâche produit ses résultats.

✓ *Relations de précédence entre les tâches:* une tâche précède une autre tâche, si la première tâche doit terminer son exécution avant que la deuxième commence. Quand une tâche Ti précède une autre tâche Tj, chaque instance de Ti doit précéder l'instance correspondante de Tj. Les précédences entre les tâches sont impliquées par des attentes de messages, ou de synchronisation.

✓ *Contraintes de ressources :* une tâche peut nécessiter l'accès à certaines ressources autres que le processeur, tels que les périphériques d'entrées/sorties, les réseaux, des fichiers et des bases de données.

✓ *Partage des données:* les tâches doivent souvent partager leurs résultats entre eux. lorsqu'une tâche a besoin de partager ses résultats avec une autre tâche, clairement, la deuxième tâche doit précéder la première tâche. En fait, la relation de précédence entre deux tâches implique parfois le partage des données entre les deux tâches (par exemple, la première tâche fait un passage des résultats vers la deuxième tâche). Cependant, ce n'est pas toujours le cas. Une tâche peut précéder une autre tâche, même s'il n'existe aucun partage de données **[32] [105]**.

2.2.1.2 Types de tâches temps réel

Une tâche peut s'exécuter en raison d'un événement interne tel qu'une interruption d'horloge qui se produise à des intervalles réguliers. Une autre tâche peut se produire de manière aléatoire en raison d'un événement extérieur tel qu'une interruption due à un appui.

Ainsi, Il existe trois types de tâches temps réel: périodiques, apériodiques et sporadiques, dans ce qui suit, nous examinons les caractéristiques importantes de ces trois grandes catégories de tâches temps réel:

✓ *Tâche périodique :* Une tâche périodique est celle qui se répète régulièrement après un certain intervalle de temps fixe (période). Une tâche Ti est complètement caractérisée par le quadruplet *Ti (Φi, Ci, Pi, Di)*, où $\Phi i \geq 0$, $Ci > 0$ et $Ci \leq \min(Pi, Di)$, $Pi > 0$, et $Di > 0$. Di peut être inférieure, égale ou supérieure à la période Pi, mais le cas le plus fréquent est celui où Di=Ti. La valeur Φi de la tâche désigne l'instant auquel la première instance de la tâche devient prête pour être exécuter (la date de son premier réveil), Ci son temps d'exécution au pire cas ou WCET (*Worst Case Execution Time*), Pi sa période d'exécution (Conformément avec la notation commune, la période est désignée par T), et Di son échéance relative, c'est-à-dire le temps maximal dont on dispose pour exécuter la tâche. Les instants précis auxquels se répètent les tâches périodiques sont délimités par un évènement périodique : horloge temps réel, message ou synchronisation émis par une tâche périodique.

En distingue deux types de tâches périodiques :

- *Tâches périodiques synchrones* : lorsque toutes les premières instances d'un ensemble de tâches périodiques sont commencées leur exécution en même temps, (les tâches sont alors à départ simultané, habituellement au temps t=0).

- *Tâches périodiques asynchrones* : lorsque les premières instances d'un ensemble de tâches périodiques sont commencées leur exécution à différents moments (les tâches sont alors à départ différé).

Le facteur d'utilisation *Ui* d'une tâche périodique Ti est le rapport entre son pire temps d'exécution et sa période : *Ui* =Ci/Ti.

Le facteur d'utilisation U_{tot}(T) d'un système composé de tâches périodiques est la somme du facteur d'utilisation de toutes les tâches qui le composent : $U_{tot}(\tau) = \sum_{i=1}^{n} U_i$.

✓ *Tâche apériodique :* Une tâche apériodique est réveillée par un événement externe (peut être un clic sur bouton). Le délai entre deux événements successifs, ainsi que la fréquence ne sont pas connus. Le temps d'arrivée d'une tâche apériodique peut être modélisé comme une variable aléatoire générée par une loi de probabilité. Les tâches apériodiques sont généralement des tâches temps réel souples. Pas de contraintes temporelles pour les tâches apériodiques.

✓ *Tâche sporadique :* une tâche sporadique est activée irrégulièrement avec un taux connu. Ce taux est caractérisé par un intervalle de temps minimum entre deux arrivées successives de deux tâches. Une tâche sporadique peut être caractérisée par un triplé *Ti (Ci, gi, Di)* tel que Ci est le temps d'exécution au pire cas (WCET) d'une instance de la tâche, gi dénote la séparation minimale entre l'arrivée successive de deux instances, ce paramètre implique que si une tâche sporadique se produit, l'instance suivante ne peut pas se produire avant le temps gi. Dans un robot une tâche qui doit gérer un obstacle qui peut apparaître soudainement est considérée comme une tâche sporadique **[32] [33] [34] [35] [36] [105]**.

2.2.2 Modèle de contraintes

2.2.2.1 Contraintes temporelles

Une contrainte temporelle typique est l'échéance (deadline), qui représente le temps avant lequel une tâche devrait terminer son exécution. On distingue trois types de contraintes temporelles :

✓ *Contraintes durs (hard constraint) :* la tâche ne doit pas dépasser leur échéance et le non-respect d'une contrainte de temps mène à des conséquences graves (humaines, économiques, écologiques) : besoin de garanties (ex : Airbag).

✓ *Contraintes souples/mous (soft constraint) :* on peut tolérer le non-respect occasionnel d'une contrainte de temps (garanties probabilistes), mais cela provoque une dégradation des performances (ex : Distributeur automatique).

✓ *Contraints fermes (firm constraint) :* le manque de l'échéance ne cause pas de dommages au système, mais le résultat de système n'a pas de valeur.. (ex : Systèmes multimédias) **[37]**.

2.2.2.2 Contraintes matérielles

Une ressource est toute structure de logiciel qui peut être utilisée par la tâche pour accomplir son exécution. Typiquement, une ressource peut être une structure de données, un ensemble de variables, une zone de mémoire principale, un fichier, une partie de programme ou un ensemble de registres d'un dispositif périphérique. Une ressource dédiée à une tâche particulière est dit privée, alors une ressource qui peut être utilisée par plusieurs tâches est appelé une ressource partagée.

Les contraintes matérielles sont dues aux restrictions des ressources dans les systèmes embarqués. La gestion de mémoire et la consommation d'énergie sont les principales contraintes étudiées dans la littérature.

2.2.2.3 Contraintes de précédences

Dans certaines applications, les tâches ne peuvent pas être exécutées dans un ordre arbitraire, mais elles doivent respecter certaines relations de précédence définies à l'étape de conception. Ces relations de précédence sont généralement décrites par un graphe dirigé acyclique (DAG), où les tâches sont représentées par des nœuds et les relations de précédence par des flèches **[37]**.

2.2.3 Modèle d'ordonnancement :

2.2.3.1 Travaux voisins

De nombreux algorithmes ont été proposés dans la littérature, afin de traiter le problème de l'ordonnancement temps réel des tâches. Par exemple, Liu et al. **[45]** ont proposés les deux algorithmes RM et EDF pour l'ordonnancement des tâches périodiques indépendantes, puis, en 1976 J. Blazewicz **[97]** a proposé une démarche qui traite les contraintes de précédences. Dans le cas où certaines tâches périodiques peuvent dépasser leurs échéances

sans pour autant compromettre le bon fonctionnement du système, on distingue les travaux de Hamdaoui et al. dans **[98]** qui ont proposé le modèle (m,k)-firm.

Srinivasan et Anderson **[99]** ont étudié l'ordonnancement dynamique de tâches sur les systèmes multiprocesseurs. Page et Naughton **[100]** ont présenté une stratégie d'ordonnancement qui utilise l'algorithme génétique pour ordonnancer de façon dynamique les tâches sur les processeurs hétérogènes dans les systèmes distribués hétérogènes. Dans **[101]** une méthode d'ordonnancement pour les systèmes mono processeur est introduite.

2.2.3.2 Concepts de base

Un ordonnanceur est un algorithme de l'exécutif temps réel qui attribue à chaque instant le processeur à la tâche la plus prioritaire.

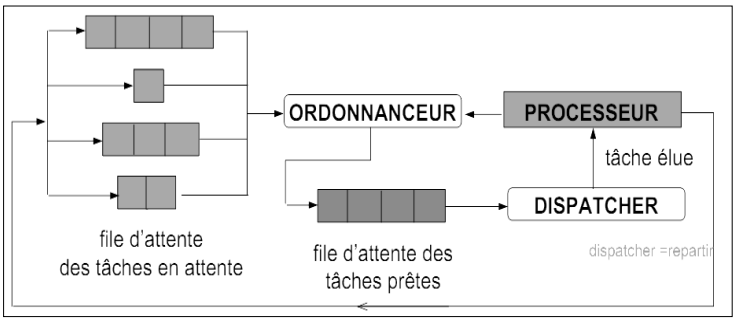


Figure 2.4 : L'ordonnancement des tâches temps-réel.

Ordonnanceur : alloue le processeur aux différentes tâches.

Dispatcheur : implémente l'ordonnanceur (élection des tâches prêtes).

On va introduire quelques concepts de base qui vont être utilisé dans notre travail :

✓ *Ordonnanceur valide :* on dit qu'un ordonnanceur est valide lorsque une seule tâche au plus est allouée au processeur, aucune tâche n'est prévue avant son temps d'arrivée. Les relations de précédence et les contraintes de ressources de toutes les tâches sont satisfaites**.**

✓ *Ordonnanceur faisable* : un ordonnanceur valide est dit faisable seulement si toutes les tâches respectent leurs contraintes temporelles par cet ordonnanceur.

✓ *Ordonnanceur optimale* : un ordonnanceur est dit optimal s'il n'existe aucun autre ordonnanceur plus compètent que l'ordonnanceur optimale **[105]**.

2.3 Classification des algorithmes d'ordonnancement

Le choix de l'algorithme d'ordonnancement est très important, ce choix est basé généralement sur les exigences spécifique du problème.

2.3.1 Critères de classification :

La classification se fait selon plusieurs critères :

- L'instant ou l'ordre d'exécution des tâches : en-ligne / hors-ligne.
- La possibilité d'interruption d'une tâche par une autre plus prioritaire: préemptif / non préemptif.
- L'ordonnancement est fixe ou réévaluer en ligne: statique / dynamique.
- La qualité de résultat de l'ordonnancement (quel est le meilleur): optimal / non optimal.
- Le respect des contraintes est avec un taux de 100% ou bien avec une tolérance: à contraintes strictes / souple.
- La topologie: centralisé / distribué.
- Le nombre de processeurs : monoprocesseur/ multiprocesseur.

2.3.2 Typologie des algorithmes d'ordonnancement

L'ordonnancement est le sujet le plus recherché dans les systèmes temps réel depuis la fin des années 1960, parce que le problème de ces systèmes est d'assurer que toutes les tâchas respectent leurs contraintes temporelles. La variété des paramètres de performance, les approches d'ordonnancement et les types de ressources de traitement utilisés par les tâches impliquent une grande variété d'algorithmes d'ordonnancement **[38].**

2.3.2.1 Ordonnancement « monoprocesseur » / « multiprocesseur »

On dit que l'ordonnancement est monoprocesseur si toutes les tâches vont être s'exécutées sur un seul et même processeur. Dans le cas où plusieurs processeurs sont disponibles dans le système, l'ordonnancement est multiprocesseur.

2.3.2.2 Ordonnancement « hors-ligne » / « en-ligne »

Hors-ligne : avant l'exécution, l'algorithme construit la séquence complète et fixe d'ordonnancement des tâches à partir des paramètres temporelles de celles-ci, puis cette

séquence est exécuté en-ligne avec ou sans préemption. Cet ordonnancement nécessite la connaissance a priori de caractéristiques des tâches.

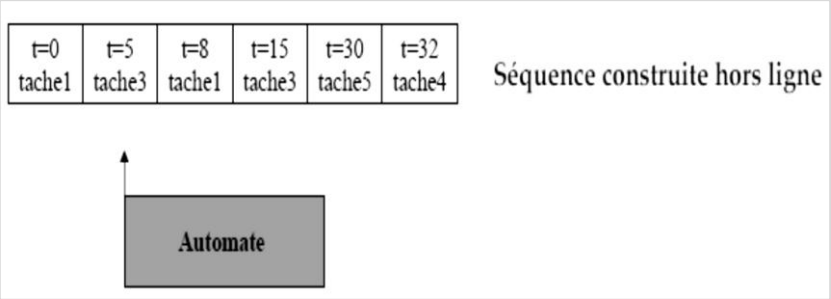


Figure 2.5 : Ordonnancement hors-ligne.

En-ligne : à tout instant de l'exécution, l'algorithme choisit la prochaine tâche à ordonnancer au cours de la vie de l'application en fonction des évènements qui surviennent ; l'ordonnanceur choisit la prochaine tâche à exécuter selon le critère de priorité **[39]**.

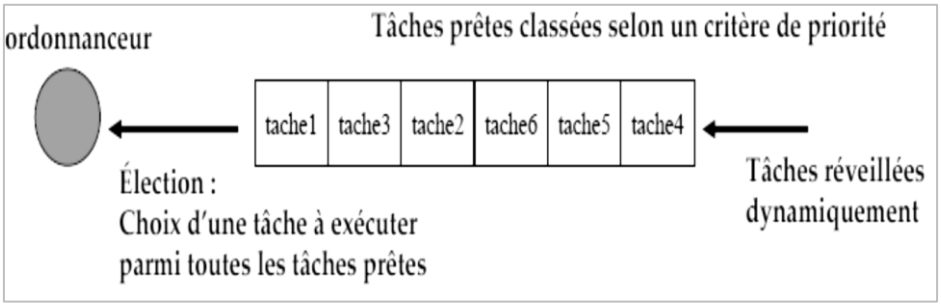


Figure 2.6 : Ordonnancement en-ligne.

2.3.2.3 Ordonnancement « Préemptif » / « Non préemptif »

Si l'exécution d'une tâche peut être interrompu pour exécuter une autre tâche plus prioritaire, on dit que cet ordonnancement est préemptif. L'ordonnanceur préemptif ajoute un temps au temps total de calcule, ce temps due au changement de contextes des tâches pendant la préemption. L'ordonnancement préemptif se fait soit sans notion de priorité : temps-partagé avec politique du tourniquet (round-robin), soit avec notion de priorité (statique ou dynamique) : la tâche la plus prioritaire obtient le processeur. Si l'exécution d'une tâche ne peut interrompre l'exécution d'une autre tâche une fois cette dernière entame son exécution, on dit que l'ordonnancement est non-préemptif. L'ordonnancement non-préemptif se fait soit selon l'ordre d'arrivée : premier arrivé, premier servi (First Come First Serve - FCFS), soit selon la durée de calcul : le plus court d'abord (Shortest Job First – SJF) **[31] [40].** Il est bien connu que l'ordonnancement préemptif dans les systèmes monoprocesseur, est plus

performant que l'ordonnancement non préemptif en termes d'ordonnançabilité des tâches **[41].**

Il est possible de trouver un système hybride dans le sens où l'ordonnancement est préemptif et non préemptif en même temps, c.à.d. que le système ne peut être préemptable qu'à certains points **[38]**.

2.3.2.4 Ordonnancement « optimal »/ « non optimal »

Un algorithme d'ordonnancement temps réel optimal, est celui qui échoue à respecter l'échéance, seulement lorsqu'aucun autre algorithme d'ordonnancement n'a réussi à respecter cette échéance **[32].**

En d'autre terme, un algorithme est dit optimal s'il est capable de trouver un ordonnancement pour tout ensemble faisable de tâches. Si un système n'est pas ordonnançable par un ordonnanceur optimal, alors l'algorithme est non optimal **[42]**.

2.3.2.5 Ordonnancement « centralisé »/ « distribué »

Dans un système temps réel distribué, il est possible que toutes les décisions d'ordonnancement soient prises par un algorithme localement en chaque nœud, dans ce cas on dit que l'ordonnancement est distribué.

De même, il est possible de prendre toutes les décisions d'ordonnancement pour tout le système, distribué ou non, d'un nœud central privilégié, dans ce cas on dit que l'ordonnancement est centralisé **[105].**

2.3.2.6 Ordonnancement « statique »/ « dynamique »

La plupart des politiques d'ordonnancement classiques traitent l'ordonnancement statique. Ce dernier réfère au fait que l'algorithme d'ordonnancement doit avoir une connaissance complète sur l'ensemble des tâches et des contraintes telle que : les deadlines, le temps de calcul, les contraintes de précédences et le temps d'activation des tâches. Toutes ces contraintes sont déterminées et assignées aux tâches du système avant leur activation et sont invariantes pendant l'exécution de l'application. Par exemple le contrôle en temps réel d'une simple expérience de laboratoire peut avoir un ensemble fixe de capteurs et actionneurs, un environnement bien défini et un ensemble de contraintes de traitements, le résultat d'ordonnancement est une solution unique et fixe tous le temps.

Par contre, l'algorithme d'ordonnancement dynamique possède une connaissance complète de l'ensemble des tâches actuellement actives, mais les nouvelles tâches arrivantes ne sont pas connues par l'algorithme dans le moment où il fait l'ordonnancement de l'ensemble courant. Donc le résultat d'ordonnancement est changé avec le temps à cause de changement des contraintes pendant l'exécution.

L'ordonnancement dynamique est nécessaire pour les systèmes temps réel telles que les applications de commande et de contrôle militaires **[32]**.

2.3.3 Ordonnancement de tâches indépendantes

Les tâches indépendantes se sont des tâches qui ne partagent pas des ressources entre eux, et qui ne possèdent pas des contraintes de précédences.

2.3.3.1 L'analyse d'ordonnançabilité (faisabilité)

Dans le cas de l'ordonnancement temps réel, l'objectif est de respecter l'échéance de chaque tâche en veillant à ce que chaque tâche peut terminer son exécution avant son échéance.

L'objectif de l'analyse d'ordonnançabilité est de déterminer avant de démarrer l'application (en phase de conception) si un ensemble de tâches répondant à certaines contraintes peuvent être ordonnancées (l'exécution de chaque tâche doit terminer avant l'échéance spécifiée) à l'aide d'un ordonnanceur spécifique. Un système est ordonnançable s'il existe un ordonnancement faisable pour ce système **[35]**.

Le test d'ordonnancement pour un algorithme d'ordonnancement donné est un algorithme qui prend en entrée une description d'un système de tâches et fournit en sortie une réponse pour savoir si le système est ordonnançable par l'algorithme d'ordonnancement donné. Un test d'ordonnançabilité est exact s'il identifie correctement toutes les tâches ordonnançables et non ordonnançables du système. Il est suffisant s'il identifie correctement toutes les tâches non ordonnançables du système.

Pour que tout algorithme d'ordonnancement soit utile pour les applications temps réel dur, il doit avoir au moins un test d'ordonnançabilité suffisant pour vérifier qu'un ensemble de travaux donnés est ordonnançable **[43]**.

2.3.3.2 Ordonnancement des tâches périodiques

2.3.3.2.1 Ordonnancement à priorités fixes (statiques)

Il existe de nombreux algorithmes d'attribution des priorités fixes :

✓ **priorité fixe au niveau des tâches** : Un algorithme à priorité fixe attribue la même priorité à toutes les tâches. En d'autres termes, la priorité de chaque tâche périodique est fixe par rapport aux autres tâches, les travaux héritent tout simplement de la priorité de leur tâche dès leur arrivée dans le système et cette priorité reste constante et ne varie pas au cours de la vie du système :(RM et DM) **[44]**.

- ***Rate monotonic (RM) :*** est un algorithme d'ordonnancement à priorités statiques (fixes) pour les tâches périodiques introduit par **[45]**, dans RM les priorités sont affectées aux tâches en fonction de leurs périodes, ç.à.d. que la tâches qui a la plus petite période a la plus grande priorité. Intuitivement l'utilité de ces priorités est que la tâche prioritaire est la première à exécuter **[46]**.

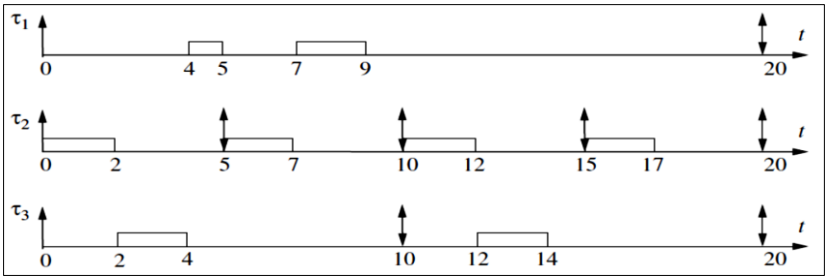


Figure 2.7 : Exemple de l'algorithme RM avec trois tâches τ1 (0, 3, 20, 20), τ2 (0, 2, 5, 5) et τ3 (0, 2, 10, 10) [47].

L'algorithme RM est optimal pour les algorithmes à priorités fixes et pour les systèmes à échéance sur requête (Di=Pi) et à départ simultané **[48]**.

Une condition nécessaire pour qu'un ensemble de tâches temps réel périodique soit ordonnançable par l'algorithme RM est :

$$\sum_{i=1}^{n} Ui = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Formule 2.1 : Condition nécessaire d'ordonnançabilité par RM.

Où : ***Ci*** est le WCET de la tâche *Ti*, ***Pi*** est la période de *Ti*, ***n*** est le nombre de tâches, ***Ui*** est le facteur d'utilisation de la tâche *Ti*, ***Ui= Ci/Pi***.

Une condition suffisante pour qu'un ensemble de tâches temps réel périodiques soit ordonnançable sous l'algorithme RM est **[105]** :

$$\sum_{i=1}^{n} u_i \leq n(2^{1/n} - 1)$$

Formule 2.2 : Condition suffisante d'ordonnançabilité par RM.

- ***Deadline monotonic (DM) :*** est un algorithme d'ordonnancement à priorités statiques (fixes) pour les tâches périodiques. Cet algorithme est proche de celui de RM à la différence que les priorités dans DM sont affectées aux tâches en fonction de leurs échéances relatives, c.à.d. que la tâche qui a la plus petite échéance à la plus grande priorité **[46]**.

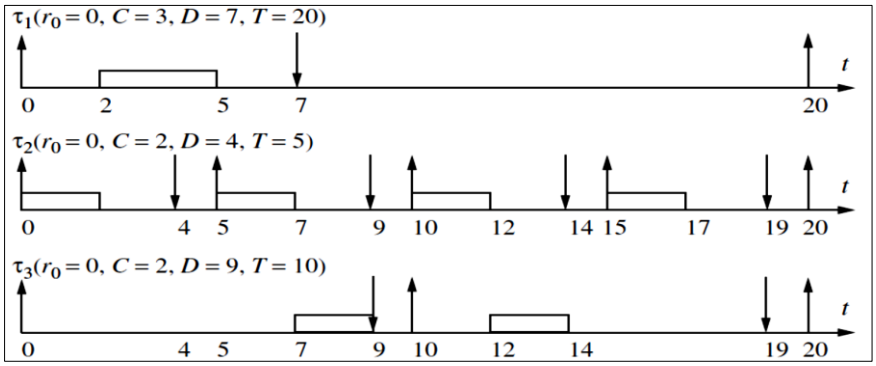


Figure 2.8 : Exemple de l'algorithme DM [47].

L'algorithme DM est optimal dans la classe des algorithmes à priorités fixes pour les systèmes à échéances contraintes et à départ simultané. On peut noter que DM et RM sont équivalents dans le cas des systèmes à échéance sur requête qui est un cas spécial des systèmes à échéance contraint, ç.à.d. que les deux algorithmes affectent des priorités aux tâches d'une manière identique **[48]**.

Une condition suffisante pour qu'un ensemble de tâches temps réel périodiques soit ordonnançable sous l'algorithme DM pour les systèmes de tâches à échéance contrainte est similaire à celle de RM :

$$\sum_{i=1}^{n} u_i \leq n(2^{1/n} - 1)$$

Formule 2.3 : Condition suffisante d'ordonnançabilité par DM.

Où : ***Ui*** est le facteur d'utilisation de la tâche *Ti, tel que :* ***Ui= Ci/Pi***, ***n*** est le nombre de tâches.

✓ **priorité fixe au niveau des travaux** : assignation des priorités fixes aux travaux. C.à.d. que les travaux d'une même tâche peuvent avoir des priorités différentes, mais la priorité reste inchangeable pour chaque travail dès leur arrivée dans le système. Ces algorithmes sont souvent appelés à priorité dynamique dans le sens où la priorité d'une tâche peut varier: (EDF) **[44]**.

- ***Earliest Deadline First (EDF):*** est un algorithme d'ordonnancement à priorités statiques (fixes) pour les tâches périodiques. les priorités dans EDF sont affectées aux tâches en fonction de leurs échéances absolue, c.à.d. que la priorité est plus forte lorsque l'échéance absolue de la tâche est plus petite (proche) **[46]**. Cet algorithme est utilisé généralement avec préemption.

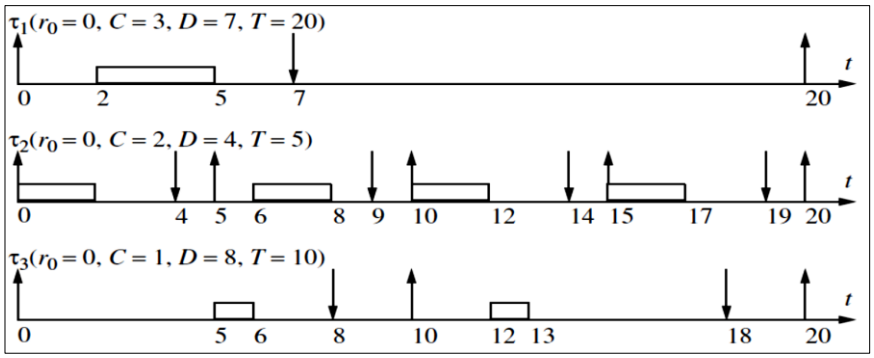


Figure 2.9 : Exemple de l'algorithme EDF [47].

L'algorithme EDF est optimal dans le sens de faisabilité : s'il existe un ordonnancement faisable pour un ensemble de tâches, alors l'algorithme EDF est capable de le trouver.

Il est important de noter qu'une condition d'ordonnançabilité nécessaire et suffisante est existée pour les tâches périodiques avec échéances sur requête. L'ensemble de tâches périodiques et à échéances sur requête est ordonnançable avec l'algorithme EDF si et seulement si le facteur d'utilisation du processeur est inférieure ou égale à 1.

$$\sum_{i=1}^{n} e_i / p_i = \sum_{i=1}^{n} u_i \leq 1$$

Formule 2.4 : Condition nécessaire et suffisante d'ordonnançabilité par EDF.

Où : ***Ui*** est le facteur d'utilisation de la tâche *Ti, tel que :* ***Ui= Ci/Pi***, ***n*** est le nombre de tâches **[47].**

EDF offre de meilleures performances que RM en terme d'ordonnançabilité dans pas mal des cas, Buttazzo en fait une étude comparative dans **[49].**

- Intérêts de l'ordonnancement à priorités fixes:

  - Un mécanisme simple.

  - S'implante naturellement dans les systèmes d'exploitation du marché.

- Inconvénients de l'ordonnancement à priorités fixes:

  - Hypothèses restrictives

  - Indépendance des tâches impérative pour l'utilisation des conditions de faisabilité.

  - Borne supérieure pour le facteur d'utilisation du processeur **[32]**.

2.3.3.2.2 Ordonnancement à priorités non fixes (dynamiques)

✓ **priorité dynamique au niveau des travaux :** un algorithme à priorités dynamiques affecte des priorités différentes à des instances individuelles de chaque tâche, ces priorités peuvent changer entre le temps d'activation et le temps de fin d'exécution. La priorité de la tâche change par rapport à celle des autres tâches à chaque nouvelle demande de la tâche qui doit être exécutée. C'est pourquoi ce type d'algorithme est dit « dynamique » **[50].**

- ***Least Laxity First (LLF):***

Dans cet algorithme, les priorités sont affectées aux tâches, selon leurs laxités, ç.à.d. que la tâche qui a la plus petite laxité sera exécutée avec une forte priorité. La laxité est la longueur de l'intervalle de temps maximum pendant lequel la tâche peut ne pas avoir le CPU sans rater leur échéance.

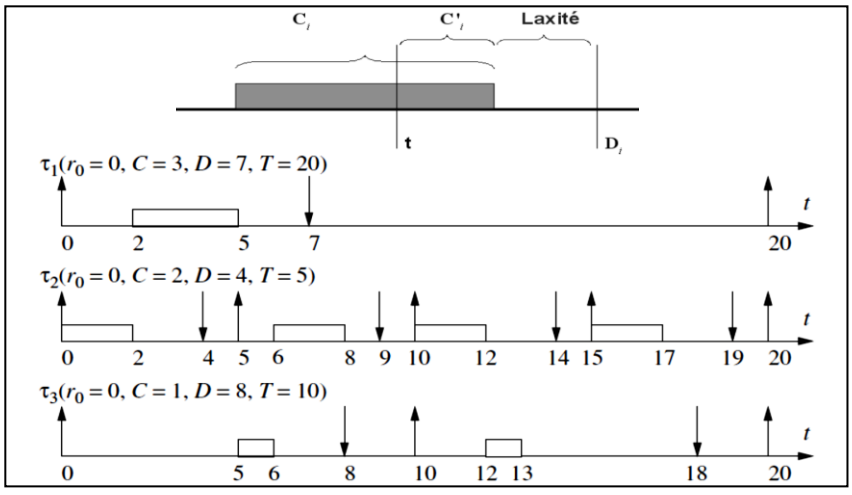


Figure 2.10 : Exemple de l'algorithme LLF [47].

Ainsi, lorsque la laxité des tâches est calculée seulement au temps d'arrivée, l'algorithme LLF est équivalent à l'algorithme EDF, mais si la laxité est calculée à chaque instant t,

plusieurs changements de contexte seront nécessaires **[47]**. Toutefois, au vue de ses inconvénients vis-à-vis du nombre de changements de contexte et du calcul des priorités nécessaire à chaque instant, LLF est rarement utilisé en pratique.

LLF est optimal dans la classe des algorithmes préemptifs pour des tâches périodiques indépendantes telles que Di≤Ti.

- Intérêts de l'ordonnancement à priorités dynamiques:

  - Simplicité de mise en œuvre.

  - Optimisation de l'usage des ressources.

  - Bien adapté aux tâches périodiques à courtes échéances.

- Inconvénients de l'ordonnancement à priorités dynamiques:

  - Indépendance des tâches impératives pour l'utilisation des conditions de faisabilité.

  - Instabilité en cas de surcharge (EDF).

  - Nombreux changements de contexte dans certains cas (LLF).

  - Difficilement implantable dans les systèmes d'exploitation actuels **[32]**.

2.3.3.3 Ordonnancement des tâches apériodiques

Les tâches apériodiques sont activées à des instants aléatoires, pour ordonnancer ce type de tâches on a deux cas : les tâches à contraintes strictes (maximiser le nombre de tâches qui peuvent être acceptées) et les tâches à contraintes relatives (minimiser le temps de réponse des tâches).

2.3.3.3.1 Ordonnancement des tâches apériodiques à contraintes strictes

On peut traiter ce cas en utilisant deux techniques principales avant l'ordonnancement : la technique du pseudo-période et la technique du test de garantie.

- **Pseudo-période :** Cette méthode consiste à doter les tâches apériodiques avec des pseudo-périodes pour transformer ces tâches en tâches périodiques. Ce pseudo-période est un intervalle de temps minimal entre deux occurrences successives d'un travail d'une tâche apériodique. Toutes les tâches ensuite seront de mêmes types (périodiques) et elle sans seront ordonnancer en utilisant les algorithmes RM ou ED. l'évaluation du pseudo-période est l'une des difficultés de cette méthode.

- **Test de garantie :** Cette technique, consiste à traiter les tâches apériodiques sans la prise en compte des hypothèses sur la fréquence d'arrivée des tâches. A chaque fois qu'une tâche apériodique est arrivée, une "routine" de garantie sert à tester dynamiquement si cette tâche peut s'exécuter en respectant ses contraintes temporels, celles des tâches périodiques et celles des autres tâches apériodiques précédemment acceptées et qui ne sont pas encore terminées. Si le test est positif, la tâche est acceptée **[51]**.

2.3.3.3.2 Ordonnancement des tâches apériodiques à contraintes relatives

Les techniques les plus importantes sont la technique de gestion en arrière-plan (background scheduling) et la technique du serveur de tâches.

- **Gestion en arrière-plan (background scheduling (BS)) :** C'est la méthode la plus simple pour gérer un ensemble de tâches apériodiques en présence de tâches périodiques, et cela quand il n'y a pas de tâches périodiques prêtes à exécuter. S'il existe plusieurs tâches apériodiques, le traitement se fait par la stratégie FIFO.

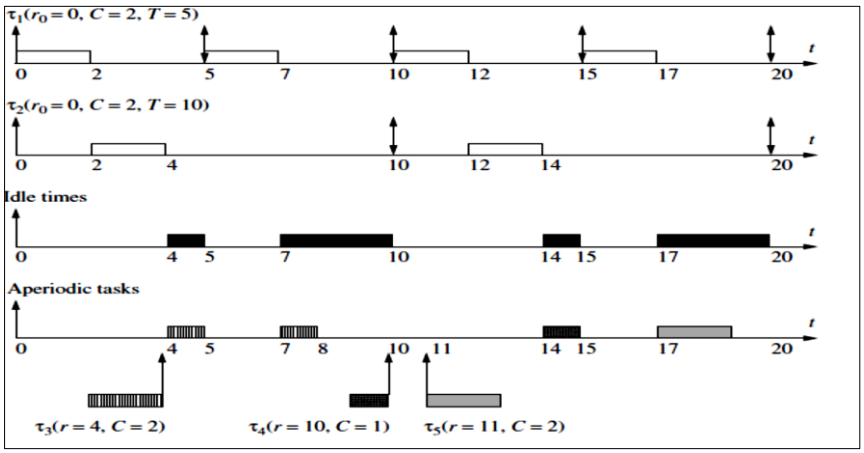


Figure 2.11 : Exemple de l'algorithme BS [47].

L'avantage majeur de cet algorithme est sa simplicité. Cependant le problème majeur est que, pour des charges élevées de tâches périodiques, le temps de réponse de tâches apériodiques peut-être trop long pour certaines applications. Pour cette raison, l'algorithme BS ne peut être adopté que lorsque les tâches apériodiques n'ont pas des contraintes de temps strictes et que la charge de tâches périodique n'est pas élevée **[47] [37]**.

- **Les serveurs de tâches:** Le temps de réponse moyen des tâches apériodiques peut être amélioré par rapport à l'ordonnancement par BS grâce à l'utilisation d'un serveur **[37]**, un

serveur est une tâche périodique dont le but est de servir les demandes apériodiques. Ce serveur est caractérisé par une période et un temps d'exécution constituant la capacité du serveur. Le serveur est souvent ordonnancé avec le même algorithme utilisé pour les tâches périodiques et, une fois qu'il est actif, il répond aux demandes apériodiques dans la limite de sa capacité. La mise en ordre des demandes apériodiques ne dépend pas de l'algorithme d'ordonnancement utilisé pour des tâches périodiques (peut être FIFO, en fonction d'échéances ou de temps d'exécution...).

Plusieurs types de serveurs ont été définis, le plus simple est appelé serveur par scrutation (polling server), d'autres types de serveurs (serveur différé, serveur à échange de priorité, serveur sporadique) qui améliorent la technique de serveur par scrutation et offrent une meilleure réactivité apériodique **[47]**.

✓ ***Serveur par scrutation (Polling Server (PS)) :*** Le PS est une tâche périodique τ_s avec une priorité Ps, une période Ts et une capacité Cs. D'un point de vue de l'analyse de faisabilité, il est absolument identique à une tâche périodique normale avec un WCET. Lorsque le serveur est déclenché, sa capacité est remise à sa valeur initiale. Si la file d'attente n'est pas vide, il commence à traiter les tâches apériodiques, jusqu'à l'épuisement de sa capacité. Si il n'y aucune tâche en attente à exécuter, le serveur se suspend jusqu'à la prochaine activation et sa capacité est récupérée par les tâches périodiques. Notant que si une tâche apériodique arrive juste après la suspension du serveur, elle doit attendre le début de la prochaine période, où la capacité du serveur est renouvelée à sa pleine valeur **[37] [47] [52]**.

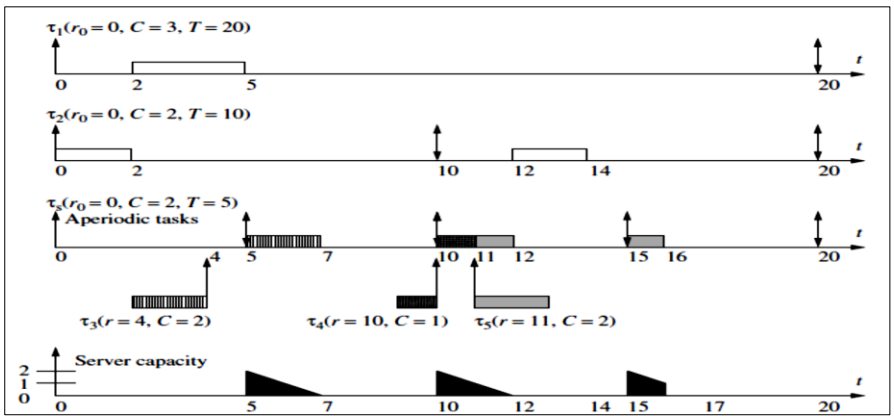


Figure 2.12 : Exemple de l'algorithme PS [47].

✓ ***Serveur sporadique (Sporadic Server (SS)) :*** L'algorithme SS est une technique, proposé par Sprunt, Sha, et Lehoczky **[53]**, qui permet l'amélioration du temps de

réponse moyen des tâches apériodiques sans dégrader le taux d'utilisation du processeur de l'ensemble de tâches périodiques. Le serveur sporadique conserve sa capacité jusqu'à ce qu'une tâche apériodique se produise. Ainsi, le serveur sporadique ne récupère pas sa capacité à sa pleine valeur au début de chaque nouvelle période, mais seulement après qu'elle a été consommée par les exécutions de tâches apériodiques **[37] [47]**.

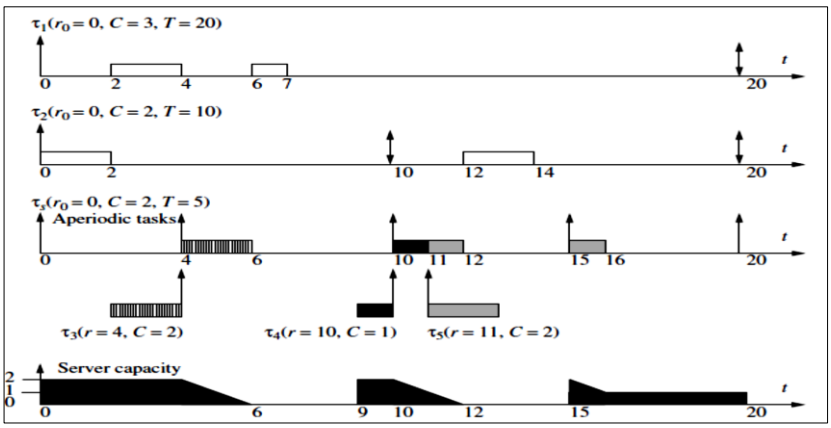


Figure 2.13 : Exemple de l'algorithme SS [47].

2.3.4 Ordonnancement des tâches dépendantes

Jusqu'à présent, nous avons étudié les systèmes qui sont constitués de tâches indépendantes. Mais dans de nombreux application, les tâches peuvent partager des ressources logicielles (comme des accès mémoire) ou matérielles (comme des capteurs). Le problème d'ordonnancement de tâches dépendantes est un problème NP complet, il consiste à satisfaire les contraintes suivantes :

- contraintes d'exclusions mutuelles pour la gestion des ressources critiques.
- contraintes de précédences entre les tâches.

2.3.4.1 Gestion des ressources

Dans le cas des ressources critiques au sein d'un système temps réel, le problème le plus important posé par l'accès à ces ressources est le blocage d'une tâche de grande priorité par une autre de moindre priorité pour une durée indéterminée, ce phénomène est appelé l'inversion de priorité. Ce dernier est dû lorsqu'une tâche est retardée par une autre tâche de plus faible priorité alors qu'elles ne partagent pas de ressources. Un autre problème existe à cause de la présence des ressources au sein des systèmes temps réel est l'interblocage, ce dernier peut se produire si une tâche nécessite une ressource possédée par une autre tâche,

mais cette dernière nécessite à son tour une autre ressource possédée par la première tâche, ce qui cause le blocage des deux tâches. Pour gérer ces différents problèmes posés à l'accès aux ressources, un ensemble de protocoles de gestion de ressources ont été introduit, le protocole à priorité héritée est l'un parmi eux, ce dernier est introduit pour résoudre le problème de l'inversion de priorités, il sert à modifier la priorité de la tâche qui provoque le blocage. Ce protocole n'empêche pas les interblocages. Un autre protocole nommé le protocole à priorité plafond est introduit à la fin des années 80 en reprenant les avantages de protocole précédant et en ajoutant l'avantage d'éviter l'interblocage **[54]**.

2.3.4.2 Contraintes de dépendances

Dans certaines applications, en plus du partage des ressources, certaines tâches ne peuvent s'exécuter que si d'autres tâches terminent les leurs. Dans ce cas les tâches ne peuvent pas être exécutées dans un ordre arbitraire, mais elles doivent respecter certaines relations de précédence définies à l'étape de conception. Ces relations de précédence sont généralement décrites par un graphe dirigé acyclique (DAG), où les tâches sont représentées par des nœuds et les relations de précédence par des flèches **[37]**.

Avant d'ordonnancer les tâches dépendantes, il faut tout d'abord les transformer en un ensemble de tâches indépendantes, ensuite on les ordonnancera par un algorithme classique. Cette transformation se fait par la modification des paramètres des tâches. Si on considère les tâches Ti, Tj telle que Ti → T j alors la règle de transformation doit respecter :

- rj >= ri
- $Prio_i > Prio_j$

Après la transformation, on valide l'ordonnançabilité (faire le test d'ordonnançabilité) selon des critères utilisés pour des tâches indépendantes, en fin on fait l'ordonnancement par un des algorithmes vu précédemment.

2.4 Ordonnancement temps réel multiprocesseur

L'ordonnancement des tâches temps réel a été beaucoup étudié en particulier sur les plates-formes monoprocesseur, où il y a exactement un seul processeur disponible partagé, et toutes les tâches dans le système sont requises pour s'exécuter sur ce processeur partagé. Depuis le début des années 80, le constat qu'il serait impossible d'augmenter indéfiniment la fréquence dans les plates-formes monoprocesseur a motivé l'intérêt pour les architectures à plusieurs processeurs. En effet, dans les plates-formes multiprocesseurs il y a plusieurs

processeurs disponibles sur lesquels les tâches peuvent s'exécuter. Toutefois, l'affectation optimale des tâches à des processeurs est dans presque tous les cas pratiques considérée comme un problème NP-difficile **[55]**.

L'ordonnancement multiprocesseur est un problème à deux dimensions :

- Allocation des tâches « organisation spatiale »: Sur quel processeur exécuter chacune des tâches.
- Ordonnancement des tâches « organisation temporelle »: quel est l'ordre d'exécution des tâches pour chaque processeur et pour tout le système **[30]**.

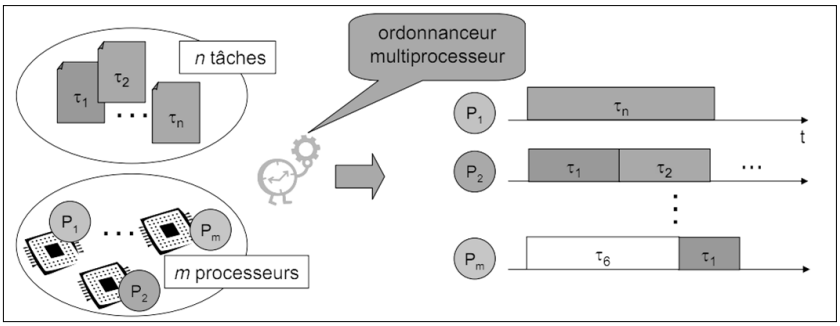


Figure 2.14 : Ordonnancement multiprocesseur [30].

2.4.1 Modèles des plates-formes multiprocesseurs

Les études sur l'ordonnancement multiprocesseur classifient les systèmes multiprocesseurs en trois catégories :

✓ ***Le modèle multiprocesseur identique :*** il suppose que chaque processeur dans la plate-forme est doté de capacités de traitement et de vitesse identiques. Plus précisément, chaque processeur est identique en termes d'architecture, de la taille du cache, de la vitesse, des E / S, de l'accès aux ressources et du temps d'accès à la mémoire partagée. Les plates-formes identiques sont donc des plates-formes homogènes **[56]**.

✓ ***Le modèle multiprocesseur uniforme :*** les processeurs ont des structures identiques, mais leurs capacités sont proportionnelles. Si un travail va s'exécuter sur un processeur de capacité de calcul s, la multiplication de WCET de ce travail par la capacité du processeur s, va donner s × t unités de travail. En fait, les modèles identiques sont des cas particuliers des modèles uniformes, dont les capacités de calcul de tous les processeurs sont égales **[55]**.

