ALLOCATION DE TÂCHES INDÉPENDANTES

3.1 Introduction

Dans ce chapitre, nous étudions les algorithmes appliqués aux tâches avant le démarrage de l'ordonnancement. Il s'agit d'une approche par partitionnement (FTII) où nous considérons que l'ordonnancement en ligne est réalisé par un algorithme de la classe FTP. Nous considérons des *tâches sporadiques* à *échéan-ces contraintes*. Ces tâches sont indépendantes dans le sens où elles ne partagent pas de ressources communes.

L'approche d'ordonnancement temps réel multiprocesseur hors-ligne la plus représentative est l'approche par partitionnement. Chaque sous-ensemble de tâches obtenu à partir d'un partitionnement est ensuite ordonnancé de manière indépendante sur chaque processeur. Cette approche interdit donc les migrations inter-processeurs. Mais nous pouvons tout à fait concevoir des approches d'ordonnancement pour lesquels des schémas de migrations sont connus à l'avance.

Nous proposons un algorithme de partitionnement qui soit robuste aux variations négatives de WCET et de périodes. Plus particulièrement, un algorithme pour lequel la marge sur le WCET ou la période des tâches est maximisée. L'aspect de la robustesse qui est ici considéré consiste à rendre le système plus tolérant à des fautes, à de mauvaises estimations de WCET ou à un mauvais dimensionnement au niveau des périodes.

Dans la section 3.2, nous décrivons les modèles et les notations utilisés dans ce chapitre. Dans la section 3.3, nous présentons la problématique de l'ordonnancement par partitionnement. Dans la section 3.5, nous discutons des propriétés de robustesse des algorithmes de partitionnement. Enfin, nous proposons une synthèse de ce chapitre dans la section 3.6.

3.2 Notations

Notation	Définition
Π	Un ensemble de processeurs
π_j	Le processeur d'indice j
τ	Un ensemble de tâches
τ_i	La tâche d'indice i
$\tau(\Pi)$	L'ensemble des tâches assignées sur un ensemble de processeurs Π
$\tau(\pi_j)$	L'ensemble des tâches assignées sur le processeur π_j

TABLE 3.1 – Notations employées dans le chapitre 3.

La table 3.1 contient la synthèse des notations utilisées dans ce chapitre.

3.3 Partitionnement

Nous nous intéressons dans cette section au modèle de tâches indépendantes. L'approche par partitionnement consiste à subdiviser un ensemble de tâches en autant de sous-ensembles disjoints qu'il y a de processeurs. Chaque sous-ensemble est ensuite ordonnancé sur un processeur, et ce sans migration. Cette approche a l'avantage qu'une fois le partitionnement connu, les résultats provenant de l'état de l'art sur l'ordonnancement temps réel monoprocesseur sont valides. La difficulté dans l'étude de l'ordonnancement par partitionnement réside dans la recherche de partitions ordonnançables.

3.3.1 Anomalie de partitionnement

Certains algorithmes de partitionnement peuvent être sujets à des anomalies. Andersson et Jonsson montrent dans [AJ02] que des anomalies peuvent se produire avec l'algorithme *R-BOUND-MP* [LMM98] lorsque les périodes des tâches sont augmentées. Ce phénomène se produit car la condition d'admission *R-BOUND* (utilisée par *R-BOUND-MP*) exploite le rapport entre la période maximale et la période minimale des tâches à assigner. Une période plus grande implique un rapport différent qui peut modifier la condition d'admission. Un ensemble de tâches pouvant être partitionné par *R-BOUND-MP* peut ne plus pouvoir l'être avec une ou plusieurs tâches ayant des périodes plus grandes.

Dans le cas où le partitionnement est effectué hors-ligne, le problème des anomalies n'est qu'un problème de dimensionnement du système. En effet, le fait qu'un ensemble de tâches ne puisse pas être partitionné avec des périodes plus grandes ne conduit pas à ce que des échéances soient dépassées.

3.4 Algorithmes de partitionnement

Nous considérons une plate-forme multiprocesseur composée de processeurs identiques. Étant donné un ensemble de tâches sporadiques caractérisées par leur utilisation, nous pouvons nous poser la question de savoir combien de processeurs sont nécessaires pour pouvoir ordonnancer cet ensemble de tâches. Cette formulation du problème est très similaire à celle du problème de BIN-PACKING.