✓ ***Le modèle multiprocesseur hétérogène :*** ou non relié, chaque processeur possède sa propre capacité de traitement et ses propres caractéristiques internes, indépendantes des autres

processeurs. Dans ce cas, le WCET de chaque tâche devra être déterminé pour chacun des processeurs**.**

La présence de plusieurs processeurs sur lesquels peuvent s'exécuter les tâches pose les problèmes suivants :

- *le problème de placement des tâches* : sur quel(s) processeur(s) une tâche va-t-elle s'exécuter ?
- *le problème de la migration des tâches* : une tâche peut-elle changer de processeur pour s'exécuter ?
- *le problème de la gestion des ressources* : une ressource ne doit pas pouvoir être prise par deux tâches en même temps s'exécutant sur deux processeurs différents.
- *le problème de l'ordonnancement des tâches* : affectation des priorités.

Les premiers travaux sur l'ordonnancement multiprocesseur classaient les algorithmes en deux catégories :

- ✓ **Ordonnancement par partitionnement** : répartition a priori de ***n*** tâches sur ***m*** processeurs, chaque tâche est assignée à un seul processeur, sur lequel chacune de ses instances seront exécutées. L'ordonnancement est réalisé localement (monoprocesseur) sur chaque processeur et les tâches ne peuvent pas migrer d'un processeur à l'autre.
- ✓ **Ordonnancement global** : l'ordonnanceur s'applique sur l'ensemble des tâches et processeurs. L'ensemble des tâches sont stockés dans une seule file d'attente ordonnée par priorité; l'ordonnanceur global sélectionne pour l'exécution des tâches la plus prioritaire à partir de cette file d'attente. Les tâches peuvent donc migrer d'un processeur à un autre **[57]**.

D'autres travaux ajoutent deux autres algorithmes d'ordonnancement multiprocesseurs: l'ordonnancement semi-partitionné et l'ordonnancement par clustering.

2.4.2 Ordonnancement par partitionnement

Il consiste à partitionner l'ensemble des tâches de telle sorte que les tâches d'une même partition soient assignées au même processeur avant de débuter leur exécution et elles resteront sur le même processeur durant toute leur vie. Après l'allocation, les tâches seront ordonnancées selon un ordonnanceur monoprocesseur (par exemple, RM ou EDF). Les tâches ne sont plus autorisées à migrer d'un processeur à autre, d'où la transformation du problème

d'ordonnancement multiprocesseur en de nombreux problèmes d'ordonnancement monoprocesseur **[55]**.

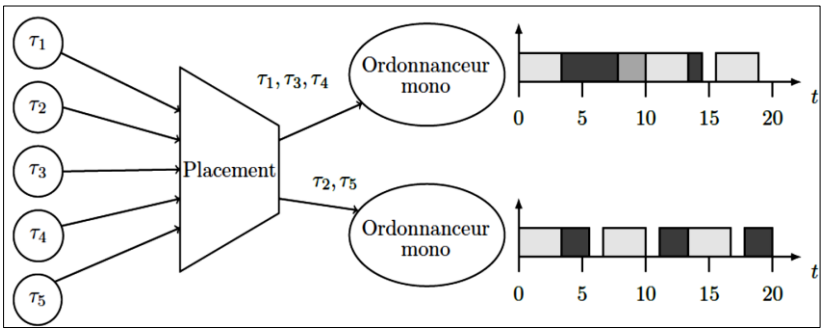


Figure 2.15 : Ordonnancement par partitionnement [58].

Le calcul du partitionnement des tâches pour allouer celles-ci sur les processeurs est un problème NP-Complet au sens fort. De nombreuses règles différentes ont été créées pour effectuer la distribution des tâches sur les différents processeurs. On considère que la tâche courante est τi. Les règles les plus populaires sont :

- ***First-Fit (FF) :*** affecter la tâche τi sur le premier processeur pouvant la contenir, dans l'ordre de son indice, tel que l'ordonnanceur local peut l'ordonnancer avec les tâches déjà affectées. Pour chaque tâche, on reprend la recherche depuis le début de la liste des processeurs.

- ***Best-Fit (BF)*** : affecter la tâche τi sur le premier processeur disposant de la plus grande capacité disponible et capable d'accepter la tâche tout en restant ordonnançable. En pratique, c'est équivalent au First-Fit mais en triant les processeurs par capacité.

- ***Worst-Fit (WF)*** : affecter la tâche τi sur le processeur ayant le plus petit niveau d'utilisation (petite capacité) pouvant la recevoir.

- ***Almost Worst-Fit (AWF)*** : c'est une variante de WF dans lequel le processeur choisi le second plus petit niveau d'utilisation.

- ***Next-Fit (NF)*** : seul le dernier processeur créé peut recevoir des tâches. Lorsqu'il n'est pas possible de placer la tâche τi, le processeur courant est fermé (il ne pourra plus recevoir de nouvelles tâches). Le processeur suivant devient alors le nouveau processeur courant. cet algorithme est similaire au First-Fit à la différence près que l'on ne reprend pas la recherche depuis le début à chaque tâche **[58]**.

2.4.3 Ordonnancement global

Il consiste à placer les tâches prêtes à s'exécuter dans une seule file d'attente qui est partagée entre tous les processeurs. Supposons qu'il existe k processeurs, à chaque instant, les k tâches plus prioritaires de la file d'attente sont sélectionnées pour l'exécution sur les k processeurs. Outre la préemption des tâches, on autorise aussi la migration de ces dernières ; ç.à.d. qu'une tâche qui commence son exécution sur un processeur Pj, peut être préemptée et reprendre son exécution ultérieurement sur un autre processeur **[55]**.

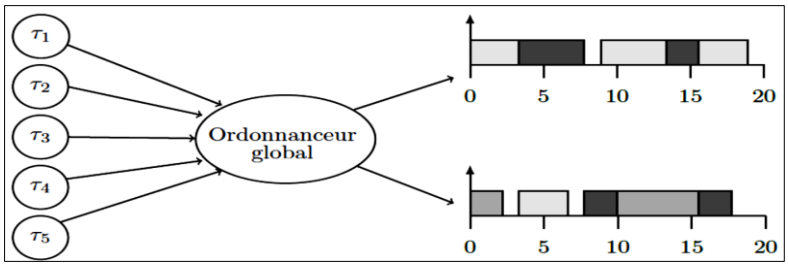


Figure 2.16 : Ordonnancement globale [58].

✓ ***Algorithme de type Pfair*** :

L'algorithme PFair (Proportionate Fair) est une technique d'ordonnancement globale prometteuse et particulièrement pour les systèmes multiprocesseurs. L'algorithme Pfair est actuellement la méthode optimale connu seulement pour l'ordonnancement des tâches périodiques sur un système multiprocesseur en temps polynomial. Sous l'ordonnancement Pfair, l'exécution des tâches est imposée explicitement à un taux régulier en divisant la tâche en une série de sous-tâches (portions) **[59]**.

Afin d'assurer le taux d'exécution de la tâche, des intervalles de tailles identiques sont réservés pour l'exécution de l'ensemble de sous-tâches appelés des fenêtres. Différentes sous-tâches d'une tâche peuvent s'exécuter sur des processeurs différents (la migration est donc autorisée) mais pas simultanément. Dans le cas particulier de l'ordonnancement multiprocesseur identique et pour des tâches périodiques à échéance sur requête, Pfair est une stratégie optimale **[60]**.

✓ **Avantages et inconvénients :**

La stratégie d'ordonnancement par partitionnement a plusieurs avantages par rapport à celle de la stratégie globale. Premièrement, une fois les tâches sont attribuées aux processeurs, le problème d'ordonnancement multiprocesseur temps réel devient un ensemble de problèmes d'ordonnancement monoprocesseurs temps réel simples, qui peuvent être résolus par des

algorithmes optimaux. Deuxièmement, pas de migration des tâches lors de l'exécution, donc le temps d'exécution réduit par rapport au cas de migration des tâches. Si l'ensemble de tâches est fixe et connu a priori, l'approche par partitionnement fournit des solutions appropriée.

La stratégie d'ordonnancement global a aussi des avantages par rapport à celle de la stratégie par partitionnement. Premièrement, si les tâches peuvent rejoindre et quitter le système lors de l'exécution, il peut être nécessaire de les réaffecter à des processeurs dans la stratégie par partitionnement. Deuxièmement, la stratégie par partitionnement ne peut pas produire des ordonnancements temps réel optimaux pour l'ensemble de tâches périodiques. Troisièmement, dans certaines architectures de processeurs embarqués de structure simple et sans cache, le surcoût de la migration a un moindre impact sur la performance. Enfin, l'ordonnancement global peut théoriquement contribuer à une meilleure compréhension sur les propriétés et les comportements des algorithmes d'ordonnancement temps réel pour les architectures multiprocesseurs **[20]**.

2.4.4 Ordonnancement semi-partitionné

Cet ordonnancement a été proposé par Anderson et al. **[61]** comme un compromis entre la stratégie par partitionnement et la stratégie globale. Ce dernier est une extension de l'ordonnancement partitionné en permettant à un petit nombre de tâches à migrer, ce qui améliore l'ordonnancement. Il consiste à répartir d'abord les tâches sur les différents processeurs en utilisant un algorithme (First Fit, best Fit,..), si une tâche ne peut être affectée à un seul processeur, alors au lieu de déclarer le système comme étant non ordonnançable par l'algorithme choisi, on va autoriser la migration de cette la tâche sur plusieurs processeurs. Ainsi donc, le système sera composé de 2 types de tâches :

- *des tâches non migrantes* : qui seront affectées à un processeur particulier et elles ne peuvent pas migrer d'un processeur à l'autre.
- *des tâches migrantes* : qui ne seront non plus affectées à un processeur particulier mais à un ensemble de processeurs, elles peuvent donc migrer d'un processeur à autre **[54]**.

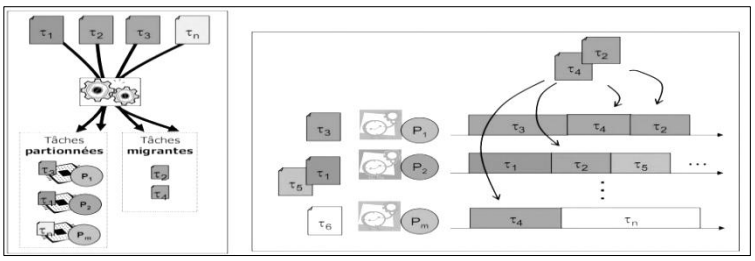


Figure 2.17 : Ordonnancement semi-partitionné [30].

2.4.5 Ordonnancement par clustering

Premièrement, les processeurs (physiquement ou virtuellement) sont organisés en groupes ou clusters, puis les tâches sont partitionnées entre ces clusters. On applique l'ordonnancement global à l'intérieur de chaque cluster.

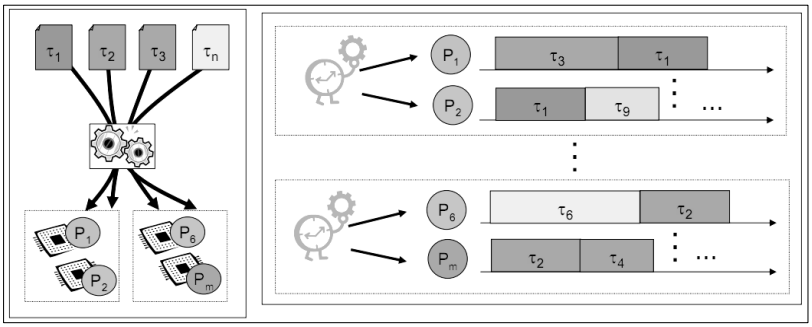


Figure 2.18 : Ordonnancement par clustering [30].

3. Conclusion

Le problème d'ordonnancement temps réel est assez général et doit être résolu dans de nombreux domaines, il consiste à organiser dans le temps la réalisation des tâches, compte tenu de contraintes temporelles et de contraintes portant sur l'utilisation et la disponibilité de ressources requises. Ce problème est très difficile pour les systèmes temps réel embarqués, et les recherches actuelles ont fourni des solutions efficaces seulement pour les petites classes de problèmes.

Un défi majeur pour les problèmes d'ordonnancement est leurs propriétés NP-difficiles, et il n'y a pas de méthodes de calcul efficaces pour trouver des solutions optimales aux problèmes d'ordonnancement, et en particulier aux problèmes d'ordonnancement multiprocesseurs, qui sont la classe la plus difficile de ce type de problèmes.

L'optimisation de l'allocation et de l'ordonnancement des tâches au sein des systèmes embarqués est donc devenue un domaine essentiel de la recherche. Plusieurs méthodes d'optimisation ont été développées dans la littérature, l'algorithme génétique est parmi ces méthodes. Afin de réaliser une meilleure exploration de l'espace des solutions, les principes de la mécanique quantique sont ajoutés aux algorithmes génétiques. Cette nouvelle technologie qui est appelée « algorithme génétique inspiré-quantique » a été démontré plus efficace, c'est ce qu'on va la définir dans le prochain chapitre.

Chapitre 3 Algorithmes génétiques inspiré-quantiques

Dans ce chapitre :

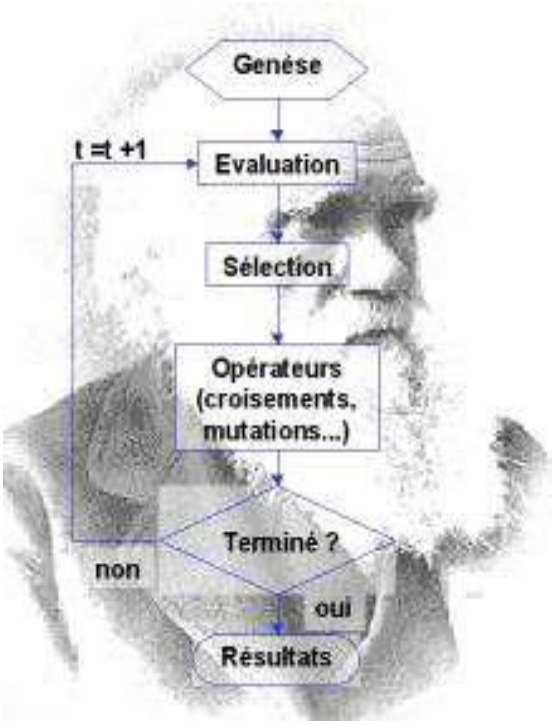
Résumé

Dans ce chapitre, nous nous intéressons dans la première partie à introduire d'abord les algorithmes génétiques classiques ainsi que leurs principales étapes, puis on va citer quelques travaux concernant l'application de ces derniers aux problèmes d'ordonnancement classiques et temps réel, en fin certains concepts de la mécanique quantique ont été associés aux algorithmes génétiques classiques afin d'améliorer leurs performances, cette nouvelle technologie qui est appelée les algorithmes génétiques inspirés-quantiques va être présentée.

1. Introduction

Les phénomènes physiques ou biologiques ont été la source d'inspiration dans de nombreux domaines, notamment dans le domaine de l'informatique. Ainsi les réseaux de neurones artificiels s'inspirent du fonctionnement du cerveau humain, l'algorithme de recuit simulé de la thermodynamique, et les algorithmes évolutionnaires (AEs) du concept de sélection naturelle élaboré par Charles Darwin en 1859 et de la génétique moderne.

Les algorithmes les plus connus des algorithmes évolutionnaires sont les algorithmes génétiques (AGs), ces derniers sont des techniques d'optimisations globales utilisées dans de nombreuses applications de la vie réelle. Ils manipulent un ensemble de plusieurs solutions simultanément et cherchent des solutions à des problèmes d'optimisation en évoluant les solutions de mieux en mieux. Pour augmenter l'efficacité, certains concepts de la mécanique quantique ont été associés aux algorithmes génétiques classiques afin d'améliorer leurs performances, cette nouvelle technologie est appelée les algorithmes génétiques inspirés-quantiques.



2. Optimisation combinatoire

L'optimisation combinatoire est une branche de l'optimisation en mathématiques appliquées et en informatique, c'est un outil très important pour résoudre les problèmes de la vie réelle souvent complexe.

Optimiser un problème combinatoire veut dire trouver une solution optimale, dans un temps raisonnable, parmi un grand nombre de solutions situées dans un espace de recherche fini ou dénombrable, en utilisant une fonction de coût à minimiser ou à maximiser et en respectant certaines contraintes.

Lorsque la solution est composée d'une seule valeur, on parle de l'optimisation mono-objectif, et lorsqu'elle est composée de plusieurs valeurs on parle alors de l'optimisation multi-objectifs, dans ce cas la solution optimale recherchée est un ensemble de bons compromis entre toutes les contraintes.

Les méthodes d'optimisation combinatoire peuvent être classées en deux catégories:

- *Méthodes exactes* : étant de complexité exponentielle, elles sont utilisées pour trouver au moins une solution optimale d'un problème par l'énumération de toutes les solutions possibles. Lorsque la taille de l'espace de recherche du problème devient importante, une énumération explicite s'avère irréalisable. Bien que les méthodes exactes trouvent des solutions optimales, leur grand inconvénient est l'explosion combinatoire, ce qui rend leur utilisation pratique difficile.

  Exemple de méthodes exactes : séparation/évaluation, programmation dynamique, diviser pour résoudre, programmation par contraintes, …

- *Méthodes approchées (heuristiques)* : fournissent une solution réalisable pour un problème d'optimisation NP-difficile, mais pas nécessairement optimale, dans un temps raisonnable. L'avantage principal de ces méthodes est qu'elles peuvent s'appliquer à n'importe quelle classe de problèmes

  Exemple de méthodes approchées : méthode de descente, recuit simulé, recherche tabou, colonies de fourmis, algorithmes génétiques, …

La difficulté d'un problème d'optimisation combinatoire est la taille de l'espace de recherche (explosion combinatoire).

3. Algorithmes génétiques classiques

« Ce qui donne à un individu sa valeur génétique, ce n'est pas la qualité propre de ses gènes. C'est qu'il n'a pas la même collection de gènes que les autres ».

François Gros Et François Jacob **[107]**.

3.1 Brève histoire sur les algorithmes évolutionnaires

Dans les années 1950 et 1960 plusieurs chercheurs en informatique ont étudié indépendamment les systèmes évolutionnaires avec l'idée que l'évolution peut être utilisée comme un outil d'optimisation pour certains problèmes. Le principe de base dans tous ces systèmes était de faire évoluer une population de solutions potentielles à un problème donné, en s'appuyant sur des techniques dérivées de la génétique et des mécanismes d'évolution de la nature : sélections, croisements, mutations..., dans l'espoir que les générations successives de solutions iront en s'améliorant.

Dans les années 1960, Rechenberg (1965, 1973) a introduit «les stratégies d'évolution», une méthode qu'il a utilisée pour optimiser les paramètres de valeurs réelles pour les dispositifs tels que les profils. Ensuite, cette idée a été développée par Schwefel (1975, 1977). Le domaine des stratégies d'évolution est resté un domaine de recherche actif.

Fogel, Owens, et Walsh (1966) ont développé «la programmation évolutionnaire », c'est une technique dans laquelle des solutions candidates d'un ensemble de tâches étaient représentés comme des machines d'états finis, qui sont évoluées par la mutation aléatoire de leurs diagrammes d'état-transition, puis en sélectionnant la meilleure.

Plusieurs autres personnes ont développé dans les années 1950 et 1960 les algorithmes évolutionnaires inspirés pour l'optimisation et l'apprentissage de la machine. Box (1957), Friedman (1959), Bledsoe (1961), Bremermann (1962), et Reed, Toombs, et Baricelli (1967) tous ont travaillé dans ce domaine.

Les algorithmes génétiques (AGs) ont été inventés par John Holland dans les années 1960 puis, ils ont développé par Holland et ses étudiants et ses collègues de l'Université du Michigan dans les années 1960 et 1970. A la différence aux stratégies d'évolution et de la programmation évolutionnaire, l'objectif initial de Holland n'était pas de concevoir des algorithmes pour résoudre des problèmes spécifiques, mais plutôt d'étudier formellement les phénomènes d'adaptation tels qu'ils se produisent dans la nature et de développer des moyens

dans lesquels les mécanismes d'adaptation naturelle pourraient être importés dans les systèmes informatiques.

Dans son livre « *Adaptation in Natural and Artificial Systems* » (1975), Holland a introduit le premier modèle formel des algorithmes génétiques. Il a présenté l'algorithme génétique comme une abstraction de l'évolution biologique, et il a posé les fondements théoriques passant du paradigme de l'évolution naturelle à celui de l'évolution artificielle, et qui constituent la base à tous les travaux sur ce type d'algorithmes. Les algorithmes génétiques constituent aujourd'hui un outil très largement utilisé dans un grand nombre de domaines différents **[62] [63]**.

3.2 Source d'inspiration

L'observation des phénomènes biologiques est une très riche source d'inspiration pour les informaticiens, et le concept d'algorithme génétique notamment est un excellent exemple. Posant la question suivante: pourquoi ils ont utilisé l'évolution comme une source d'inspiration pour résoudre certains problèmes informatiques? La réponse est que pour les chercheurs de calcul évolutionnaire, les mécanismes de l'évolution semblent bien adaptés à certains problèmes informatiques les plus urgents dans de nombreux domaines.

Les algorithmes génétiques se basent principalement sur une inspiration informatique simplifiée de la très célèbre théorie de Darwin (1859). En d'autres termes, faire une imitation au sein d'un programme de la capacité d'adaptation d'une population d'organismes vivants à son environnement à l'aide de mécanismes de sélection et d'héritage génétique.

Darwin a essayé de montrer par sa théorie que l'apparition d'espèces distinctes se fait par le biais de la sélection naturelle des variations individuelles. Il en résulte que les individus les plus adaptés tendent à survivre plus longtemps et à se reproduire plus aisément. Les êtres vivants ce sont donc, sous l'influence des contraintes extérieures, graduellement adaptés à leur milieu naturel à travers le processus de reproduction.

La transposition des principes d'évolution Darwiniste en technique d'optimisation stochastique globale est née indépendamment des deux côtés de l'océan Atlantique il y a une quarantaine d'années **[62] [64] [65]**.

3.3 Définition et concepts de base des algorithmes génétiques

3.3.1 Définition des algorithmes génétiques

Les algorithmes génétiques (AGs) sont des techniques d'optimisation globale utilisées dans de nombreuses applications da la vie réelle. Ce sont les algorithmes les plus connus des algorithmes évolutionnaires, qui reposent principalement sur la compétition entre les individus d'une population : les individus qui sont mieux adaptés aux contraintes survivent alors les autres périssent en laissant par conséquent leurs places à d'autres nouveaux formés par la descendance des premiers. Donc leur fonctionnement est très simple, avec ces algorithmes la recherche des solutions pour les problèmes d'optimisation est effectuée en évoluant les solutions de mieux en mieux.

L'objectif des algorithmes génétiques est d'optimiser une fonction prédéfinie qui traduit la capacité d'adaptation des individus au problème considéré et mesure la qualité des solutions, appelée « fonction objective ou fitness ». Pour cela, ils commencent par un ensemble de solutions possibles (candidates) pour l'application désirée, appelés « population, chromosomes ou individus » choisis soit aléatoirement, soit par des heuristiques ou par des méthodes spécifiques au problème, soit encore par mélange de solutions aléatoires et heuristiques. Cette population doit être suffisamment diversifiée pour que l'algorithme ne reste pas bloqué dans un optimum local. C'est ce qui se produit lorsque trop d'individus sont semblables.

Chaque chromosome de la population qui représente un codage d'une solution du problème donné est constitué d'un ensemble d'élément appelé « gène », qui peuvent prendre plusieurs valeurs (binaire, réel, ..).

3.3.2 Terminologie et concepts de base des algorithmes génétiques

Avant d'entamer le principe de base des algorithmes génétiques nous devons également donner quelques vocabulaires:

- ✓ **Chromosome** : En biologie, il est défini comme le support de l'information génétique nécessaire à la construction et au fonctionnement d'un organisme. Dans le cadre des AG, il se réfère généralement à une solution candidate à un problème donné, souvent codée comme une chaîne de bits.

- ✓ **Gène** : En biologie, il représente un segment d'ADN porté par les chromosomes et qui détermine la transmission des caractéristiques héréditaires des êtres vivants. Pour un AG,

chaque chromosome est divisé en un ensemble d'unités le constituant dites gènes. Ces derniers, sont des bits individuels adjacents sert à coder la solution candidate.

✓ **Génotype** : En biologie, c'est l'ensemble des gènes qui caractérisent un être vivant, dans les algorithmes génétiques, l'ensemble des chaînes est appelé structure.

✓ **Phénotype** : En biologie, c'est l'ensemble des caractères manifestes spécifiques qui constituent l'expression vivante du patrimoine génétique. Dans les algorithmes génétiques, les structures décodées forment un ensemble de paramètres donné, ou une solution ou un point dans l'espace des solutions.

✓ **Allèle** : En biologie, représente une des formes que peut prendre un gène ou une séquence d'ADN à un endroit précis. Dans les algorithmes génétiques, l'allèle est également appelé valeur caractéristique.

✓ **Locus** : représente la place qu'un gène occupe sur le chromosome.

✓ **Individu** : En biologie, c'est un spécimen représentatif d'une espèce, Pour un algorithmes génétiques, il est réduit à un chromosome et on l'appelle donc chromosome ou individu pour désigner un même objet.

✓ **Population** : c'est un ensemble d'êtres vivants appartenant à la même espèce et vivant dans le même lieu, c'est l'ensemble de chromosomes dans les algorithmes génétiques.

✓ **Parents** : En biologie, afin d'assurer la continuité de la vie, les individus peuvent se reproduire en créant de nouveaux individus formant une nouvelle génération. Dans le cadre d'un AG, les parents correspondent aux meilleurs individus ce qui donne naissance à de nouveaux individus descendants afin de former une nouvelle génération.

Les autres concepts comme le croisement et la mutation seront discutés dans la partie suivante **[62] [65]**.

Le concept de base des algorithmes génétique repose sur une boucle qui enchaîne des étapes de sélections et des étapes de reproductions. Ils consistent dans un premier temps à effectuer la sélection des meilleurs individus de la population, en se basant sur la fonction objective. Les nouveaux individus crées sont appelés « parents », ce sont ceux autorisés à se reproduire, puis avec l'utilisation des opérateurs génétiques tel que le croisement et la mutation plusieurs autres individus sont créés, ce sont « les fils ». Les nouveaux individus sont évalués grâce à la fonction fitness et ajoutés à la population initiale, en remplaçant les mauvaises solutions. Ce processus est répété pendant une certaine succession d'itérations

appelées « générations » afin de générer à chaque fois de nouveaux individus plus performants que leurs prédécesseurs, jusqu'à ce que le critère d'arrêt soit vérifié. La meilleure solution est ensuite tirée, c'est la dernière solution trouvée **[65] [66] [67]**.

3.3.3 Etapes de l'algorithme génétique

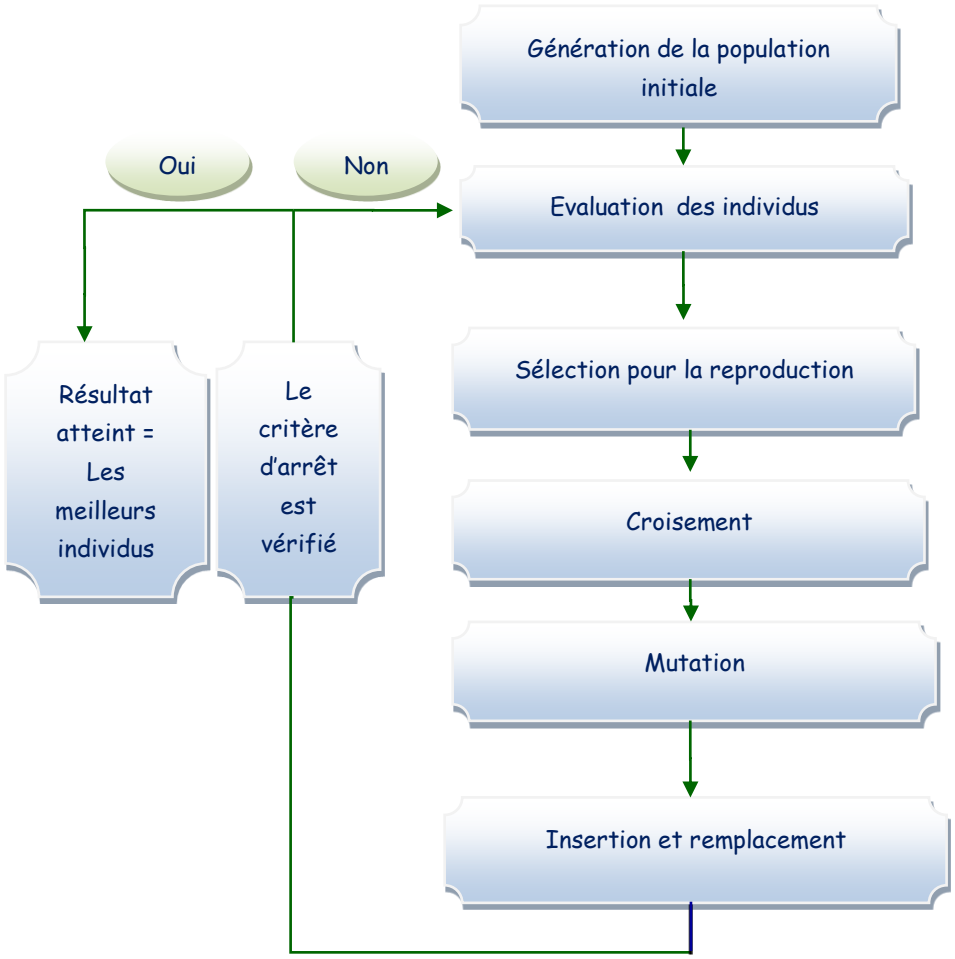


Figure 3.1 : Fonctionnement général de l'algorithme génétique.

Avant d'expliquer chacune des étapes figurant dans la figure ci-dessus, nous devons d'abord trouver une manière de coder chaque allèle différent de façon unique selon le problème donné.

3.3.3.1 Codage des chromosomes

La première étape de la mise en œuvre d'un algorithme génétique consiste à construire « un codage » des solutions potentielles aux problèmes. La structure de codage la plus utilisée pour les algorithmes génétiques est le codage en vecteurs. L'efficacité de ces algorithmes va donc dépendre du choix du codage des individus. Parmi les techniques les plus fréquemment utilisées pour coder les individus, on distingue :

- **Le codage binaire :** initialement retenu par John Holland **[63]**. C'est le codage le plus utilisé, chaque individu de la population est représenté sous forme de vecteur de bits 0-1 de longueur fixe. Ce codage binaire a permis de résoudre beaucoup de problèmes. Les raisons pour lesquelles ce type de codage est le plus utilisé sont tout d'abord historiques. En effet, lors des premiers travaux de Holland, les théories ont été élaborées en se basant sur ce type de codage.

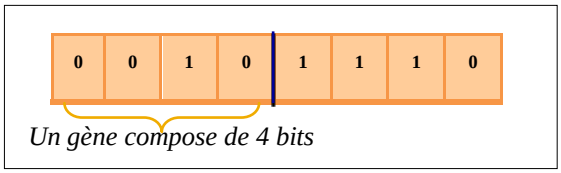


Figure 3.2 : Structure d'un chromosome en codage binaire.

Il existe cependant au moins un côté négatif à ce type de codage qui entraine certains problèmes. En effet, ce codage est souvent peu naturel par rapport à un problème donné.

- **Le codage réel :** Il y a plusieurs raisons pour lesquelles il est préférable de coder l'algorithme génétique en nombres réels. Une raison est que le codage réel permet une plus grande marge de valeurs possibles des paramètres.

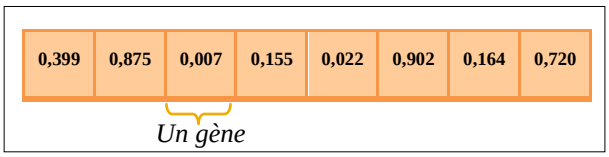


Figure 3.3 : Structure d'un chromosome en codage réel.

- **Le codage de gray :** En codage binaire deux éléments voisins (en distance de Hamming) ne codent pas toujours deux solutions proches dans l'espace de recherche. Cet inconvénient peut être évité en utilisant un « codage de Gray » : le codage de Gray est un codage qui redéfinit les nombres binaires de sorte que la distance de Hamming entre deux nombres consécutifs (voisins dans l'espace de recherche) soit de 1, ç.à.d. un seul bit diffère **[62] [65] [68]**.

3.3.3.2 Initialisation de la population

La première étape à partir de laquelle l'AG démarre son exécution est de générer la population initiale. Le choix de cette population est important car il peut rendre plus ou moins rapide la convergence vers l'optimum global, donc il est essentiel qu'elle soit répartie sur tout le domaine de recherche. Pour cela, l'espace de recherche des individus doit être fixé, ensuite des fonctions pour le codage et le décodage des individus doivent être établies. La façon la plus simple et la plus naturelle est d'initialiser aléatoirement les valeurs des gènes des individus initiaux. La taille de la population et la manière par laquelle les individus sont choisis sont les principales questions à considérer. La taille de la population qui désigne le nombre des individus dans la population (dans une génération), soumit aux plusieurs facteurs à savoir le type de codage ainsi que le problème à optimiser. Si la taille de la population est trop grande, le temps de calcul augmente et demande un espace mémoire important. Par contre, si la taille est très petite, la solution obtenue n'est pas satisfaisante. Il faut donc attacher une attention particulière aux choix de la taille de la population **[69] [70]**.

3.3.3.3 Evaluation des individus

« Il y a toujours quelques individus que le hasard isole, ou que la génétique favorise ».

Ronald Wright **[107]**.

Dans les algorithmes génétiques, chaque individu de la population doit être évalué pour quantifier sa qualité ou son fitness, ce qui offre la possibilité de comparer les individus. Pour calculer le coût d'un individu, on utilise une fonction dite d'adaptation en fonction du critère que l'on désire optimiser. Le résultat fournit par cette fonction va permettre de ne choisir que les individus ayant le meilleur coût par rapport aux autres individus de la population. Donc cette fonction permet de s'assurer que les individus performants seront conservés pour les prochaines générations, alors que les individus peu adaptés seront progressivement éliminés de la population **[65] [67] [69]**.

3.3.3.4 Sélection

Après avoir réalisé l'évaluation des individus, l'opérateur de sélection sert à choisir les individus de la population selon leurs valeurs d'adaptation pour la reproduction. On distingue deux étapes de sélection : l'une pour déterminer quels seront les individus, parmi les individus de la population totale qui vont se reproduire entre eux, et former des enfants « ce sont les parents ». Et une seconde étape pour déterminer quels sont ceux qui vont survivre **[62] [66]**.

Un nombre important de principes de sélection existe dans la littérature, nous présentons ici les quatre les plus utilisés :

- **Sélection par roulette biaisée:** La méthode RWS (Roulette Wheel Selection) consiste à associer un secteur d'une roue pour chaque individu, de telle sorte que l'angle du secteur est proportionnel à la qualité de l'individu qu'il représente. On lance la roue autant de fois et on obtient à chaque fois un individu correspondant au secteur désigné par une sorte de "curseur". Par cette méthode, les meilleurs individus ont plus de chance d'être choisis.

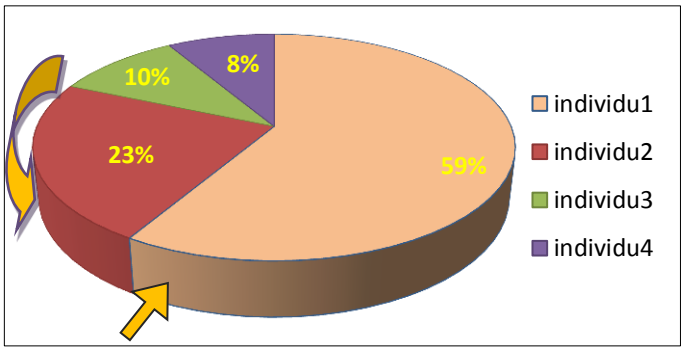


Figure 3.4 : Tirage par roulette.

Malgré qu'une certaine diversité est maintenue avec cette technique, mais les mauvais individus peuvent être choisis autant de fois pour passer à la génération suivante et de même, il se peut qu'aucun des bons individus ne soit sélectionné.

- **Sélection par élitisme :** introduit par Kenneth De Jong en 1975, elle permet de trier les individus de la population selon leurs valeurs d'adaptation en mettant en avant les meilleurs individus. Ensuite, les premiers individus de la moitié supérieure de la population seront sélectionnés pour se reproduire. Cette méthode a l'avantage de permettre une convergence (plus) rapide des solutions, mais au détriment de la diversité des individus.

- **Sélection par tournoi :** elle consiste à choisir aléatoirement un petit nombre d'individus (généralement deux) de la population « P » pour se combattre, le gagnant qui est de meilleure qualité est autorisé à reproduire avec une probabilité p comprise entre 0.5 et 1. Le processus est répété n fois de manière à obtenir les n individus de « P' » qui serviront de parents.

- **Sélection par rang :** elle consiste à ranger les individus de la population en fonction de leurs qualités et à attribuer un rang à chacun, de telle sorte que les individus de moins bonne qualité obtiennent un rang faible (à partir de 1). ensuite, implémenter une roulette

basée sur ces rangs. L'angle de chaque secteur de la roue sera proportionnel au rang de l'individu qu'il représente **[62] [108]**.

Chromosomes	1	2	3	4	5	6	Total
Probabilités initiales	85%	5%	1%	4%	3%	2%	100%
Rang	6	5	1	4	3	2	21
Probabilités finales	29%	24%	5%	19%	14%	9%	100%

Tableau 3.1 : Exemple de sélection par rang pour 6 chromosomes.

3.3.3.5 Croisement

Après avoir sélectionné les parents, ils peuvent commencer à se reproduire, pour construire une nouvelle génération. L'idée derrière l'opérateur de croisement est que le nouvel individu peut être meilleur que les deux parents s'il hérite les meilleures caractéristiques de ses parents. On pourrait dire que la principale caractéristique d'un AG est l'utilisation du cet opérateur, ce qui permet d'enrichir la diversité de la population permettant ainsi l'exploration de l'espace de recherche. Un croisement (crossover) est effectué avec deux parents et génère deux enfants. Il est appliqué avec une probabilité P_{cross}, appelée probabilité de croisement. Les gènes des parents sont alors copiés et recombinés de façon à former deux enfants possédant des caractéristiques issues des deux parents. Plusieurs opérateurs de croisement sont proposés :

- **Croisement en un point :** l'opérateur de croisement sélectionne au hasard un point de coupure pour les deux parents, puis il copie tout ce qui est avant ce point du premier parent, et tout ce qui est après le point de l'autre parent dans un nouveau chromosome créant ainsi le premier enfant. On inverse l'opération pour créer le deuxième enfant. La figure ci-dessous présente un exemple illustratif de ce type de croisement :

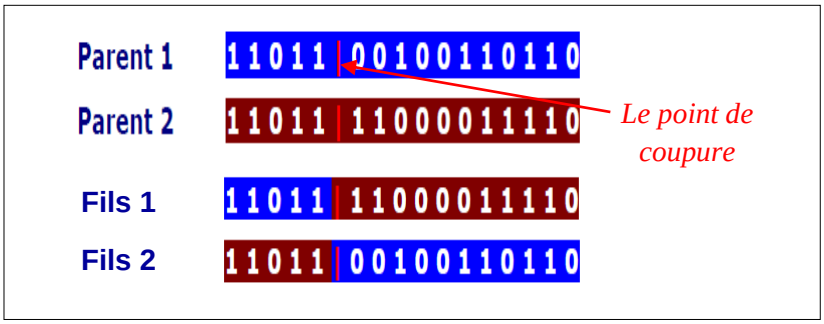


Figure 3.5 : Croisement en un point de deux chromosomes.

- **Croisement en deux points :** dans ce cas, on choisit au hasard deux points de croisement pour dissocier chaque parent en 3 fragments, puis on échange les fragments situés entre ces deux points. Cette définition peut également se généraliser pour effectuer un croisement à *n* points.

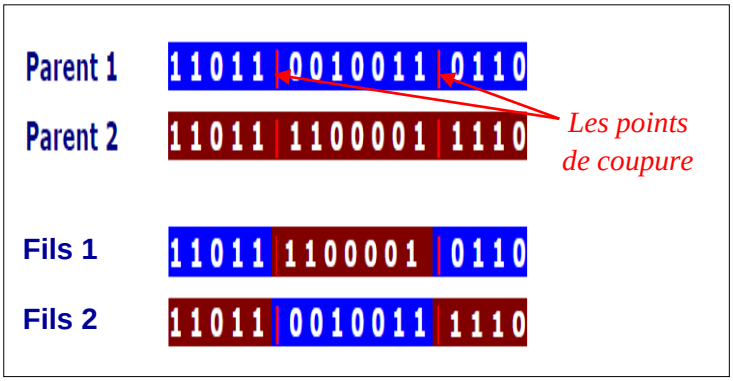


Figure 3.6 : Croisement en deux points de deux chromosomes.

- **Croisement uniforme :** cet opérateur utilise un masque de croisement binaire pour chaque couple d'individus, ce dernier est de même longueur que les individus. un «1 » dans la position « i » désigne un échange entre les parents dans la même position, « 0 » désigne pas d'échange.

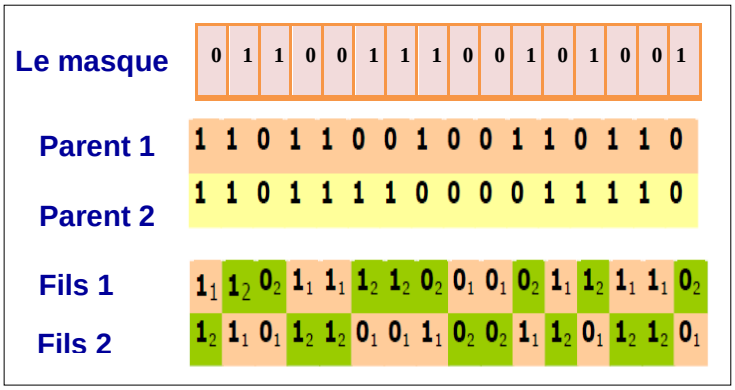


Figure 3.7 : Croisement uniforme de deux chromosomes.

3.3.3.6 Mutation

Après que le croisement est effectué, l'opérateur de mutation est utilisé pour trouver de nouveaux points dans l'espace de recherche, il consiste à changer la valeur allélique d'un gène choisi de façon aléatoire avec une probabilité *Pm* très faible, généralement comprise entre 0.01 et 0.001. Prenant l'exemple suivant : dans le cas d'un codage binaire, si *Pm* est la probabilité de mutation, $0 \leq r \leq 1$ un nombre tiré au hasard et que l'on ait $r < Pm$, alors le chromosome va subir une mutation. Un site de mutation est choisi aléatoirement, et le bit correspondant est inversé. Il existe aussi une variante où plusieurs bits peuvent muter au sein d'un même chromosome.

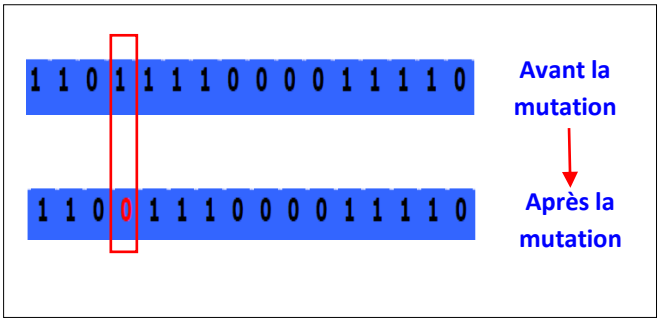


Figure 3.8 : Opérateur de mutation.

Cet opérateur joue un rôle très important, leur but consiste à introduit du bruit, et en particulier de nouveaux allèles, au sein de la population. Ceci est utile à s'échapper des minima locaux car il aide à explorer de nouvelles régions de l'espace de solution, ce qui permet d'introduire et de maintenir la diversité.

On peut conclure que le croisement sert à explorer globalement l'espace de recherche, tandis que la mutation sert à optimiser des solutions déjà rapprochées **[62] [65] [66] [69] [71]**.

3.3.3.7 Insertion et remplacement

Après les croisements et les mutations, les coûts liés aux enfants et aux individus mutés sont calculés pour sélectionner ceux qui vont continuer à participer à l'amélioration de notre population. On utilise ensuite une méthode d'insertion pour générer la nouvelle population. Plusieurs stratégies ont été présentées :

- **remplacement total** : Les enfants remplacent automatiquement les parents sans tenir compte de leurs performances respectives.
- **remplacement partiel** : On garde au moins l'individu possédant les meilleures performances d'une génération à la suivante.

3.3.3.8 Critère d'arrêt

En l'absence de toute information sur la valeur cible de l'optimum recherché, différents critères d'arrêt de l'algorithme peuvent être considérés :

- Le nombre de générations que l'on souhaite exécuter peut être fixé a priori selon le problème à résoudre (atteindre un nombre maximal d'itérations). C'est ce que l'on est tenté de faire lorsque l'on doit trouver une solution dans un temps déterminé.
- L'algorithme peut être arrêté lorsque la population n'évolue plus rapidement (état de stagnation).

En effet, la bonne gestion de l'arrêt de l'évolution contribue de façon importante à l'efficacité des algorithmes génétiques **[64] [72]**.

Le processus d'évolution des individus est répété jusqu'à ce que le critère d'arrêt soit vérifié.

4. Algorithmes génétiques appliqués aux problèmes d'ordonnancement

Le problème d'ordonnancement multiprocesseur est un problème bien connu, dont plusieurs solutions ont été proposées à travers les années. Ce problème nécessite l'allocation d'un graphe acyclique orienté (DAG), qui décrit un ensemble de tâches et leurs relations de précédences de données, sur un système distribué. En fait, la littérature est très riche et plusieurs heuristiques ont été développées pour résoudre ce problème. Plus récemment, avec la croissance de puissance de calcul, et l'augmentation de l'importance des systèmes embarqués, les algorithmes génétiques ont suscité un intérêt considérable dans la résolution du problème d'ordonnancement multiprocesseur.

Dans cette partie, nous allons scinder les travaux les plus pertinents portants sur l'application des algorithmes génétiques aux problèmes d'ordonnancement multiprocesseurs en deux classes : l'ordonnancement multiprocesseur classique et l'ordonnancement multiprocesseur temps réel.

4.1 Ordonnancement multiprocesseur classique

Les premiers travaux qui appliquent les algorithmes génétiques aux problèmes d'ordonnancements multiprocesseurs ont commencé depuis le début des années 90s. La distinction principale entre eux étant la représentation de chromosomes (la manière de représenter les solutions). La structure de chromosome et les restrictions imposées à leur représentation, jouent un rôle important pour la complexité des opérateurs génétiques ainsi que la convergence de l'algorithme vers un ordonnancement optimal.

Dans le contexte des problèmes d'ordonnancements classiques, Hou et al. **[73]** ont proposé un algorithme génétique, dont la solution est représentée sous forme de plusieurs listes, de telle sorte que chaque liste représente l'ensemble de tâches qui seront exécutées par un processeur, et l'ordre des tâches dans la liste indique l'ordre d'exécution.

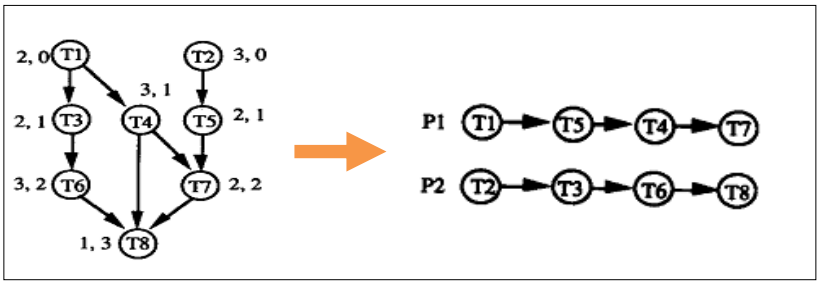


Figure 3.9 : Représentation de la solution par deux listes.

Cette représentation préserve la relation de précédence entre les tâches exécutées dans le même processeur et ignore cette relation dans des processeurs différents.

Pour générer la population initiale, chaque solution générée doit être légale, c.-à-d. vérifie la condition d'ordre d'exécution des tâches. L'algorithme qui a été proposé pour la génération de la population initiale repose principalement sur la hauteur de chaque tâche dans le DAG. La fonction objective cherche à minimiser le temps de fin d'exécution d'une solution. Le croisement se fait par l'échange de portion entre deux solutions choisi. Pour choisir les sites de croisement, deux conditions doivent être vérifiées :

1- La hauteur des tâches à côté des sites de croisement doit être différente.

2- Toutes les tâches devant les sites de croisement doivent être de même hauteur.

L'opération de mutation se fait par l'échange aléatoire de deus tâches de même hauteur.

Wang et Korfhage **[74]** ont proposé aussi un algorithme génétique mais avec l'utilisation de matrice bidimensionnelle pour coder les solutions. Ils ont utilisé un codage binaire d'une matrice qui sert à enregistrer l'affectation et l'ordre d'exécution des tâches sur chaque processeur. Mais, les règles qui sont utilisées pour se diriger à une forme d'une solution valide ne sont pas prises en compte par les opérateurs de croisement et de mutation et, par conséquent, il y a une possibilité de produire une solution invalide.

Bien que les opérations de réparation soient mises en œuvre pour corriger ces solutions, ces opérations consomment beaucoup de temps ce qui influe sur l'optimisation.

Dans les deux algorithmes cités précédemment aucune connaissance appropriée sur le problème est prise en considération et la recherche est effectuée seulement sur un sous ensemble de l'espace de recherche. Et bien que les restrictions imposées à la représentation de chaîne empêchent la production de solutions invalides Corrêa, Ferreira, et Rebreyend **[75]** ont prouvé que cette représentation ne peut pas exprimer la gamme complète de solutions possibles. Par conséquent, il peut être impossible pour l'algorithme génétique à converger vers une solution optimale quel que soit la période consacrée à l'optimisation. Dans **[75]**, les

auteurs ont amélioré le travail de Hou et al. **[73]** par l'utilisation de l'algorithme génétique de recherche complète (the full-search genetic algorithm FSG), qui utilise une représentation de chaîne capable d'explorer tous l'espace de solutions possibles. Bien que FSG excède de manière significative son prédécesseur, une liste supplémentaire qui exploite des connaissances sur le problème d'ordonnancement est ajoutée à la FSG pour améliorer encore la qualité de ses solutions. Ces connaissances sont représentées par l'utilisation des listes heuristiques dans les opérateurs de croisement et de mutation ce qui conduit à une amélioration de l'AG par rapport au AG pure de **[73]** et **[74]**. L'inconvénient est que le temps d'exécution de l'algorithme peut être beaucoup plus important par rapport à ce de l'AG pure.

Une autre représentation différente du chromosome a été proposée par M.K Dhodhi, I.Ahmad et Ishfaq Ahmed **[76]** pour minimiser le temps de fin d'exécution. Un gène *i* dans le chromosome représente la priorité de nœud *i* dans le DAG.

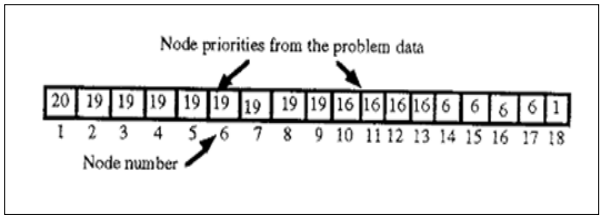


Figure 3.10 : La représentation d'un chromosome [76].

La priorité de nœud *i* du premier chromosome de la population initiale est le poids total de plus long chemin en partant de *i* jusqu'au dernier nœud. Ainsi que la priorité des nœuds dans les autres chromosomes de la population initiale est générée par une perturbation aléatoire des priorités du premier chromosome comme suit :

$$(W_r)^i = (W_{ro})^i + Uniform\ (-\eta, \eta\ )$$

Formule 3.1 : Perturbation aléatoire des priorités du premier chromosome.

Tel que :$(W_{ro})^i$ la priorité du nœud *i* dans le premier chromosome.

(-η, η) est un nombre aléatoire généré entre –η et η / $\eta = Max\ ((W_{ro})^i\ \forall\ i\ )$

Les opérations de croisement et de mutation sont appliquées de la même manière que dans les algorithmes génétiques de base.

De même, M.Rinehart, V.Kianzad, et S.S. Bhattacharyya **[77]** ont développé un algorithme génétique modulaire pour l'ordonnancement des tâches multiprocesseurs qui se base sur la représentation bi-chromosomique (Bi-Chromosomal Genetic Algorithm (BCGA)). Le but de la fonction objective de cet algorithme est de minimiser le makespan d'un graphe de

tâches particulier. La représentation de la solution dans BCGA consiste à décomposer la solution en deux structures indépendantes: une matrice d'allocation des tâches ; qui enregistre l'allocation des tâches aux processeurs, et un vecteur trié topologiquement représente l'ordre d'exécution des tâches d'une solution particulière. Cette représentation implique une modularisation du contenu de l'information, ce qui permet de faciliter les opérateurs génétiques pour la manipulation de solutions. La figure5 illustre la représentation bi-chromosomique de BCGA.

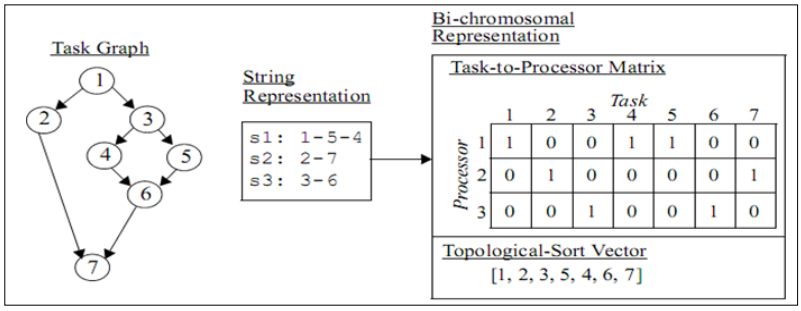


Figure 3.11 : Illustration de la représentation bi chromosomique.

La population initiale dans BCGA est générée de façon aléatoire ; c'est-à-dire générer aléatoirement les deux structures : la matrice d'allocation de tâches et le vecteur d'ordre d'exécution de tâches. L'application des opérateurs génétiques dans le BCGA est un peu différente. En effet, le croisement dans le BCGA est un processus en deux étapes. En premier lieu, l'une des deux structures de données dans BCGA est choisie au hasard (par exemple la matrice d'allocation des tâches) pour la manipulation, puis l'opération de croisement est appliquée sur cette structure. Le résultat est composé de deux enfants de telle sorte que chaque enfant reçoit une copie de la nouvelle structure générée après le croisement, et la structure restante est directement copiée à partir des deux parents.

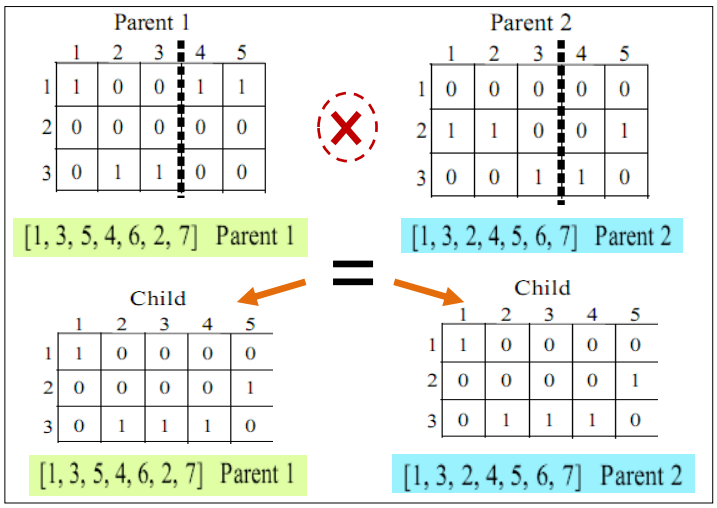


Figure 3.12 : Illustration de l'operateur de croisement au niveau de la matrice d'allocation de tâches.

La mutation dans le BCGA est aussi un processus en deux étapes. En premier lieu, l'une des deux structures de données dans BCGA est choisie au hasard (par exemple la matrice d'allocation des tâches) pour la manipulation, puis l'opération de mutation est appliquée sur cette structure. Le résultat est composé d'un seul enfant de telle sorte qu'il reçoive la nouvelle structure générée après la mutation, et la structure restante est directement copiée à partir du parent.

L'utilisation d'une représentation bi-chromosomique dans BCGA sert à réduire le contenu d'informations, et par conséquence; la complexité des opérateurs de croisement et de mutation diminue, et l'algorithme génétique sera capable d'explorer de manière plus significative l'espace des solutions.

Dans tous les travaux cités précédemment, les auteurs ont traité le problème d'ordonnancement multiprocesseur classique. Plusieurs autres chercheurs ont traité le problème d'ordonnancement sur des systèmes prenant en considération les contraintes temporelles, la priorité de chaque tâche et le type de système à traiter qui peut être dur, souple ou bien mixte.

4.2 Ordonnancement multiprocesseur temps réel

Pour résoudre le même problème, Y. Li, Y. Yang, M. Ma et R. Zhu **[78]** ont proposé un nouvel algorithme génétique qui tire profit de la connaissance du problème spécifique tout au long du processus de recherche pour résoudre le problème d'ordonnancement non préemptives des tâches indépendantes dans un système temps réel souple et statique sur une architecture multiprocesseurs centralisé dans laquelle toutes les tâches sont arrivées à un processeur central. Ce dernier est associé par une file d'attente pour mémoriser les nouvelles tâches arrivées. L'ordonnanceur central distribue les tâches vers les autres processeurs pour les exécuter. Le but est de minimiser le temps moyen de réponse et minimiser le nombre des tâches respectant leurs échéances.

Pour le système de codage, un gène est composé d'un couple décimale de deux attributs <Tj,Pi> tel que : Tj représente les tâches et Pi représente les processeurs. Chaque chromosome a une longueur de taille fixe égale au nombre maximum de tâches.

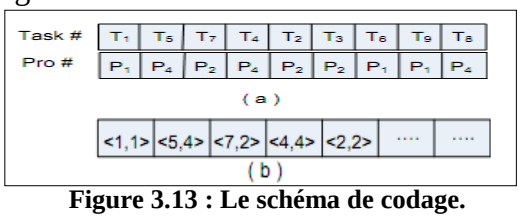


Figure 3.13 : Le schéma de codage.

Ils ont employé dans **[78]** certaines heuristiques pour générer la population initiale désirée. Par telle utilisation, des bonnes solutions (individus) peuvent être choisis et ça va aider l'AG à arriver à une solution plus effective et plus efficace.

La population initiale a été générée en faisant une permutation des tâches selon une heuristique puis en attribuant un nombre aléatoire *m* qui représente le processeur pour chaque tâche de gauche à droite de la séquence des tâches.

Pour l'opération de sélection, ils ont utilisé la méthode de l'échantillonnage stochastique sans remplacement qui a été démontré mieux que les autres méthodes. Pour le croisement, ils ont appliqué le croisement en un point choisi de manière aléatoire. Mais ce processus peut donner un résultat avec deux tâches dupliquées. Pour l'obtention d'un résultat valide, l'opérateur de croisement doit aussi vérifier la partie droite de chaque fils pour remplacer la tâche dupliquée avec une autre qui n'apparait pas dans cette séquence.

L'opération de mutation est également un processus en deux parties; faire une commutation de deux tâches sélectionnées de façon aléatoire, puis un autre changement aléatoire est appliqué à chacun des processeurs associés aux ces tâches.

Dans le même contexte, Ils ont développé dans **[79]** une nouvelle représentation pour résoudre le problème d'ordonnancement des tâches en utilisant les algorithmes génétiques.

Ce nouvel algorithme génétique est appliqué sur un ensemble de tâches non-préemptives et dépendantes sur une architecture multiprocesseur. Les tâches peuvent être apparues sur plusieurs processeurs, c'est-à-dire que la duplication d'une tâche est autorisée à condition que le coût de la communication entre les processeurs est plus grand que celui de la tâche elle-même. Cette duplication peut réduire les coûts de communication et par conséquent peut réaliser un meilleur temps d'exécution.

La représentation de la solution dans **[79]** est sous forme de deux sections; la première s'occupe de l'affectation des processeurs aux différentes tâches, tandis que la deuxième représente le mécanisme de la duplication des tâches.

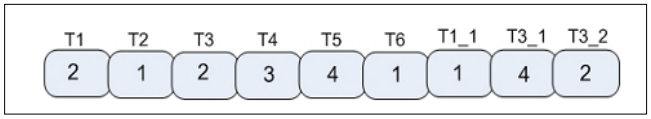


Figure 3.14 : Un simple individu (Ti_j est la j^{eme} duplication de Ti).

La section1, l'indice de la cellule représente le numéro de la tâche et le contenu représente le numéro du processeur qui est entre 1 et 4.

La section 2, s'occupe de la duplication des tâches: si la tâche est une tâche « feuille » il n'est pas nécessaire d'être dupliquée, si elle a un seul fils elle peut être placée dans le même processeur que son fils, dans le cas où la tâche a plusieurs fils elle doit être copiée dans aux moins « nombre des fils – 1 » processeurs car elle a déjà été attribuée à un processeur dans la première section.

Pour les opérateurs génétiques, ils ont utilisé la méthode de sélection par roulette, la méthode de croisement en un point et un taux égal à 0.02 pour la mutation. La fonction objective ici sert à minimiser le temps d'exécution total (qui représente le temps d'exécution de chaque tâche plus le coût de communication entre les processeurs) tout en respectant les contraintes requises.

En raison de l'utilisation d'une représentation plus améliorée de l'algorithme génétique, le rendement est amélioré par cette approche. L'inconvénient est qu'elle ne garantit pas une couverture complète de l'espace de recherche.

De même, M. R. Miryani, M. Naghibzadeh **[80]** ont proposé aussi un nouvel algorithme génétiques pour résoudre le problème d'ordonnancement non préemptives des tâches dépendante dans un système temps réel dur sur un ensemble des processeurs hétérogènes et avec un temps de rechargement de cache.

Le temps de rechargement de cache est important parce que dans les systèmes pratiques toutes les contraintes temporelles prisent autrement, le système peut se bloquer. Ils ont essayé dans ce travail de minimiser le temps d'exécution total et le nombre des processeurs en même temps à condition que tous les deadlines soient respectés.

Et comme il y a un conflit entre les deux objectifs, ils ont utilisé dans **[80]** l'approche des poids adaptative qui utilise certaines informations de la population actuelle pour justifier le choix des poids et pour pouvoir déplacé dans l'espace de recherche.

La solution (chromosome) dans cet algorithme est sous forme de deux parties: *u(.)* et *v(.)*. *u(.)* représente l'ordre d'exécution des tâches et *v(.)* représente l'information d'allocation des tâches aux processeurs. L'ordre d'exécution doit suivre un tri topologique et doit respecter le graphe des tâches. Ils ont supposé aussi que le nombre total des processeurs dans le système égal au nombre de tâches.

l	1	2	3	4	5	6	7
u(.)	6	2	4	7	1	3	5
v(.)	3	1	2	5	1	4	3

Figure 3.15 : Le chromosome résultant.

Pour les opérateurs génétiques ils ont utilisé la méthode de sélection par roulette, le croisement en un point et la mutation standard mais uniquement sur la partie *v(.)* du chromosome.

Un autre algorithme intelligent tolérant aux fautes a été proposé dans **[81]**, basé sur les AG et la réplication des tâches. Ce nouvel algorithme est conçu pour traiter les tâches indépendantes et périodiques dans un système temps réel souple sur une architecture multiprocesseur.

Ils ont supposé que chaque tâche possède une copie primaire et une autre copie de sauvegarde, ces deux copies sont allouées aux processeurs différents. La copie de sauvegarde ne peut pas être exécutée sauf que la copie primaire est échouée à cause d'une panne.

L'objectif alors est que cet algorithme doit être tolérant aux fautes et que toutes les tâches doivent utiliser les processeurs de manière égale.

Pour le codage du chromosome, chaque chromosome est constitué de tâches primaire *T1...*, *Tn*, et de tâches de sauvegarde *B1…*, *Bn*. Ces tâches doivent être exécutées sur les *m* processeurs. Chaque processeur possède une liste des tâches et une variable *R* qui représente le temps restant de l'utilisation du processeur. Initialement, tous les *R* sont mis à 1 et les listes des tâches de tous les processeurs sont vides.

En cas d'une allocation de la tâche *Ti* au processeur *Pj*, le nom de *Ti* est ajouté à la liste des tâches de *Pj*, et *R* est modifié par *R=R-Ci/Di*.

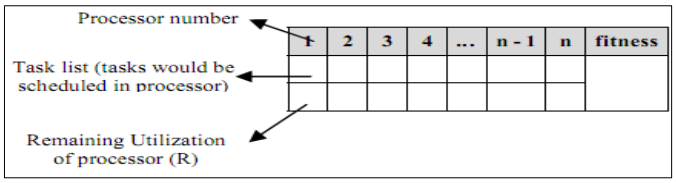


Figure 3.16 : Exemple de codage du chromosome.

La population initiale est générée selon un algorithme principalement basé sur les valeurs de *R*, La valeur du fitness de chaque chromosome est l'écart type de la valeur *R* comme l'indique la formule suivante :

$$Fitness = \sqrt{\sum_{J=1}^{m}(\frac{(R_J - \overline{R})^2}{m-1})}$$

Formule 3.2 : La valeur du fitness de chaque chromosome.

Pour le croisement, ils ont appliqué le croisement en un point, les listes des tâches et R sont copiés complètement et sans aucune modification, puis vérifier l'unicité et la condition de complétude des deux fils générés à savoir, vérifier si les tâches ne sont pas exécutées deux fois et que toutes les tâches doivent être exécutées, enfin calculer le fitness pour les deux fils générés.

Comme l'objectif aussi est d'utiliser les processeurs de manière égale, ils ont appliqué dans **[81]** deux algorithmes pour équilibrer la charge des processeurs basés sur la valeur *R*. Ces deux algorithmes sont pris en considération pour améliorer l'équilibrage de la charge des processeurs et par conséquent améliorer le fitness du chromosome. Ce nouvel algorithme proposé dans **[81]** est adapté aussi pour l'ordonnancement des tâches apériodique mais il ne prend pas en considération les relatons de précédence entres les différentes tâches.

Nous restons toujours dans le même sujet, N. Sedaghat, H. T. Yazdi, et M. Akbarzadeh **[82]** ont proposé un algorithme génétique multi-objectifs pour résoudre le problème d'ordonnancement et l'allocation des tâches dépendantes dans un système temps réel souple et sur un réseau arbitraire constitué des processeurs hétérogènes et non structurés, afin de minimiser deux objectifs : le temps d'exécution total et le temps de retard global des tâches. Pour cela ils ont utilisé la méthode de Pareto optimal.

Les messages sont aussi ordonnancés et alloués comme les tâches sur des liens appropriés au cours de la minimisation, et pour trouver le chemin de transfert de message entre les processeurs ils ont utilisé l'algorithme de routage classique qui se fonde sur la vitesse de lien. La figure suivante montre le codage utilisé:

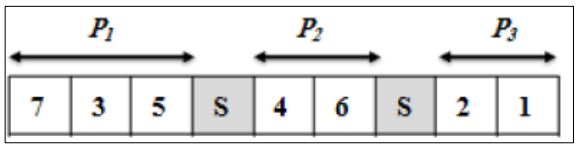


Figure 3.17 : Un individu dans l'AG proposée

Chaque caractère représente une allocation entre les tâches et les processeurs, ainsi que chaque caractère contient un nombre d'identification unique de la tâche, S est utilisé pour limiter les files d'attente des processeurs.

L'ordonnancement des tâches est diffère de celui des messages, en effet ces derniers peuvent être ordonnancés sur plus d'un lien et la communication entre deux tâches situant dans deux processeurs différents implique le transfert de message sur des voies intermédiaire entre ces deux processeurs. Et comme nous avons dit précédemment, ils ont utilisé dans **[82]** une méthode multi-objective pour déterminer la qualité de chaque chromosome, dans cette méthode il faut déterminer pour chaque solution l'ensemble des solutions qui lui ont dominé, et l'ensemble des solutions qu'elle domine: (une solution *x (1)* domine une solution *x (2),* si la solution *x (1)* n'est pas pire que *x (2)* à tous les objectifs. Ainsi, les solutions sont comparées sur la base de leurs valeurs de la fonction objectif).

Tous les points (solutions) qui ne sont pas dominés par aucun autre point sont appelés les point non dominés. L'ensemble des points non dominés forme un front dans l'espace des solutions et ils sont souvent visualisés pour représenter un front non dominé.

Donc les points situés dans le front non dominé par définition, ne sont pas dominé par aucun autre point dans l'espace des solutions, ce sont des points Pareto-optimal (ils constituent ensemble : le front Pareto optimal).

En fin, calculer l'ensemble non dominé pour chaque individu de la population et le choix de la population est basé sur la non-domination dans chaque front.

Dans cette méthode ils ont utilisé la sélection par tournoi, le croisement par cycle et le changement aléatoire des gènes d'un individu choisi aléatoirement pour l'operateur de mutation. Après l'application des opérateurs génétiques, la sélection des bons individus dépend de leur non domination.

On peut noter que les travaux cités précédemment concernant l'application des AGs aux problèmes d'ordonnancement temps réel sur des architectures multiprocesseurs sont aussi applicable pour les systèmes embarqués distribués. Ces derniers possèdent certaines contraintes particulières qui les distinguent des autres systèmes multiprocesseurs classiques. En effet, ce sont des systèmes intégrés sur une seule puce. La plupart d'entre eux sont spécifiques à une application, c'est à- dire qu'ils sont conçus « sur mesure » pour l'application et qu'ils possèdent des performances élevés.

5. Adaptation de l'informatique quantique

« En science naturelle, la nature nous a donné un monde qu'il ne nous reste qu'à découvrir ses lois. Quant aux ordinateurs, nous pouvons les garnir de lois pour créer un monde ».

Alan Kay.

5.1 Informatique quantique

5.1.1 Source d'inspiration et brève histoire

Le calcul, l'informatique et les ordinateurs quantiques forment l'essentiel de ce que les anglo-saxons désignent sous le vocable unique de Quantum Computing (QC), ce dernier repose principalement sur des lois et phénomènes de la mécanique quantique (comme la superposition d'états) pour le traitement de l'information. Cette nouvelle discipline qui est encore au stade d'enfance, permet de donner des solutions dans des temps de calcul raisonnables à des systèmes avec bien plus de précision qu'il ne sera jamais possible de le faire avec des ordinateurs conventionnels.

Le physicien américain Richard Feynman a observé au début des années 80 **[83],** que certains effets de la mécanique quantique ne peuvent pas être simulés de manière efficace sur un ordinateur classique. Il a émis l'idée de tirer avantage des propriétés des éléments quantiques apparaissant à l'échelle microscopique pour construire des machines d'un type nouveau, seules capables de simuler de manière efficace le comportement de systèmes quantiques. C'est ce qu'on appelle « les ordinateurs quantiques » **[84]**. Mais, son projet ne retient pas l'attention car il ne propose pas d'application concrète. En 1985, David Deutsch a formalisé l'intuition de Feynman et a proposé un premier modèle de calcul quantique. Dix ans après, cette nouvelle technologie a suscité beaucoup d'attention, sa supériorité a été démontrée par quelques algorithmes quantiques tels que l'algorithme de factorisation des nombres entiers proposé par Peter Shor en 1994 **[85]**, qui est le premier algorithme spécifiquement destiné aux ordinateurs quantiques, et l'algorithme de recherche dans les bases de données non triées proposé par Grover en 1996 **[86]**. En 2011, Rainer Blatt, de l'université d'Innsbruck, a présenté le dispositif le plus complexe qui comporte 14 qubits des ions de calcium. En mai 2013, La société canadienne DWave commercialise un simulateur équivalent à un ordinateur quantique, mais dont le principe de fonctionnement et les capacités sont encore débattus.

5.1.2 De quoi s'agit l'informatique quantique

L'informatique quantique est une nouvelle branche de l'informatique qui connait un intérêt grandissant en s'inspirant des principes de la mécanique quantique pour le traitement et la transmission de l'information. Elle joue maintenant un rôle important dans de nombreux domaines: la physique, la chimie, l'informatique et les mathématiques.

D'abord, Qu'est-ce que la mécanique quantique ? La mécanique quantique est la description la plus précise et complète du monde connu. C'est une branche de la physique qui décrit la manière dont se comportent les objets microscopiques : les molécules, les atomes ou les particules. Elle a été découverte dans les années 1920-1940 lorsque les physiciens ont voulu décrire le comportement des atomes et les échanges d'énergie entre la lumière et la matière à l'échelle de l'atome. La mécanique quantique est également la base pour la compréhension du calcul quantique et de l'informatique quantique.

Le développement de l'informatique quantique a créé une nouvelle technologie inattendue aide à diminuer remarquablement la complexité algorithmique grâce notamment de calcul parallèle. Ce dernier permet d'effectuer des calculs d'une manière radicalement nouvelle, et peut être exploité pour les problèmes d'optimisation combinatoire qui manipulent une grande quantité d'information. Les champs d'application de l'informatique quantique concernent essentiellement deux secteurs :

- Le calcul quantique : ensemble des algorithmes quantiques exploitent des propriétés des états quantiques ;
- La communication de l'information : comme la sécurisation de l'échange de clés secrètes de codage (cryptographie quantique).

Un bit peut porter l'un des deux états : 0 ou 1, qui sont suffisant pour la modélisation et la réalisation des solutions dans de nombreuses applications. Mais ces deux états, dans certains autres domaines tels que l'optimisation combinatoire et la cryptographie qui nécessitent des calculs parallèles, se heurtent à une limite au niveau de temps de réponse qui est très grand. Cela signifie qu'on a besoin d'une grande puissance de calcul pour pouvoir traiter ces problèmes dans un temps raisonnable. L'ordinateur quantique apparait comme la réponse à ces attentes **[87] [88] [89] [90] [109]**.

5.1.3 Ordinateur quantique : l'ordinateur de demain !

« L'ordinateur quantique, si nous parvenons un jour à en construire un, ne sera pas une machine conventionnelle utilisant simplement des transistors plus rapides, mais fonctionnera selon des principes radicalement différents »

Par cette phrase a expliqué Daniel Estève, responsable du groupe « Quantronique » au CEA de Saclay la définition de l'ordinateur quantique **[110]**. Ces ordinateurs pourraient résoudre certains problèmes exponentiellement plus vite qu'un ordinateur classique.

Gordon Moore avait prévu dès 1965 par sa fameuse "loi", que le nombre de transistors est multiplié tous les deux ans par puce. En effet, en 1971 le premier microprocesseur comptait environ 2000 transistors. Si en faisant un peu de calcul, on peut constater qu'après 40 ans, le nombre de transistor devient plus de deux milliards par puce mesurant à peine 1 cm^2. Cela nous mène à déduire que la puissance des ordinateurs allait croître de manière exponentielle. En 2007, Gordon Moore a déclaré que la croissance du nombre de transistors dans une puce se heurterait aux environ de 2020 à une limite physique: celle de la taille des atomes.

Aujourd'hui, les chercheurs ne considèrent pas ces lois comme un obstacle qui ne peuvent pas le franchir, et ils essayent de trouver des solutions à ces problèmes en changeant du paradigme du mode de calcul binaire sur lequel repose l'informatique depuis son invention, dans les années 1940, et se tourner vers l'informatique quantique **[91]**.

Un ordinateur quantique repose sur des propriétés quantiques comme la superposition et intrication d'états quantiques. De petits ordinateurs quantiques ont déjà été construits dans les années 90 et des progrès sont en cours. en 2007, la société D-Wave a présenté leur produit « Orion » et affirmé que ce dernier est capable d'exécuter 64000 calculs en parallèle. Pour démontrer la qualité et la puissance de son produit, D-Wave a essayé de résoudre le fameux problème du voyageur de commerce. Orion ne demande que quelques cycles pour résoudre ce problème. Cette démonstration montre que les calculateurs quantiques deviennent petit à petit une réalité et qu'ils vont continuer sur le chemin du progrès **[111]**.

Certains algorithmes qui s'appuient sur les caractéristiques des ordinateurs quantiques rendent possibles la résolution des problèmes de décryptage ainsi que l'accès à l'information sera plus vite que tout ordinateur classique. Le paradigme d'informatique quantique se heurte à de nombreux difficultés et problèmes qui restent encore à résoudre comme le phénomène de décohérence (perte des effets quantiques sur le long terme) **[112]**.

Les futurs ordinateurs quantiques auront donc une grande puissance de calcul qui leurs permet de réaliser en quelques secondes des simulations numériques qui demandent aujourd'hui des semaines ou des mois de travail aux meilleurs superordinateurs **[91]**.

5.1.4 Concepts fondamentaux du traitement quantique de l'information

5.1.4.1 Les bites quantiques (qubits)

La théorie classique de l'information est basée sur l'existence de la notion de bit, une variable discrète pouvant prendre deux états contrastés notés '0' et '1' signifiant que le courant passe ou pas. En informatique quantique, cette théorie est carrément écartée où les bits sont remplacés par une autre structure élémentaire de stockage de l'information : qubit.

Un bit quantique ou qubit, peut prendre un ensemble de valeurs beaucoup plus large. En effet, la physique quantique, avec son principe de superposition, permet à un état d'être un "mélange" d'autres états. Ainsi, un qbit peut être dans l'état 0 ou 1, et il peut être dans les deux états en même temps, ou plus précisément dans une superposition des deux états |0> ou |1>. En fait, tout état de la forme : $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, de telle sorte que: $|\alpha|^2 + |\beta|^2 = 1$.

Pour distinguer les deux états 0 et 1 des états classiques, on les note |0> et |1> suivant la convention introduite par le physicien P.A.M. Dirac dans les années 30.

Un état constitue de 10% de 0 et 90% de 1, signifie que quand on mesure la valeur du qubit, on a 10% de chances de trouver 0 et 90% de trouver 1 **[90]**.

Le tableau suivant illustre un comparatif entre les bits classiques et les bits quantiques :

Bit classique	**Bit quantique (qubit)**
Un bit a toujours une valeur définie	Pas de valeur définie sans observation
Un bit vaut seulement 0 ou 1	Un qubit peut être dans une superposition de 0 et 1 simultanément.
Un bit peut être copié sans être affecté	Un qubit dans un état inconnu ne peut être copié
Un bit peut être lu sans affecter sa valeur	La lecture d'un qubit qui est initialement dans une superposition changera sa valeur
Lire un bit n'affecte pas un autre	Si un qubit est enchevêtré avec un autre, la lecture de l'un affectera le second.

Tableau 3.2 : Bit classique et bit quantique (Qubit) [65].

5.1.4.2 Registres quantiques

Un ensemble de n bits, s'appelle un registre. Un registre classique de n bits peut stocker 2^n valeurs différents, par contre dans le cas de registre quantique, il peut faire beaucoup mieux car la superposition quantique lui permet de les stocker tous en même temps.

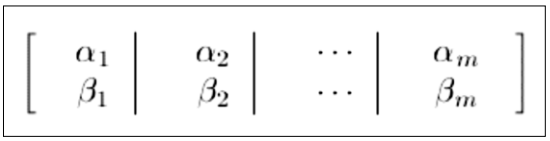


Figure 3.18 : Exemple d'un registre quantique.

Comme la montre la figure ci-dessus, le registre contient la superposition de toutes les permutations possibles. Chaque colonne représente un simple qubit, les amplitudes de probabilité $\boldsymbol{\alpha i}$ et $\boldsymbol{\beta i}$ sont des valeurs réelles satisfaisant la condition: $|\boldsymbol{\alpha}_i|^2 + |\boldsymbol{\beta}_i|^2 = \mathbf{1}$.

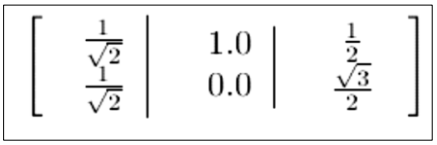


Figure 3.19 : Attribution des valeurs à α et β.

Pour chaque qubit, une valeur binaire est ensuite calculée en fonction de ses deux probabilités $|\boldsymbol{\alpha}_i|^2$ et $|\boldsymbol{\beta}_i|^2$ qui sont interprétés comme les probabilités d'avoir respectivement 0 ou 1, en d'autre terme le vecteur $\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$ représente le qubit dont la mesure renvoie soit l'état |0> avec probabilité $|\boldsymbol{\alpha}_i|^2$, soit l'état |1> avec probabilité $|\boldsymbol{\beta}_i|^2$ **[92]**. Pour manipuler le contenu de ces registres, on a deux types d'opérations : une opération simple dite « porte quantique » et un ensemble d'opérations quantiques successives appelé « circuits quantique ».

5.1.4.3 Portes quantiques

Dans l'informatique quantique, l'information est stockée sous forme des bits quantiques et elle est traitée par des portes logiques quantiques. Ces dernières sont équivalentes aux portes logiques en informatique classique (qui prennent des bits d'un état à un autre). Un ordinateur quantique est construit de circuits quantiques contenant des portes quantiques élémentaires effectuant des opérations logiques de traitement de l'information quantique. Une porte quantique est une transformation unitaire de l'espace de Hilbert de dimension 2^n, qui peut s'écrire comme des combinaisons de transformations locales agissant sur seulement quelques qubits **[93]**.

Nous avons sélectionné trois portes quantiques s'appliquant à un, deux et trois qubits afin de cerner leur mode de fonctionnement.

• **Porte de Hadamard** : Un grand nombre d'algorithmes quantiques utilisent la transformation de Hadamard comme première étape, puisqu'elle transforme ***n*** qubits initialisés avec |0> en une superposition de tous les 2^n états orthogonaux exprimés dans la base |0>, |1> avec une pondération égale. La porte de Hadamard s'applique à un seul qubit.

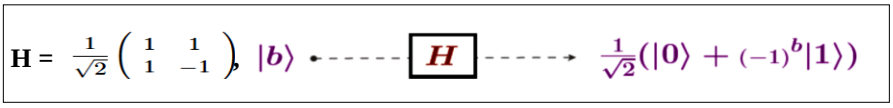


Figure 3.20 : La porte quantique de Hadamard.

Et il est intéressant d'essayer de visualiser son fonctionnement en tenant compte de l'image de sphère de Bloch.

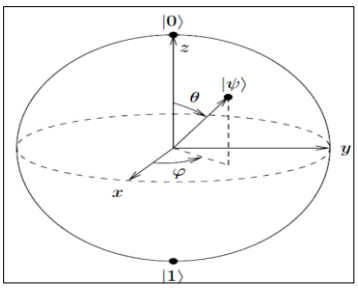


Figure 3.21 : Sphère de Bloch.

• **Porte Non Contrôlée (CNOT) :** s'applique à deux qubits, le deuxième qubit est inversé si le premier est dans l'état |1> : **|00> ⟶ |00> ; |01> ⟶ |01> ; |10> ⟶ |11> ; |11> ⟶ |10> ;**

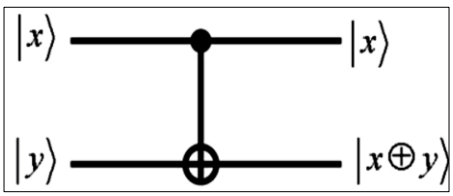


Figure 3.22 : La porte quantique CNOT.

• **Porte de Toffoli :** elle s'applique à trois qubits. Le troisième qubit est inversé si les deux premiers sont dans l'état |1>.

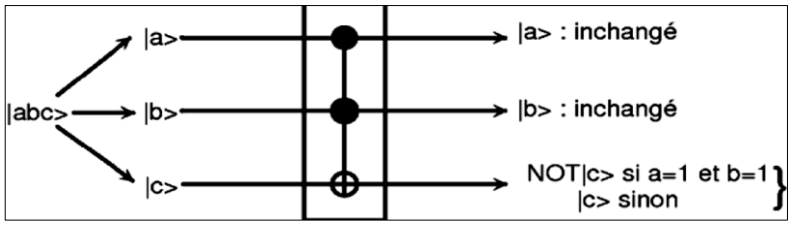


Figure 3.23 : La porte quantique de Toffoli.

Un circuit quantique est la combinaison de deux ou plusieurs portes quantiques pour effectuer un traitement plus compliqué sur un système quantique.

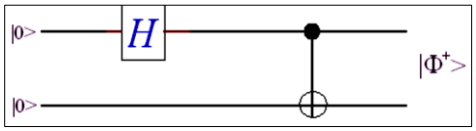


Figure 3.24 : Un circuit quantique.

5.2 Algorithme génétique inspiré-quantique (AGIQ)

Depuis la fin des années 90s, le couplage entre l'informatique quantique et le calcul évolutionnaire a été prouvé très efficace dans le cas des problèmes complexes. Ce couplage fait la naissance de ce qu'on appelle « les algorithmes génétiques inspirés-quantiques ».

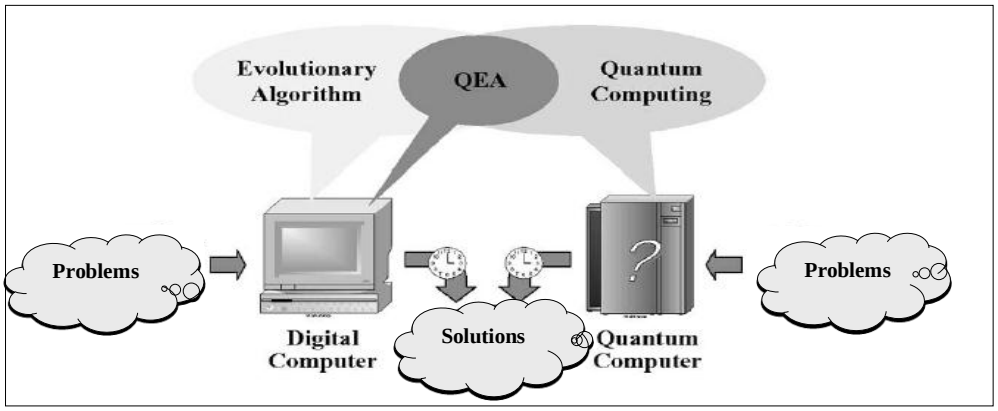


Figure 3.25 : Algorithme génétique inspiré-quantique.

Le premier travail qui a appliqué ces algorithmes est celui de Narayanan et Moore En 1996 **[94]**, ils ont proposé un AG utilisant les principes de l'informatique quantique, puis en 2000, Han et Kim **[95]** ont proposé un nouveau modèle d'AG utilisant les principes de qubit, la superposition d'états et de la porte quantique pour résoudre le problème de sac-à-dos. Ces nouveaux algorithmes ont ainsi marqué avec force leur présence ces dernières années dans le domaine de l'optimisation combinatoire.

5.2.1 Définition des algorithmes génétiques inspirés-quantiques

Comme tout autre AG, un algorithme génétique inspiré-quantique (AGIQ) repose sur la représentation de l'individu, la fonction d'évaluation et la dynamique de la population. La distinction principale est que ces algorithmes sont caractérisés par l'utilisation des principes de l'informatique quantique, y compris les concepts de qubits et la superposition d'états.

En fait, les AGIQ utilisent le concept du qubit pour la représentation des chromosomes, au lieu d'utiliser une représentation binaire, réel ou symbolique. Ceci signifie que les opérations génétiques seront totalement redéfinies afin de pouvoir s'adapter avec la nouvelle représentation des chromosomes **[96] [95]**.

5.2.2 Fonctionnement des algorithmes génétiques inspirés-quantiques

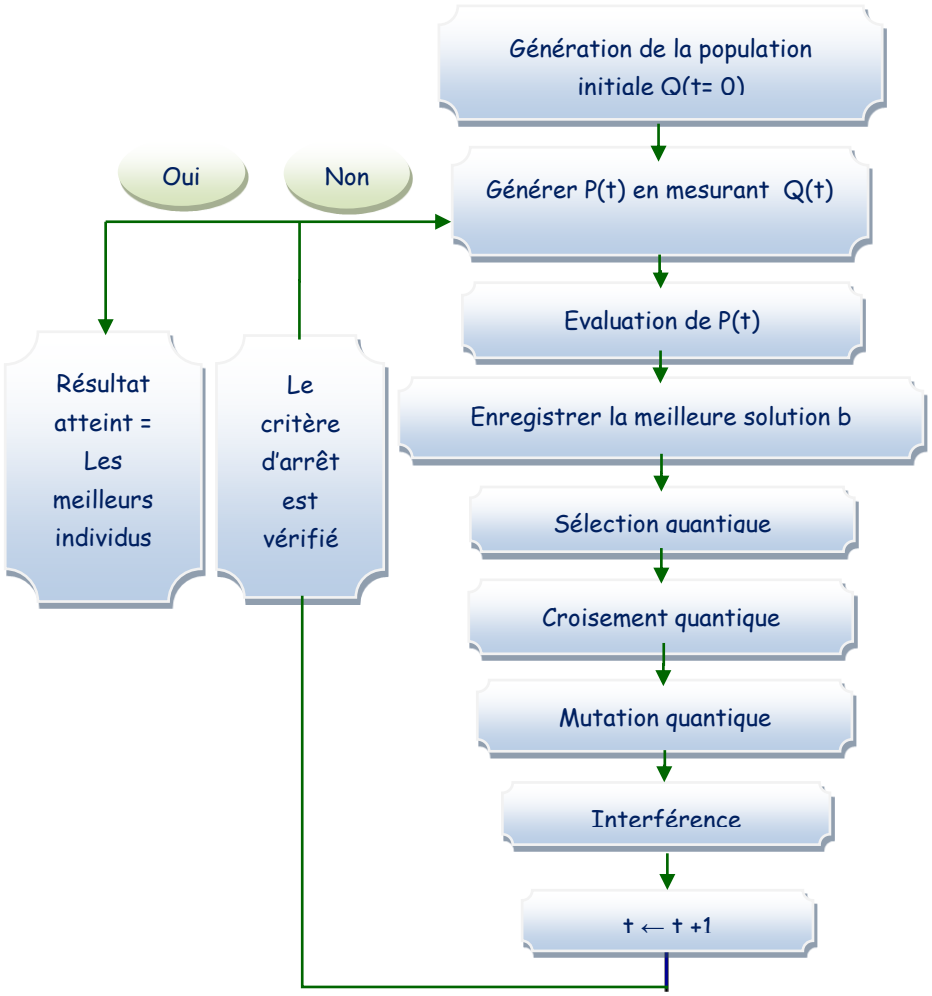


Figure 3.26 : Principe de fonctionnement d'un AGQ.

5.2.2.1 Codage des chromosomes quantiques

Les AGIQ emploient une nouvelle représentation des individus basée sur le concept du qubit. Un chromosome est représenté sous forme de chaîne de qubits de longueur *m* formant un registre quantique.

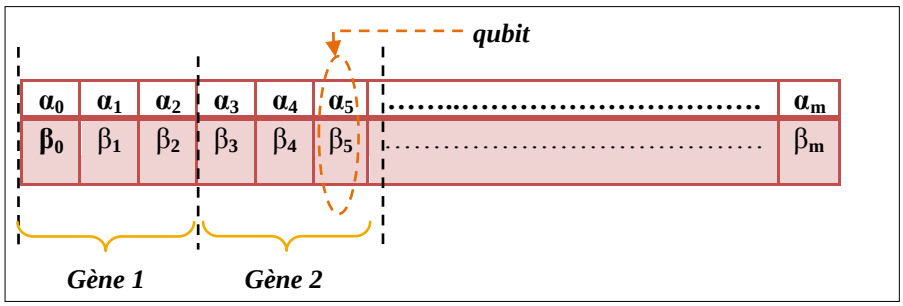


Figure 3.27 : Représentation du chromosome quantique.