Définition 3.1 (Problème de BIN-PACKING)**.** *Étant donné des boîtes de taille V et un ensemble d'objets de taille inférieure ou égale à V à ranger dans les boîtes, le problème consiste à trouver le nombre minimum de boîtes nécessaires pour ranger les objets.*

Il est aisé de remarquer que nous pouvons transformer une instance de notre problème de partitionnement d'un ensemble de tâches en une instance du problème BIN-PACKING. Il suffit de considérer nos processeurs comme étant des boîtes dont la taille serait la puissance de calcul des processeurs et de considérer nos tâches comme des objets à ranger dont la taille serait l'utilisation de ces tâches. Nous pouvons maintenant résoudre notre problème de partitionnement en utilisant un algorithme permettant de résoudre le problème de BIN-PACKING. Malheureusement, il a été montré que ce dernier est un problème *NP-difficile* [GJ79]. Cela signifie qu'à moins qu'il soit prouvé que $P = NP$, aucun algorithme ne peut garantir de trouver la solution optimale à ce problème en temps polynomial. Une solution optimale correspondrait, pour notre problème, au nombre minimal de processeurs nécessaires pour pouvoir ordonnancer un ensemble de tâches. Heureusement, le problème de BIN-PACKING a été largement étudié et il existe de nombreuses heuristiques permettant de le résoudre sans garantie de trouver la solution optimale.

Une autre question que nous pouvons nous poser est de savoir si il existe un partitionnement d'un ensemble de tâches pour une plate-forme multiprocesseur donnée, en d'autres termes lorsque le nombre de processeurs est fixé. Ce problème correspond au problème de décision associé au problème BIN-PACKING. La description du problème donnée dans la définition 3.1 correspond au problème de minimisation. Pour résoudre le problème de décision, il suffit d'appliquer un algorithme résolvant le problème de minimisation et de vérifier que la valeur de la solution retournée est inférieure ou égale au nombre de processeurs.

Bien que nous n'ayons pas à notre disposition d'algorithme garantissant de trouver les solutions optimales en temps polynomial, la littérature concernant le problème BIN-PACKING contient plusieurs algorithmes donnant de bons résultats en temps polynomial. Naturellement, la communauté de l'ordonnancement temps réel s'est inspirée de ces derniers pour proposer des heuristiques de partitionnement.

3.4.1 Algorithmes pour BIN-PACKING dans le cas du partitionnement

Nous comparons ici les différents algorithmes utilisés pour résoudre le problème BIN-PACKING dans le but de choisir celui qui permettra de fournir des partitions les plus robustes.

```
   Entrées : τ                                  /* Un ensemble de tâches */
   Sorties : k                                /* Un nombre de processeurs */
 1 Π ← π_1;
 2 k ← 1;
 3 pour τ_i ∈ τ faire
 4     pour j ∈ {1, ..., k} faire                        /* Ordre croissant */
 5         si τ_i est ordonnançable sur π_j alors
 6             τ(π_j) ← τ(π_j) ∪ τ_i;
 7             sortie de boucle;
 8         fin
 9     fin
10     si τ(Π) ∩ τ_i = ∅ alors                  /* τ_i n'est pas assignée */
11         k ← k + 1;
12         Π ← Π ∪ π_k;                       /* Ajout d'un processeur */
13         τ(π_k) ← τ(π_k) ∪ τ_i;                          /* Allocation */
14     fin
15 fin
16 retourner k
```

Algorithme 3.1: FF

First-Fit L'algorithme *First-Fit (FF)* associe un ordre à l'ensemble des tâches. Chaque tâche est ensuite assignée sur le premier processeur pour lequel le test d'ordonnançabilité réussit. Ce test dépend de l'algorithme d'ordonnancement choisi pour ordonnancer les ensembles de tâches sur les processeurs. Les processeurs sont considérés dans l'ordre croissant de leur indice. Si une tâche ne peut être assignée à aucun processeur, alors un processeur est ajouté à l'ensemble des processeurs et la tâche y est assignée.

Last-Fit L'algorithme *Last-Fit (LF)* est similaire à FF à la différence que les processeurs sont pris dans l'ordre décroissant de leur indice.

Next-Fit L'algorithme *Next-Fit (NF)* se comporte comme FF, mais avec la particularité qu'un processeur n'ayant pu admettre une tâche n'est plus considéré pour l'affectation. Cet algorithme a donc une complexité en temps réduite par rapport aux autres car pour chaque tâche, un seul processeur est considéré.