Notant que : $|\alpha|^2$ est la probabilité que le qubit se trouve dans l'état '0' et $|\beta|^2$ est la probabilité que le qubit se trouve dans l'état '1'.

Cette représentation a l'avantage qu'elle peut représenter une superposition linéaire des états d'une manière probabilistique et elle peut produire la diversité dans la population.

5.2.2.2 Initialisation de la population

Pour générer la population initiale, il faut créer un ensemble de chromosomes quantiques. Pour cela, toutes les amplitudes des qubits sont initialisées en donnant les mêmes probabilités pour tous les états de superposition.

5.2.2.3 Fonction mesure

Elle est aussi appelée observation. Le but de cette fonction est d'extraire un chromosome classique à partir d'un autre quantique pour l'évaluer. Donc chaque qubit est transformé à un bit classique qui porte une seule valeur : 0 ou 1.

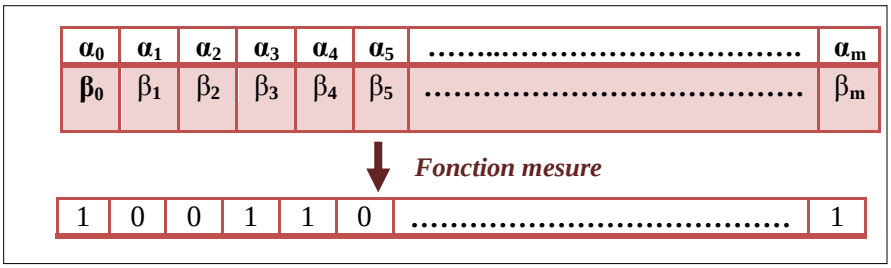


Figure 3.28 : Mesure d'un chromosome quantique.

L'algorithme suivant illustre bien l'implémentation de cette fonction :

```
Fonction   mesure ()
{   r = tirer r dans [0,1]
    si  (r> α²)  retourner  1
    sinon        retourner  0
}
```

Figure 3.29 : Pseudo code de la fonction mesure.

5.2.2.4 Evaluation des individus

Après l'application de la fonction mesure sur l'ensemble de chromosomes de la population, un ensemble de chromosomes binaires sont ainsi créés. La quantification de ces derniers se fait pour distinguer les chromosomes proches de la solution optimale. De la même

manière que pour un AG ordinaire, l'évaluation des individus est effectuée à l'aide d'une fonction d'adaptation.

5.2.2.5 Sélection quantique

De la même manière que d'un AG ordinaire, la sélection quantique se fait selon les valeurs fournies par la fonction d'adaptation. Les techniques de sélection utilisées ici sont les mêmes que ceux des AG conventionnels.

5.2.2.6 Croisement quantique

Après la sélection des meilleurs individus, l'operateur du croisement quantique peut être appliqué sur chaque paire des individus de la même façon que pour les AG ordinaire. La seule distinction est que cet opérateur est appliqué sur des chromosomes quantiques.

Le croisement quantique entre deux individus en un point donné se fait par l'échange des deux fragments situés après ce point ce qui permet de générer deux nouveaux individus.

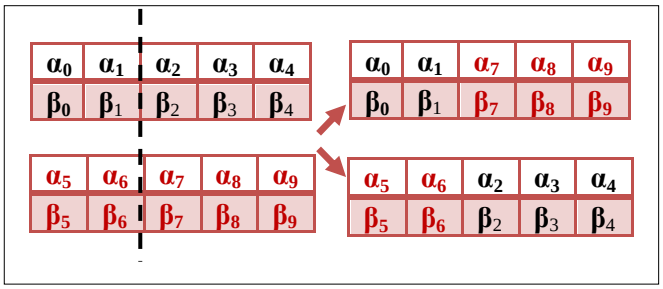


Figure 3.30 : Croisement quantique.

De la même manière, on peut aussi appliquer le croisement en des points multiples pour générer de nouveaux individus.

5.2.2.7 Mutation quantique

La mutation quantique opère sur un qubits choisi de façon aléatoire. Cet opérateur est réalisée en permutant les coefficients α_i et β_i du qubit choisi, ce qui permet d'inverser les probabilités d'avoir la valeur 1 ou 0.

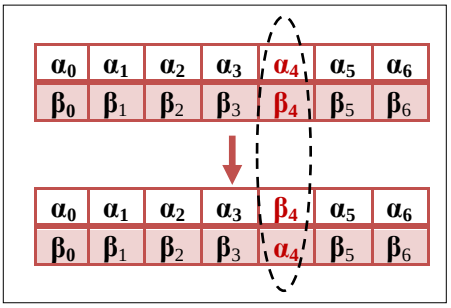


Figure 3.31 : Mutation quantique.

5.2.2.8 Interférence

L'interférence permet d'augmenter ou de diminuer la probabilité d'avoir un état. On distingue deux types d'interférence : constructive et destructive.

L'interférence constructive permet d'augmenter la probabilité d'un état tandis que l'interférence destructive permet de diminuer la probabilité d'obtenir un état.

Cette opération permet de faire des changements dans les amplitudes des individus pour bien exploiter l'espace de recherche. Elle sert essentiellement à déplacer l'état de chaque qubit dans le même sens que de la valeur du bit correspondant dans la meilleure solution enregistrée, pour rester toujours autour de la meilleure solution trouvée, c'est le cas constructif. Le cas destructif consiste en le déplacement de l'état de chaque qubit dans le sens inverse de la valeur du bit correspondant dans la meilleure solution enregistrée. L'avantage de cette technique est l'exploration des autres solutions afin d'échapper aux minimums locaux. Cette opération est réalisée à l'aide des portes quantiques choisies selon le problème donné. Ces portes quantiques permettent une rotation dont l'angle est en fonction des valeurs de *ai, bi,* et de la valeur du bit correspondant dans la solution enregistrée. Le choix de la valeur de l'angle $\delta\theta$ se fait de telle sorte à éviter la convergence prématurée. Elle est souvent fixée empiriquement et sa direction est déterminée en fonction des valeurs de ai, bi et de la valeur du qubit situant à la position i de l'individu en cours de modification **[92]**.

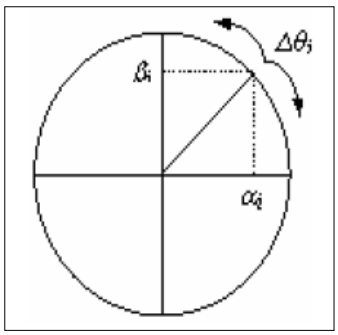


Figure 3.32 : Interférence quantique basé sur la rotation.

6. Conclusion

Dans ce chapitre, nous avons d'abord exposé le paradigme des algorithmes génétiques classiques et les fondations nécessaires à leur compréhension. Ces méthodes qui sont inspirées des mécanismes de sélection naturelle et de la génétique, connaissent un développement très important et donnent de bons résultats à de nombreux problèmes réels dans des temps de calcul raisonnables. Ensuite, nous avons découvert le domaine de l'informatique quantique, qui est un domaine très riche et qui offre une capacité de traitement et de stockage des données très élevée. Son gain considérable en temps et en calcul a conduit à l'apparition des

approches hybrides. Une hybridation entre les algorithmes génétiques classiques et l'informatique quantique a donné naissance aux algorithmes génétiques inspirés-quantiques qui ont montré leurs efficacités dans de nombreux problèmes d'optimisation comparable à celles des algorithmes génétiques classiques.

Chapitre 4 Contribution, tests et résultats

Dans ce chapitre :

1. Introduction.
2. Modélisation du problème.
   - Le graphe de tâches.
   - Le graphe d'architecture.
3. Exemple d'application.
4. Représentation des solutions.
   - Première stratégie: application des algorithmes génétiques classiques AGC1.
   - Deuxième stratégie: application des algorithmes génétiques classiques AGC2.
   - Troisième stratégie: application des algorithmes génétiques inspirés-quantiques AGIQ1.
   - Quatrième stratégie : application des algorithmes génétiques inspirés-quantiques AGIQ2
5. Comparaison des résultats des quatre stratégies et discussion
6. Conclusion

Résumé

Nous présentons dans ce chapitre l'approche proposée pour l'allocation et l'ordonnancement de tâches temps réel mixtes et dépendantes sur un système embarqué et sur une architecture distribuée tout en optimisant leurs performances en utilisant quatre stratégies d'optimisation: optimisation par algorithmes génétiques classiques (AGC1 et AGC2) et optimisation par algorithmes génétiques inspirés-quantiques (AGIQ1 et AGIQ2).

1. Introduction

Nous introduisons dans ce chapitre, l'approche proposée et toutes les techniques de résolution que nous avons adoptés. Nous allons également présenter en détail tout ce qui se rapporte aux deux problèmes d'allocation et d'ordonnancement des tâches temps réel mixtes et dépendantes sur un système embarqué, distribué sur une architecture composée d'un ou de plusieurs bus et processeurs embarqués hétérogènes.

Puisque notre objectif vise à optimiser les performances au niveau des systèmes embarqués distribués, à savoir le temps de réponse moyen et le nombre de tâches respectant leurs échéances, nous proposons d'utiliser une méta-heuristique récente, qui est l'optimisation par algorithme génétique (AG), à laquelle nous joindrons ensuite les concepts de l'informatique quantique. Nous appelons cette fusion l'optimisation par algorithme génétique inspiré-quantique (AGIQ) (en anglais : Quantum-Inspired Genetic Algorithms QIGA).

Notre approche comporte quatre techniques de résolution, qui sont les suivantes :

- L'optimisation des performances par algorithmes génétiques classiques en affectant des priorités statiques aux tâches (AGC1).
- L'optimisation des performances par algorithmes génétiques classiques en affectant des priorités dynamiques aux tâches (AGC2).
- L'optimisation des performances par algorithmes génétiques inspirés-quantiques en affectant des priorités statiques aux tâches (AGIQ1).
- L'optimisation des performances par algorithmes génétiques inspirés-quantiques en affectant des priorités dynamiques aux tâches (AGIQ2).

Enfin, nous présenterons une série d'expérimentations ainsi qu'un ensemble de résultats expérimentaux afin d'évaluer les différentes stratégies.

2. Modélisation du problème

Le but de l'allocation est de distribuer les différentes tâches temps réel sur les différents processeurs de l'architecture de telle sorte que les performances satisfassent les différentes contraintes imposées par le concepteur et/ou l'environnement. L'ordonnancement consiste à fixer des dates d'exécutions pour un ensemble de tâches ayant certaines caractéristiques connues sur un ensemble limité d'unités de traitement dont le but est d'optimiser un ou plusieurs objectifs. L'allocation et l'ordonnancement sont fortement interdépendants ; c'est pour cette raison et pour aboutir à des bons résultats, nous préférons les traiter en parallèle.

Les systèmes embarqués distribués peuvent être décrits par deux graphes : un graphe de tâches représentant les fonctionnalités à implanter ainsi que leurs dépendances, et un graphe d'architecture matérielle représentant les processeurs interconnectés chargés d'exécuter ces fonctionnalités. On présente donc dans la section suivante le graphe de tâches et le graphe d'architecture matérielle, en précisant quelques hypothèses de travail concernant ces graphes.

2.1 Le graphe de tâches

2.1.1 Graphe de tâche acyclique

L'ensemble de tâches de notre système est modélisé par un graphe de tâches. Il s'agit d'un graphe orienté acyclique (en anglais Directed Acyclic Graph (DAG)) G = (N, $\mathcal{E}$) où, N = {ti / i = 1, …, N} est un ensemble de N nœuds et $\mathcal{E}$ = {$e_{i,j}$ / (i, j) ⊂ {1, …, N} × {1, …, N}} est un ensemble de $\mathcal{E}$ arêtes, dans lequel chaque nœud représente une tâche temps réel et chaque arête définit une relation de précédence (message reliant deux tâches). Par définition, une tâche d'entrée du graphe n'a aucun prédécesseur et une tâche de sortie est sans successeur.

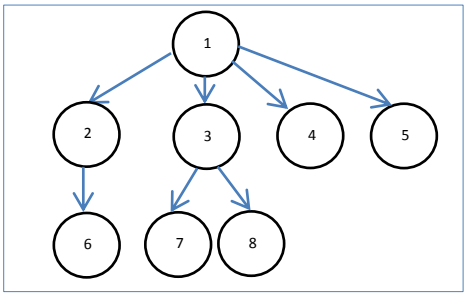


Figure 4.1 : Exemple d'un graphe de tache.

Les tâches temps réel de notre système peuvent être périodiques ou apériodiques, de contrainte dure ou souple, ce qui implique que le graphe de tâches peut être composé de

plusieurs sous-graphes, chacun représente une dépendance entre des taches temps réel de même type et de même contrainte, comme le montre la figure suivante :

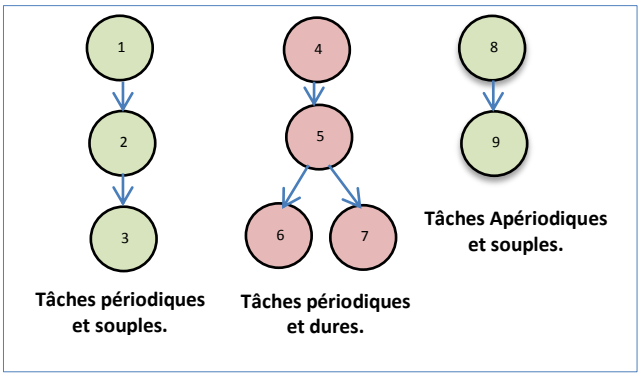


Figure 4.2 : Exemple d'un graphe de tâches composé de trois sous-graphes.

2.1.2 Hypothèses sur le graphe de tâches

Par la suite, nous allons considérer un graphe de tâches, constitué d'un ensemble de sous-graphes. Une tâche peut être reliée à une autre tâche de même type par un message dans le sous-graphe.

2.1.2.1 Sous-graphes de tâches périodiques

Un sous-graphe de tâches périodique est composé d'un ensemble de tâches périodique reliées entre eux par des messages.

Nous avons supposé que les tâches périodiques de tous les sous-graphes sont synchrones, cela veut dire que toutes les premières instances d'un ensemble de tâches périodiques ont commencés leur exécution en même temps (les tâches sont alors à départ simultané, par exemple: la date d'arrivée des tâches est « t= 0 »).

Chaque sous-graphe est doté d'une période ***P*** fixée à l'avance, cela signifie que toutes les tâches appartenant au même sous-graphe ont la même période ***P,*** cette période est choisie de telle sorte qu'on doit garantir l'exécution de toutes les tâches périodique du sous-graphe dans cette période. Généralement la période doit être supérieure ou égale à la somme des temps d'exécution de toutes les tâches du sous-graphe.

Aussi, chaque tâche périodique possède une échéance ***D*** (deadline), avant laquelle elle doit terminer son exécution. La valeur ***D*** doit être inférieure ou égale à la valeur ***P***.

Le sous-graphe peut être composé d'un ensemble de tâches périodiques souples ou dures, dans le cas souple, le non-respect de la valeur ***D*** ne conduit à aucun risque mais cela peut

provoquer une dégradation des performances du système. Dans le cas dur, la tâche ne doit pas dépasser l'échéance ***D***.

En plus, nous avons attribué un temps d'exécution ***C*** pour chaque tâche. Dans le cas souple, la valeur ***C*** est calculée en utilisant une loi de distribution pour trouver le temps moyen d'exécution ou ACET « Average-Case Execution Time ». Dans le cas dur, on attribue une valeur à ***C,*** cette dernière représente le temps d'exécution au pire des cas ou WCET « Worst-Case Execution Time ».

Lorsque la période d'un sous-graphe est terminée, tous les traitements sont arrêtés, les paramètres des tâches sont réinitialisés et un nouveau cycle est commencé.

2.1.2.2 Sous-graphes de tâches apériodiques

Comme le cas des tâches périodiques, un sous-graphe de tâches apériodique est composé d'un ensemble de tâches apériodique reliées entre eux par des messages.

Pour générer le temps inter-arrivé de chaque tâche apériodique, nous avons implémenté la loi de poisson de paramètre (λ) qui indique qu'un événement arrive en moyenne (λ) fois dans une période donnée. Cette loi est donnée par la formule suivante :

$$p(k) = P(X = k) = \mathrm{e}^{-\lambda}\frac{\lambda^k}{k!}$$

Formule 4.1 : La loi de poisson.

Chaque tâche apériodique possède une échéance ***D*** (deadline), avant laquelle elle doit terminer son exécution.

Nous avons considéré que le sous-graphe de tâches apériodiques est toujours de contraintes souples, puisque les dates d'arrivées des tâches apériodiques ne sont pas fixées comme celles des tâches périodiques, et par conséquent l'échéance d'une tâche peut être achevée avant que la tâche soit déclenchée. En plus, Nous avons attribué un temps d'exécution ***C*** pour chaque tâche en utilisant une loi de distribution pour trouver le temps moyen d'exécution ou ACET « Average-Case Execution Time ».

Pour simplifier, nous avons affecté une pseudo-période pour chaque tâche apériodique, cette pseudo-période est une valeur qui sépare deux occurrences successives d'un travail d'une tâche apériodique. Pour évaluer la pseudo-période, nous avons calculé la moyenne des temps inter-arrivée généré par la loi de poisson de chaque tâche.

Nous avons adopté dans notre travail deux types de priorité pour les deux types de tâches (périodique ou apériodique), une priorité locale entre les tâches du même sous-graphe, et une priorité globale entre les tâches appartenant aux sous-graphes différents. Par exemple, une tâche t_0 est plus prioritaire par rapport à une tâche t_1 si t_0 est le prédécesseur de t_1 dans le même sous-graphe, ou t_0 possède une échéance relative (ou une période) inférieure à celle de t_1 qui est situé dans un autre sous-graphe.

Nous avons supposé aussi qu'une tâche (périodique ou apériodique), ne peut recevoir un message qu'au début de son exécution, et ne peut l'envoyer qu'à la fin de son exécution.

2.1.2.3 Les messages

La communication entre les tâches du système se fait par passage de message. Un message est activé seulement lorsque la tâche qui le produit termine son temps d'exécution. Leur temps de transfert est ainsi calculé en divisant sa taille sur le débit du support de communication chargé de la transmission du message. Si le message termine son temps de transfert, la tâche de destination sera alors prête à être exécutée.

Nous avons adopté une priorité fixe pour chaque message. Un message m_0 et plus prioritaire par rapport à un autre message m_1 si la tâche de destination de m_0 possède une échéance relative (ou une période) inférieure à celle de m_1.

Nous avons supposé que les messages ne possèdent pas des périodes, mais dans le cas où la période d'un sous-graphe est terminée, toutes les transmissions sont arrêtées et les paramètres de tous les messages de ce sous-graphe sont réinitialisés.

2.2 Le graphe d'architecture

Le graphe d'architecture sert à modéliser l'ensemble de composants matériels des infrastructures informatiques : les unités de traitement, les liens physiques de communications et les interconnexions entre les composants matériels.

2.2.1 Modélisation de l'architecture

Notre architecture distribuée peut être constituée de plusieurs processeurs embarqués hétérogènes interconnectés tous ensemble. Les liens physiques de communications sont des bus, des ponts ou des liens bipoints.

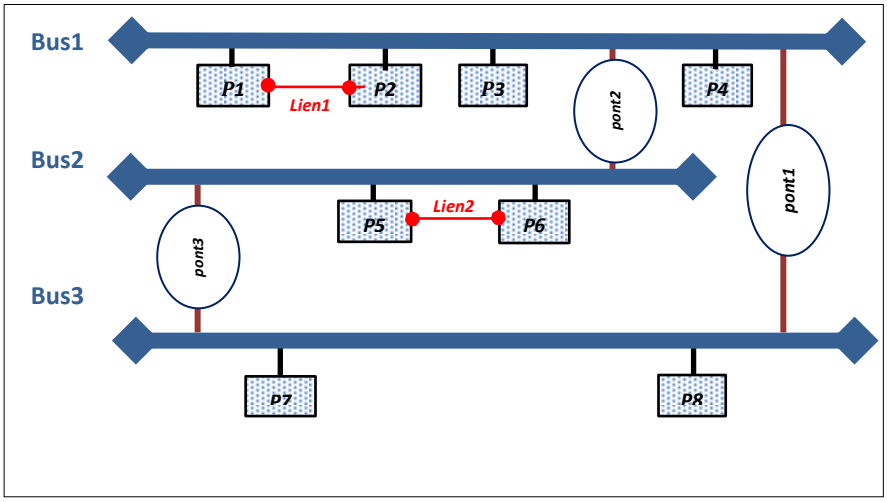


Figure 4.3 : Exemple d'un graphe d'architecture composé de 3 bus, 8 processeurs, 3 ponts et 2 liens.

2.2.2 Hypothèses sur le graphe d'architecture

Par la suite, nous allons considérer un graphe d'architecture, constitué d'un ensemble de composants matériels sur lesquelles les tâches sont implantées.

2.2.2.1 Les processeurs

Un processeur représente la machine chargée d'exécuter les tâches. Chaque processeur est connecté par un bus ou par un lien de communication.

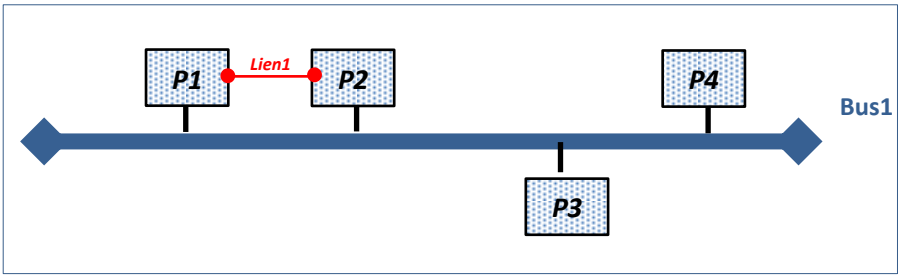


Figure 4.4 : Ensemble de processeurs connectés à un bus.

Comme nous avons dit précédemment, les processeurs de notre architecture sont hétérogènes dans le sens où chaque processeur est caractérisé par une puissance de calcul différente selon le type de processeur. Cela signifie que le nombre de cycle d'exécution pour chaque tâche varie d'un processeur à un autre, comme le montre le tableau ci-dessous:

	Cpu_1	Cpu_2	Cpu_3	Cpu_4	……	Cpu_N
Tâche 1	4 cycles	6 cycles	2 cycles	3 cycles	……	8 cycles
Tâche 2	5 cycles	7 cycles	3 cycles	4 cycles	……	9 cycles
Tâche 3	3 cycles	5cycles	2 cycles	3 cycles	……	7 cycles
:	:	:	:	:	:	:
Tâche M	6 cycles	7cycles	4 cycles	5 cycles	……	9 cycles

Tableau 4.1 : Le nombre de cycle d'exécution de chaque tâche par processeur.

Nous avons fait l'hypothèse que chaque processeur est doté d'une file d'attente de taille limitée. Les processeurs embarqués ne partagent pas de mémoire et ont leur propre mémoire qui est ainsi dite distribuée. Les communications inter-processeurs peuvent ainsi s'effectuer par passage de messages. Notant que tous les processeurs utilisent la même politique d'ordonnancement temps réel pour l'exécution des tâches et qu'aucune ressource n'est partagée entre plusieurs processeurs (hormis les supports de communication).

Nous avons considéré dans cette étude que la migration des tâches entre les processeurs n'est pas autorisée, cela signifie qu'une tâche ne peut s'exécuter que sur un et un seul processeur.

2.2.2.2 Les bus

Un bus est un support physique de communication chargé de transmettre les messages inter-processeurs. Chaque bus est caractérisé par un débit, c'est la quantité de données qu'il peut transporter par unité de temps (généralement en bit/second).

Nous avons également fait l'hypothèse que les bus de notre architecture sont bidirectionnelle et que les messages sont transféré sur ces bus avec un temps de transfert qui varie d'un bus à un autre. Chaque deux bus sont reliés par un pont.

	Bus_1	**Bus_2**	**Bus_3**	**……**	**Bus_x**
$Message_1$	3 unités	2unités	4 unités	……	5 unités
$Message_2$	2 unités	3 unités	5 unités	……	4 unités
$Message_3$	5unités	2 unités	3 unités	……	4 unités
:	:	:	:	:	:
$Message_Y$	4 unités	3 unités	5 unités	……	2 unités

Tableau 4.2 : Le temps de transfert de chaque message par bus.

2.2.2.3 Les ponts

Un pont est un équipement chargé d'interconnecter deux bus différents dans l'architecture. Lorsqu'une tâche est affecté à un processeur connecté à un bus, termine son temps d'exécution, elle doit transmettre un message à sa tâche successeur, mais cette dernière est affectée à un processeur différent connecté à son tour à un autre bus, dans ce cas le message est transmis premièrement sur le bus qui contient le processeur de la tâche source puis sur le pont et en fin sur le bus qui contient le processeur de la tâche de destination. Donc le but du pont et de recevoir un message sur l'une de ses interfaces, puis il le fait transmettre sur l'autre interface.

2.2.2.4 Les liens

Le lien est aussi un support chargé de transmettre les messages. Dans l'architecture proposée, un nœud (processeur) peut être relié à un autre par un lien direct. Cela signifie que le message est directement transmi sur le lien reliant les deux processeurs émetteur et récepteur au lieu de le transmettre sur le bus. L'avantage est que le lien est plus rapide que le bus ce qui permet de transmettre le message dans un temps réduit.

Enfin, nous avons supposé que tous ces supports de communication sont fiables, c'est-à-dire que les transmissions des messages s'effectuent sans erreurs.

3. Exemples d'applications

Pour chaque stratégie implémentée, nous allons présenter les résultats obtenus en exécutant notre logiciel sur un ensemble d'exemples d'architectures que nous avons jugé significatifs pour valider notre approche. Ainsi, les trois exemples présentés traitent le problème d'ordonnancement\ allocation sur:

- Une architecture avec un seul bus partagé.
- Une architecture avec au moins deux bus.
- Une architecture avec au moins deux bus et avec l'existence de quelques liens directs entre les processeurs.

4. Représentation des solutions

Notre objectif consiste à optimiser l'ordonnancement /allocation dans les systèmes embarqués distribués temps réel. Pour ce faire, nous avons besoin d'une bonne représentation des solutions.

4.1 Première stratégie (AGC1): application des algorithmes génétiques classiques en affectant des priorités statiques aux tâches

Comme nous l'avons cité dans le chapitre précèdent, les algorithmes génétiques sont des méthodes basées sur les mécanismes biologiques tels que les lois de Mendel et le principe de sélection de Charles Darwin (1859). Holland a présenté l'algorithme génétique comme une abstraction de l'évolution biologique et il a exposé les principes de ces algorithmes passant du paradigme de l'évolution naturelle à celui de l'évolution artificielle. Ces principes constituent la base à tous les travaux sur ce type d'algorithmes.

Le grand avantage des algorithmes génétiques est qu'ils parviennent à trouver de bonnes solutions sur des problèmes très complexes, et trop éloignés des problèmes combinatoires classiques. Ils doivent simplement déterminer entre deux solutions qui est la meilleure, afin d'opérer leurs sélections. Ces algorithmes sont employés dans les domaines où un grand nombre de paramètres entrent en jeu, et où l'on a besoin d'obtenir de bonnes solutions en quelques itérations seulement.

Par la suite nous allons voir comment la résolution du notre problème est fait par le biais de l'algorithme génétique classique puis de l'algorithme génétique inspiré-quantique.

4.1.1 Organigramme de la première stratégie (AGC avec priorités statiques)

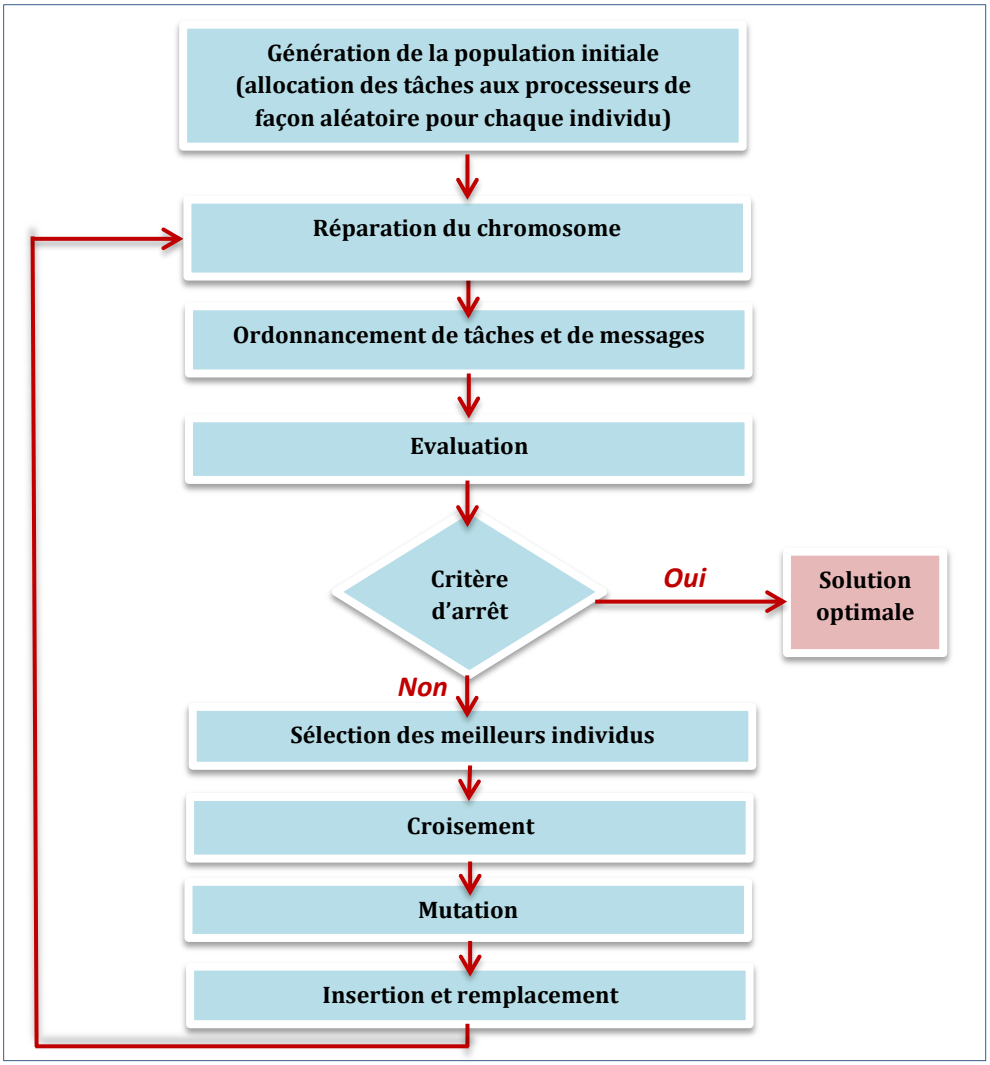
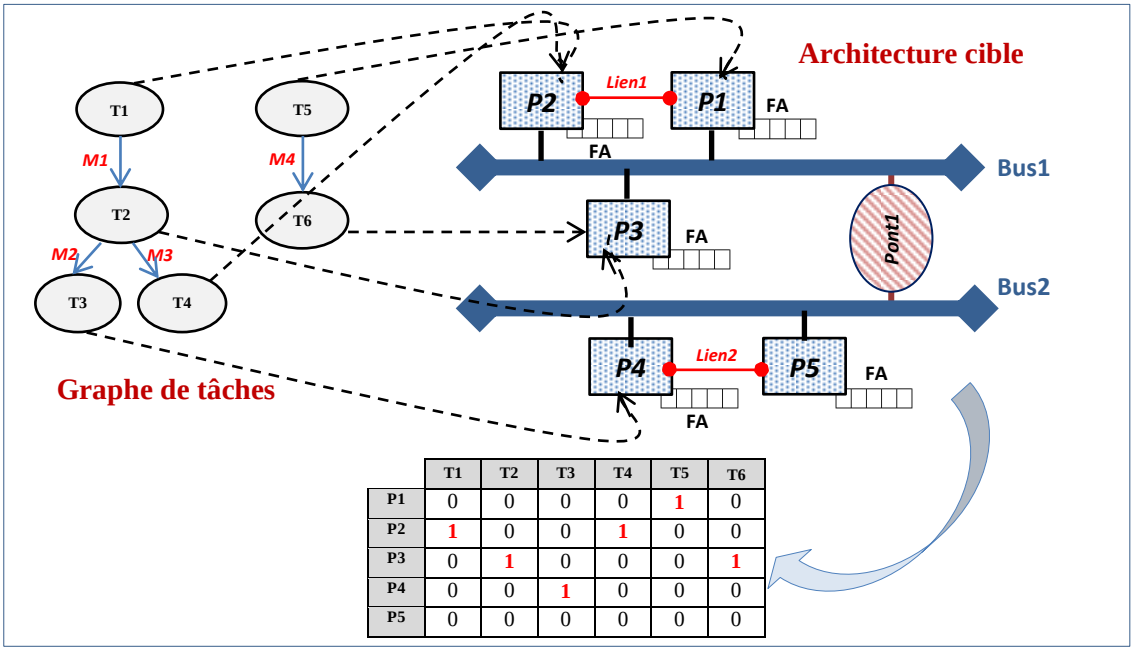


Figure 4.5 : Organigramme général de la première stratégie (AGC1).

4.1.2 Allocation des tâches

Les différentes tâches de notre système et leurs dépendances (les messages) représentées par le graphe de tâches devront être distribuées et ordonnancées sur les processeurs et les supports physiques de communication de l'architecture matérielle.

Nous avons utilisé une méthode aléatoire pour l'allocation des différentes tâches aux différents processeurs de l'architecture.



	T1	T2	T3	T4	T5	T6
P1	0	0	0	0	1	0
P2	1	0	0	1	0	0
P3	0	1	0	0	0	1
P4	0	0	1	0	0	0
P5	0	0	0	0	0	0

Figure 4.6 : Allocation des tâches aux processeurs.

4.1.2.1 Représentations du chromosome

La représentation adoptée est celle d'une matrice binaire, dont les colonnes correspondent au nombre de tâches de l'application, et les lignes correspondent au nombre de processeurs de l'architecture cible auxquels sont associées les tâches. Chaque case de cette matrice contient une valeur binaire. Si la valeur de la case (*i*, *j*) est égale à **1** cela signifie que la tâche numéro *j* est affectée au processeur numéro *i*.

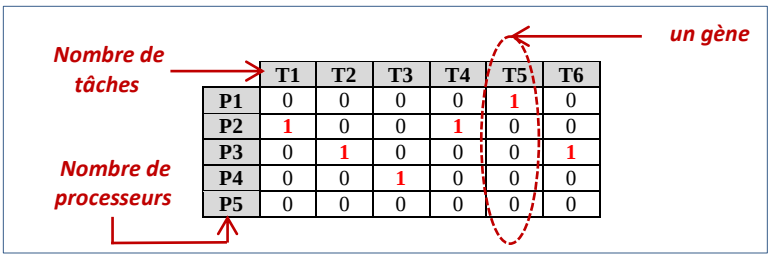


Figure 4.7 : Représentation du chromosome.

4.1.2.2 Initialisation de la population

La population initiale est générée de façon aléatoire. La population doit être suffisamment diversifiée pour une meilleure exploration de l'espace de recherche.

Notre initialisation consiste à allouer chaque tâche à un seul processeur de façon aléatoire. En effet, pour chaque tâche nous avons généré un nombre aléatoire pour tous les processeurs, la tâche sera ensuite attribuée au processeur qui a la plus grande valeur.

4.1.3 Réparation du chromosome

Cette phase est nécessaire pour éliminer le cas de l'existence d'un processeur avec une file d'attente vide et par conséquent améliorer la fitness du chromosome. Nous avons adopté une simple méthode afin de garantir une bonne distribution des tâches sur les processeurs.

La taille de la file d'attente de chaque processeur (*taille_fa*) est égale à :

- Au maximum : *taille_max_fa* = (*nb_tâches* /*nb_cpu*) +1 ;
- Au minimum : *taille_min_fa = nb_tâches* /*nb_cpu*.

Si la file d'attente d'un processeur contient un nombre de tâche supérieur à *taille_max_fa* alors :

- sélectionner une tâche **T**i de façon aléatoire;
- parcourir toutes les autres files d'attente, puis faire migrer **T**i à une file d'attente d'un processeur qui contient une file d'attente vide;
- refaire l'opération pour chaque file d'attente jusqu'à ce que toutes les files contiennent un nombre de tâches dans l'intervalle [*taille_min_fa* , *taille_max_fa*].

Après l'allocation de l'ensemble de tâches et la réparation du chromosome, les files d'attente des processeurs sont remplies, et l'algorithme d'ordonnancement sera prêt à être exécuté.

4.1.4 Ordonnancement de tâches et de messages

Nous nous intéressons ici, aux tâches temps réel modélisées à l'aide d'un graphe de tâches décrivant les dépendances entre ces tâches, et d'un graphe d'architecture décrivant les processeurs et les média de communication, qui peuvent être de types différents, sur lesquels devront s'exécuter les tâches en respectant des contraintes temps réel.

L'algorithme d'ordonnancement est utilisé dans le but de donner l'ordre d'exécution de l'ensemble de tâches temps réel de notre système afin de calculer en final le temps de réponse moyen, le taux d'utilisation des processeurs et le nombre de tâches respectant leurs échéances, chacune de ces valeurs représentent une fonction objective.

Nous avons choisi d'utiliser l'approche d'ordonnancement multiprocesseur par partitionnement qui consiste à partitionner l'ensemble de tâches à ordonnancer en sous-ensembles de tâches ordonnancés chacun sur un processeur, ce qui nous permet d'utiliser une stratégie d'ordonnancement temps réel monoprocesseur pour chaque processeur.

Nous présenterons donc dans la partie suivante l'algorithme d'ordonnancement que nous avons proposé, c'est un algorithme d'ordonnancement temps réel multiprocesseur et préemptif avec des contraintes de précédence, formé de trois algorithmes qui s'exécutent en parallèle sur l'architecture donnée, qui sont: l'algorithme d'ordonnancement des tâches périodiques, l'algorithme d'ordonnancement des tâches apériodiques et l'algorithme d'ordonnancement des messages.

4.1.4.1 Ordonnancement des tâches périodiques

Après la phase d'allocation des tâches aux différents processeurs de l'architecture, l'algorithme d'ordonnancement a encore besoin d'un ensemble de données pour qu'il puisse commencer son exécution, ces données sont des informations sur les tâches, les processeurs et les supports de communication de l'architecture.

Chaque tâche périodique est caractérisée par une période qui est égale à la période de leur sous-graphe, une échéance à respecter inferieur ou égale à sa période, une contrainte temporelle souple ou dure, un temps d'exécution au pire ou au moyen cas (WCET/ACET) et une date d'arrivée.

Bien que les systèmes temps réel soient infinis (les tâches périodiques s'exécutent infiniment), nous avons fait l'hypothèse que le temps de simulation (Temps-Sim) représentant la période d'étude de notre application est égale au plus petit commun multiple (PPCM) de périodes des sous-graphes périodiques.

A partir de ces informations, l'algorithme d'ordonnancement (l'ordonnanceur) peut générer deux listes, la première liste L_1 est construite pour l'insertion de toutes les tâches (périodiques et apériodiques) nouvellement arrivées et ordonnée de manière croissante selon

la date d'arrivée des tâches, tandis que la deuxième liste L_2 qui est associée à chaque processeur, est construite pour l'insertion des tâches périodiques prêtes à être exécutées et ordonnée de manière croissante selon les échéances relatives ou les périodes des tâches.

Nous utilisons les politiques d'ordonnancement temps réel « Deadline Monotonic (DM) et Rate Monotonic (RM) », pour l'ordonnancement des tâches.

Lorsqu'une tâche est arrivée, l'ordonnanceur vérifie d'abord si cette dernière est indépendante ou non, si c'est le cas, il doit l'insérer directement dans la liste L_2 pour qu'elle puisse être s'exécuté sur son processeur, sinon la tâche est inséré dans la liste L_1 selon sa date d'arrivée et elle doit attendre jusqu'à ce qu'elle devienne indépendante (reçoit un message de sa tâche prédécesseur).

Après la génération de ces deux listes, l'exécution des tâches aura lieu. L'ordonnanceur accède à la liste des tâches prêtes pour faire l'exécution de la première tâche T_i dans la liste sur le processeur associé. Si le processeur est libre, la tâche T_i est exécutée immédiatement et sans aucune attente, si non l'ordonnanceur vérifie si la tâche en cours d'exécution T_j a une priorité inferieure à la tâche T_i, si c'est le cas, la tâche T_j est interrompue et insérée dans la liste des tâches prêtes, tandis que la tâche T_i commence son exécution.

Si la tâche *Ti* termine son temps d'exécution, elle doit activer tous les messages sortant d'elle, puis elle doit être insérée dans la liste L_1 avec la prochaine date d'arrivée (égale à la nouvelle période + la date d'arrivée).

Nous sauvegardons la date de fin d'exécution de la tâche pour une utilisation ultérieure dans le calcul de temps de réponse moyen.

Si la période de la tâche est terminée, tous les paramètres de cette tâche sont réinitialisés.

Il convient de noter qu'il existe une tâche périodique parmi toutes les tâches du système, créé spécialement pour servir les demandes apériodiques.

4.1.4.2 Ordonnancement des tâches apériodiques

En plus des tâches périodiques, l'ordonnanceur doit faire l'exécution des tâches apériodique en même temps. Chaque tâche apériodique est caractérisée par un pseudo-période qui est égal à la moyenne des temps inter-arrivé de la tâche, une échéance à respecter, une

contrainte temporelle souple, un temps d'exécution au moyen cas (ACET) et une date d'arrivée.

L'ordonnanceur utilise ces informations pour construire la liste L_3 associée à chaque processeur. Cette liste contient toutes les tâches apériodiques prêtes à être exécutées et elle est ordonnée de manière croissante selon les échéances relatives ou les périodes de ces tâches.

Nous avons utilisé la méthode de serveur par scrutation (Polling Server) pour l'ordonnancement des tâches apériodiques, c'est une tâche périodique T_x chargée de la gestion de ce type de tâches, appelée serveur. Ce dernier est doté d'une priorité P_s, une période T_s et une capacité C_s et il est généralement ordonnancé suivant le même algorithme que les autres tâches périodiques.

Nous avons supposé que le serveur possède la plus basse priorité par rapport aux tâches périodiques, cela signifie qu'une tâche apériodique ne peut jamais interrompre une autre tâche périodique exécutée sur le même processeur. Notant que la politique d'ordonnancement des tâches apériodiques est similaire à celle des tâches périodiques (DM ou RM).

Lorsque le serveur est déclenché, sa capacité est remise à sa valeur initiale. Si la liste L_3 n'est pas vide, il commence à traiter les tâches apériodiques sur les processeurs associés, jusqu'à l'épuisement de sa capacité. S'il n'y aucune tâche en attente à exécuter, le serveur se suspend jusqu'à la prochaine activation et sa capacité est récupérée par les tâches périodiques. Notant que si une tâche apériodique arrive juste après la suspension du serveur, elle doit attendre le début de la prochaine période, où la capacité du serveur est renouvelée à sa pleine valeur.

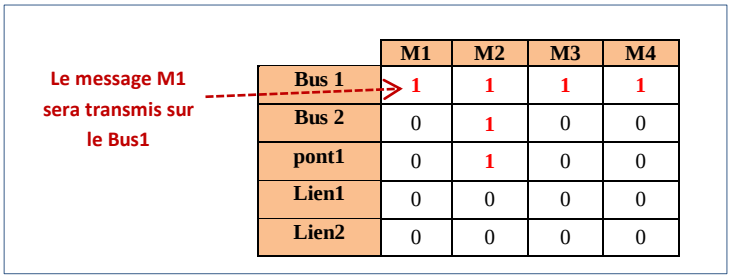
Si les processeurs associés aux tâches apériodiques sont occupés par des tâches périodiques, le serveur doit attendre jusqu'à ce que ces processeurs deviennent libres.

Dans le cas où une tâche apériodique termine son temps d'exécution, elle doit activer tous les messages sortant d'elle, puis elle doit être insérée dans la liste L_1 avec la prochaine date d'arrivée (égale à la nouvelle période + la date d'arrivée). Nous sauvegardons la date de fin d'exécution de la tâche pour l'utiliser dans le calcul de temps de réponse moyen.

Si la période de la tâche est terminée, tous les paramètres de cette tâche sont réinitialisés.

4.1.4.3 Ordonnancement des messages

Après la phase d'allocation des tâches aux différents processeurs de l'architecture, on peut facilement déduire le media de transfert de chaque message.



	M1	M2	M3	M4
Bus 1	1	1	1	1
Bus 2	0	1	0	0
pont1	0	1	0	0
Lien1	0	0	0	0
Lien2	0	0	0	0

Figure 4.8 : Allocation des messages de la figure 4.6 sur les supports physiques.

L'ordonnancement des messages est fait en parallèle avec les tâches. Chaque message est caractérisé par une taille, un mode activé ou désactivé, un numéro de la tâche source et un numéro de la tâche de destination.

Si un message est activé, l'ordonnanceur doit le transféré sur le bon support vers le bon processeur pour activer la tâche de destination. Et comme notre architecture est composée de supports de communication partagés, l'ordonnanceur doit trouver une politique pour ordonner les messages sur ces supports.

Nous avons supposé que chaque support (bus, pont ou lien) contient deux listes, la première pour les messages que le support va transférer et la deuxième pour seulement les messages prêts. Ces listes sont ordonnées de manière croissante selon les échéances relatives ou les périodes des tâches de destination des messages.

Si un message est activé, il doit premièrement être transféré sur un bus ou un lien. S'il existe un lien entre le processeur de la tâche source et le processeur de la tâche de destination, le message est obligatoirement transféré sur ce lien au lieu de bus, et le temps de transfert de message est ainsi calculé en divisant sa taille sur le débit de lien.

Si non, l'ordonnanceur accède à la liste des messages prêts des bus pour faire l'exécution du premier message M_i dans la liste sur le bus associé. Le temps de transfert de message est calculé en divisant sa taille sur le débit de ce bus. Si le bus est libre le message est directement transféré, si non l'ordonnanceur vérifie si l'échéance relative de la tâche de destination du message en cours d'exécution M_j est supérieure à celle du message M_i. Si c'est le cas, le

message M_j est interrompu et inséré dans la liste des messages prêts, tandis que le message M_i commence son temps de transfert.

Dans le cas où le message termine son temps de transfert sur le bus, l'ordonnanceur doit vérifier l'emplacement du processeur de la tâche de destination de ce message. Si ce dernier est connecté au même bus, la tâche de destination sera alors activée, si non le message est transféré sur le pont approprié (qui relie les deux bus). Le temps de transfert du message est encore calculé en divisant sa taille sur le débit du pont.

Après le transfert du message sur le pont, il sera transféré finalement sur le bus qui contient le processeur de la tâche de destination pour l'activer.

Il est convient de noter que si la tâche source et la tâche de destination d'un message sont affectées au même processeur, dans ce cas le temps de transfert de message est annulé et la tâche de destination est activée directement par la tâche source, cela nous montre que la méthode d'allocation des tâches a un grand effet sur la minimisation du temps de réponse moyen du système.

Lorsque la période du sous-graphe contenant le message est achevée, son transfert est arrêté et tous ses paramètres sont réinitialisés.

En fin, on associe pour chaque processeur et pour chaque support de communication un chronogramme de longueur *Temps-Sim*, pour représenter le déroulement de l'algorithme d'ordonnancement sur ces composants.

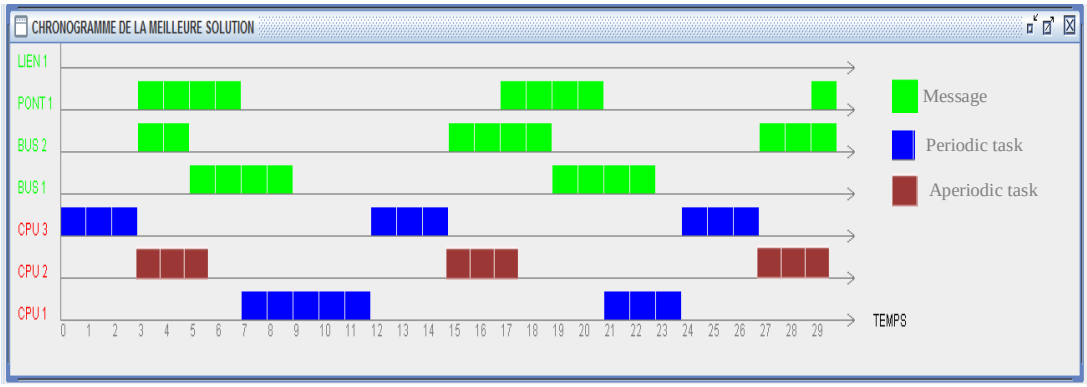


Figure 4.9 : Exemple d'un chronogramme général.

4.1.5 Evaluation des individus

Chaque individu de la population doit être évalué pour quantifier sa qualité ou sa fitness, en utilisant la fonction d'adaptation que l'on désire optimiser. Le résultat fournit par cette

fonction va permettre de ne choisir que les individus ayant le meilleur coût. Cela permet de s'assurer que les individus performants seront conservés pour les prochaines générations, alors que les individus moins adaptés seront progressivement éliminés de la population.

La fonction d'adaptation que nous avons voulu optimiser dans ce travail est représentée par l'un de ces deux facteurs: le temps de réponse moyen ou le nombre de tâches respectant leurs échéances.

Après l'exécution de l'algorithme d'ordonnancement, nous pouvons calculer ces facteurs.

4.1.5.1 Calcul du temps moyen de réponse

Le temps de réponse d'une tâche T_i est le temps qui s'écoule entre une demande d'activation d'une tâche et la fin d'exécution de cette tâche. Prise en compte du retard introduit par les autres tâches dans l'exécution de T_i.

A titre indicatif, une tâche est faisable si son temps de réponse dans le pire des cas est inférieur ou égal à son échéance. Un système est faisable si toutes les tâches qui le composent sont faisables.

Pour calculer le temps de réponse moyen de chaque tâche (TRM_i), nous avons calculé d'abord le temps de réponse de la tâche (Ti) dans chaque période (TR_i), puis la somme de toutes les valeurs obtenues est divisée par le nombre de fois d'activation de la tâche (nb_activ) dans le temps de simulation. La formule suivante montre la manière de calculer le temps de réponse moyen pour chaque tâche :

$$TRM_i = \frac{\sum_{i=1}^{nb\_activ} TRi}{nb\_activ}$$

Formule 4.2 : Calcul du temps de réponse moyen pour chaque tâche.

Le temps de réponse moyen de tout le système ($TRMS$) est ainsi calculé en divisant la somme de temps de réponse moyen de toutes les tâches par le nombre de tâches du système, comme le montre la formule suivante :

$$TRMS = \frac{\sum_{i=1}^{nb\_tâches} TRMi}{nb\_tâches}$$

Formule 4.3 : Calcul du temps de réponse moyen du système.

4.1.5.2 Calcul du taux d'utilisation des processeurs

Le taux d'utilisation (d'occupation) de chaque processeur *(TU_j)* est égal au nombre d'unités d'occupation du processeur (U_{occup}), divisé par le temps de simulation *Temps-Sim.*

$$TU_j = \frac{\sum_{k=1}^{Temps\_Sim} Uoccup}{Temps\_Sim}$$

Formule 4.4 : Calcul du taux d'utilisation de chaque processeur.

Pour calculer le taux moyen d'utilisation de tous les processeurs du système (*TMU*), nous avons divisé la somme des taux d'utilisation de chaque processeur (TU_j) sur le nombre total de processeurs (*nb_cpu*).

$$TUM = \frac{\sum_{j=1}^{nb\_cpu} TUj}{nb\_cpu}$$

Formule 4.5 : Calcul du taux moyen d'utilisation de tous les processeurs.

4.1.5.3 Calcul du nombre de tâches respectant leurs échéances

Dans notre algorithme d'ordonnancement, une variable (*compteur)* est incrémentée à chaque fois qu'une tâche ne respecte pas son échéance.

Si l'échéance d'une tâche est achevée, l'ordonnanceur vérifie si la tâche n'a pas encore terminé son temps d'exécution, dans ce cas la tâche ne respecte pas son échéance et la valeur du *compteur* est incrémentée.

A titre indicatif la minimisation du nombre de tâches souples dépassant leurs échéances sert à augmenter la qualité du résultat.

4.1.6 Critère d'arrêt

Le critère d'arrêt joue un rôle très important dans le jugement de la qualité des individus. Nous avons choisi d'utiliser un nombre fixe de générations comme critère d'arrêt. Si ce dernier est atteint, l'algorithme est arrêté et la solution optimale est ainsi obtenue. Si non, l'application des opérateurs génétiques aura lieu.

4.1.7 Sélection

Cet opérateur consiste à choisir les individus à partir desquels va être obtenue la génération suivante ou les chromosomes enfants.

La sélection des parents est basée sur la valeur de la fitness des chromosomes. Nous avons adopté deux méthodes de sélection : par élitisme et par tournoi.

Avant de sélectionner les meilleurs individus, nous avons d'abord choisi un facteur à optimiser parmi les deux cités dans la section précédente.

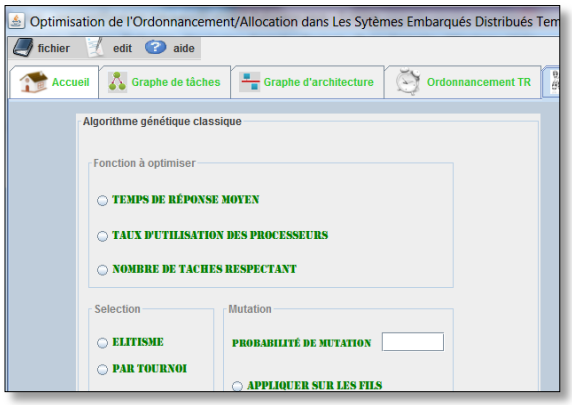


Figure 4.10 : Interface graphique pour choisir un facteur à optimiser.

A base de ce facteur les individus de la population sont triés, et mis dans une liste nommée (L_{select}).

4.1.8 Croisement

Dans un algorithme génétique, cet opérateur permet d'enrichir la population. Il permet de croiser deux parents et génère un ou deux enfants.

Après avoir sélectionné un certain nombre d'individus, l'opérateur de croisement aura lieu. Deux méthodes de croisement sont aussi implémentées, un simple croisement en un seul point et un autre en deux points. La particularité du croisement que nous avons employé est que chaque deux parents sélectionné pour le croisement représentent respectivement le meilleur et le mauvais individu de la population. Cela signifie que le premier individu de la liste (L_{select}) est croisé avec le dernier, ensuite le second individu est croisé avec l'avant dernier, et ainsi de suite.

Dans le cas du croisement en un seul point, le point de coupure n'est pas choisi de façon aléatoire, il est calculé à base de la valeur de la fitness des parents. En effet, nous avons calculé le pourcentage de la fitness du premier parent (fit_1) par rapport à celui du deuxième (fit_2) comme suit :

$$Pourcentage = \frac{fit1}{fit1 + fit2} * 100$$

Formule 4.6 : Calcul du pourcentage de la fitness d'un parent.

Puis avec l'utilisation de la règle de trois nous avons déduit le point de coupure *pc* correspondant :

$$\left.\begin{array}{l} nb\_tâches \rightarrow 100\,\% \\ pc \rightarrow pourcentage \end{array}\right\} \quad pc = \frac{\text{nb\_tâches} * \text{pourcentage}}{100}$$

Formule 4.7 : Calcul du point de coupure de croisement.

Supposant que fit_1= 50, fit_2=30 et nb_tâches = 5, cela nous donne :

→$Pourcentage = \frac{50}{80} * 100 = 62.5\%$.

→Le point de coupure *pc* est alors égale à : 5 * 62.5 / 100 ≈ 3.

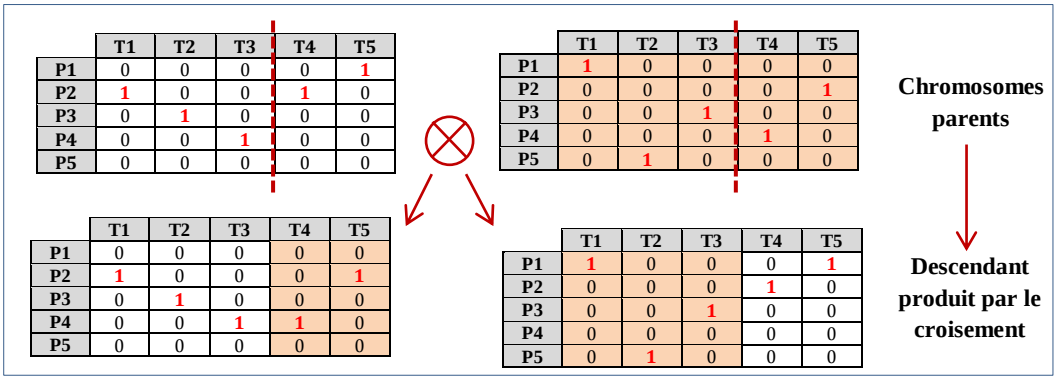


Figure 4.11 : Exemple de croisement en un point.

Dans le cas du croisement en deux points, les points de coupure sont choisis de façon aléatoire :

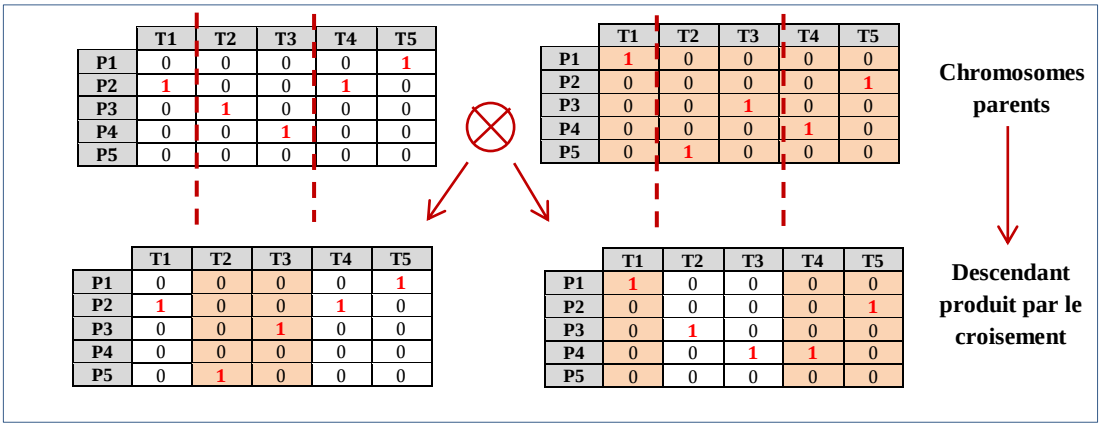


Figure 4.12 : Exemple de croisement en deux points.

4.1.9 Mutation

La mutation vise à explorer l'espace de solutions et permet d'échapper aux minimums locaux. Cet opérateur est utilisé avec une probabilité P_{mut}.

Dans notre travail, nous générons pour chaque individu enfant un nombre aléatoire, si ce nombre appartient à [0, P_{mut}], alors nous appliquons l'opérateur de mutation sur cet individu.

Une fois l'individu est choisi pour la mutation, un de ses gènes est choisi aléatoirement, puis nous affectons une permutation entre le processeur chargé d'exécuter la tâche et un autre choisi aléatoirement.

Nous avons appliqué une mutation soit directement sur les individus de la population initiale, ou soit sur les individus résultants du croisement. La figure 4.13 présente une mutation appliquée sur un individu choisi de façon aléatoire :

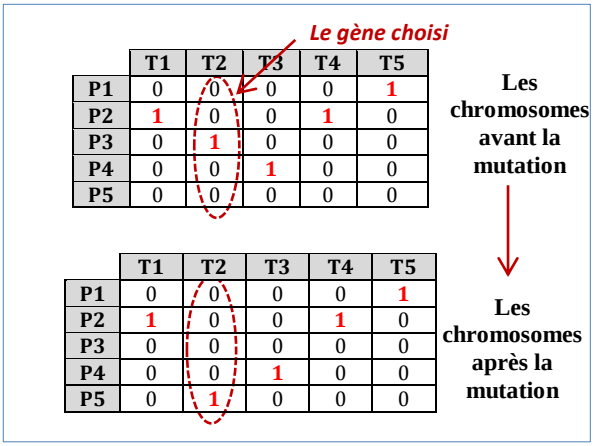


Figure 4.13 : Exemple de la mutation.

4.1.10 Insertion et remplacement

Après l'application des opérateurs de croisement et de mutation, nous avons utilisé une méthode d'insertion pour remplacer les mauvais individus et générer la nouvelle population.

4.1.11 Tests et résultats de la première stratégie

Nous avons réalisé un ensemble de tests en appliquant l'AGC avec priorités statiques.

- **Premier exemple :**

Nous avons choisi de traiter le problème de l'ordonnancement\allocation premièrement sur une architecture contenant un seul bus partagé :

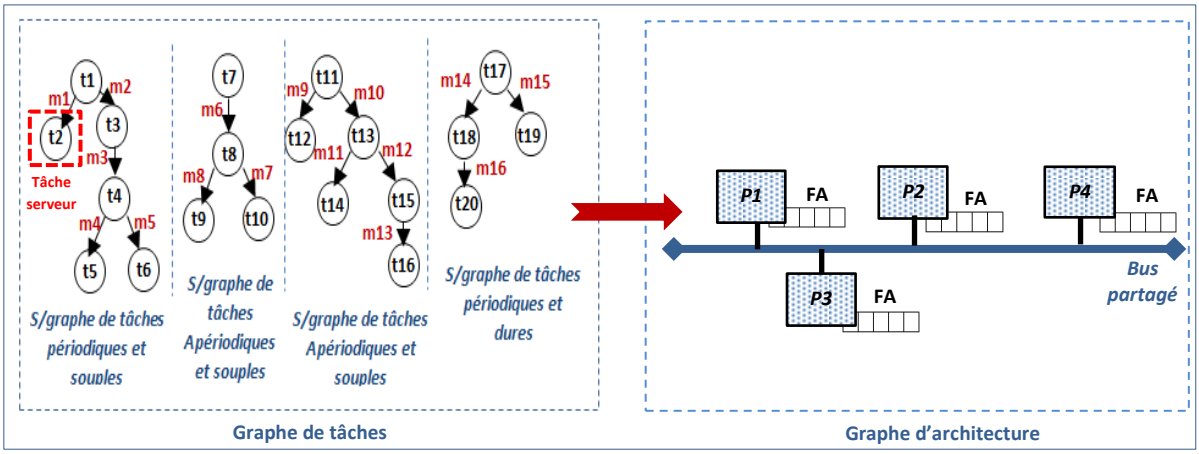


Figure 4.14 : Le premier exemple (exemple d'un seul bus partagé).

Le tableau ci-dessous résume les différents jeux de paramètres appliqués au premier exemple, ainsi que les résultats obtenus. Les abréviations suivantes sont utilisées:

Nb_Tach : nombre de toutes les tâches (périodiques et apériodiques), **Nb_Cpu :** nombre de processeurs de l'architecture, **Nb_Bus :** nombre de bus, **Nb_Pont :** nombre de ponts, **Nb_Lien :** nombre de liens, **Nb_Gen :** nombre de générations, **Nb_indiv :** nombre d'individus, **Tps_Sim :** le temps de simulation, **Factr à Optim :** soit optimiser le temps de réponse moyen du système **« Trm »**, soit le nombre de tâches dépassant leurs échéances **« N.t.d »**, **Selc :** la méthode de sélection choisie (élitisme ou tournoi), **Crois :** la méthode du croisement choisi (en 1 ou en 2 points), **Mut :** probabilité de mutation, **Politiq_ordon:** affecter des priorités statiques aux tâches selon les deadlines **(DM)** ou selon les périodes **(RM), T.u.m Cpu :** taux d'utilisation moyen des processeurs.

test	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Selc	Crois	Mut	Politiq ordon	Résultats		
														Trm	T.u.m Cpu%	N.t.d
A1	20	03	01	00	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	RM	13.8	33.0	6
A2	20	03	01	00	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	DM	12.55	33.0	3
A3	20	03	01	00	00	100	60	60	Trm	Tn	1 Pnt	P=0.3	DM	10.75	33.0	2
A4	20	03	01	00	00	100	60	60	Trm	Tn	1 Pnt	P=0.8	DM	12.4	33.0	4
A5	20	03	01	00	00	100	60	60	N.t.d	Tn	1 Pnt	P=0.3	DM	11.06	33.0	0
A6	25	03	01	00	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	RM	11.96	43.0	3
A7	25	04	01	00	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	DM	11.08	35.0	3
A8	25	04	01	00	00	100	100	60	Trm	Elit	2 Pnts	P=0.3	DM	10.48	35.0	2

- **Deuxième exemple :**

Le problème est traité maintenant sur une architecture contient au moins deux bus :

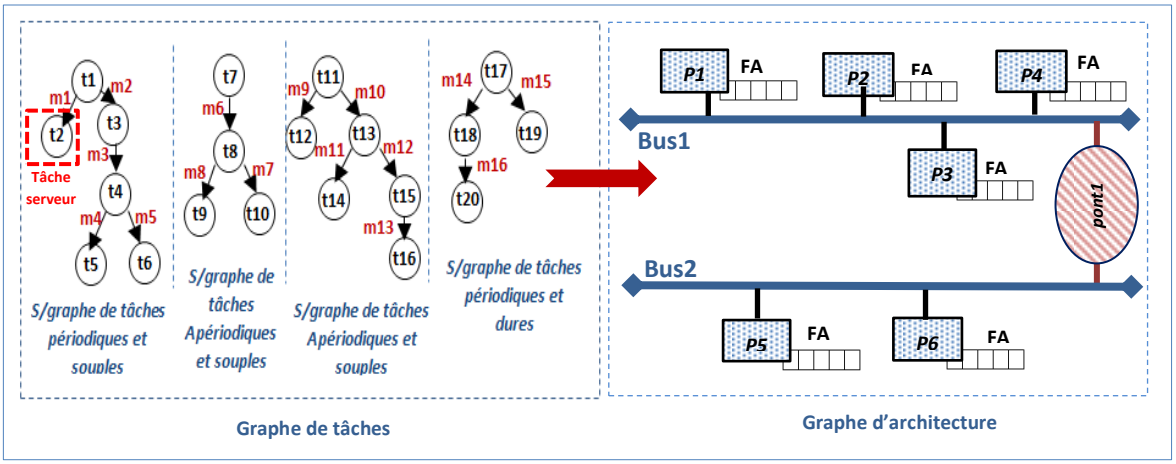


Figure 4.15 : Le deuxième exemple (exemple avec au moins deux bus).

Le tableau ci-dessous résume les différents jeux de paramètres appliqués au deuxième exemple:

Test	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indi	Tps Sim	Factr à Optim	Selc	Crois	Mut	Politiq ordon	Résultats		
														Trm	T.u. Cpu%	N.t.d
B1	20	03	02	01	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	RM	13.95	33.0	7
B2	20	03	02	01	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	DM	11.65	33.0	4
B3	20	03	03	03	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	DM	12.3	33.0	6
B4	20	03	02	01	00	100	60	60	Trm	Tn	1 Pnt	P=0.8	DM	12.65	33.0	0
B5	20	03	02	01	00	100	60	60	N.t.d	Tn	1 Pnt	P=0.3	DM	14.05	33.0	0
B6	25	03	02	01	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	RM	14.92	43.0	5
B7	25	04	02	01	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	DM	12.68	35.0	4
B8	25	04	03	03	00	100	100	60	Trm	Elit	2 Pnts	P=0.3	DM	10.0	35.0	2

Tableau 4.4 : Paramètres et résultats de l'AGC1 appliqués au 2eme exemple.

- ## Troisième exemple :

Le troisième exemple traite le problème sur une architecture contient au moins deux bus avec quelques liens directs entre les processeurs:

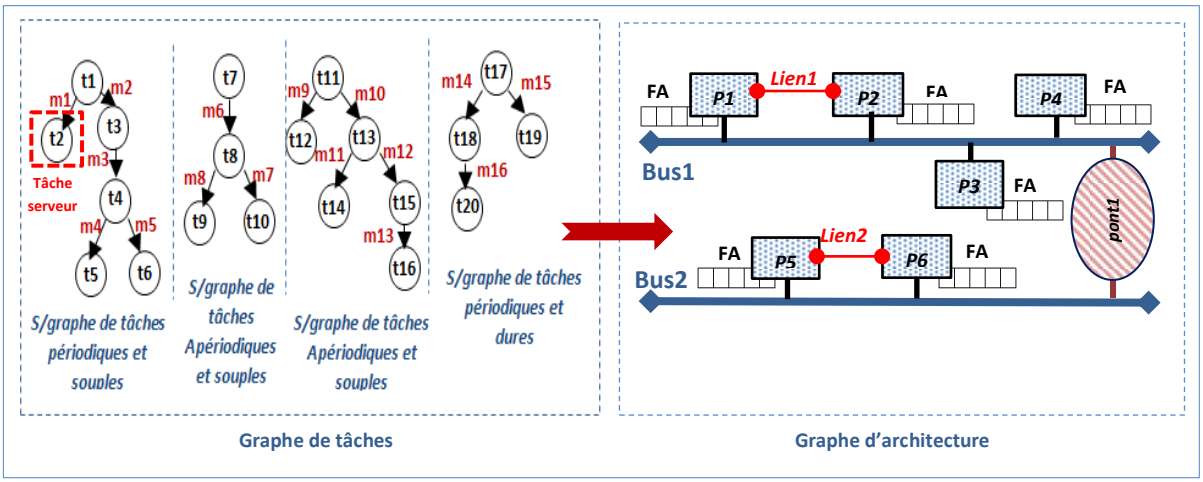


Figure 4.16 : Le troisième exemple (au moins deux bus + liens directs).

Le tableau ci-dessous résume les différents jeux de paramètres appliqués au troisième exemple:

N° test	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Selc	Crois	Mut	Politiq ordon	Résultats		
														Trm	T.u. Cpu%	N.t.d
C1	20	03	02	01	01	100	60	60	Trm	Elit	1 Pnt	P=0.3	RM	11.3	33.0	3
C2	20	03	02	01	01	100	60	60	Trm	Elit	1 Pnt	P=0.3	DM	12.0	33.0	4
C3	20	04	02	01	01	100	60	60	Trm	Elit	1 Pnt	P=0.3	DM	10.6	27.0	2
C4	20	03	02	01	01	100	60	60	Trm	Tn	1 Pnt	P=0.8	DM	11.0	33.0	1
C5	20	03	02	01	01	100	60	60	N.t.d	Tn	1 Pnt	P=0.3	DM	10.4	33.0	0
C6	25	03	02	01	01	100	60	60	Trm	Elit	1 Pnt	P=0.3	RM	14.84	43.0	5
C7	25	04	02	01	02	100	60	60	Trm	Elit	1 Pnt	P=0.3	DM	11.1	35.0	1
C8	25	04	03	03	01	100	100	60	Trm	Elit	2 Pnts	P=0.3	DM	11.84	35.0	4

Tableau 4.5 : Paramètres et résultats de l'AGC1 appliqués au 3eme exemple.

4.1.12 Discussion

Nous avons essayé de fixer à chaque fois un paramètre et d'apercevoir leur influence sur les résultats.

D'après les résultats des trois tableaux ci-dessus, il est possible de remarquer que dans la plupart des tests effectués sur les trois exemples, le *Trm* est diminué à chaque fois que le *Ntd* est réduit (*exp : C1,C2 et C3*), cela s'explique par le fait que si *Ntd =0,* cela implique que le temps de réponse de chaque tâche est limité dans l'intervalle : *In* =] 0, *Dt*], (*Dt = deadline de la tâche*), et à chaque fois que le *Ntd* augmente, l'intervalle « *In* » a étendu et par conséquent le *Trm* est augmenté.

Mais d'après ce qu'on remarque aussi (*exp: test A3 et A5*), il se peut que le *Trm* soit diminué alors que le *Ntd* augmente. Cela peut être s'expliqué par le fait que le temps de réponse de chaque tâche est petit malgré que certaines tâches dépassent leurs échéances, ce qui nous donne au total un *Trm* réduit.

Il est convient d'ajouter que le *Trm* est calculé seulement lorsque toutes les tâches respectent leurs périodes, cela signifie que la solution qui contient au moins une tâche dépassant sa période, sera écartée complètement de l'ensemble de solutions candidates.

Donc le *T.u.m.Cpu* de trois processeurs vaut toujours une valeur max (*exemple : T.u.m.Cpu=33% pour 20 tâches et T.u.m.Cpu =43% pour 25 tâches*), parce que toutes les tâches ont réussi à terminer leurs temps d'exécution avant ses périodes.

Aussi, nous avons essayé d'augmenter le nombre de tâches avec le même nombre de processeurs, le résultat obtenu montre une augmentation du *T.u.m. Cpu (exp : test C6)*.

Nous avons également essayé de changer la politique d'ordonnancement (*exp test B1et B2*) et selon le nombre de tests effectué sur les trois exemples, nous avons remarqué que dans la plupart des cas l'algorithme *DM* donne les meilleurs résultats par rapport à *RM*, cela est due aux périodes des tâches qui sont un peu grandes et que les tâches les plus urgentes sont négligées par *RM*, ce qui influe directement sur le *Ntd* ainsi que le *Trm*.

Si on compare maintenant entre les résultats des trois tableaux, on peut dire que l'augmentation du nombre des bus sert à augmenter le *Ntd* (*exp : A1 et B1*) et cela due au temps perdue qui s'écoule pour le transfert des messages sur les média de transfert (bus et pont) ce qui retarde l'exécution des tâches de destination de ces messages. L'augmentation du nombre de processeurs sert à diminuer le *Ntd*.

L'utilisation des liens directs entre les processeurs améliore un peu les résultats (*exp : test B1 et C1*). Cela s'explique par le fait que le débit des liens directs est plus élevé par rapport à celui des bus, ce qui influe sur la vitesse de transfert du message. Cet avantage joue un rôle très important pour la minimisation du *Trm* de tout le système.

Après la réalisation de plusieurs tests, nous avons déduit que la valeur la plus intéressante du temps de réponse moyen pour chaque exemple est indiquée par le tableau suivant :

Exemple	Temps de réponse moyen (3 Cpus)	Taux d'utilisation des Cpus	Nombre de tâches dépassant l'échéance
1	10.75	33.0 %	3
2	11.65	33.0 %	4
3	10.4	33.0 %	0

Tableau 4.6: les meilleurs résultats obtenus par l'AGC1 pour 20 tâches.

4.1.13 Résultats des trois exemples

Nous avons refait les tests sur les trois exemples. Cette fois, nous avons fixé les paramètres de la stratégie (**nb_tâches = 30, nb_cpu = 6 , Politiq_ordon = DM, Tps_Sim = 60, taille_pop = 100**), et nous avons varié le paramètre du nombre de génération :

Exemples	Nombre de génération	Temps de réponse moyen	Taux d'utilisation des Cpu	Nombre de tâches dépassant l'échéance
Exemple 01 **(bus partagé)**	Itér = 30	13.87	23%	9
	Itér = 60	12.33	23%	9
	Itér = 80	12.1	23%	9
	Itér = 100	11.8	23%	9
Exemple 02 **(3 bus, 3 ponts)**	Itér = 30	11.03	23%	4
	Itér = 60	10.7	23%	7
	Itér = 80	10.7	23%	6
	Itér = 100	10.7	23%	4
Exemple 03 **(3 bus, 3 ponts, 2 liens)**	Itér = 30	11,77	23%	9
	Itér = 60	11,6	23%	9
	Itér = 80	10,97	23%	8
	Itér = 100	10.6	23%	3

Tableau 4.7: Les résultats des trois exemples obtenus par l'AGC1.

Les graphes (a) et (b) de la figure 4.17 résument les résultats des trois exemples du tableau 4.7 :

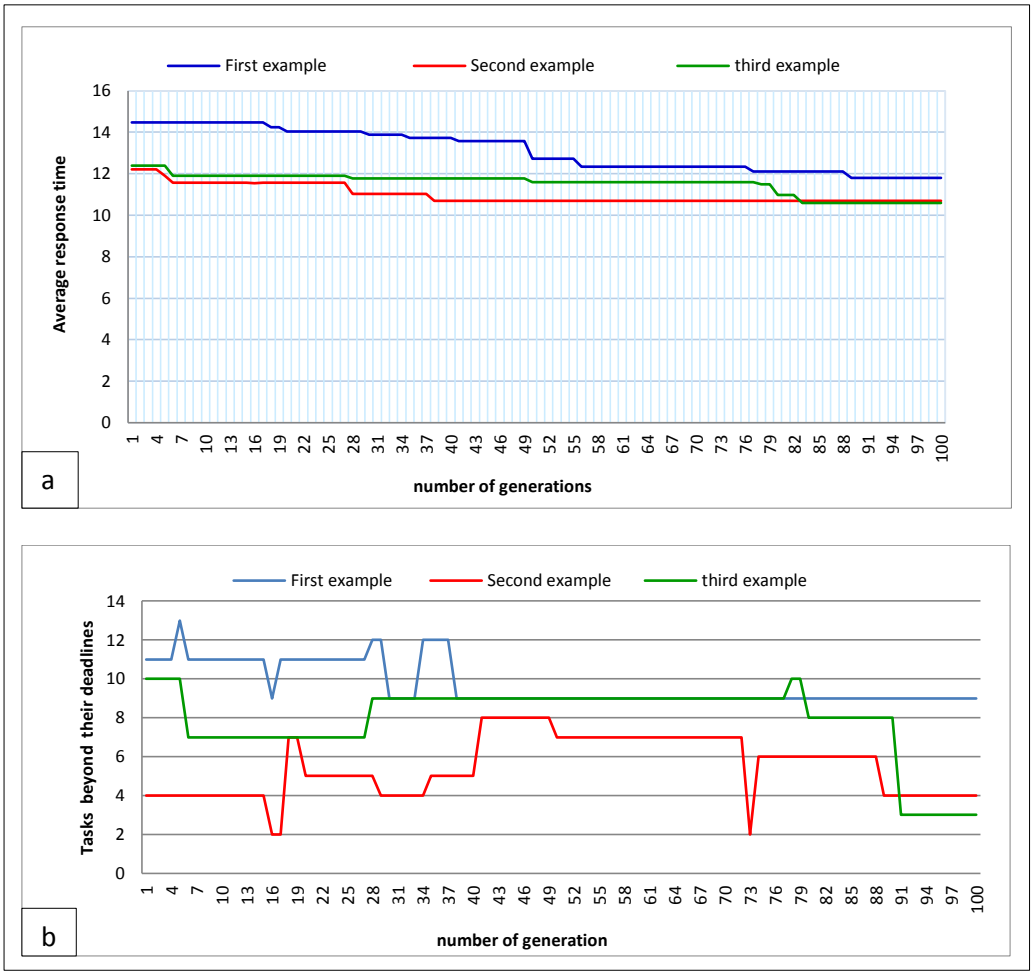


Figure 4.17 : Résultats des trois exemples sous forme de graphes.

- **Discussion**

D'après ce qu'on apercevait sur les courbes de la figure (*4.17 -a-*), on remarque bien que dans cette exemple (*30 tâches*), le *Trm* atteint une valeur minimale dans le troisième exemple (*Trm=10.6*) où nous avons utilisé une architecture composé de trois bus reliés par trois ponts et avec l'existence de deux liens directs.

Les valeurs du *Trm* du deuxième exemple sont un peu proches par rapport à celles du troisième exemple, par contre le premier exemple donne toujours des grandes valeurs. Cela est expliqué par le fait que l'augmentation du nombre de bus dans cet exemple, sert à minimiser le conflit des messages par rapport au cas d'un seul bus ce qui accélère leurs transfert et par conséquent minimiser le *Trm* des tâches de destination. L'utilisation des liens directs entre les processeurs sert à réduire le *Trm* en déchargeant les bus de transférer certains messages ce qui permet de gagner un peu de temps.

Le *Ntd* est un paramètre très important pour mesurer la qualité des solutions. Sur les courbes de la figure (*4.17 -b-*), on remarque que les meilleures valeurs de ce paramètre sont atteint par le troisième exemple (*Ntd= 3*), ainsi que si le *Ntd* diminue le *Trm* diminue aussi. L'explication de cette remarque c'est que l'utilisation des liens aide à transférer certains messages dans des délais un peu courts par rapport au bus ce qui aide à baisser le *Ntd* et par conséquent, l'intervalle des valeurs de temps de réponse de chaque tâche est diminué, ce qui donne un *Trm* réduit. Il existe quelques cas (*exemple 2 : itér =60 et itér= 80*) où l'augmentation du *Ntd* n'a aucun effet sur le *Trm*, cela signifie que le temps de réponse des tâches est petit, ce qui donne au total un *Trm* réduit.

Les figures 4.18, 4.19 et 4.20 représentent les chronogrammes des trois exemples du tableau 4.7 :

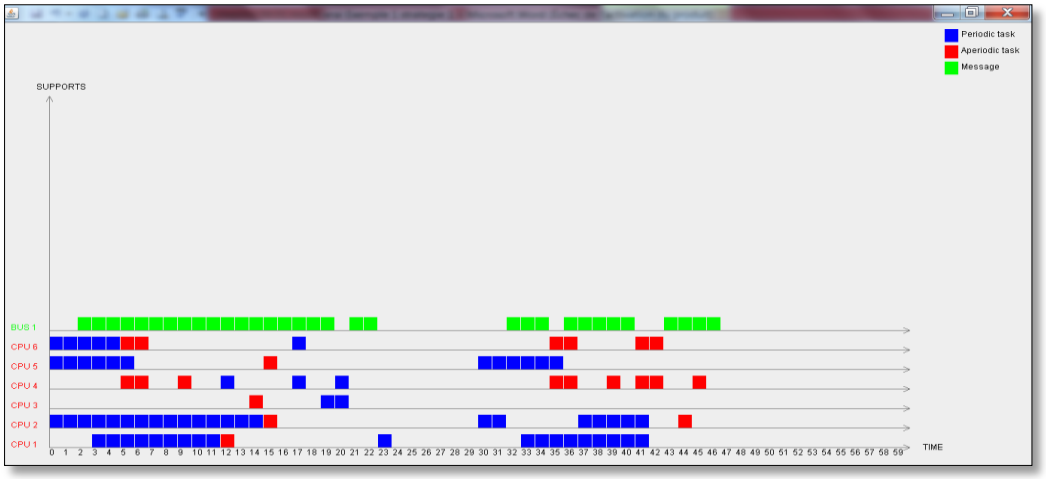


Figure 4.18 : Chronogramme de l'ordonnancement de la meilleure solution (le premier exemple).

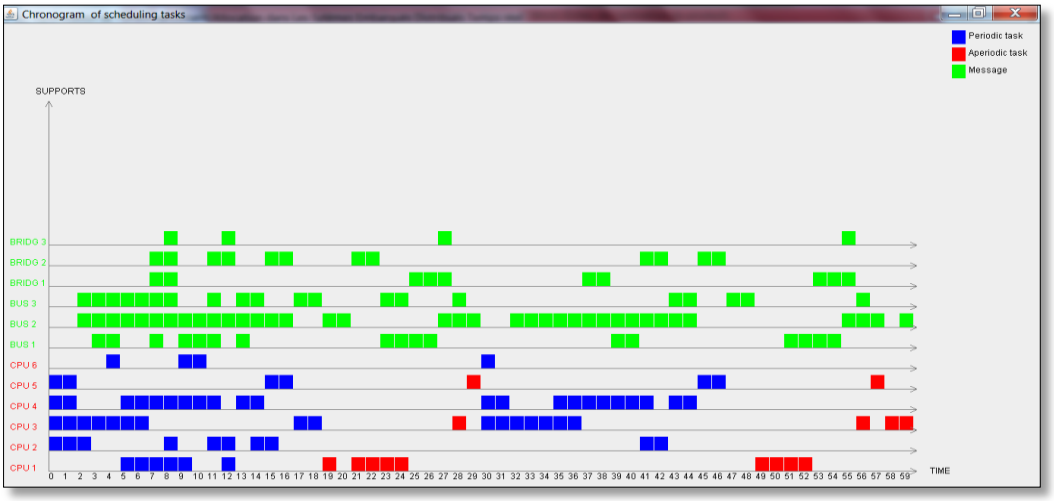


Figure 4.19 : Chronogramme de l'ordonnancement de la meilleure solution (le deuxième exemple).

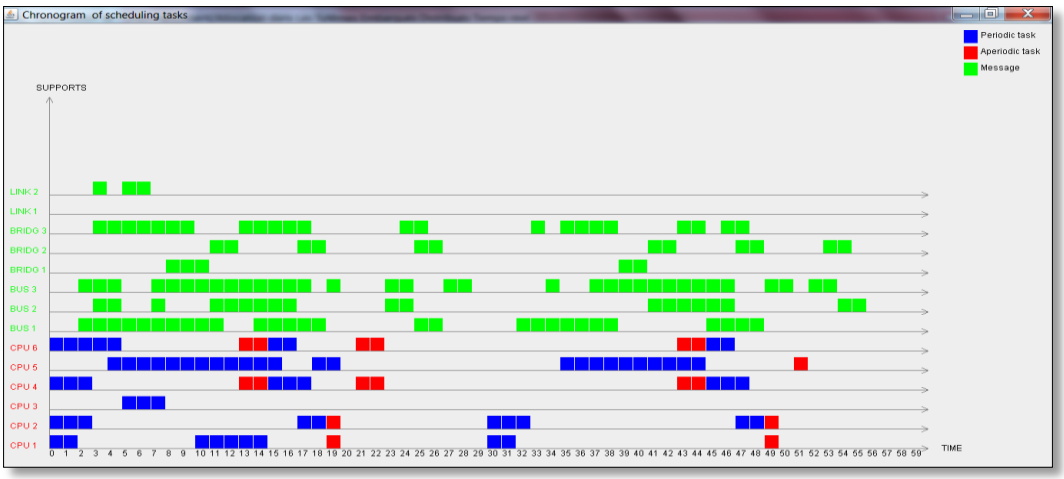


Figure 4.20 : Chronogramme de l'ordonnancement de la meilleure solution (le troisième exemple).

Les figures 4.21, 4.22 et 4.23 représentent les taux d'occupation des processeurs des trois exemples du tableau 4.7 :

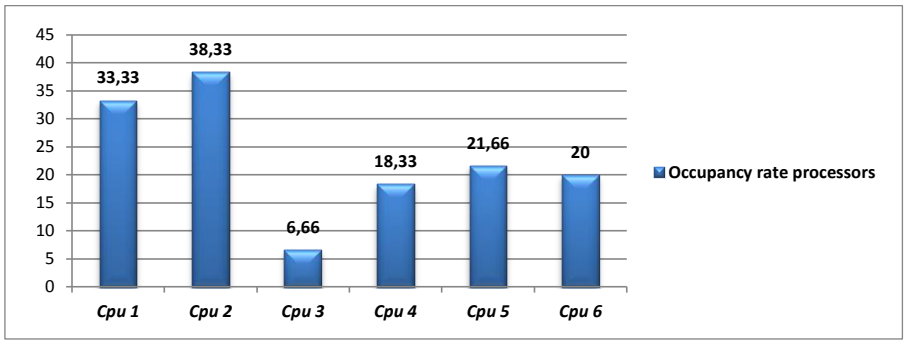


Figure 4.21 : Taux d'occupation des processeurs de la meilleure solution (premier exemple).

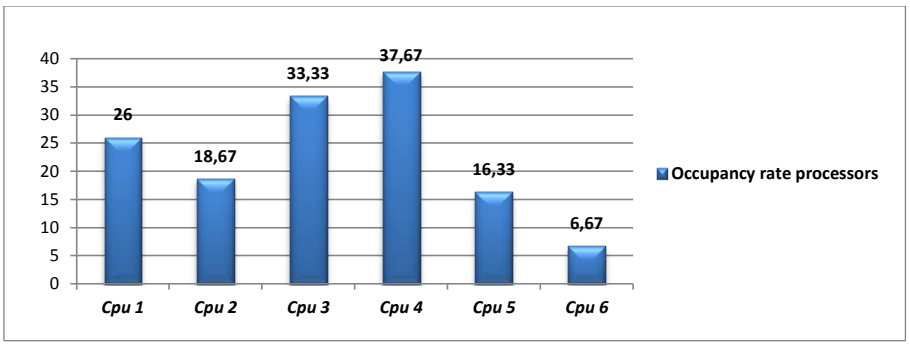


Figure 4.22 : Taux d'occupation des processeurs de la meilleure solution (deuxième exemple).

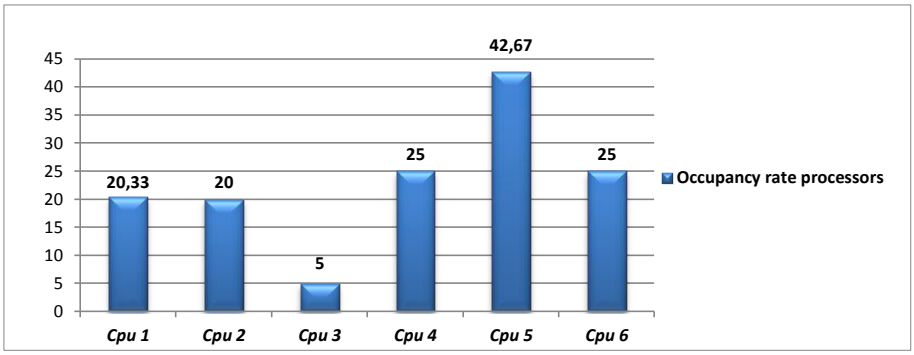


Figure 4.23 : Taux d'occupation des processeurs de la meilleure solution (troisième exemple).