Best-Fit L'algorithme *Best-Fit (BF)* se comporte comme FF mais les processeurs sont considérés dans un ordre associé à une métrique. Dans notre cas, la métrique corres-

```
   Entrées : τ                                   /* Un ensemble de tâches */
   Sorties : k                                 /* Un nombre de processeurs */
1  Π ← π_1;
2  k ← 1;
3  pour τ_i ∈ τ faire
4      si τ_i n'est pas ordonnançable sur π_k alors
5          k ← k + 1;
6          Π ← Π ∪ π_k;                          /* Ajout d'un processeur */
7      fin
8      τ(π_k) ← τ(π_k) ∪ τ_i;                                /* Allocation */
9  fin
10 retourner k
```

Algorithme 3.2: NF

```
   Entrées : τ                                   /* Un ensemble de tâches */
   Sorties : k                                 /* Un nombre de processeurs */
1  Π ← π_1;
2  k ← 1;
3  pour τ_i ∈ τ faire
4      {1', ..., k'} ← decrcost({1, ..., k};              /* Coût décroissant */
5      pour j ∈ {1', ..., k'} faire
6          si τ_i est ordonnançable sur π_j alors
7              τ(π_j) ← τ(π_j) ∪ τ_i;                            /* Allocation */
8              sortie de boucle;
9          fin
10     fin
11     si τ(Π) ∩ τ_i = ∅ alors                    /* τ_i n'est pas assignée */
12         k ← k + 1;
13         Π ← Π ∪ π_k;                          /* Ajout d'un processeur */
14         τ(π_k) ← τ(π_k) ∪ τ_i;                            /* Allocation */
15     fin
16 fin
17 retourner k
```

Algorithme 3.3: BF

pond à l'utilisation de l'ensemble des tâches assignées sur le processeur. C'est-à-dire que le premier processeur testé, pour l'affectation d'une tâche, est le processeur le plus chargé. Mais ce n'est qu'un exemple et d'autres fonctions de coût peuvent être utilisées. La fonction `decrcost()` (algorithme 3.3, ligne 4) retourne l'ensemble des indices des processeurs triés par ordre de leur coût décroissant.

Worst-Fit L'algorithme *Worst-Fit (WF)* se comporte comme BF mais les processeurs sont considérés cette fois-ci dans l'ordre de leur utilisation croissante. C'est-à-dire que le premier processeur, sur lequel la tâche à assigner est testée, est le processeur le moins

chargé. Comme dans le cas de *Best-Fit*, une autre fonction de coût peut être utilisée.

```
   Entrées : τ                                   /* Un ensemble de tâches */
   Sorties : k                                 /* Un nombre de processeurs */
1  Π ← π_1;
2  k ← 1;
3  pour τ_i ∈ τ faire
4  |  {1', ..., k'} ← iu({1, ..., k};                  /* Util. croissante */
5  |  pour j ∈ {2', 1', ..., k'} faire
6  |  |  si τ_i est ordonnançable sur π_j alors
7  |  |  |  τ(π_j) ← τ(π_j) ∪ τ_i;                          /* Allocation */
8  |  |  |  sortie de boucle;
9  |  |  fin
10 |  fin
11 |  si τ(Π) ∩ τ_i = ∅ alors                  /* τ_i n'est pas assignée */
12 |  |  k ← k + 1;
13 |  |  Π ← Π ∪ π_k;                          /* Ajout d'un processeur */
14 |  |  τ(π_k) ← τ(π_k) ∪ τ_i;                          /* Allocation */
15 |  fin
16 fin
17 retourner k
```

Algorithme 3.4: AWF

Almost-Worst-Fit L'algorithme *Almost-Worst-Fit (AWF)* se comporte comme WF dans le sens où les processeurs sont considérés dans l'ordre de leur utilisation croissante. La différence avec ce dernier est que le premier processeur testé pour l'affectation d'une tâche est le deuxième processeur le moins chargé.

3.4.2 Algorithmes modifiés

Les algorithmes proposés précédemment permettent de trouver une solution à un problème d'optimisation. En effet, pour un ensemble de tâches donné, ces algorithmes renvoient un nombre de processeurs nécessaires pour pouvoir les ordonnancer. Ce type de problème est adapté dans le cas du dimensionnement d'un système. Par contre, dans le cas d'une analyse d'ordonnançabilité sur une plate-forme dont le nombre de processeurs est fixé, c'est une solution au problème de décision qui doit être trouvée. Il s'agit de décider si l'ensemble de tâches peut être partitionné sur m processeurs. La solution à ce problème repose sur la solution au problème d'optimisation. Il suffit d'appliquer ce dernier et de vérifier que $k \leq m$. Dans notre étude, nous voulons maximiser la marge des tâches. Notre objectif est donc une répartition au mieux des tâches sur les différents processeurs. Malheureusement, les algorithmes présentés précédemment ne sont pas spécialement adaptés pour ce genre de comportement. En effet, il ne peuvent pas répartir les tâches sur un ensemble de m processeurs car ils procèdent pour ajout

itératif de processeur au cours du déroulement de l'algorithme. C'est pourquoi nous proposons des variantes des algorithmes WF et AWF. Ces variantes considèrent en entrée, en plus de l'ensemble des tâches, un ensemble de processeurs.

```
   Entrées : τ                                   /* Un ensemble de tâches */
   Sorties : Π                                 /* Un ensemble de processeurs */
 1 m ← |Π|;
 2 pour τ_i ∈ τ faire
 3 |   {π_1', ..., π_m'} ← iu(Π);                          /* Util. croissante */
 4 |   pour π_j ∈ {π_1', ..., π_m'} faire
 5 |   |   si τ_i peut être assignée sur π_j alors
 6 |   |   |   τ(π_j) ← τ(π_j) ∪ τ_i;
 7 |   |   |   sortie de boucle;
 8 |   |   fin
 9 |   fin
10 |   if τ(Π) ∩ τ_i = ∅ then                    /* τ_i n'est pas assignée */
11 |   |   retourner échec
12 |   end
13 fin
14 retourner succès
```

Algorithme 3.5: F-WF

Fixed-Worst-Fit L'algorithme WF choisit toujours le processeur pour lequel la fonction de coût retourne la plus petite valeur. Dans notre cas, il s'agit de l'utilisation processeur qui doit être la plus petite possible. Mais ce choix est fait parmi un ensemble de processeurs qui grandit au fur et à mesure du déroulement de l'algorithme. Finalement, on obtient une partition pour laquelle les premiers processeurs sont les plus remplis. En effet, un processeur n'est rajouté à l'ensemble que lorsque qu'une tâche ne peut être assignée sur un processeur déjà présent dans l'ensemble. Le comportement de *Fixed-Worst-Fit (F-WF)* décrit dans l'algorithme 3.5 est sensiblement le même que WF. Toutefois, les m processeurs sont considérés comme déjà présents. Cet algorithme produit une partition pour laquelle les tâches sont réparties sur tous les processeurs plutôt que concentrées sur les premiers.

Fixed-Almost-Worst-Fit L'algorithme *Fixed-Almost-Worst-Fit (F-AWF)* présente la même modification que F-WF. Pour chaque tâche, le processeur sur lequel assigner cette dernière est choisi parmi l'ensemble des m processeurs du système.

3.4.3 Ordre sur les tâches

L'ordre dans lequel les tâches sont prises pour être assignées sur les processeurs impacte grandement sur les performances de l'algorithme. Il est d'ailleurs souvent fait

```
   Entrées : τ                                   /* Un ensemble de tâches */
   Sorties : Π                                 /* Un ensemble de processeurs */
 1 m ← |Π|;
 2 pour τ_i ∈ τ faire
 3 |   {π_1', ..., π_m'} ← iu(Π);                          /* Util. croissante */
 4 |   pour π_j ∈ {π_2', π_1', ..., π_m'} faire
 5 |   |   si τ_i peut être assignée sur π_j alors
 6 |   |   |   τ(π_j) ← τ(π_j) ∪ τ_i;
 7 |   |   |   sortie de boucle;
 8 |   |   fin
 9 |   fin
10 |   si τ(Π) ∩ τ_i = ∅ alors                    /* τ_i n'est pas assignée */
11 |   |   retourner échec
12 |   fin
13 fin
14 retourner succès
```

Algorithme 3.6: F-AWF

référence à l'algorithme *First-Fit Decreasing* pour parler de l'algorithme FF où les tâches sont prises dans l'ordre décroissant de leur utilisation. Il existe différentes manières de trier les tâches par ordre de leur :

- utilisation décroissante (*Decreasing Utilization*) (DU) ;
- utilisation croissante (*Increasing Utilization*) (IU) ;
- échéances décroissantes (*Decreasing Deadline*) (DD) ;
- échéances croissantes (*Increasing Deadline*) (ID) ;
- périodes décroissantes ;
- périodes croissantes (*Increasing Period*) (IP) ;
- WCET décroissant (*Decreasing WCET*) (DW) ;
- WCET croissants (*Increasing WCET*) (IW).