4.2 Deuxième stratégie (AGC2):application des algorithmes génétiques classiques en affectant des priorités dynamiques aux tâches

Dans cette stratégie, nous avons essayé d'appliquer un ordonnancement à priorité dynamique. Nous avons premièrement affecté de façon aléatoire des priorités aux tâches, puis nous avons tenté de les changer à chaque génération dans l'espoir de trouver une bonne affectation. Les grandes lignes de cette stratégie sont décrites dans l'organigramme suivant :

4.2.1 Organigramme de la deuxième stratégie (AGC avec priorités dynamiques)

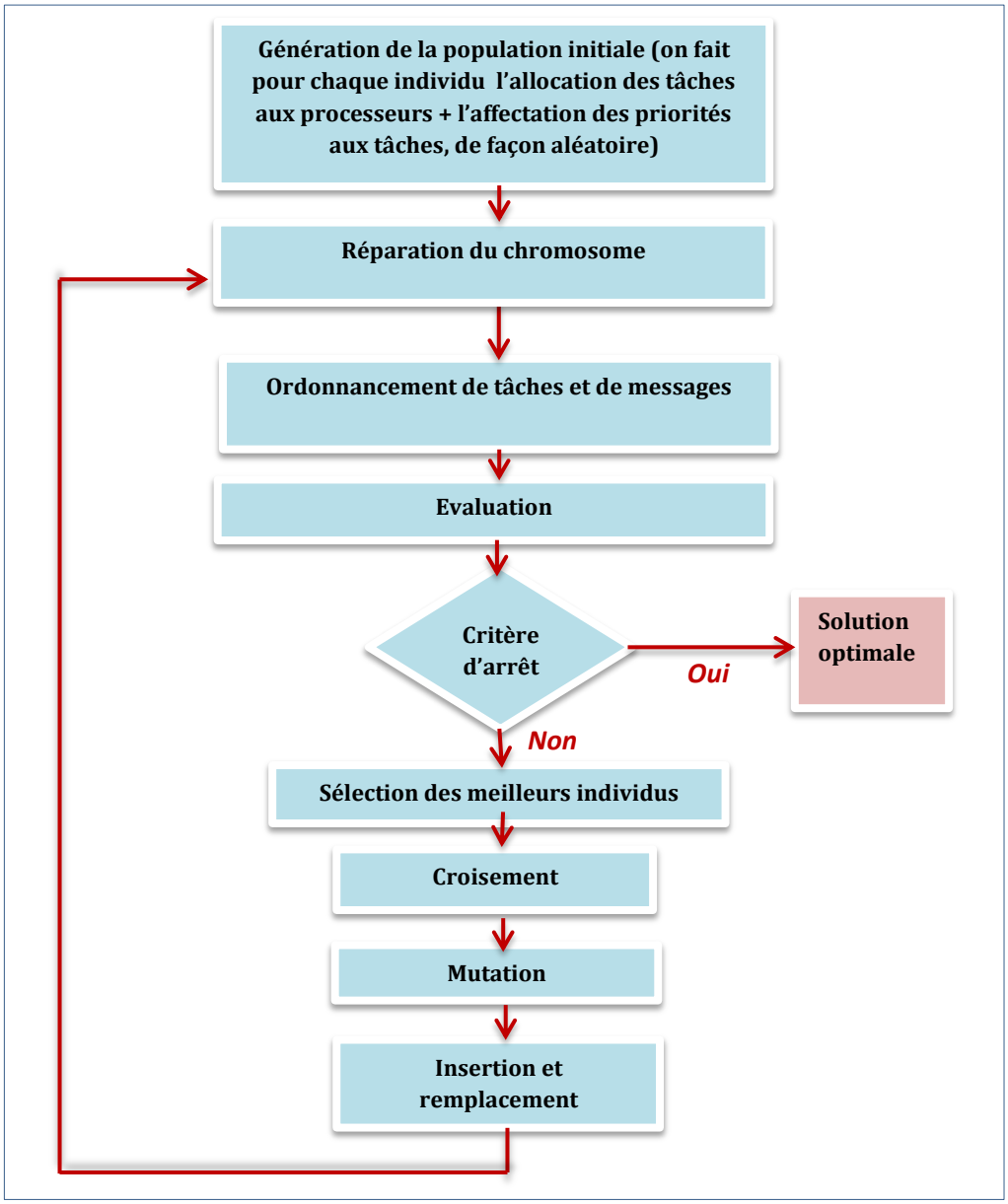


Figure 4.24: Organigramme général de la deuxième stratégie (AGC2).

4.2.2 Allocation des tâches

Nous avons utilisé une méthode aléatoire pour l'allocation des différentes tâches aux différents processeurs de l'architecture.

4.2.2.1 Représentations du chromosome

Dans cette stratégie, chaque chromosome est représenté par deux matrices binaires. La première matrice représente l'allocation des tâches aux processeurs dont les colonnes correspondent au nombre de tâches de l'application, et les lignes correspondent aux nombre de processeurs de l'architecture cible auxquels sont associées les tâches. Chaque case de cette matrice contient une valeur binaire. Si la valeur de la case (*i*, *j*) est égale à **1** cela signifie que la tâche numéro *j* est affectée au processeur numéro *i*. Tandis que la deuxième matrice représente l'affectation des priorités aux tâches dont les colonnes correspondent au nombre de tâches de l'application, et les lignes correspondent aux nombre de priorités affectés aux tâches.

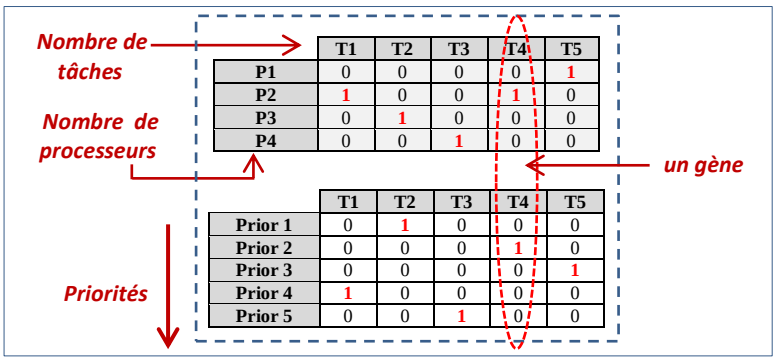


Figure 4.25 : Représentation du chromosome de la deuxième stratégie (AGC2).

Nous avons supposé que chaque tâche possède une priorité différente, cela signifie que la taille de la matrice d'affection des priorités aux tâches est égale à $(nb\_tâches)^2$. Si la valeur de la case (*i*, *j*) de cette matrice est égale à **1** cela signifie que la tâche numéro *j* possède une priorité égale à $\mathbf{prior}_i$. Notant que les priorités sont ordonnées de façon croissante, cela désigne que « **Prior 1** » est la plus faible priorité.

4.2.2.2 Initialisation de la population

Comme dans la première stratégie, nous avons utilisé une population initiale générer de façon aléatoire. Chaque individu est composé d'une matrice d'allocation des tâches et d'une matrice d'affectation des priorités, générées de façon aléatoire.

4.2.3 Réparation du chromosome

Cette phase est nécessaire, d'une part pour garantir que les files d'attente de tous les processeurs ne sont pas vides, et d'autre part pour garantir que l'affectation des priorités aux tâches respecte certaines conditions.

Pour chaque individu, la fonction de réparation est appliquée premièrement sur la matrice d'allocation des tâches comme dans la première stratégie, et une autre fonction de réparation est appliquée sur la matrice d'affectation des priorités afin de garantir les conditions suivantes :

- Respecter les relations de précédence : une tâche T_i qui précède une tâche T_j doit toujours être plus prioritaire par rapport à la tâche T_j.
- Deux tâches indépendantes ne doivent pas porter la même priorité.

4.2.4 Ordonnancement de tâches et de messages

Le déroulement de l'algorithme d'ordonnancement est presque le même que dans la première stratégie. La distinction principale réside dans les priorités des tâches qui sont devenus dynamiques.

L'ordonnanceur accède à la liste des tâches prêtes pour faire l'exécution de la première tâche T_i dans la liste sur le processeur associé. Si le processeur est libre, la tâche T_i est exécutée immédiatement et sans aucune attente, si non, à partir de la matrice d'affectation des priorités, l'ordonnanceur vérifie si la tâche en cours d'exécution T_j a une priorité inferieure à la tâche T_i.

Supposant que c'est le cas, cela implique que la tâche T_j est interrompue et insérée dans la liste des tâches prêtes, tandis que la tâche T_i commence son exécution.

Si T_i et T_j veulent s'exécuter en même temps dans la prochaine génération, dans ce cas, il se peut que les priorités des tâches soient changées, ce qui permet à T_j de s'exécuté cette fois-ci en première. Le changement des priorités est fait grâce aux opérateurs génétiques.

4.2.5 Evaluation des individus

L'évaluation de chaque individu se fait après l'ordonnancement de toutes les tâches du système, où les facteurs que nous avons voulus optimiser (le temps de réponse moyen

« **TRM** » ou le nombre de tâches respectant leurs échéances « **NTR** ») sont calculés, et les individus de la population sont alors quantifiés.

4.2.6 Critère d'arrêt

Après la phase d'évaluation des individus, un petit test se fait pour savoir si le critère d'arrêt (qui représente un nombre fixe de génération) est atteint. Si c'est le cas, tous les calculs sont arrêtés et la solution optimale est ainsi obtenue. Si non, poursuivre l'algorithme tant que le nombre de générations fixé au préalable n'est pas atteint.

4.2.7 Sélection

Après les calculs effectués dans la phase d'évaluation, chaque individu doit porter deux valeurs : **TRM** et **NTR.** La sélection des meilleurs individus se fait à base d'un seul facteur choisi par l'utilisateur. Les méthodes de sélection sont restées les mêmes que dans la première stratégie.

4.2.8 Croisement

Deux méthodes de croisement sont aussi implémentées (croisement en un seul point ou en deux points). La figure ci-dessous montre l'opérateur du croisement en un seul point :

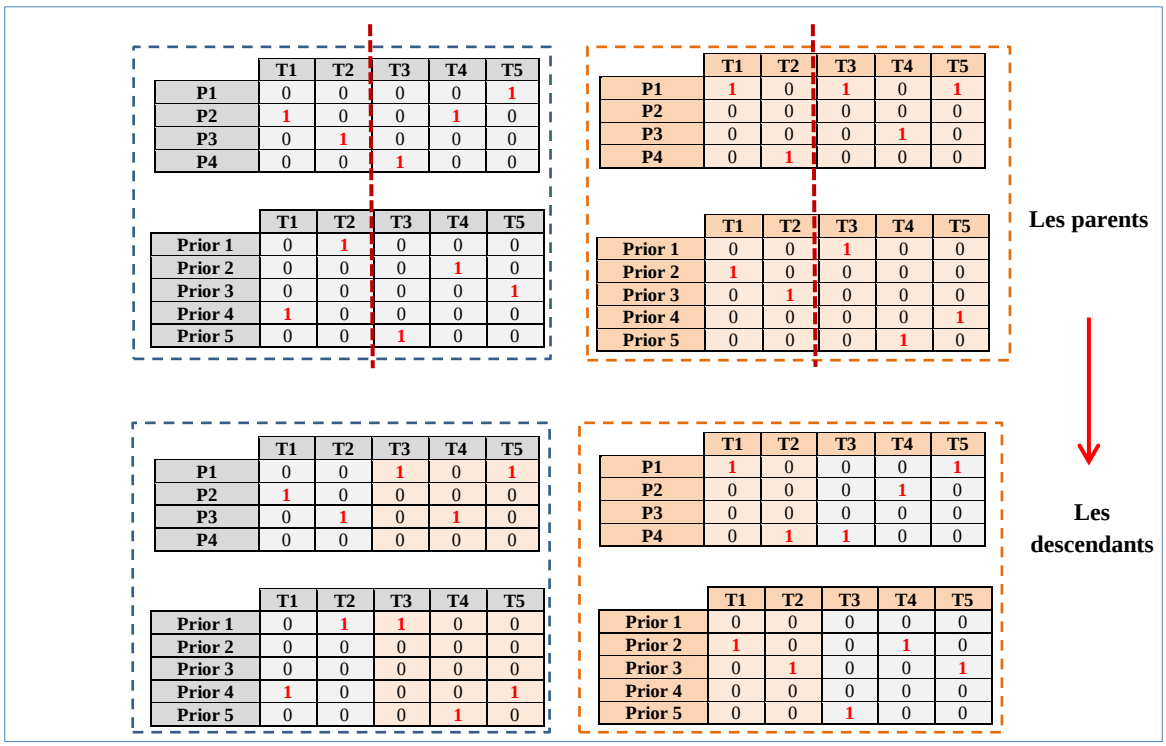


Figure 4.26 : L'opérateur du croisement de la deuxième stratégie (AGC2).

4.2.9 Mutation

Nous avons appliqué l'opérateur de mutation sur les deux matrices en même temps (changement de la position d'un seul bit dans chaque matrice). La figure ci-dessous présente une mutation appliquée sur un individu choisi de façon aléatoire:

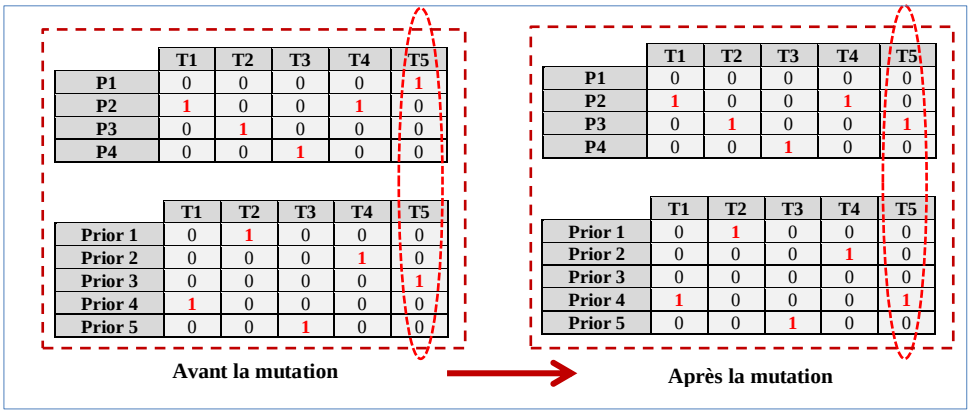


Figure 4.27 : L'opérateur de mutation de la deuxième stratégie (AGC2).

4.2.10 Insertion et remplacement

C'est la dernière étape avant de refaire la boucle, elle sert à remplacer les mauvais individus.

4.2.11 Tests et résultats de la deuxième stratégie

Nous avons réalisé un ensemble de tests en appliquant l'AGC avec priorités dynamiques sur les trois exemples cités précédemment. Notant que les priorités des tâches changent d'une génération à une autre grâce aux opérateurs génétiques.

Le tableau ci-dessous résume les différents jeux de paramètres appliqués au premier exemple :

Num tests	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Selc	Crois	Mut	Résultats		
													Trm	T.u. Cpu%	N.t.d
A1	20	03	01	00	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	10.5	33.0	2
A2	20	03	01	00	00	100	60	60	Trm	Tn	1 Pnt	P=0.3	14.3	33.0	5
A3	20	03	01	00	00	100	60	60	Trm	Tn	1 Pnt	P=0.8	11.2	33.0	1
A4	20	03	01	00	00	100	60	60	N.t.d	Tn	1 Pnt	P=0.3	11.08	33.0	0
A5	25	03	01	00	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	12.52	43.0	1
A6	25	04	01	00	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	11.76	35.0	3
A7	25	04	01	00	00	100	100	60	Trm	Elit	2 Pnts	P=0.3	11.3	35.0	2

Tableau 4.8 : Paramètres et résultats de l'AGC2 appliqués au 1er exemple.

Le tableau ci-dessous résume les différents jeux de paramètres appliqués au deuxième exemple:

Num tests	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Selc	Crois	Mut	Résultats		
													Trm	T.u. Cpu%	N.t.d
B1	20	03	02	01	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	10.15	33.0	3
B2	20	03	03	03	00	100	60	60	Trm	Tn	1 Pnt	P=0.3	10.7	33.0	1
B3	20	03	02	01	00	100	60	60	Trm	Elit	1 Pnt	P=0.8	11.4	33.0	0
B4	20	03	02	01	00	100	60	60	N.t.d	Elit	1 Pnt	P=0.3	11.45	33.0	0
B5	25	03	02	01	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	11.48	43.0	3
B6	25	04	02	01	00	100	60	60	Trm	Elit	1 Pnt	P=0.3	10.36	35.0	0
B7	25	04	03	03	00	100	100	60	Trm	Elit	2 Pnts	P=0.3	11.52	35.0	5

Tableau 4.9 : Paramètres et résultats de l'AGC2 appliqués au 2eme exemple.

Le tableau ci-dessous résume les différents jeux de paramètres appliqués au troisième exemple:

Num tests	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Selc	Crois	Mut	Résultats		
													Trm	T.u. Cpu%	N.t.d
C1	20	03	02	01	01	100	60	60	Trm	Elit	1 Pnt	P=0.3	11.3	33.0	1
C2	20	04	03	03	01	100	60	60	Trm	Tn	1 Pnt	P=0.3	9.25	27.0	1
C3	20	03	02	01	01	100	60	60	Trm	Elit	1 Pnt	P=0.8	12.85	33.0	2
C4	20	03	02	01	01	100	60	60	N.t.d	Elit	1 Pnt	P=0.3	11.4	33.0	0
C5	25	03	02	01	01	100	60	60	Trm	Elit	1 Pnt	P=0.3	13.76	43.0	1
C6	25	04	02	01	02	100	60	60	Trm	Elit	1 Pnt	P=0.3	10.44	35.0	1
C7	25	04	03	03	01	100	100	60	Trm	Elit	2 Pnts	P=0.3	10.36	35.0	3

Tableau 4.10 : Paramètres et résultats de l'AGC2 appliqués au 3eme exemple.

4.2.12 Discussion

Dans cette stratégie, nous avons affecté des priorités dynamiques aux tâches, et d'après les résultats des 3 tableaux, on peut dire aussi que le *Trm* est diminué à chaque fois que le *Ntd* est réduit (*exp : test A2 et A 3*). Le changement des priorités sert à diminuer dans la plupart des cas le *Ntd* et par conséquent le *Trm*. Ceci est dû au fait que l'on a donné plus d'importance à chaque individu composé d'un ensemble de tâches possédant des priorités aident à donner un *Trm* un peu plus réduit et on a négligé les autres. La probabilité de mutation $P_{mut} = 0.1$ donne les meilleurs résultats.

Après la réalisation de plusieurs tests, nous avons déduit que la valeur la plus intéressante du temps de réponse moyen pour chaque exemple est indiquée par le tableau suivant :

Exemple	**Temps de réponse moyen (3 Cpus)**	**Taux d'utilisation des Cpus**	**Nombre de tâches respectant l'échéance**
1	10.5	33.0	2
2	10.15	33.0	3
3	11.3	33.0	1

Tableau 4.11: les meilleurs résultats obtenu par l'AGC2 pour 20 tâches.

4.2.13 Résultats des trois exemples

Nous avons refait les tests sur les trois exemples. Cette fois, nous avons fixé les paramètres de la stratégie (**nb_tâches = 30, nb_cpu = 6, Tps_Sim = 60, taille_pop = 100**), et nous avons varié le paramètre du nombre de génération :

Exemples	Nombre de génération	Temps de réponse moyen	Taux d'utilisation des Cpu	Nombre de tâches dépassant l'échéance
Exemple 01 **(bus partagé)**	Itér = 30	12.2	23%	8
	Itér = 60	11.53	23%	5
	Itér = 80	11.53	23%	4
	Itér = 100	11.53	23%	6
Exemple 02 **(3 bus, 3 ponts)**	Itér = 30	11.47	23%	4
	Itér = 60	10.27	23%	6
	Itér = 80	10.43	23%	5
	Itér = 100	10.4	23%	5
Exemple 03 **(3 bus, 3 ponts, 2 liens)**	Itér = 30	10.82	23%	5
	Itér = 60	10.5	23%	6
	Itér = 80	10.77	23%	5
	Itér = 100	10.5	23%	5

Tableau 4.12: Les résultats des trois exemples obtenus par l'AGC2.

Les graphes de la figure 4.28 résument les résultats des trois exemples du tableau 4.12 :

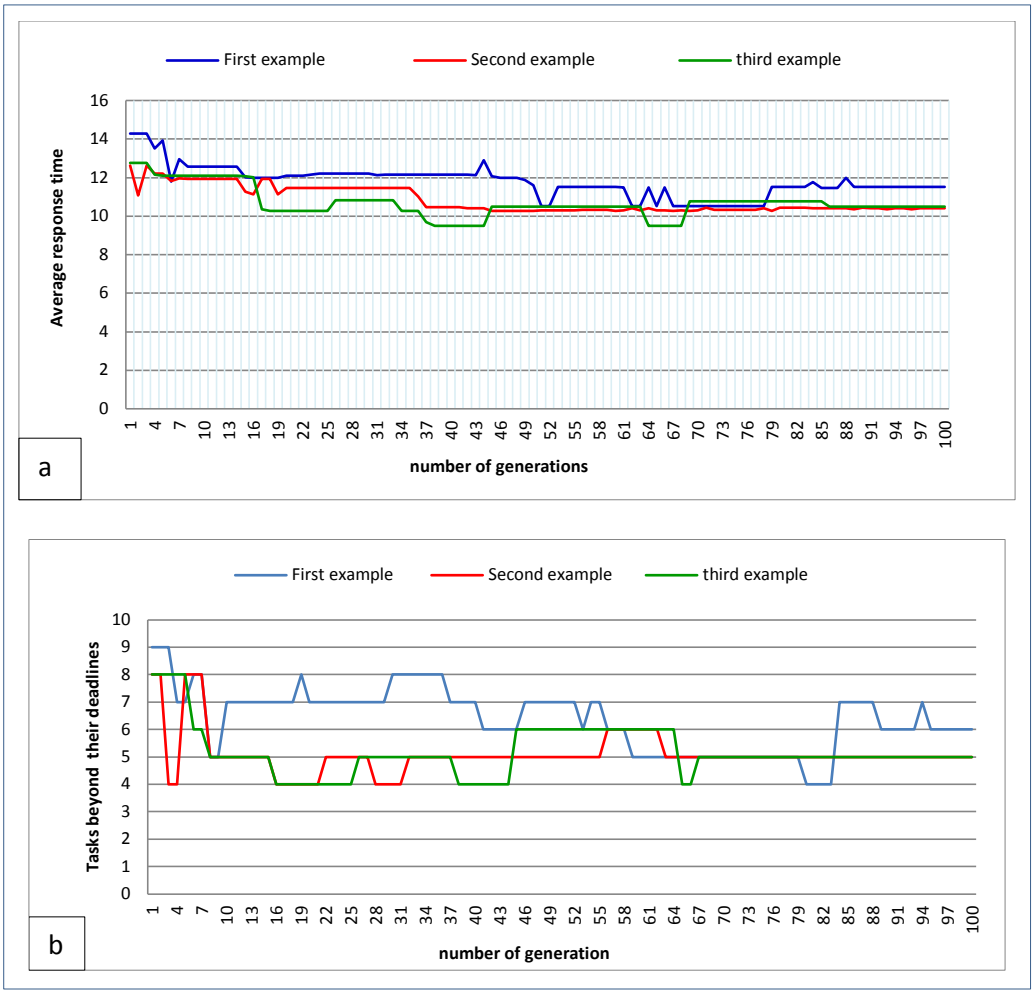


Figure 4.28 : Résultats des trois exemples sous forme de graphes.

- ## Discussion

Le comportement du paramètre *Trm* du premier exemple est le même que dans la première stratégie et l'explication est la même. Le deuxième et le troisième exemple prennent des valeurs très proches.

D'après les résultats obtenus sur cet exemple (*30 tâches*), on remarque que l'architecture composé de trois bus et trois ponts donne au final la meilleure valeur du *Trm (Trm =10.4)*. Cela est interprété par le fait que la distribution des tâches sur les processeurs peut donner une valeur de *Trm* plus réduit sans la nécessité d'utiliser des liens directs entre les processeurs. Le changement des priorités ne donne pas toujours des bons résultats mais on peut dire qu'en moyen les résultats sont mieux que dans la première stratégie où nous avons utilisé des priorités fixes. Le comportement du paramètre *Ntd* est le même que dans la première stratégie.

Les figures 4.29, 4.30 et 4.31 représentent les chronogrammes des trois exemples du tableau 4.12 :

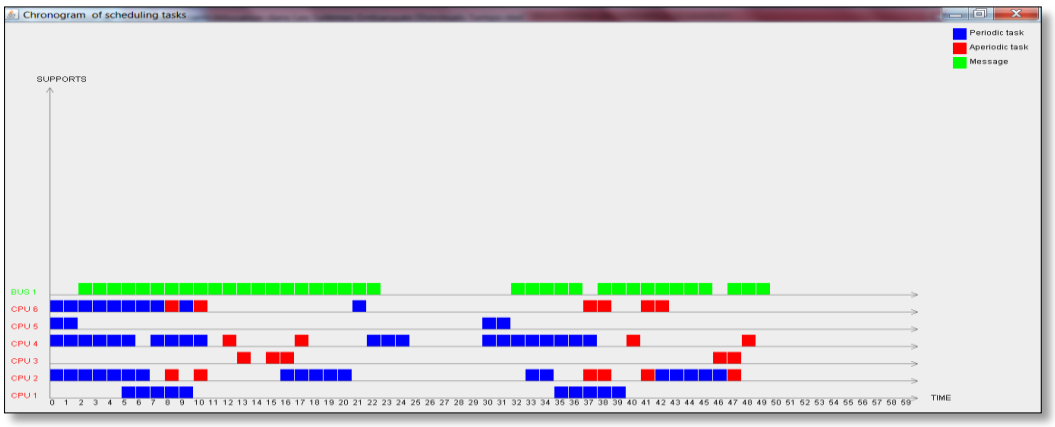


Figure 4.29 : Chronogramme de l'ordonnancement de la meilleure solution (le premier exemple).

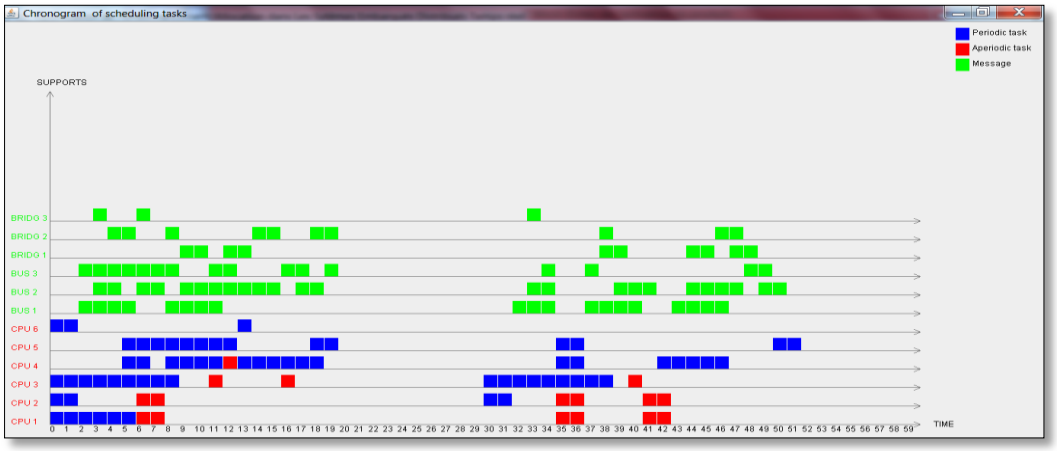


Figure 4.30 : Chronogramme de l'ordonnancement de la meilleure solution (le deuxième exemple).

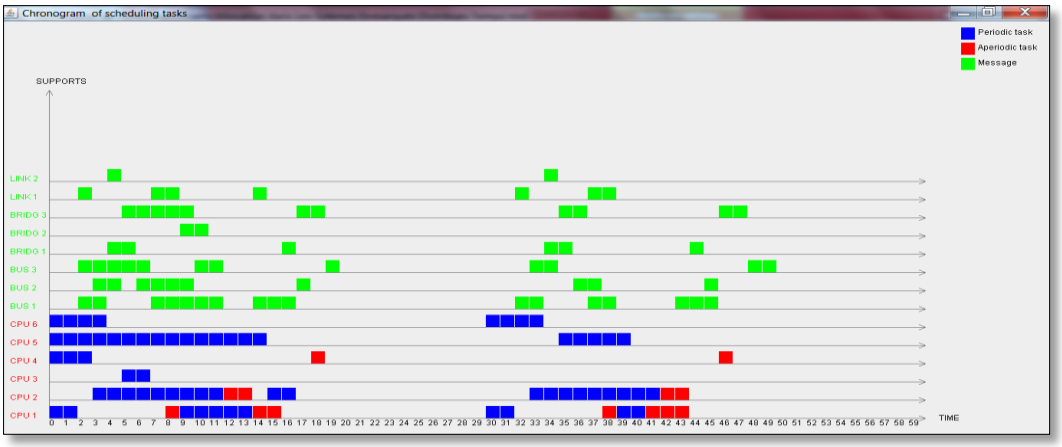


Figure 4.31 : Chronogramme de l'ordonnancement de la meilleure solution (le troisième exemple).

Les figures 4.32, 4.33 et 4.34 représentent les taux d'occupation des processeurs des trois exemples du tableau 4.12 :

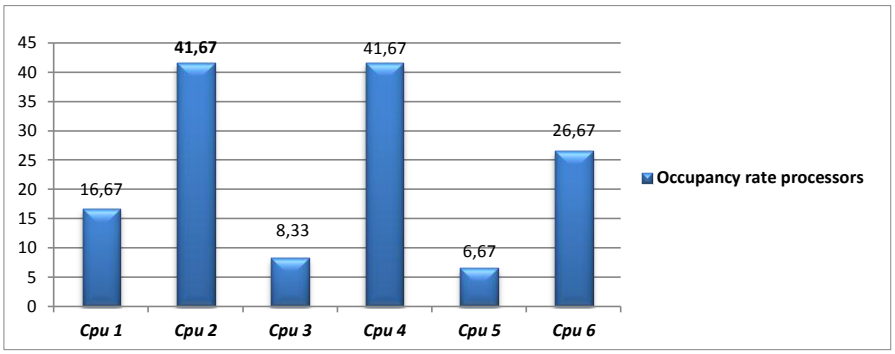


Figure 4.32 : Taux d'occupation des processeurs de la meilleure solution (premier exemple).

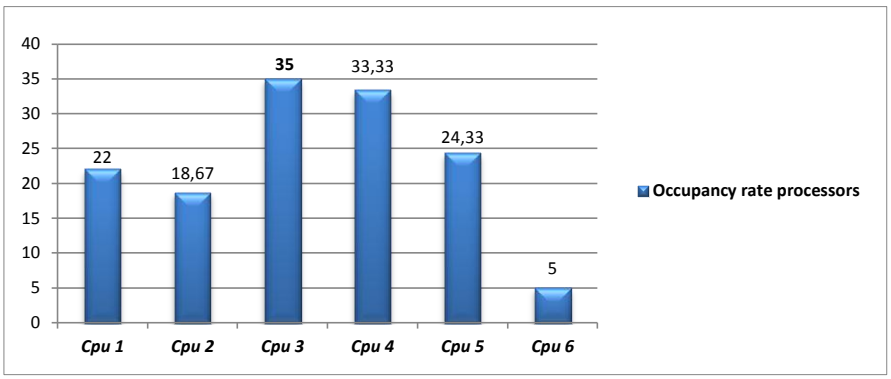


Figure 4.33 : Taux d'occupation des processeurs de la meilleure solution (deuxième exemple).

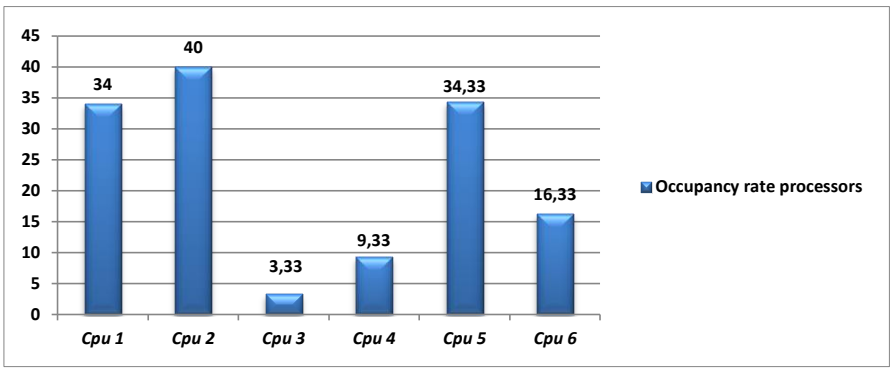


Figure 4.34 : Taux d'occupation des processeurs de la meilleure solution (troisième exemple).

Dans ce qui suit, nous présenterons les deux autres approches que nous avons proposées. Elles se basent principalement sur une hybridation entre l'algorithme génétique et l'informatique quantique, adapté au problème de l'ordonnancement/allocation dans les systèmes embarqués distribués temps réel. Nous l'avons appelé " optimisation par algorithmes génétique inspiré-quantique (AGIQ).

Nous avons développé deux algorithmes pour une telle hybridation:

- AGIQ1 qui est une hybridation entre l'informatique quantique et l'AGC1.

- AGIQ2 qui est une hybridation entre l'informatique quantique et l'AGC2.

4.3 Troisième stratégie(AGIQ1): application des algorithmes génétiques inspirés-quantiques en affectant des priorités statiques aux tâches

Les recherches dans le domaine de l'hybridation des algorithmes évolutionnaires et de l'informatique quantique ont commencé à la fin des années 1990. Cette hybridation a donné naissance aux algorithmes génétiques inspirés-quantiques (QIGA). C'est une hybridation qui a prouvé son efficacité dans le domaine de l'optimisation.

Dans cette section, après la phase d'allocation et d'ordonnancement des tâches temps réel dans les systèmes embarqués distribués, nous avons proposé un AGIQ pour optimiser les performances de ces systèmes.

4.3.1 Organigramme de la troisième stratégie (AGIQ avec priorités statiques)

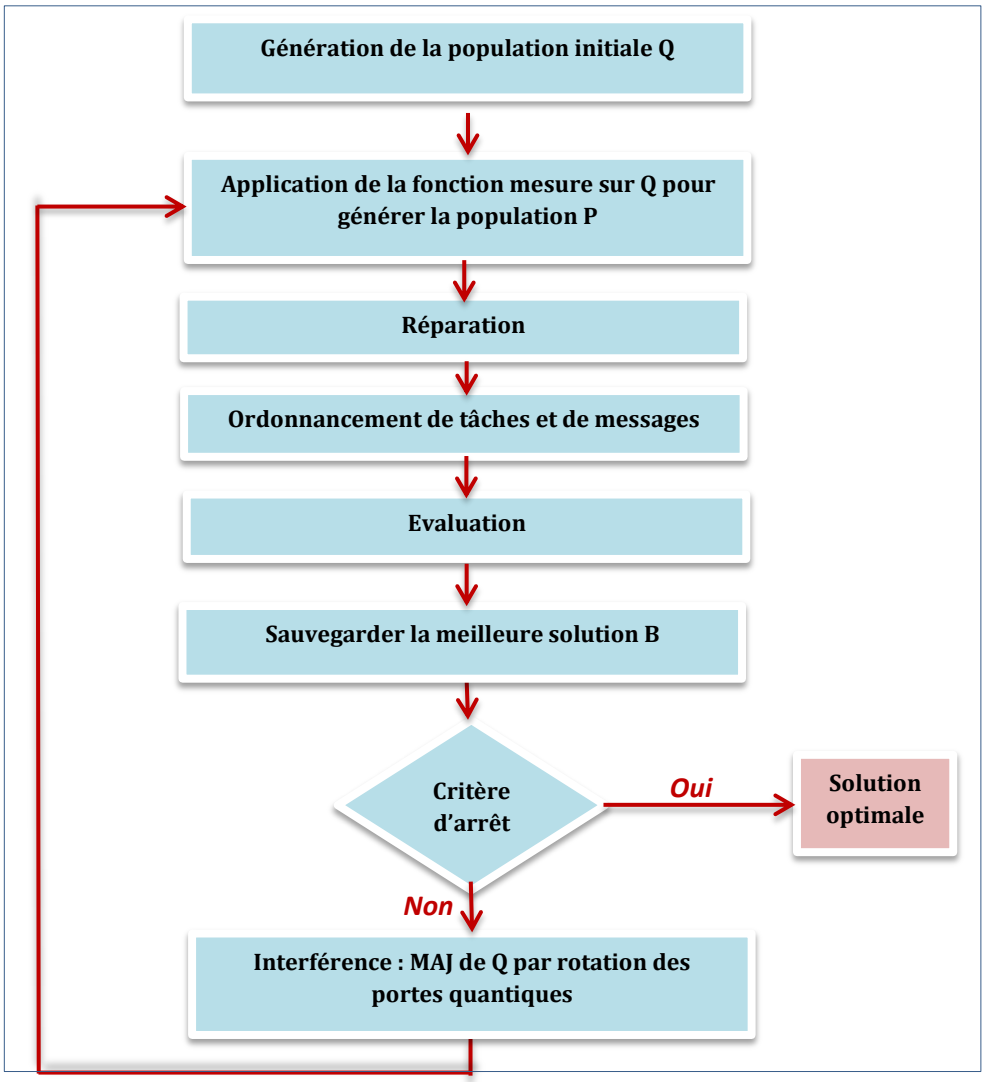


Figure 4.35 : Organigramme général de la troisième stratégie (AGIQ1).

4.3.2 Principe de fonctionnement de notre algorithme

Nous avons essayé en premier lieu d'utiliser les opérateurs génétiques telle que la sélection le croisement et la mutation en plus de l'opérateur de l'interférence, pour tester nos exemples.

Puis nous avons répété le même processus mais dans cette fois-ci, nous avons essayé d'éliminer ces opérateurs et de les remplacé par l'opérateur d'interférence afin de percevoir l'effet de ces opérateurs sur les résultats finaux. Dans ce cas, l'interférence est le seul opérateur génétique à appliquer.

Nous avons cherché à réduire le nombre des paramètres pour rendre l'algorithme plus facile à utiliser.

4.3.3 Génération de la population initiale « Q »

4.3.3.1 Structure proposé du chromosome quantique

Contrairement à la représentation des chromosomes présentés dans les deux premières stratégies (sous forme de matrice binaire), les chromosomes ici sont représentés sous forme d'une matrice de qubits de longueur *nb_tâche* et de largeur *nb_cpu*. La figure ci-dessous montre la structure d'un chromosome quantique :

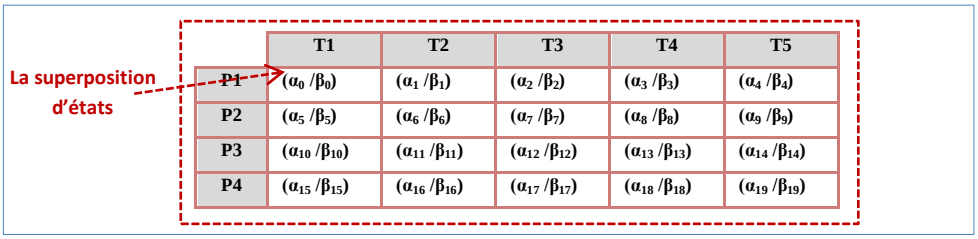


Figure 4.36 : Représentation du chromosome quantique de la troisième stratégie (AGIQ1).

4.3.3.2 Initialisation de la population

Pour générer la population initiale, il faut créer un ensemble de chromosomes quantiques. Pour cela, toutes les amplitudes des qubits sont initialisées en donnant la même valeur (la valeur $2^{-1/2}$) pour tous les états de superposition. Cela signifie qu'un chromosome quantique représente tous les états de superpositions avec la même probabilité.

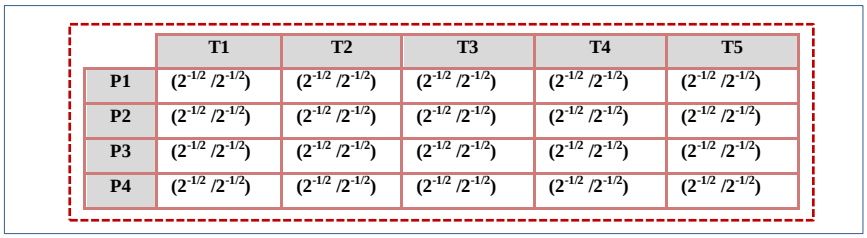


Figure 4.37 : Initialisation de chaque chromosome quantique.

4.3.4 Application de la fonction mesure sur « Q »

La solution quantique ne peut pas être évaluée, du fait que l'affectation exacte d'une tâche n'est pas connue. Le but de cette fonction (dite aussi d'observation) est d'extraire un chromosome classique (matrice binaire) à partir d'un autre quantique pour l'évaluer. Donc pour chaque chromosome de la population « Q », nous avons fait le transfert de chaque qubit à un bit classique qui porte une seule valeur : 0 ou 1. En effet, pour mesurer un chromosome quantique, nous avons généré pour chaque case de la matrice un nombre aléatoire dans [0,1],

si ce nombre est strictement supérieur à la valeur α^2 de cette case, alors la valeur de la case devient 1, sinon elle devient 0. La nouvelle population générée est appelée « P ».

	T1	T2	T3	T4	T5
P1	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$
P2	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$
P3	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$
P4	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$	$(2^{-1/2}/2^{-1/2})$

↓

	T1	T2	T3	T4	T5
P1	0	0	1	1	1
P2	1	0	1	0	0
P3	1	1	1	0	1
P4	0	0	0	0	1

Figure 4.38 : Mesure de chaque chromosome quantique.

4.3.5 Réparation de « P »

Après la fonction mesure, tous les chromosomes quantiques sont transférés aux chromosomes binaires. Mais, il est possible de constater qu'une colonne de la matrice contient plusieurs 1, ou que toutes les valeurs d'une colonne sont des 0. L'interprétation de ces deux cas est que le premier cas signifie que la tâche est affecter aux plusieurs processeurs, tandis que le deuxième cas signifie que la tâche n'est affecté à aucun processeur. Donc, chaque tâche doit être affectée à un seul processeur pour qu'elle puisse s'exécuter.

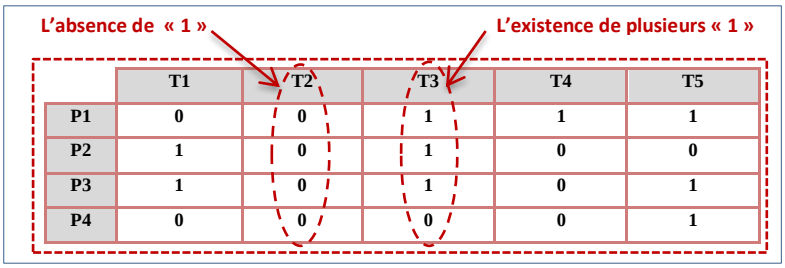


Figure 4.39 : L'exigence d'une réparation.

Nous avons implémenté deux fonctions de réparation (réparation1 et réparation2) pour rectifier ces cas. La fonction réparation1 est employée pour l'acceptation d'un seul 1 dans la colonne en cas de l'existence de plusieurs 1, elle génère un nombre aléatoire pour chaque 1 de la colonne, et celui qui a le plus grand nombre reste dans la colonne. La fonction réparation2 est employée pour garantir l'existence d'un seul 1 dans chaque colonne en cas d'absence de la valeur 1 dans la colonne. Pour cela elle met un 1 dans la même ligne que la tâche

prédécesseur de cette tâche, c'est-à-dire que cette tâche est affectée au même processeur que sa tâche prédécesseur pour éliminer le temps de transfert de message entre les deux tâches.

4.3.6 Ordonnancement de tâches et de messages

Après avoir réparé tous les chromosomes de la population, cela signifie que toutes les tâches sont allouées aux différents processeurs de l'architecture.

L'ordonnancement des tâches temps réel et des messages est fait de la même manière que dans la première stratégie.

4.3.7 Evaluation et sauvegarde du meilleur individu

Comme dans les deux premières stratégies, Après la phase d'allocation et d'ordonnancement des tâches et des messages, chaque individu de la population doit être évalué, en utilisant une fonction d'adaptation.

La fonction d'adaptation que nous avons voulus optimiser dans ce travail est représentée par l'un des deux facteurs: le temps de réponse moyen ou le nombre de tâches respectant leurs échéances. Après l'exécution de l'algorithme d'ordonnancement, nous pouvons calculer ces facteurs. Nous avons utilisé les mêmes formules utilisées dans les deux premières stratégies pour calculer les deux facteurs.

Dans chaque itération, nous avons sauvegardé la meilleure solution (ayant la valeur d'adaptation la plus élevée) une fois la population est évaluée.

4.3.8 Critère d'arrêt

Un petit test se fait avant de poursuivre la boucle, pour savoir si le nombre max d'itération est atteint. Dans ce cas tout le traitement est arrêté et la solution est obtenue.

4.3.9 Interférence

L'interférence a comme rôle d'augmenter (interférence constructive) ou diminuer (interférence destructive) l'amplitude d'un état et par conséquence sa probabilité d'être observé. L'interférence est très importante en calcul quantique. Elle permet d'augmenter la probabilité d'avoir les résultats espérés.

L'interférence quantique peut être définit comme étant une rotation spéciale. Cette rotation est faite en fonction de la solution courante, la meilleure solution et les signes des amplitudes de la solution courante (*voir Tableau 4.13*).

Chaque élément q_{ij} de l'individu quantique est mis à jour suivant ces étapes:

1. Déterminer $\Delta\theta_i$ avec la table de recherche (*voir Tableau 4.13*).

2. Calculer les nouvelles valeurs α'_{ij}, β'_{ij} en utilisant la formule :

$$\begin{bmatrix} \alpha'_{ij} \\ \beta'_{ij} \end{bmatrix} = \begin{bmatrix} \cos\left(\Delta\theta_{ij} \times s(\alpha_{ij},\beta_{ij})\right) & -\sin\left(\Delta\theta_{ij} \times s(\alpha_{ij},\beta_{ij})\right) \\ \sin\left(\Delta\theta_{ij} \times s(\alpha_{ij},\beta_{ij})\right) & \cos\left(\Delta\theta_{ij} \times s(\alpha_{ij},\beta_{ij})\right) \end{bmatrix} \begin{bmatrix} \alpha_{ij} \\ \beta_{ij} \end{bmatrix}$$

Formule 4.8 : Politique de rotation.

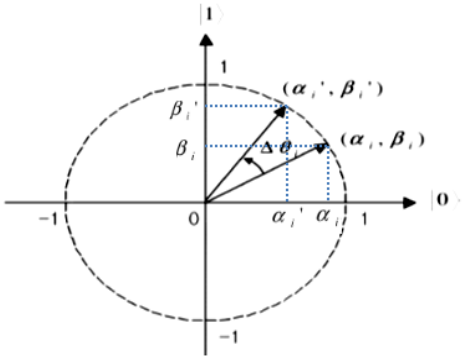


Figure 4.40 : Coordonnées polaires de la porte de rotation pour les individus Q-bits

Où $\mathbf{\Delta\theta_i}$ est l'angle de rotation faisant dévier chaque qubit vers l'état 0 ou 1 dépendant de son signe. Le signe de $\mathbf{\Delta\theta_i}$ change selon la table présentée dans la figure ci-dessous :

xij	bij	*f*(x) >*f* (b)	Δθij	S (aij, bij)			
				aij* bij > 0	aij* bij < 0	aij=0	bij = 0
0	0	false	θ_1	±	±	±	±
0	0	true	θ_2	±	±	±	±
0	1	false	θ_3	±	±	±	±
0	1	true	θ_4	±	±	±	±
1	0	false	θ_5	±	±	±	±
1	0	true	θ_6	±	±	±	±
1	1	false	θ_7	±	±	±	±
1	1	true	θ_8	±	±	±	±

Tableau 4.13: Table de recherche générale pour la rotation des portes quantiques.

xij et ***bij*** sont les bits de la case [i,j] de ***x*** et ***b*** (la meilleure solution trouvée) respectivement. ***f*** est la fonction fitness et ***S (aij, bij)*** est le signe de l'angle ***θij***. La valeur de l'angle de rotation ***Δθij*** est choisie par un raisonnement intuitif. Quand la condition *f(x)>f(b)* n'est pas satisfaite, $\Delta\theta_{ij}$ prend une valeur de telle sorte qu'on doit mettre à jour la solution binaire « ***x*** » autour de la meilleure solution « ***b*** ». Cela permet la recherche de nouvelles solutions dans le voisinage de la meilleure solution, mais quand la condition est satisfaite, $\Delta\theta_{ij}$ prend de très petite valeurs ou tout simplement une valeur nulle. Cela permet d'explorer d'autres solutions et pour échapper au minimum local.

4.3.10 Tests et résultats de la troisième stratégie

Nous avons réalisé un ensemble de tests en appliquant l'AGIQ avec priorités statiques sur les trois exemples cités dans la section 3. Nous avons réalisé ces tests sans, puis avec opérateurs génétiques telle que la sélection le croisement et la mutation en plus de l'opérateur de l'interférence.

Les angles de rotation utilisée dans ces tests sont figurés dans le tableau suivant :

xi	bi	f(x) >f (b)	Δθi	S (ai bi)			
				ai* bi > 0	ai* bi < 0	ai =0	bi = 0
0	0	false	0	±	±	±	±
0	0	true	0	±	±	±	±
0	1	false	0.02 π	+	-	±	±
0	1	true	0	±	±	±	±
1	0	false	0.02 π	-	+	±	±
1	0	true	0	±	±	±	±
1	1	false	0	±	±	±	±
1	1	true	0	±	±	±	±

Tableau 4.14: Table de recherche adoptée pour la rotation des portes quantiques.

Les tableaux ci-dessous résument les différents jeux de paramètres appliqués aux trois exemples ainsi que les résultats obtenus:

N° test	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Politiq ordon	Résultats		
											Trm	T.u. Cpu%	N.t.d
A1	20	03	01	00	00	100	40	60	Trm	DM	9.9	33.0	1
A2	20	03	01	00	00	100	40	60	Trm	RM	11.5	33.0	2
A3	20	04	01	00	00	100	40	60	Trm	DM	9.75	27.0	1
A4	20	03	01	00	00	100	40	60	N.t.d	DM	9.8	33.0	0
A5	25	03	01	00	00	100	40	60	Trm	DM	12.04	43.0	3
A6	25	04	01	00	00	100	40	60	Trm	RM	10.8	35.0	0
A7	25	04	01	00	00	100	60	60	Trm	DM	10.25	35.0	0

Tableau 4.15 : Paramètres et résultats de l'AGIQ1 appliqués au 1er exemple.

N° test	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Politiq ordon	Résultats		
											Trm	T.u. Cpu%	N.t.d
B1	20	03	02	01	00	100	40	60	Trm	DM	10.25	33.0	1
B2	20	03	02	01	00	100	40	60	Trm	RM	11.8	33.0	3
B3	20	04	02	01	00	100	40	60	Trm	DM	9.15	27.0	0
B4	20	03	02	01	00	100	40	60	N.t.d	DM	9.9	33.0	0
B5	25	03	02	01	00	100	40	60	Trm	RM	12.52	43.0	3
B6	25	04	02	01	00	100	40	60	Trm	RM	10.36	35.0	0
B7	25	04	03	03	00	100	60	60	Trm	DM	11.68	35.0	2

Tableau 4.16 : Paramètres et résultats de l'AGIQ1 appliqués au 2eme exemple.

N° test	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Politiq ordon	Résultats		
											Trm	T.u. Cpu%	N.t.d
C1	20	03	02	01	01	100	40	60	Trm	DM	8.7	33.0	1
C2	20	03	02	01	01	100	40	60	Trm	RM	9.45	33.0	2
C3	20	04	02	01	01	100	40	60	Trm	DM	10.4	27.0	1
C4	20	03	02	01	01	100	40	60	N.t.d	DM	9.3	33.0	1
C5	25	03	02	01	01	100	40	60	Trm	RM	11.88	43.0	0
C6	25	04	02	01	02	100	40	60	Trm	RM	10.24	35.0	0
C7	25	04	03	03	01	100	60	60	Trm	DM	11.64	35.0	0

Tableau 4.17 : Paramètres et résultats de l'AGIQ1 appliqués au 3eme exemple.

4.3.11 Discussion

Dans cette stratégie, nous avons présenté quelques résultats sur les trois exemples cités ci-dessus, l'algorithme est exécuté sans et avec opérateurs génétiques.

Selon le nombre de tests effectué sur les trois exemples, nous avons remarqué que les valeurs: $\theta_3 = 0.02\pi$, $\theta 5 = 0.02\pi$ et les restes égale à *0*, donnent les meilleurs résultats.

La modification des amplitudes des qubits se fait selon l'individu *x*, le meilleur individu et les signes des amplitudes de l'individu *x*. Si ces amplitudes se situent sur le premier ou le troisième quadrant, le signe est positif pour θ_3 et négatif pour θ_5, si elles se situent sur le deuxième ou le quatrième quadrant, le signe est négatif pour θ_3 et positif pour θ_5.

D'après les résultats des trois tableaux ci-dessus, et si on met l'accent sur l'opérateur d'interférence, on remarque bien que l'AGIQ1 est significativement meilleur que la première et la deuxième stratégie, où on constate que le *Trm* atteint une valeur minimale (*Trm= 8,7*).

On remarque aussi que dans la plupart des cas, la politique d'ordonnancement deadline monotonic (*DM)* donne les meilleurs résultats.

Pour voir maintenant l'effet des opérateurs génétiques sur les résultats, on a testé les trois exemples en ajoutant l'opérateur de croisement en un seul point et l'operateur de mutation avec un taux de mutation égale à 0,3 (c'est la valeur ayant donné après un certain nombre d'essai le meilleur comportement par rapport à d'autre valeurs).

Exemples	θij	croisement	mutation	resultats	
				Trm	Ntd
Exemples 1	0,1 π	oui	non	11.1	2
		non	0,3	11.4	2
		oui	0,3	11.24	1
	0,05 π	oui	non	11.0	3
		non	0,3	11.3	3
		oui	0,3	10.08	0
	0,02 π	oui	non	9.7	0
		non	0,3	10.2	2
		oui	0,3	10.02	1
Exemples 2	0,1 π	oui	non	13.94	3
		non	0,3	13.6	3
		oui	0,3	12.1	2
	0,05 π	oui	non	11.22	4
		non	0,3	12.56	3
		oui	0,3	11.2	3
	0,02 π	oui	non	11.3	2
		non	0,3	10.8	1
		oui	0,3	9.92	0

Exemples 3	0,1 π	oui	non	11.06	2
		non	0,3	12.4	1
		oui	0,3	11.3	1
	0,05 π	oui	non	10.02	0
		non	0,3	11.12	2
		oui	0,3	11.1	1
	0,02 π	oui	non	10.1	1
		non	0,3	10.15	1
		oui	0,3	10.1	1

Tableau 4.18 : Effet des opérateurs génétiques sur les résultats des trois exemples.

On remarque que pour la plupart des tests, l'opérateur de croisement améliore considérablement les résultats obtenus. Par contre, l'opérateur de mutation donne les meilleurs résultats pour des angles plus petits. Donc l'operateur de mutation sert à faire échapper l'algorithme d'un minimum local qui peut résulter d'un petit angle de rotation. L'utilisation des deux opérateurs à la fois améliore encore plus les performances.

Les meilleurs résultats obtenus pour chaque exemple selon le *Trm* sont les suivants :

Exemple	Temps de réponse moyen (3 Cpus)	Taux d'utilisation des Cpus	Nombre de tâches respectant l'échéance
1	9.8	33.0 %	0
2	9.9	33.0 %	0
3	8.7	33.0 %	1

Tableau 4.19: les meilleurs résultats obtenu par l'AGIQ1 pour 20 tâches.

4.3.12 Résultats des trois exemples

Nous avons refait les tests sur les trois exemples. Cette fois, nous avons fixé les paramètres de la stratégie (**nb_tâches = 30, nb_cpu = 6, Politiq_ordon = DM, Tps_Sim = 60, taille_pop = 40, θ = 0,02π**), et nous avons varié le paramètre du nombre de génération :

Exemples	Nombre de génération	Temps de réponse moyen	Taux d'utilisation des Cpu	Nombre de tâches dépassant l'échéance
Exemple 01 **(bus partagé)**	Itér = 30	12.5	23%	4
	Itér = 60	12.03	23%	5
	Itér = 80	11.93	23%	1
	Itér = 100	10.5	23%	3
Exemple 02 **(3 bus, 3 ponts)**	Itér = 30	11.67	23%	2
	Itér = 60	11.43	23%	3
	Itér = 80	12.03	23%	4
	Itér = 100	10.0	23%	2

Exemple 03 (3 bus, 3 ponts, 2 liens)	Itér = 30	11,27	23%	2
	Itér = 60	10,6	23%	2
	Itér = 80	11,1	23%	4
	Itér = 100	10.43	23%	3

Tableau 4.20: Les résultats des trois exemples obtenus par l'AGIQ1.

Les graphes de la figure 4.41 résument les résultats des trois exemples du tableau 4.20 :

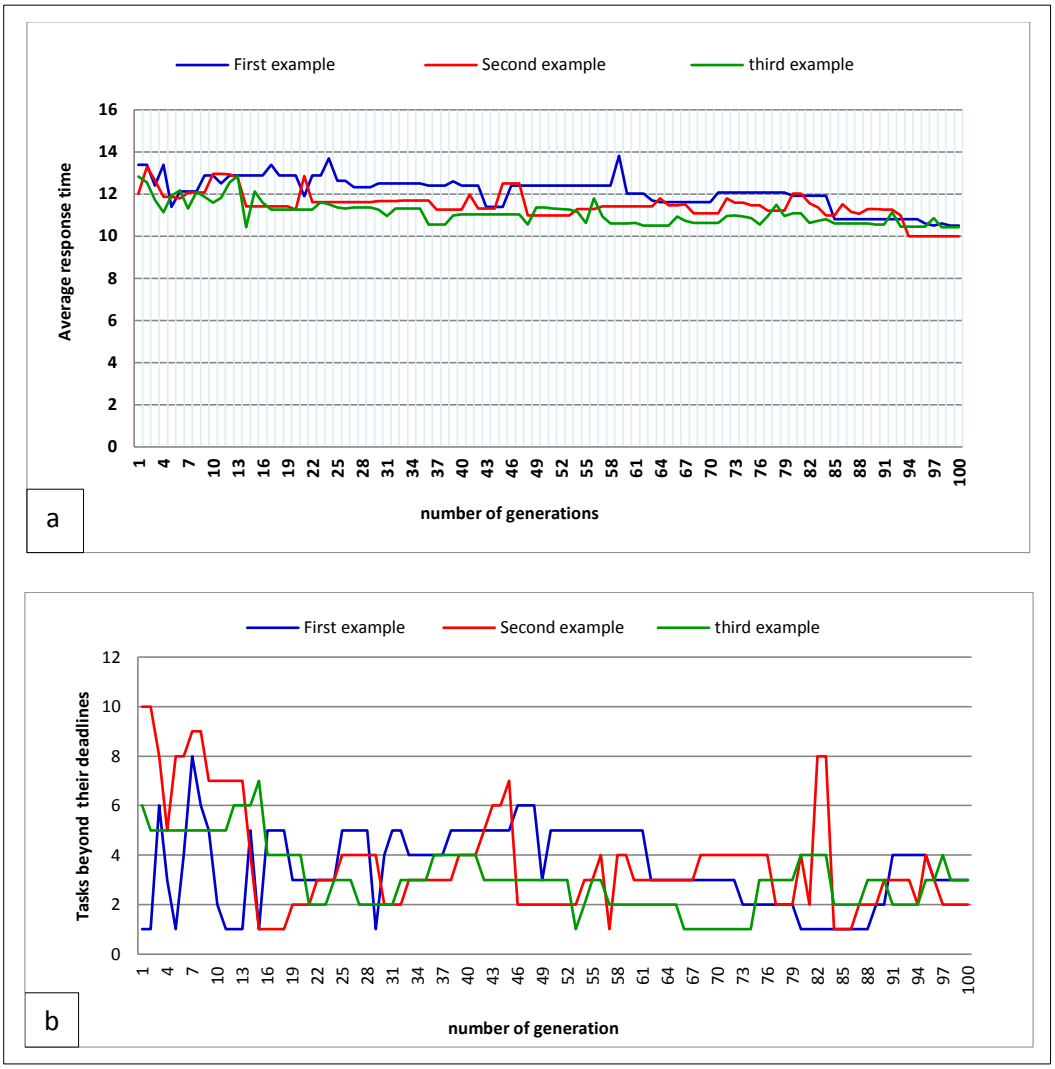


Figure 4.41 : Résultats des trois exemples sous forme de graphes.

- **Discussion**

Selon les courbes de la figure 4.41, on remarque que les résultats obtenus sont remarquablement bons en comparant les résultats avec ceux des algorithmes génétiques classiques qui nécessitent une grande taille de la population.

Les figures 4.42, 4.43 et 4.44 représentent les chronogrammes des trois exemples du tableau 4.20 :

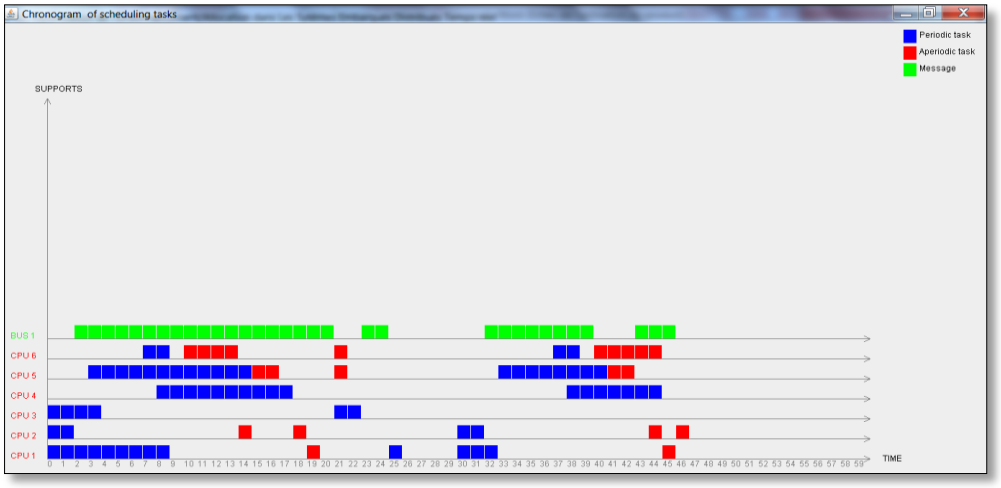


Figure 4.42 : Chronogramme de l'ordonnancement de la meilleure solution (le premier exemple).

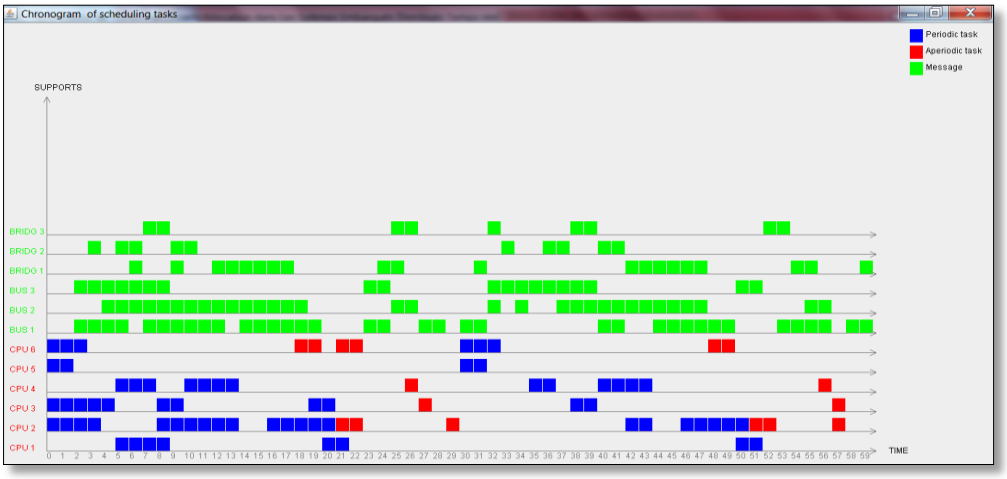


Figure 4.43 : Chronogramme de l'ordonnancement de la meilleure solution (le deuxième exemple).

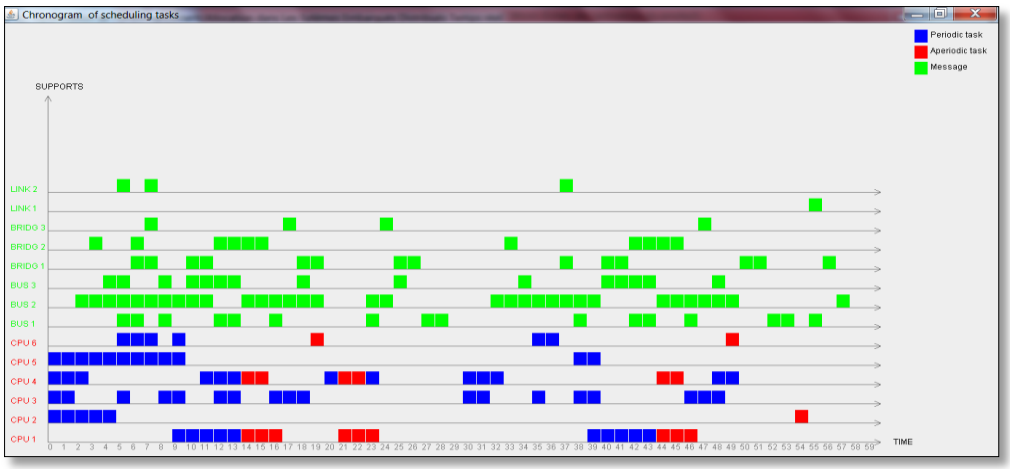


Figure 4.44 : Chronogramme de l'ordonnancement de la meilleure solution (le troisième exemple).

Les figures 4.45, 4.46 et 4.47 représentent les taux d'occupation des processeurs des trois exemples du tableau 4.20 :

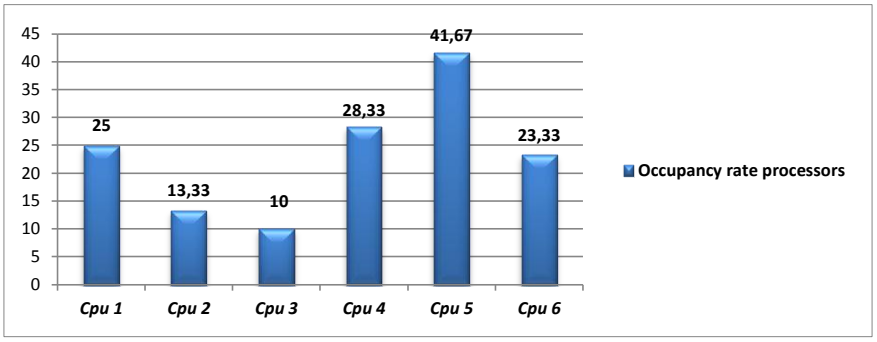


Figure 4.45 : Taux d'occupation des processeurs de la meilleure solution (premier exemple).

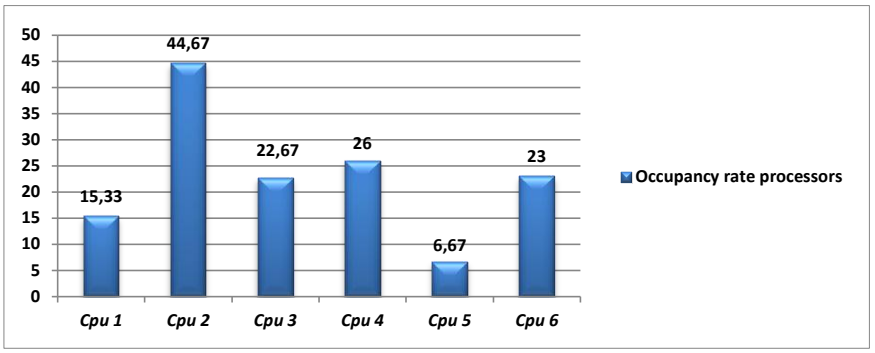


Figure 4.46 : Taux d'occupation des processeurs de la meilleure solution (deuxième exemple).

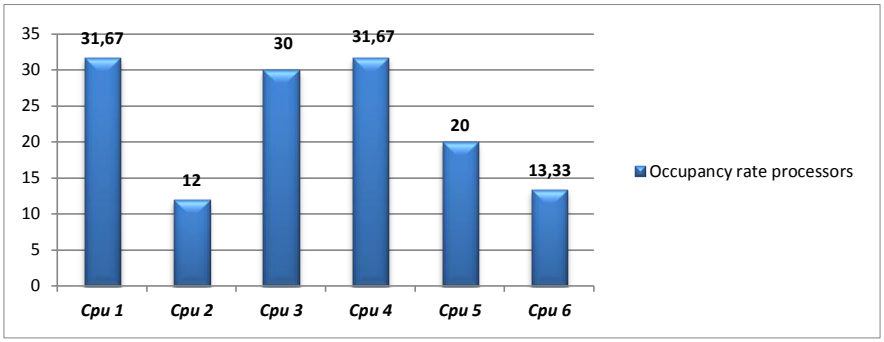


Figure 4.47 : Taux d'occupation des processeurs de la meilleure solution (troisième exemple).

4.4 Quatrième stratégie(AGIQ2):application des algorithmes génétiques inspirés-quantiques en affectant des priorités dynamiques aux tâches

Dans cette stratégie, nous avons essayé d'appliquer un ordonnancement à priorité dynamique.

4.4.1 Organigramme de la quatrième stratégie (AGIQ avec priorités dynamiques)

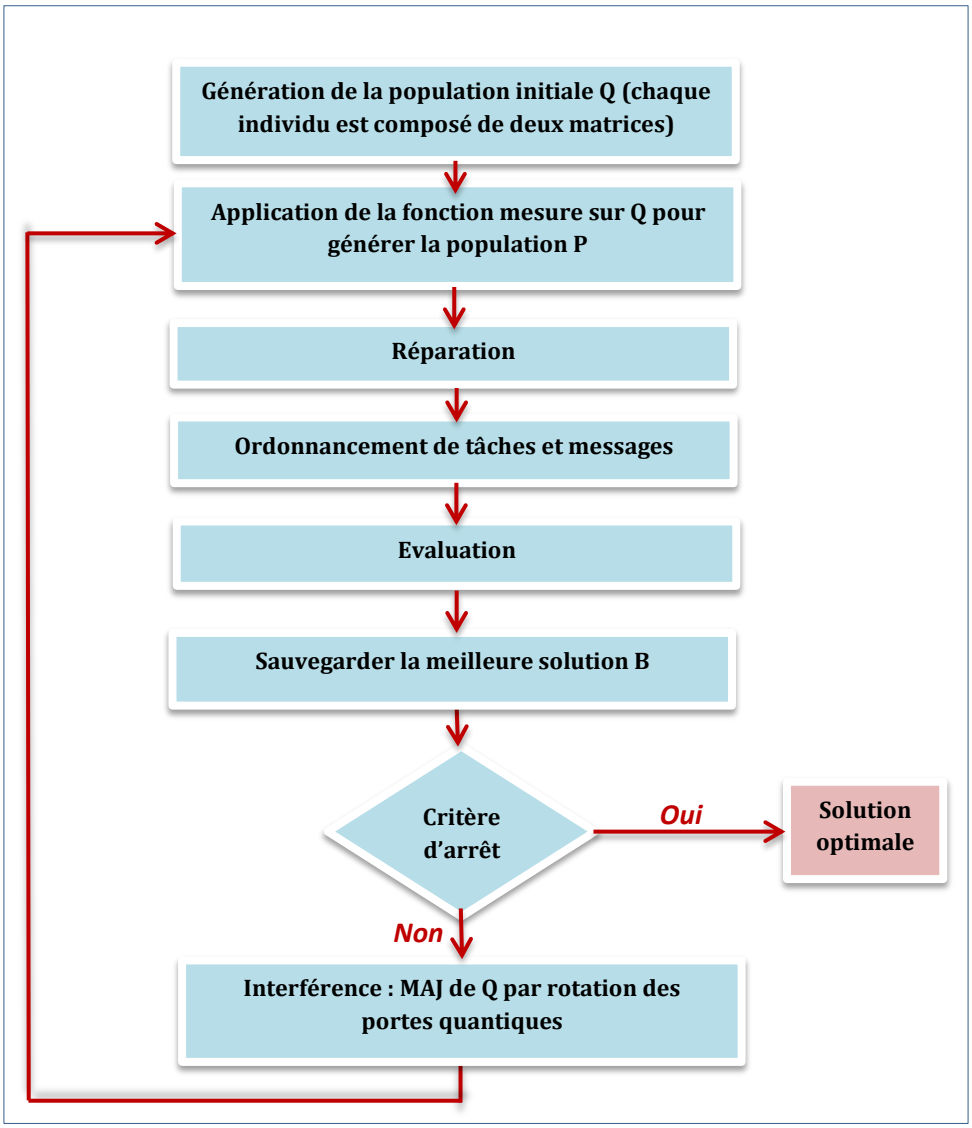


Figure 4.48 : Organigramme général de la quatrième stratégie (AGIQ2).

4.4.2 Principe de fonctionnement de notre algorithme

Comme dans la troisième stratégie, l'algorithme proposé ici fonctionne avec puis sans opérateurs génétiques. L'opérateur d'interférence est appliqué sur les deux matrices à la fois.

4.4.3 Génération de la population initiale « Q »

4.4.3.1 Structure proposé du chromosome quantique

Chaque chromosome est représenté sous forme de deux matrices de qubits, la première pour l'allocation des tâches aux processeurs de longueur *nb_tâche* et de largeur *nb_cpu*, et la deuxième pour l'affectation des priorités aux tâches de longueur *nb_tâche* et de largeur *nb_tâche*. La figure ci-dessous montre la structure d'un chromosome quantique dans AGIQ2:

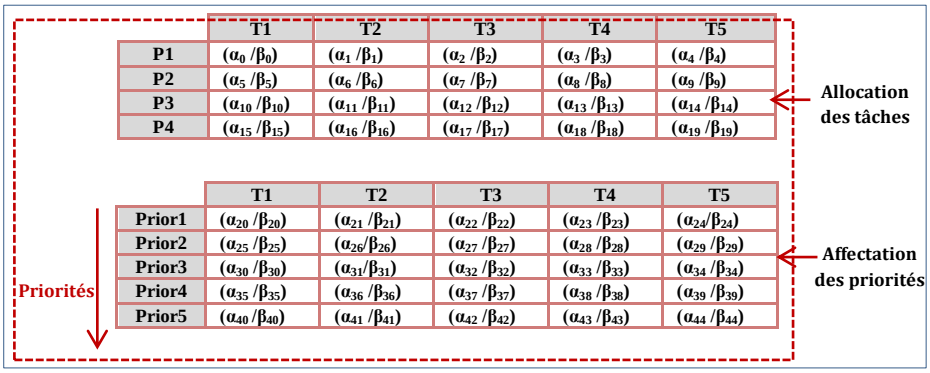


Figure 4.49 : Représentation du chromosome quantique de la quatrième stratégie (AGIQ2).

4.4.3.2 Initialisation de la population

Pour générer la population initiale, il faut créer un ensemble de chromosomes quantiques. Pour cela, toutes les amplitudes des qubits des deux matrices sont initialisées en donnant la même valeur (la valeur $2^{-1/2}$) pour tous les états de superposition.

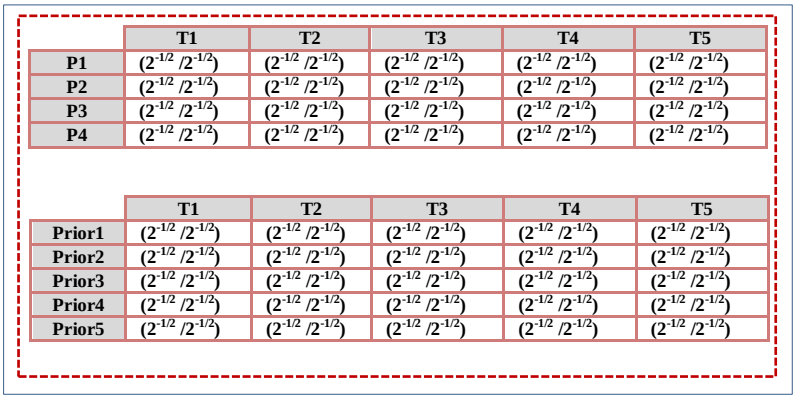


Figure 4.50 : Initialisation de chaque chromosome quantique.

4.4.4 Application de la fonction mesure sur « Q »

Pour chaque chromosome de la population « Q », nous avons fait le transfert de chaque qubit à un bit classique qui porte une seule valeur : 0 ou 1. En effet, pour mesurer un chromosome quantique, nous avons généré pour chaque case des deux matrices un nombre aléatoire dans [0,1], si ce nombre est strictement supérieur à la valeur α^2 de cette case, alors la valeur de la case devient 1, sinon elle devient 0. La nouvelle population générée est appelée « P ».

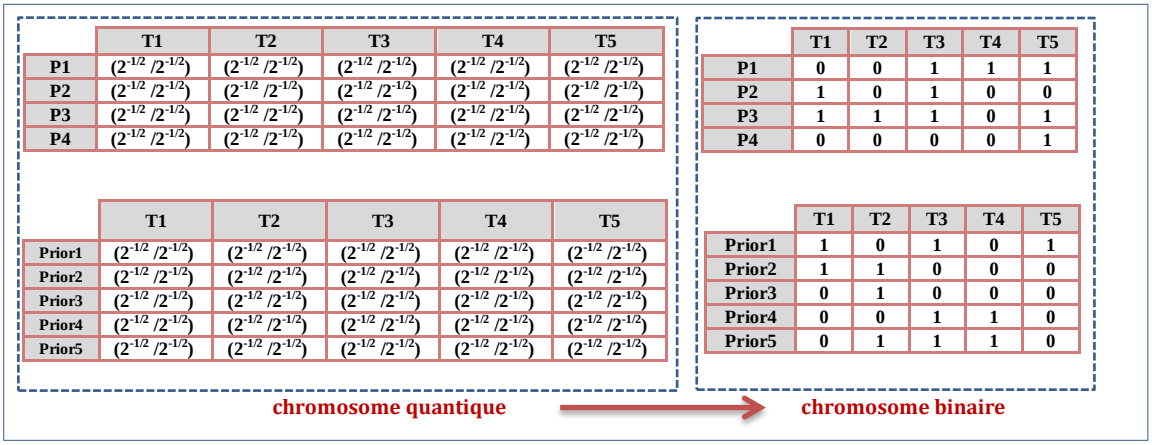


Figure 4.51 : Mesure de chaque chromosome quantique.

4.4.5 Réparation de « P »

Après la fonction mesure, tous les chromosomes quantiques sont transférés aux chromosomes binaires. Mais, il est possible de constater qu'une colonne de la première et / ou de la deuxième matrice contient plusieurs 1, ou que toutes les valeurs de d'une colonne dans l'une ou dans les deux matrices sont des 0. L'interprétation de ces deux cas est que le premier cas signifie que la tâche est affecter aux plusieurs processeurs (pour la matrice d'allocation des tâches) et/ou que la tâches possède plus qu'une seule priorité (pour la matrice d'affectation des priorités), tandis que le deuxième cas signifie que la tâche n'est affectée à aucun processeur et/ou ne possède aucune priorité. Donc, chaque tâche doit être affectée à un et un seul processeur et possède une et une seule priorité pour qu'elle puisse s'exécuter.

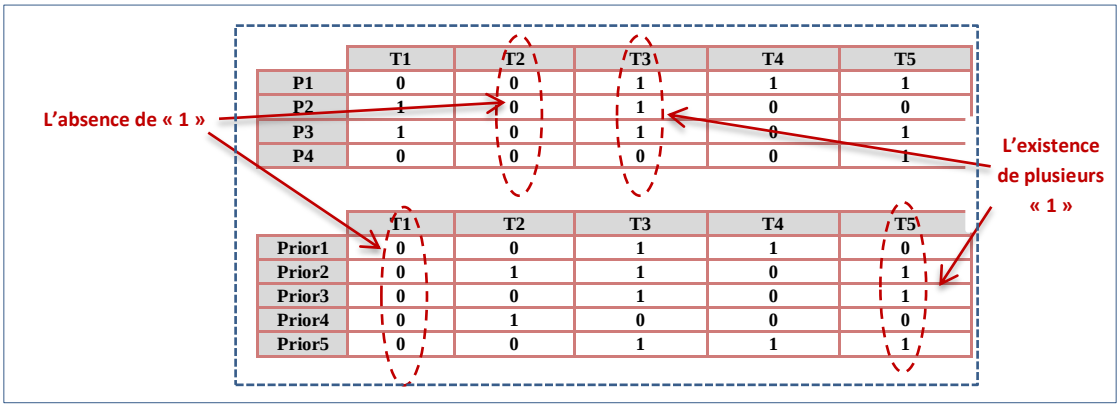


Figure 4.52 : L'exigence d'une réparation dans les deux matrices du chromosome.

Nous avons employé les mêmes fonctions de réparation (réparation1 et réparation2) comme dans la troisième stratégie mais cette fois-ci pour chaque matrice à part.

4.4.6 Ordonnancement de tâches et de messages

Après avoir réparé tous les chromosomes de la population, cela signifie que toutes les tâches sont allouées aux différents processeurs et qu'elles possèdent des priorités différentes.

L'ordonnancement des tâches temps réel et des messages peut se dérouler de la même manière que dans la troisième stratégie, sauf que le changement des priorités des tâches dans chaque itération se fait grâce à l'opérateur d'interférence.

4.4.7 Evaluation et sauvegarde du meilleur individu

Comme dans les deux premières stratégies, Après la phase d'allocation et d'ordonnancement des tâches et des messages, chaque individu de la population doit être évalué, en utilisant une fonction d'adaptation.

La fonction d'adaptation que nous avons voulus optimiser dans ce travail est représentée par l'un des deux facteurs: le temps de réponse moyen ou le nombre de tâches respectant leurs échéances. Après l'exécution de l'algorithme d'ordonnancement, nous pouvons calculer ces facteurs. Nous avons utilisé les mêmes formules utilisées dans les deux premières stratégies pour calculer les deux facteurs.

Dans chaque itération, nous avons sauvegardé la meilleure solution (ayant la valeur d'adaptation la plus élevée) une fois la population est évaluée.

4.4.8 Critère d'arrêt

Un petit test se fait avant de poursuivre la boucle, pour savoir si le nombre max d'itération est atteint. Dans ce cas tout le traitement est arrêté et la solution est obtenue.

4.4.9 Interférence

L'interférence est réalisée à l'aide des portes quantiques, c'est un moyen très puisant pour renforcer la recherche autour de la meilleure solution en cours. Les chromosomes sont alors tout le temps guidés par cette solution actuelle.

Notant que dans cette stratégie, cet opérateur est appliqué sur les deux matrices de chaque individu à la fois.

4.4.10 Tests et résultats de la quatrième stratégie

Nous avons réalisé un ensemble de tests en appliquant l'AGIQ avec priorités dynamique sur les trois exemples cités dans la section 3. La table de recherche adoptée, est la même que dans la troisième stratégie pour les deux matrices de chaque individu (la matrice d'allocation des tâches ainsi que la matrice des priorités).

Les tableaux ci-dessous résument les différents jeux de paramètres appliqués aux trois exemples ainsi que les résultats obtenus:

N° test	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Résultats		
										Trm	T.u. Cpu%	N.t.d
01	20	03	01	00	00	100	40	60	Trm	12.5	33.0	4
02	20	03	01	00	00	100	60	60	Trm	10.1	33.0	3
03	20	04	01	00	00	100	40	60	Trm	10.25	27.0	1
04	20	03	01	00	00	100	40	60	N.t.d	10.08	33.0	0
05	25	03	01	00	00	100	40	60	Trm	13.52	43.0	4
06	25	04	01	00	00	100	40	60	Trm	11.66	35.0	2
07	25	04	01	00	00	100	60	60	Trm	11.3	35.0	2

Tableau 4.21 : Paramètres et résultats de l'AGIQ2 appliqués au 1er exemple.

N° test	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Résultats		
										Trm	T.u. Cpu%	N.t.d
01	20	03	02	01	00	100	40	60	Trm	11.5	33.0	2
02	20	03	02	01	00	100	60	60	Trm	10.93	33.0	2
03	20	04	02	01	00	100	40	60	Trm	13.15	27.0	4
04	20	03	02	01	00	100	40	60	N.t.d	10.2	33.0	0
05	25	03	02	01	00	100	40	60	Trm	14.82	43.0	3
06	25	04	02	01	00	100	40	60	Trm	12.36	35.0	1
07	25	04	03	03	00	100	60	60	Trm	13.91	35.0	4

Tableau 4.22 : Paramètres et résultats de l'AGIQ2 appliqués au 2eme exemple.

N° test	Nb Tach	Nb Cpu	Nb Bus	Nb Pont	Nb Lien	Nb Gen	Nb indiv	Tps Sim	Factr à Optim	Résultats		
										Trm	T.u. Cpu%	N.t.d
01	20	03	02	01	01	100	40	60	Trm	13.7	33.0	4
02	20	03	02	01	01	100	60	60	Trm	10.15	33.0	2
03	20	04	02	01	01	100	40	60	Trm	12.4	27.0	3
04	20	03	02	01	01	100	40	60	N.t.d	10.3	33.0	1
05	25	03	02	01	01	100	40	60	Trm	14.88	43.0	5
06	25	04	02	01	02	100	40	60	Trm	12.44	35.0	4
07	25	04	03	03	01	100	60	60	Trm	12.24	35.0	3

Tableau 4.23 : Paramètres et résultats de l'AGIQ2 appliqués au 3eme exemple.

4.4.11 Discussion

D'après ce qu'on remarque, le changement des priorités n'aide pas à améliorer les résultats. Les valeurs du *Trm* pour les trois exemples sont un peu grandes par rapport à la stratégie précédente. Cela peut être expliqué par le fait que le changement des priorités peut diminuer la possibilité de trouver les bons individus. Une tâche qui possède une grande priorité peut être moins urgente (possède une grande échéance) qu'une autre de basse priorité.

On refaire les mêmes tests en ajoutant les opérateurs génétiques, juste pour voir leurs effets sur les résultats :

Exemples	θij	croisement	mutation	resultats	
				Trm	Ntd
Exemples 1	0,1 π	oui	non	12.1	3
		non	0,3	12.33	3
		oui	0,3	11.14	1
	0,05 π	oui	non	13.0	3
		non	0,3	11.2	1
		oui	0,3	11.24	1
	0,02 π	oui	non	10.1	1
		non	0,3	10.9	1
		oui	0,3	10.09	0
Exemples 2	0,1 π	oui	non	13.55	4
		non	0,3	13.2	3
		oui	0,3	12.7	3
	0,05 π	oui	non	12.29	4
		non	0,3	12.9	4
		oui	0,3	12.2	3
	0,02 π	oui	non	11.5	2
		non	0,3	13.8	1
		oui	0,3	10.92	2

Exemples 3	0,1 π	oui	non	13.83	2
		non	0,3	13.3	3
		oui	0,3	12.1	2
	0,05 π	oui	non	12.14	3
		non	0,3	11.33	1
		oui	0,3	11.3	1
	0,02 π	oui	non	10.1	1
		non	0,3	10.3	2
		oui	0,3	10.2	2

Tableau 4.24 : Effet des opérateurs génétiques sur les résultats des trois exemples.

On remarque que dans la plupart des tests, l'utilisation des deux opérateurs à la fois améliore considérablement les résultats obtenus.

Les meilleurs résultats obtenus pour chaque exemple selon le *Trm* sont les suivants :

Exemple	**Temps de réponse moyen (3 Cpus)**	**Taux d'utilisation des Cpus**	**Nombre de tâches respectant l'échéance**
1	10.08	33.0 %	0
2	10.2	33.0 %	0
3	10.15	33.0 %	2

Tableau 4.25: les meilleurs résultats obtenu par l'AGIQ2 pour 20 tâches.

4.4.12 Résultats des trois exemples

Nous avons refait les tests sur les trois exemples. Cette fois, nous avons fixé les paramètres de la stratégie (**nb_tâches = 30, nb_cpu = 6, Tps_Sim = 60, taille_pop = 40**), et nous avons varié le paramètre du nombre de génération :

Exemples	Nombre de génération	Temps de réponse moyen	Taux d'utilisation des Cpu	Nombre de tâches dépassant l'échéance
Exemple 01 **(bus partagé)**	Itér = 30	12.53	23%	6
	Itér = 60	11.53	23%	6
	Itér = 80	11.27	23%	6
	Itér = 100	10.77	23%	4
Exemple 02 **(3 bus, 3 ponts)**	Itér = 30	10.8	23%	3
	Itér = 60	11.2	23%	5
	Itér = 80	10.8	23%	4
	Itér = 100	10.6	23%	3
Exemple 03 **(3 bus, 3 ponts, 2 liens)**	Itér = 30	11.33	23%	4
	Itér = 60	10.73	23%	3
	Itér = 80	10.17	23%	4
	Itér = 100	10.2	23%	1

Tableau 4.26: Les résultats des trois exemples obtenus par l'AGIQ2.

Les graphes de la figure 4.53 résument les résultats des trois exemples du tableau 4.26 :

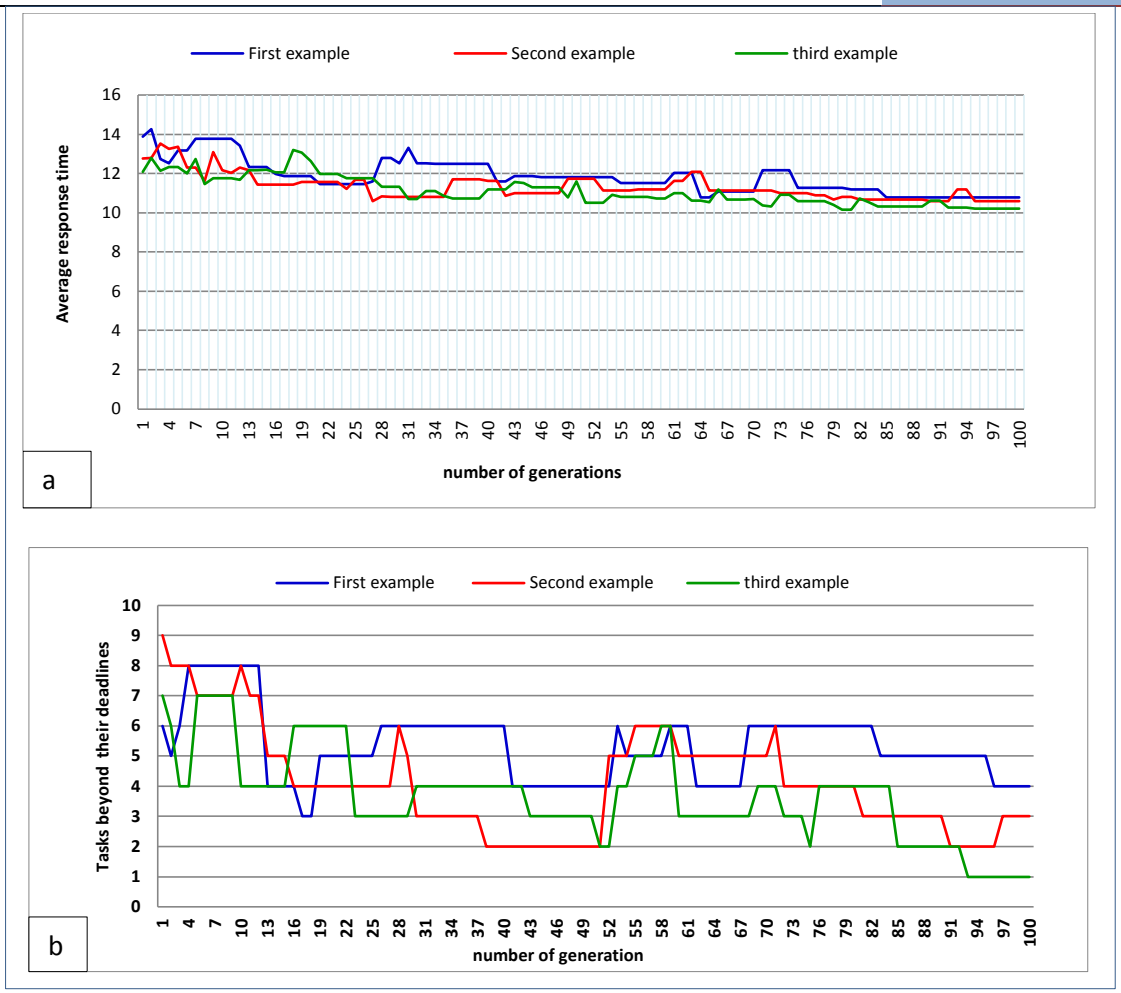


Figure 4.53 : Résultats des trois exemples sous forme de graphes.

- **Discussion**

Selon les courbes de la figure 4.53, on remarque bien que cette stratégie n'apporte pas grand-chose. Les valeurs de *Trm* sont très proches les unes des autres pour les trois exemples. Les valeurs du *Ntd* sont un peu grandes en moyenne, par rapport à ceux du troisième exemple.

Les figures 4.54, 4.55 et 4.56 représentent les chronogrammes des trois exemples du tableau 4.26 :

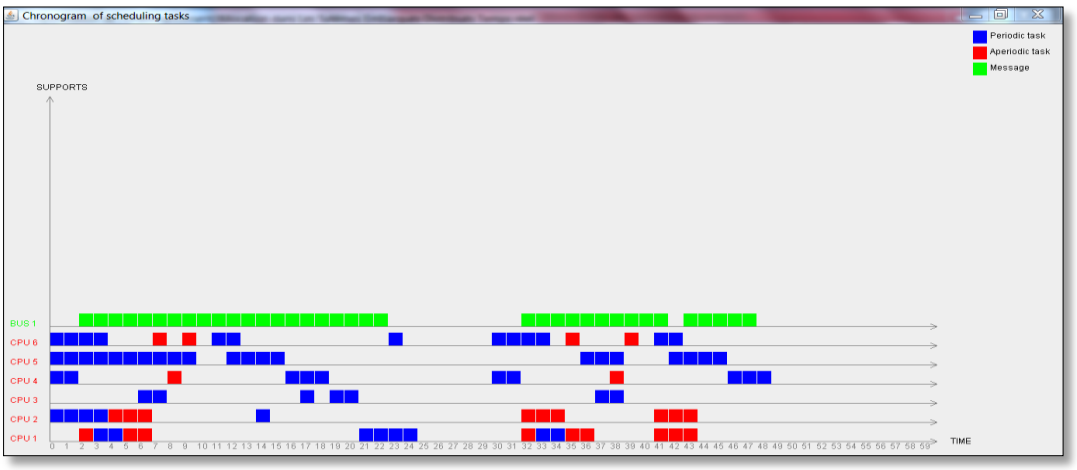


Figure 4.54 : Chronogramme de l'ordonnancement de la meilleure solution (le premier exemple).

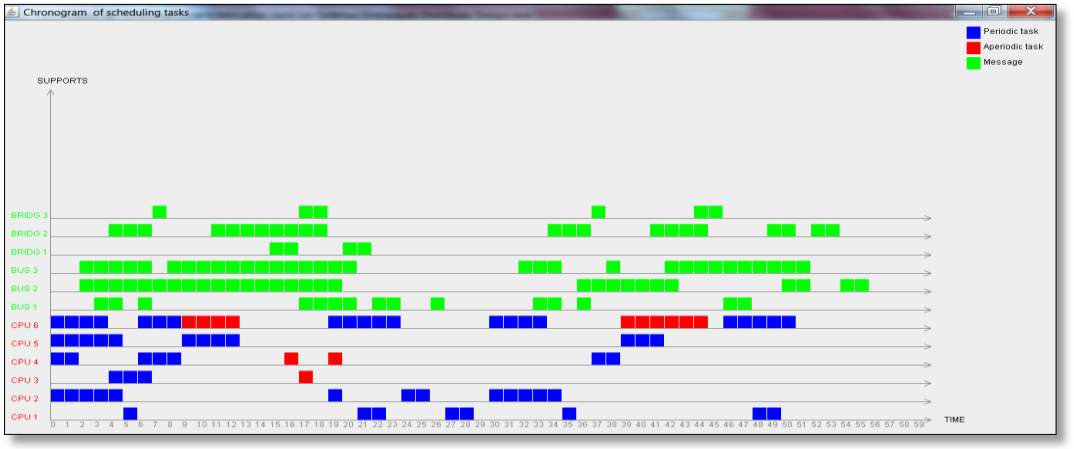


Figure 4.55: Chronogramme de l'ordonnancement de la meilleure solution (le deuxième exemple).

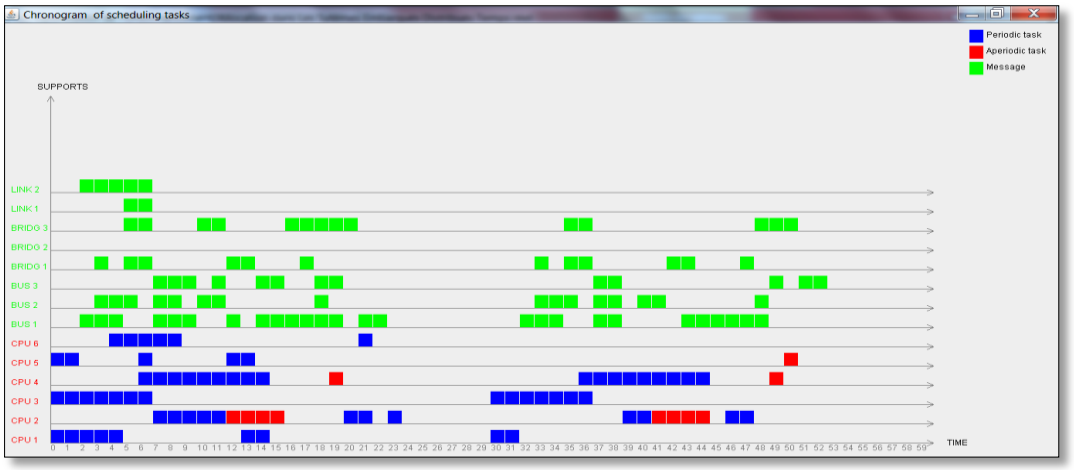


Figure 4.56 : Chronogramme de l'ordonnancement de la meilleure solution (le troisième exemple).

Les figures 4.57, 4.58 et 4.59 représentent les taux d'occupation des processeurs des trois exemples du tableau 4.26 :

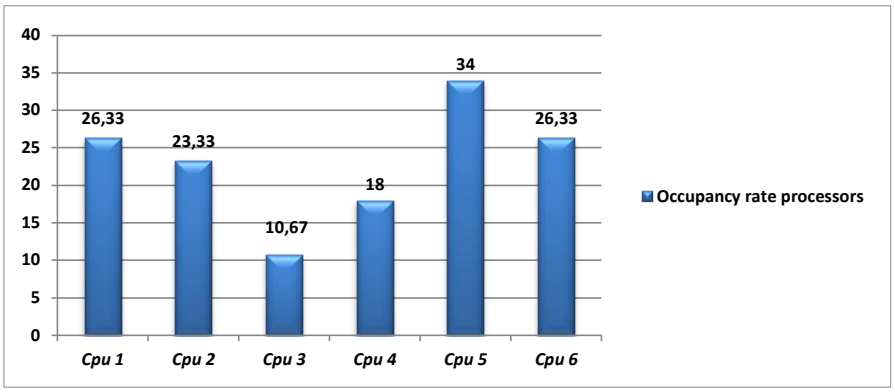


Figure 4.57 : Taux d'occupation des processeurs de la meilleure solution (premier exemple).

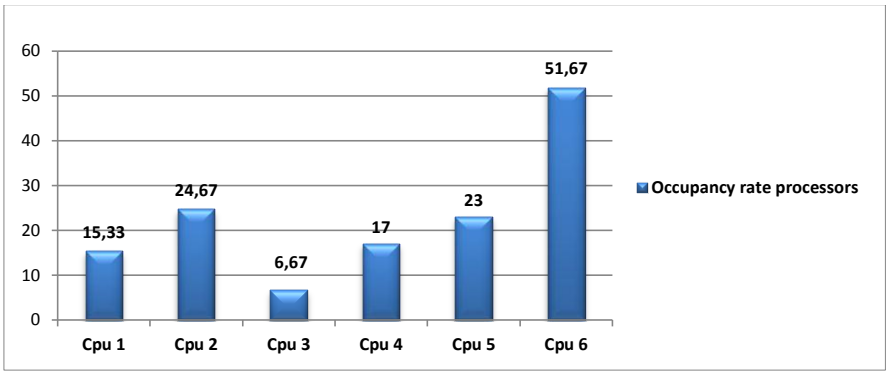


Figure 4.58 : Taux d'occupation des processeurs de la meilleure solution (deuxième exemple).

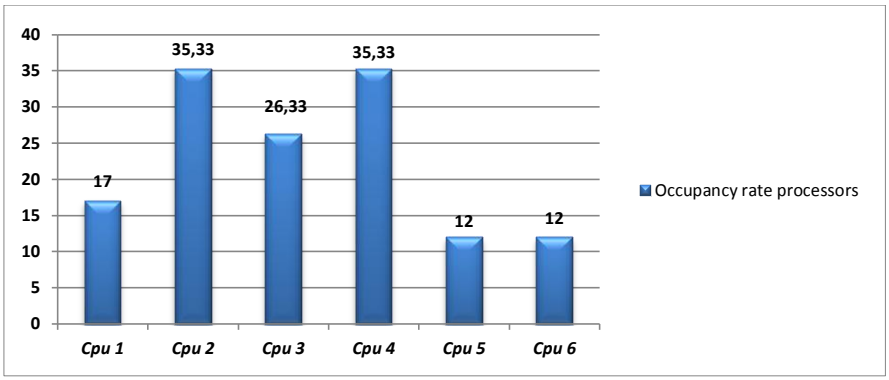


Figure 4.59 : Taux d'occupation des processeurs de la meilleure solution (troisième exemple).

5. Comparaison des résultats des quatre stratégies et discussion

Les quatre stratégies utilisées permettent d'obtenir des solutions réalisables en des temps très acceptables.

Si on compare premièrement entre les deux premières stratégies, on constate que les résultats sont très proches, mais le temps nécessaire (*sur un precesseur Intel core ™ i3 2.40 GHz*) pour trouver la solution optimale de chaque exemple de la première stratégie (environ 13 minutes) est plus petit par rapport à celui de la deuxième stratégie (environ 18 minutes). Cela se traduit par le fait que le changement des priorités dans la deuxième stratégie consomme un peu de temps dû aux opérateurs génétiques qui sont appliqués encore sur les chromosomes représentant les priorités.

Les deux dernières stratégies donnent les meilleures valeurs de *Trm* et de *Ntd* par rapport aux deux premières stratégies, et elles nécessitent un temps plus réduit (environ 9 minutes pour la troisième et 11 minutes pour la quatrième stratégie) pour trouver la solution optimale surtout dans le cas où nous avons utilisé seulement l'opérateur d'interférence. Ce dernier aide à renforcer la recherche dans le voisinage de la meilleure solution et sert à diriger les chromosomes vers la bonne solution.

L'ajout des opérateurs génétiques quantiques peuvent faire changer les probabilités de la superposition des états quantiques. Mais, comme dans les AGIQ la diversité génétique est principalement causée par la représentation du qubit, il n'est pas si nécessaire d'utiliser les autres opérateurs génétiques.

Donc, deux grands avantages évidents de nos algorithmes génétiques inspirés quantiques sont la taille de la population et le temps nécessaire pour trouver la solution optimale qui sont les deux réduits. Les algorithmes génétiques classiques nécessitent souvent plusieurs chromosomes, et une longue durée pour converger.

D'après ce qu'on aperçoit aussi comme résultat pour les quatre stratégies, on ne peut pas juger qu'il existe une relation fixe entre le *Trm* et le *Ntd,* cela s'explique par le fait que le *Trm* est calculé à partir des tâches qui respectent leurs périodes, et dans le cas où le temps de réponse moyen de chaque tâche est grand malgré que le *Ntd* est nul cela va donner en total un *Trm* grand, mais dans le cas où le temps de réponse de la majorité des tâches est un peu petit

malgré que le *Ntd >0* cela nous donne un *Trm* plus réduit. Donc l'augmentation du *Ntd* n'augmente pas forcément le *Trm.*

Le taux moyen d'utilisation (d'occupation) de trois processeurs prend toujours sa valeur max dans les quatre stratégies, et il est égal à *33%* pour *20* tâches et *43%* pour *25* tâches.

L'architecture matérielle peut réduire dans la pluparts des cas le *Trm* et le *Ntd,* et plus particulièrement dans le cas de l'existence des liens directs entre les processeurs, mais une mauvaise distribution des tâches sur les processeurs peut mener à des mauvais résultats.

On peut déduire à la fin que l'adaptation de l'informatique quantique aux algorithmes génétiques, permet de trouver les meilleurs résultats dans des temps de recherche très rapides. L'avantage majeur apporté par cette hybridation réside dans l'indéterminisme, ce facteur permet une large exploration de l'espace de recherche. En plus, les AGC nécessitent le réglage de plusieurs paramètres comme le taux de croisement, le taux de mutation, la méthode de sélection, la méthode du croisement, la méthode de mutation, le point de coupure,… tandis que les AGIQ ne nécessitent que le réglage d'un seul paramètre qui est l'angle de rotation, pour se diriger vers la solution optimale.

Les résultats obtenus par les quatre stratégies montrent le fait qu'il est impossible de choisir une méthode particulière pour résoudre les problèmes d'allocation et d'ordonnancement des tâches dans les systèmes embarqués. Les résultats dépendent fortement du type et de la taille des données d'entrées. La répartition des solutions influe fortement sur l'efficacité des stratégies et des valeurs des paramètres.

6. Conclusion

A travers ce chapitre, nous avons présenté la conception des quatre approches que nous avons proposées pour la résolution du problème d'allocation et d'ordonnancement dans les systèmes embarqués distribués temps réel.

Les deux premières approches représentent l'optimisation par algorithme génétique classique, qui montre une grande efficacité pour la résolution du problème traité. Les deux dernières approches consistent essentiellement dans la représentation de la population manipulée en utilisant un codage quantique et en définissant des opérateurs spécifiques tels que la mesure et l'interférence. En comparant les quatre approches, les approches basées quantique ont l'avantage d'avoir plus de diversité.

Conclusion et perspectives

Les systèmes temps-réel se distinguent des autres systèmes informatiques par la prise en compte de contraintes temporelles dont le respect est aussi important que l'exactitude du résultat. Notre étude s'inscrive principalement dans le domaine de l'ordonnancement multiprocesseur dans les systèmes embarqués distribués temps-réel mixtes avec des contraintes mixtes (dur et souple).

Les tâches temps réel considérées sont des tâches périodiques et apériodiques, à départ simultané et avec des contraintes de précédence. Tandis que l'ordonnancement considéré est un ordonnancement temps réel multiprocesseur par partitionnement, préemptif et hors-ligne. Il convient de noter que nous avons adopté dans notre travail, des politiques d'ordonnancement temps réel comme DM, RM, BS et PS.

Le problème d'ordonnancement préemptif de tâches temps réel avec les contraintes de précédence est un problème complexe, il appartient à la classe des problèmes NP-Complets.

La résolution des problèmes d'allocation et d'ordonnancement, nécessite l'utilisation des méthodes approchées. Nous avons également choisi les méta-heuristiques qui résolvent ce problème comme un problèmed'optimisation.

Pour cela, nous avons premièrement conçu et mis en œuvre, une méthode d'optimisation par algorithmes génétiques classiques (OAGC). Dans un second temps, nous avons décidé d'aborder ce travail à l'aide des algorithmes génétiques inspirés quantiques (AGIQ), en insistant sur l'importance de leur nature quantique.

Nous avons donc proposé deux approches OAGC et OAGIQ, dont la deuxième est une extension et un enrichissement de la première. En effet, les deux approches traitent le problème d'allocation et d'ordonnancement temps réel comme un problème d'optimisation.

L'optimisation par algorithmes génétiques classique a présenté des solutions de qualité intéressante pour la résolution de ce type de problèmes complexes, elle a permis d'obtenir une solution robuste de bonne performance.

L'hybridation entre les algorithmes génétiques et les principes de l'informatique quantique permet d'avoir une stratégie efficace pour la recherche de la bonne solution.

Cette stratégie repose sur un ensemble complémentaire d'opérations quantiques permettant de faire un compromis entre l'exploration et l'exploitation de l'espace de recherche. Elle sert à améliorer le rendement de la première approche en permettant d'accélérer les temps de traitement et de réduire la taille des informations traitées.

Pour concrétiser ce travail et permettre le test des différents algorithmes, nous avons développé un outil composé de quatre stratégies :

- La première stratégie est l'optimisation par algorithmes génétiques classiques en affectant des priorités fixes aux tâches.

- La deuxième stratégie est l'optimisation par algorithmes génétiques classiques en affectant des priorités dynamiques aux tâches.

- La troisième stratégie est l'optimisation par algorithmes génétiques inspirés quantiques en affectant des priorités fixes aux tâches.

- La quatrième stratégie est l'optimisation par algorithmes génétiques inspirés quantiques en affectant des priorités dynamiques aux tâches.

Les quatre stratégies sont appliquées sur plusieurs graphes de tâches. Les résultats montrent que dès les premières itérations, l'algorithme réduit le nombre de tâches dépassant leurs échéances sur chaque processeur. Ce qui signifie que l'algorithme trouve des allocations correctes rapidement.

On peut également conclure, à la lumière des résultats présentés, que l'hybridation avec les principes de l'informatique quantique sert à effectuer des intensifications sur la solution, et permet de produire les meilleurs résultats.

Le résultat final dépend de la politique d'ordonnancement utilisée et des caractéristiques des tâches du système. Cependant, pour une classe de problèmes dits NP-Difficiles, il n'est pas possible de trouver la solution optimale. Les méta-heuristiques représentent alors une alternative pour résoudre ces problèmes et sont reconnues pour donner des résultats très satisfaisants en un temps raisonnable.

Nous avons également rencontré plusieurs difficultés au cours de ce travail, et elles peuvent être résumées dans les points suivants:

- L'absence de règles spécifiques pour choisir les bonnes valeurs des différents paramètres, ce qui nous a obligé de faire plusieurs tests avec différentes combinaisons des valeurs de paramètres (les périodes, les échéances, les temps d'exécution des

différentes tâches, le nombre de processeurs, la taille de la population, le nombre d'itération, méthode de sélection, taux de mutation, angle de rotation, ...etc) ce qui a consommé énormément de temps.

- La difficulté de trouver les meilleurs angles de rotation qui donnent toujours les meilleurs résultats.

Comme perspectives, nous envisageons de :

1. Améliorer nos algorithmes proposés en effectuant plus de tests afin de trouver les meilleurs paramètres et surtout les paramètres probabilistes tels que le taux de mutation et les angles de rotation qui aboutissent aux résultats optimums.
2. Tester d'autres stratégies d'ordonnancement temps réel et en particulier les stratégies traitant les systèmes mixtes avec des tâches périodiques et apériodiques.
3. Prise en compte de la consommation d'énergie comme critère à optimiser et appliquer les nouveaux algorithmes d'ordonnancement temps réel prenant en considération la réduction de la consommation d'énergie (Energy-aware Scheduling Algorithms).
4. Etendre notre démarche pour qu'elle supporte les architectures distribuées embarquées au-delà de la topologies bus et hiérarchisation de bus tels que les réseaux sur puce (NOC : Network On Chip).

Bibliographie

[1] Luigi Zaffalon, Pierre Breguet, « Programmation concurrente et temps reél avec ADA 95 », Presses polytechniques et universitaires romandes, Italie, 2003.

[2] Rémy kocik, « L'optimisation des systèmes distribués temps réel embarqués : application au prototypage rapide d'un véhicule électrique autonome », thèse de doctorat, l'université de Rouen, France, mars 2000.

[3] John A. Stankovic, « Misconceptions About Real-Time Computing : A Serious Problem for Next-Generation Systems». IEEE Computer, 21:10–19, October 1988.

[4] Samia Bouzefrane, « Introduction aux systèmes temps réel», CEDRIC – CNAM Dunod Editeur, Collection SCIENCES SUP, pp. 566, Octobre 2003

[5] Wolfgang A.Halang et Krzysztof M.Sacha, « real-time systems implementation of industrial computeriesd process automation », World scientific Publishing Go.Pte. Ltd, Singapore 2001.

[6] Rajib Mall, « real time systems theory and practice », Dorling Kindersley, India, Ptv. Ltd, 2008.

[7] Mohamed.L Boukhanoufa,« Adaptabilité et reconfiguration des systèmes temps-réel embarqués », thèse de doctorat, l'université paris-sud école doctorale : EDIPS, France, septembre 2012.

[8] Phillip A.Laplante et Seppo J.Ovaska, « real time systems design and analysis», IEEE Inc, Canada, 2012.

[9] Z. Mammeri « Systèmes temps réel et embarqués, Concepts de base, expression des contraintes temporelles », université Paul Sabatier, Toulouse III, France, 2000

[10] Kasim M. Al-Aubidy, « Real-time systems, Classification of Real-Time Systems », Computer Engineering, Department Philadelphia University, Summer Semester, 2011

[11] Shibu K V , « Introduction to embedded systems» , Tata Mc Graw-Hill Education Private Limited, India, 2009.

[12] Etienne Messerli, « Les systèmes embarqués introduction », haute école d'ingénierie et de gestion du Canton de Vaude, Septembre, 2013

[13] Sebastian Fischmeister, « Introduction to Programming Embedded Systems », Department of Computer and Information Science, University of Pennsylvania.

[14] Raj Kamal, « Embedded systems architecture, programming and design », Tata Mc Graw Hill Publishing Company Limited, Nagar New Delhi, 2008.

[15] A.P.Godse et A.O.Mulani, « Embedded systems », Technical Publication Pune, India, 2009.

[16] Traian Pop, «Analysis and Optimisation of Distributed Embedded Systems with Heterogeneous Scheduling Policies» ,memoire de doctorat, université de Linköpings, Sweden,2007.

[17] Mohamed Marouf, «Ordonnancement temps réel dur multiprocesseur tolérant aux fautes appliqué à la robotique mobile », pastel-00720934, version 1 -, Paris, 26 Juil 2012.

[18] Fadia Nemer, «Optimisation de l'estimation du wcet par analyse inter-tâche du cache d'instructions», Université de Toulouse III – Paul Sabatier, France, 2008.

[19] K. Jeffay, D. F. Stanat, C. U. Martel, « On Non-Preemptive Scheduling of Periodic and Sporadic Tasks», In Proceedings of the Twelfth IEEE Real-Time Systems Symposium, San Antonio, Texas, IEEE Computer Society Press, pp. 129-139, December 1991.

[20] H.Cho , B. Ravindran , E. D. Jensen, «An Optimal Real-Time Scheduling Algorithm for Multiprocessors», Proceedings of the 27th IEEE International Real-Time Systems Symposium (RTSS'06), 2006.

[21] Paul Pop, «Scheduling and Communication Synthesis for Distributed Real-Time Systems», mémoire du doctorat, Université de Linköping, Sweden, 2000.

[22] A. Gantman, P.N. Guo, J. Lewis, F. Rashid, «Scheduling Real-Time Tasks in Distributed Systems:A Survey », Université de California, San Diego,1998.

[23] K.Y. Chen, A. Liu and C.H. L. Lee, «A Multiprocessor Real-Time Process Scheduling Method», Proceedings of the IEEE Fifth International Symposium on Multimedia Software Engineering, Taichung, Taiwan, 2003.

[24] Li Jie ; Guo Ruifeng ; Shao Zhixiang , «The research of scheduling algorithms in real-time system», Proceedings of the IEEE International Conference on Computer and Communication Technologies in Agriculture Engineering, Vol.1, Chengdu 2010

[25] L. Zhang, «Scheduling algorithm for real-time applications in grid environment», Systems, Man and Cybernetics, IEEE International Conference, Vol.5, 2002.

[26] K. Zhang , B. Qi , Q. Jiang et L. Tang, «Real-time periodic task scheduling considering load-balance in multiprocessor environment», Network Infrastructure and Digital Content (IC-NIDC), 3rd IEEE International Conference, Beijing, 2012.

[27] Y.X. Dai, B.S. Chen, «An alternate approach for real-time scheduling», Proceedings of TheIEEE International Conference on Industrial Technology, Shanghai, 1996.

[28] I. Stierand, P.Reinkemeier, T. Gezgin, P.Bhaduri, «Real-time scheduling interfaces and contracts for the design of distributed embedded systems», 8th IEEE International Symposium on Industrial Embedded Systems (SIES), Porto, 2013.

[29] Yasmina Berrou, « Evaluation de la fiabilité des systèmes temps réel distribués embarqués», mémoire de magister, Universite El-hadj Lakhdhar Batna, Algérie, 2009.

[30] Anne.M Déplanche, «Ordonnancement temps réel multiprocesseur panorama sur les politiques globales», Ecole d'Eté Temps Réel, 2011.

[31] Audrey Marchand, « Problématique de l'ordonnancement temps-réel », Module E4 : Systèmes temps-réel 2005-2006.

[32] Jobn A. Stankovic, Marco Spuri, Kritbi R, Giorgio C. Buttazzo, « Deadline scheduling for real-time systems edf and Related Algorithms », Kluwer Academic Publishers, Etats-Unis d'Amérique, 1998.

[33] Arezou Mohammadi et Selim G. Akl, «Scheduling Algorithms for Real-Time Systems » Technical Report No. 2005-499, 2005.

[34] Lalatendu B, Durga P M, « Schedulability Analysis of Task Scheduling in Multiprocessor Real-Time Systems Using EDF Algorithm », IEEE International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, INDIA, Jan. 10 – 12, 2012.

[35] Albert M. K. Cheng, « Real-time systems Scheduling, Analysis, and Verification », Publié par John Wiley & Sons, Inc., Hoboken, New Jersey, 2002.

[36] Sanjoy B, Joël G, « Scheduling Real-time Tasks: Algorithms and Complexity», Supported in part by the National Science Foundation, 2003.

[37] Giorgio C. Buttazzo, « Hard Real Time Computing Systems: Predictable Scheduling Algorithms and Applications », Springer Science and Business Media, LLC 2011.

[38] Filip Thoen, Francky Catthoor, « Modeling, Verification and Exploration of Task-Level Concurrency in Real-Time Embedded Systems », Springer Science and Business Media Dordrech, Boston in 2000.

[39] S.Bouzefrane, « Ordonnancement centralisé des tâches temps réel » CEDRIC – CNAM Dunod Editeur, 2003.

[40] Claude Kaiser, « Concepts et modèles pour le Temps Réel », version novembre 2001.

[41] P. Jayachandran, T.Abdelzaher, «The Case for Non-Preemptive Scheduling in Distributed Real-Time Systems », University of Illinois, Urbana-Champaign, IL 61801.

[42] F.Singhof, « Ordonnancement temps réel Monoprocesseur », Bureau C-207 Université de Brest, France.

[43] Insup Lee, Joseph Y-T. Leung, Sang H. Son, «Handbook of Real-Time and Embedded Systems», Taylor & Francis Group, LLC, Etats-Unis d'Amérique, 2008.

[44] Nikita Veshchikov, « Techniques avancées de systèmes d'exploitation »,INFO-F-404.

[45] C. L. Liu. et J. W. Layland, « Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment », Journal of the Association for Computing Machinery, Vol, 20, No. 1, January 1973, pp. 46-61.

[46] Ed Overton, « A Foray into Uniprocessor Real-Time Scheduling Algorithms and Intractibility », thèse de doctorat, December 3, 1997.

[47] F.Cottet, J.Delacroix, C.Kaiser Et Z.Mammeri, «Scheduling in Real-Time Systems», John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2002.

[48] Thi Huyen Chau NGUYEN, «Approximation des temps de réponse des tâches sporadiques à priorité fixe dans les systèmes monoprocesseurs », thèse de doctorat, France 2010.

[49] G. C. Buttazzo, « Rate Monotonic vs. EDF: Judgment Day ». Real-Time Systems, 29, 5–26, Springer Science and Business Media, Inc. Manufactured in The Netherlands, 2005.

[50] Jane W. S. Liu, « Real-Time systems »,Integre Technical Publishing Co., Inc. Liu January 13, 2000.

[51] Francisco Vasques de Carvalho, « L'integration de mecanismes d'ordonnancement et de communication dans la sous-couche mac de reseaux locaux temps-reel », Chapitre II, 25 Juin 1996.

[52] M. Teresa, A.J. Wellings, « Distributed, Embedded and Real-time Java Systems », Springer Science and Business Media, LLC, Spain, 2012.

[53] B. Sprunt, L. Sha, and J. Lehoczky,« Aperiodic task scheduling for hard real time systems». Journal of Real-Time Systems, 1, July 1989.

[54] F. DORIN, « Contributions à l'ordonnancement et l'analyse des systèmes temps réel critiques », thèse de doctorat, 30 septembre 2010.

[55] Arezou Mohammadi and Selim G. Akl , « Scheduling Algorithms for Real-Time Systems», supported by the Natural Sciences and Engineering Research Council of Canada July, 2005.

[56] Nathan W Fisher, « The Multiprocessor Real-Time Scheduling of General Task Systems», mémoire de doctorat, University of North Carolina at Chapel Hill, 2007.

[57] J.Carpenter, S.Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, «A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms», Department of Computer Science, University of North Carolina at Chapel Hill, 2004.

[58] M. Chéramy, A.M. Déplanche, et P.E. Hladik, « Ordonnancement temps réel, des politiques monoprocesseurs aux politiques multiprocesseurs », hal-00662741, version 1 – France, 25 Jan 2012.

[59] D. Aoun, A.M. Déplanche, Y. Trinquet, « Pfair scheduling improvement to reduce interprocessor migrations », published in "16th International Conference on Real-Time and Network Systems (RTNS), 2008.

[60] Joël Goossens, « Ordonnancement multiprocesseur », Université Libre de Bruxelles ETR'07 — 6 septembre 2007.

[61] J. Anderson, V. Bud, et U. Devi, « An EDF-based scheduling algorithm for multiprocessor soft real-time systems ». In Proc. of the 17th Euromicro Conf. on Real-Time Sys., pp. 199–208, 2005.

[62] Mitchell Melanie, « An Introduction to Genetic Algorithms », Massachusetts Institute of Technology, London, England, 1996.

[63] John H. Holland, « Adaptation in Natural and Artificial Systems », The MIT Press, 1992.

[64] Evelyne Lutton, « Darwinisme artificiel: une vue d'ensemble », traitement du signal 2005_volume 22_numéro 4 Méthodologie de la gestion intelligente des senseurs, France, 2005.

[65] Laboudi Zakaria, « Evolution d'automates cellulaires par algorithmes génétiques quantiques sur un environnement parallèle », mémoire de magister, Constantine, Algérie, 2009.

[66] Rustem Popa, « Genetic algorithms in applications », ISBN 978-953-51-0400-1, 328 pages, Publisher: InTech, Chapters published March 21, 2012.

[67] Tarek CHAARI, « Un algorithme génétique pour l'ordonnancement robuste : application au problème du flow shop hybride », thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis, 2010.

[68] K De Jong, L Fogel, H-P Schwefel, « Handbook of Evolutionary Computation », IOP Publishing Ltd and Oxford University Press, 1997.

[69] RC Chakraborty, « Fundamentals of genetic algorithms: AI course », Lecture 39-40, june 01,2010.

[70] Ulrich Bodenhofer, « Genetic Algorithms: Theory and Applications », Département de l'algèbre, Stochastiques et systèmes mathématiques fondées sur le savoir, l'Université Johannes Kepler A-4040 Linz, Autriche, 2002

[71] Franz Rothlauf, « Representations for Genetic and Evolutionary Algorithms », Soficover reprint of the hardcover 1st edition, 2002.

[72] Randy L. Haupt, Sue Ellen Haupt, «Practical Genetic Algorithms », Published by John Wiley & Sons, Inc., Hoboken, New Jersey. 2004

[73] E. Hou, N. Ansari, and H. Ren, « A Genetic Algorithm for Multiprocessor Scheduling », IEEE Trans. Parallel and Distributed Systems, vol. 5, no. 2, pp. 113±120, Feb. 1994.

[74] P.-C. Wang, W. Korfhage, « Process Scheduling Using Genetic Algorithms », IEEE Symp. Parallel and Distributed Processing, pp. 638±641 San Antonio, Texas, Oct. 1995.

[75] R. C. Corrêa, A. Ferreira, and P.Rebreyend, « Scheduling Multiprocessor Tasks with Genetic Algorithms », IEEE transactions on parallel and distributed systems, Vol. 10, No. 8, August 1999.

[76] M.K Dhodhi, I.AhmadetIshfaq Ahmed, « A multiprocessor scheduling scheme using problem-space genetic algorithms », Evolutionary Computation, IEEE International Conference, Vol.1, Perth, WA, Australia, Dec.1995 .

[77] M. Rinehart, V. Kianzad, and S. S. Bhattacharyya, « A Modular Genetic Algorithm for Scheduling Task Graphs », Technical Report UMIACS-TR-2003-66, Institute for Advanced Computer Studies, University of Maryland at College Park, June 2003.

[78] Y. Li, Y. Yang, M. Ma, R. Zhu, «A Problem-Specific Genetic Algorithm for Multiprocessor Real-time Task Scheduling», The 3rd Intetnational Conference on Innovative Computing Information and Control IEEE 2008.

[79] M. S. Jelodar, S. N. Fakhraie, F. Montazeri, S. M. Fakhraie, M. NiliAhmadabadi, « A Representation for Genetic-Algorithm-Based Multiprocessor Task Scheduling», 2006 IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada July 16-21, 2006.

[80] M. R. Miryani, M.Naghibzadeh, «Hard Real-Time Multiobjective Scheduling in Heterogeneous Systems Using Genetic Algorithms», 2009 IEEE Proceedings of the 14th International CSI Computer Conference (CSICC'09), 2009.

[81] G.Zarinzad, A. m. Rahmani, N.Dayhim, « A Novel Intelligent Algorithm for Fault-Tolerant Task Scheduling in Real-Time Multiprocessor Systems »,2008 IEEE, Third 2008 International Conference on Convergence and Hybrid Information Technology, 2008.

[82] N.Sedaghat, H. T. Yazdi, and M. Akbarzadeh-T3, « Pareto Front Based Realistic Soft Real-Time Task Scheduling with Multi-objective Genetic Algorithm in Unstructured Heterogeneous Distributed System», 5th International Conference, GPC 2010, Hualien, Taiwan, May 10-13, 2010.

[83] Richard P. Feynman « Simulating Physics with Computers», International Journal of Theoretical Physics, VoL 21, Nos. 6/7, 1982.

[84] Eleanor R, Wolfgang P, « An Introduction to Quantum Computing for Non-Physicists», FX Palo Alto Laboratory , jan 2000.

[85] Peter.W. Shor, « Algorithms for Quantum Computation: Discrete Logarithms and Factoring», », IEEE Inc, USA, 1994.

[86] Lov K. Grover, « A fast quantum mechanical algorithm for database search», in Proceedings, STOC 1996, Philadelphia PA USA, pages 212-219, 1996.

[87] B Georgeot, D L. Shepelyansky , «Les ordinateurs quantiques affrontent le chaos », Laboratoire de physique théorique, université Paul Sabatier, 31062 Toulouse, 2003.

[88] Gregg Jaeger, « Quantum Information: An Overview », Springer Science and Business Media, LLC, 2007.

[89] Michael A. Nielsen & Isaac L. Chuang, « Quantum Computation and Quantum Information », Published in the United States of America by Cambridge University Press, New York, 2010.

[90] Y. Leroyer, Enseirb-Matmeca, « Introduction à l'information quantique », 2013.

[91] Trégouët, René, « L'ordinateur quantique va révolutionner l'informatique », Hegel Vol. II N° 4, ALN Editions, Nancy, France, 2012.

[92] Abdesslem Layeb, Djamel-Eddine Saidouni, « Quantum Genetic Algorithm for Binary Decision Diagram Ordering Problem », IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.9, 2007.

[93] Jonathan A. Jones, Dieter Jaksch, « Quantum Information, Computation and Communication », Published in the United States of America by Cambridge University Press, New York, 2012.

[94] A.Narayanan, M.Moore, « Quantum-inspired genetic algorithm», In Proceedings of the IEE International Conference on Evolutionary Computation,pp41,46, 1996.

[95] Kuk-Hyun Han, Jong-Hwan Kim, « Genetic Quantum Algorithm and its Application to Combinatorial Optimization Problem», IEEE Proc. Of the 2000 Congress on Evolutionary Computation, pp. 1354-1360, 2000.

[96] Kuk-Hyun Han, Jong-Hwan Kim, « Quantum-Inspired Evolutionary Algorithms With a New Termination Criterion, H ϵ Gate, and Two-Phase Scheme», IEEE Transactions On Evolutionary Computation, Vol. 8, No. 2, April 2004.

[97] J. Blazewicz, «Scheduling dependent tasks with different arrival times to meet deadlines» in E. Gelembre, H. Beiher Modelling and performance evaluation of computer systeme Nort-holland, amsterdem, 1976.

[98] M. Hamdaoui, P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines", IEEE Transactions on Computers 44, December 1995, p. 1443–1451.

[99] A. Srinivasan and J. Anderson, "Fair scheduling of dynamic task systems on multiprocessor", The Journal of Systems, vol. 77, (2005), pp. 67-80.

[100] A. Page and J. Naughton, "Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing", The proceedings of the 19th International Parallel & Distributed Processing Symposium, Denver, USA IEEE Computer Society, (2005).

[101] B. Andersson and J. Jonsson, “Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition”, Seventh International Conference on Real-Time Computing Systems and Applications (RTCA’00), (2000).

Webographie

[102] http://www.webopedia.com/TERM/R/real_time.html. Consulté le 07/11/2013.

[103] http://en.wikipedia.org/wiki/Embedded_system. Consulté le 08/11/2013.

[104] http://fr.slideshare.net/Flashdomain/embedded-systemppt. Consulté le 11/11/2013.

[105] https:// docs.google.com / file/ d/ 0B21HoBq6u9TsUWhidWc1RGxidWc/ edit?pli = 1. Consulté le 02/12/2013.

[106] http : // david.decotigny. free.fr/ rt/ intro-ordo/ intro-ordo004. html. Consulté le 28/11/2013.

[107] http://www.citation-celebre.com/citation/genetique. Consulté le 20/12/2013.

[108] http://khayyam.developpez.com/articles/algo/genetic/. Consulté le 31/12/2013.

[109] http: // www. terresacree. org/ actualites/ 1643/ actualite - l- ordinateur – quantique - la-nouvelle revolution-informatique-108398. Consulté le 04/12/2013.

[110] http://www.lesechos.fr/26/05/2013/lesechos.fr/0202786184905_l-ordinateur quantique--la-nouvelle-revolution-informatique.htm. Consulté le 05/01/2014.

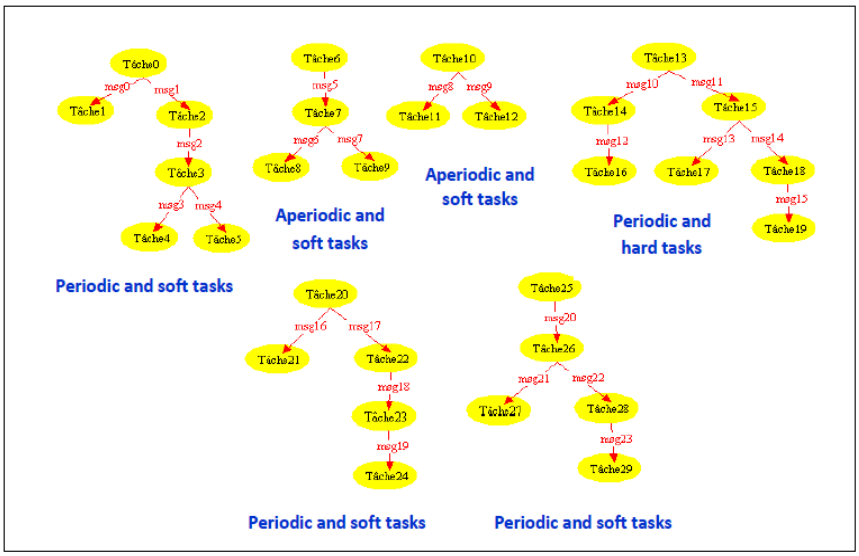
[111] http://www.tomshardware.fr/articles/D-Wave-Orion,1-19849.html, consulté le : 18/11 /2013.

[112] http://www.techno-science.net/?onglet=glossaire&definition=8007, consulté le : 05/01 /2014.

Annexe

Cette annexe présente les principaux jeux de données sur lesquels ont été testés les algorithmes AGC et AGIQ présentés dans le chapitre 4.

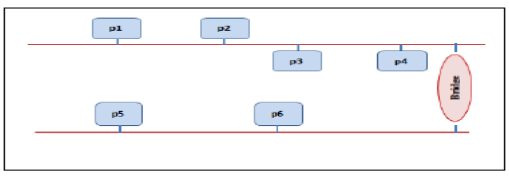
1- ***le graphe de tâches :*** composé de 30 tâches de types et de contraintes mixtes.



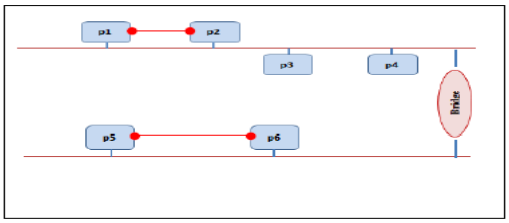
2- ***le graphe d'architecture :***

- Un seul bus partagé :

- Au moins deux bus :



- Au moins deux bus + liens :



3- les jeux de données associés :

Tâches Périodiques									
Nom tâche	**Num tâche**	**Num Sous /graphe**	**deadline**	**periode**	**Msg d'entré**	**Msg de sortie**	**serveur**	**contraine**	**Wcet /Acet**
Tâche0	0	0	22	30	/	0, 1	non	soft	2
Tâche1	1	0	22	30	0	/	oui	soft	12
Tâche2	2	0	25	30	1	2	non	soft	2
Tâche3	3	0	25	30	2	3, 4	non	soft	2
Tâche4	4	0	25	30	3	/	non	soft	2
Tâche5	5	0	28	30	4	/	non	soft	3
Tâche13	13	3	50	60	/	10, 11	non	hard	2
Tâche14	14	3	50	60	10	12	non	hard	2
Tâche15	15	3	50	60	11	13, 14	non	hard	1
Tâche16	16	3	50	60	12	/	non	hard	2
Tâche17	17	3	55	60	13	/	non	hard	1
Tâche18	18	3	55	60	14	15	non	hard	2
Tâche19	19	3	55	60	15	/	non	hard	1
Tâche20	20	4	20	30	/	16, 17	non	soft	3
Tâche21	21	4	20	30	16	/	non	soft	2
Tâche22	22	4	20	30	17	18	non	soft	1
Tâche23	23	4	28	30	18	19	non	soft	3
Tâche24	24	4	28	30	19	/	non	soft	2
Tâche25	25	5	52	60	/	20	non	soft	3
Tâche26	26	5	52	60	20	21, 22	non	soft	2
Tâche27	27	5	55	60	21	/	non	soft	1
Tâche28	28	5	55	60	22	23	non	soft	2
Tâche29	29	5	55	60	23	/	non	soft	1

Tâches Apériodiques								
Nom tâche	**Num tâche**	**Num Sous /graphe**	**Date d'arrivé**	**Msg d'entré**	**Msg de sortie**	**deadline**	**contraine**	**Wcet /Acet**
Tâche6	6	1	1	/	5	18	soft	2
Tâche7	7	1	1	5	6, 7	18	soft	1
Tâche8	8	1	2	6	/	18	soft	1
Tâche9	9	1	1	7	/	20	soft	1
Tâche10	10	2	1	/	8, 9	18	soft	2
Tâche11	11	2	2	8	/	20	soft	1
Tâche12	12	2	1	9	/	20	soft	1

Messages					
Nom message	**Num message**	**Num sous graphe**	**Tâche source**	**Tâche cible**	**Taille**
Msg0	0	0	0	1	2
Msg1	1	0	0	2	2
Msg2	2	0	2	3	3
Msg3	3	0	3	4	2
Msg4	4	0	3	5	3
Msg5	5	1	6	7	2
Msg6	6	1	7	8	1
Msg7	7	1	7	9	2
Msg8	8	2	10	11	2
Msg9	9	2	10	12	1
Msg10	10	3	13	14	2
Msg11	11	3	13	15	2

Messages					
Nom message	**Num message**	**Num sous graphe**	**Tâche source**	**Tâche cible**	**Taille**
Msg12	12	3	14	16	2
Msg13	13	3	15	17	3
Msg14	14	3	15	18	3
Msg15	15	3	18	19	2
Msg16	16	4	20	21	2
Msg17	17	4	20	22	2
Msg18	18	4	22	23	2
Msg19	19	4	23	24	3
Msg20	20	5	25	26	3
Msg21	21	5	26	27	2
Msg22	22	5	26	28	2
Msg23	23	5	28	29	2