Nous considérons l'ordre DU car c'est l'un des plus utilisés dans l'état de l'art.

3.4.4 Travaux existants

First-Fit L'algorithme *Rate-Monotonic-First-Fit* (RMFF) a été proposé par Dhall et Liu [DL78], puis étudié à nouveau par Oh et Son [OS93]. Il s'agit de l'un des premiers algorithmes de partitionnement basé sur un algorithme pour le problème BIN-PACKING. Il a été proposé pour partitionner des tâches périodiques à échéances implicites.

L'algorithme *First-Fit Decreasing-Utilization Factor* (FFDUF) a été proposé par Davari et Dhall [DD86].

L'algorithme *Fisher-Baruah-Backer First-Fit-Decreasing* (FBB-FFD) a été proposé par Fisher, Baruah et Baker [FBB06b]. Il est capable de partitionner un ensemble de tâches sporadiques à échéances arbitraires.

Nom	Algorithme	Ordre	Priorité	Modèle de tâche
RMFF [DL78]	FF	IP	RM	périodique / échéances implicites
RMNF [DL78]	NF	IP	RM	périodique / échéances implicites
FFDUF [DD86]	FF	DU	RM	périodique / échéances implicites
RMBF [OS93]	BF	IP	RM	périodique / échéance implicites
RM-DU-NFS [AJ02]	NF	DU	RM	périodique / échéances implicites
FBB-FFD [FBB06b]	FF	ID	DM	sporadique / échéances arbitraires

TABLE 3.2 – Algorithmes de partitionnement

Best-Fit L'algorithme *Rate-Monotonic-Best-Fit* (RMBF) [OS93] a été proposé par Oh et Son. Il est basé sur l'algorithme BF et permet de partitionner un ensemble de tâches périodiques à échéances implicites.

Next-Fit L'algorithme *Rate-Monotonic-Next-Fit* (RMNF) a été proposé par Dhall et Liu [DL78], puis étudié à nouveau par Oh et Son [OS93]. Il s'agit de l'un des premiers algorithmes de partitionnement basé sur un algorithme pour le problème BIN-PACKING. Il a été proposé pour ordonnancer des tâches périodiques à échéances implicites.

L'algorithme *Rate-Monotonic Decreasing-Utilization Next-Fit-Scheduling* (RM-DU-NFS) a été proposé par Andersson et Jonsson [AJ02]. Il a été conçu pour pallier aux anomalies de partitionnement. On parle d'anomalies lorsqu'un ensemble de tâches peut être partitionné mais qu'un ensemble de tâches avec une plus petite utilisation ne le peut pas.

Tous ces algorithmes référencées dans la table 3.2 ont été conçus dans le but d'offrir une meilleure ordonnançabilité ou d'éviter les anomalies de partitionnement dans le cas de RM-DU-NFS. Mais aucun de ces algorithmes ne prend en considération la tolérance aux dépassements de WCET ou à la réduction de période.

3.5 Partitionnement robuste

Dans cette section, nous proposons un algorithme de partitionnement dont l'objectif est de maximiser la marge des tâches. Cet algorithme est décliné en deux version :

- *Allowance-Fit-WCET (AF^C)* pour la marge sur le WCET ;
- *Allowance-Fit-Frequency (AF^f)* pour la marge sur la période.

3.5.1 Description de l'algorithme

```
   Entrées : τ                                  /* Un ensemble de tâches */
   Sorties : Π                                  /* Un ensemble de processeurs */
1  m ← |Π|;
2  minMarge[m]i;
3  pour τ_i ∈ τ faire
4      pour j ∈ {1, ..., m} faire
5          minMarge[j] ← marge_minimale(τ(π_j) ∪ τ_i);
6          si max_{j=1}^{m} minMarge[j] = −1 alors
7              retourner échec
8          fin
9      fin
10     π_k ← max_{j=1}^{m} minMarge[i];
11     τ(π_k) ← τ(π_k) ∪ τ_i;
12 fin
13 retourner succès
```

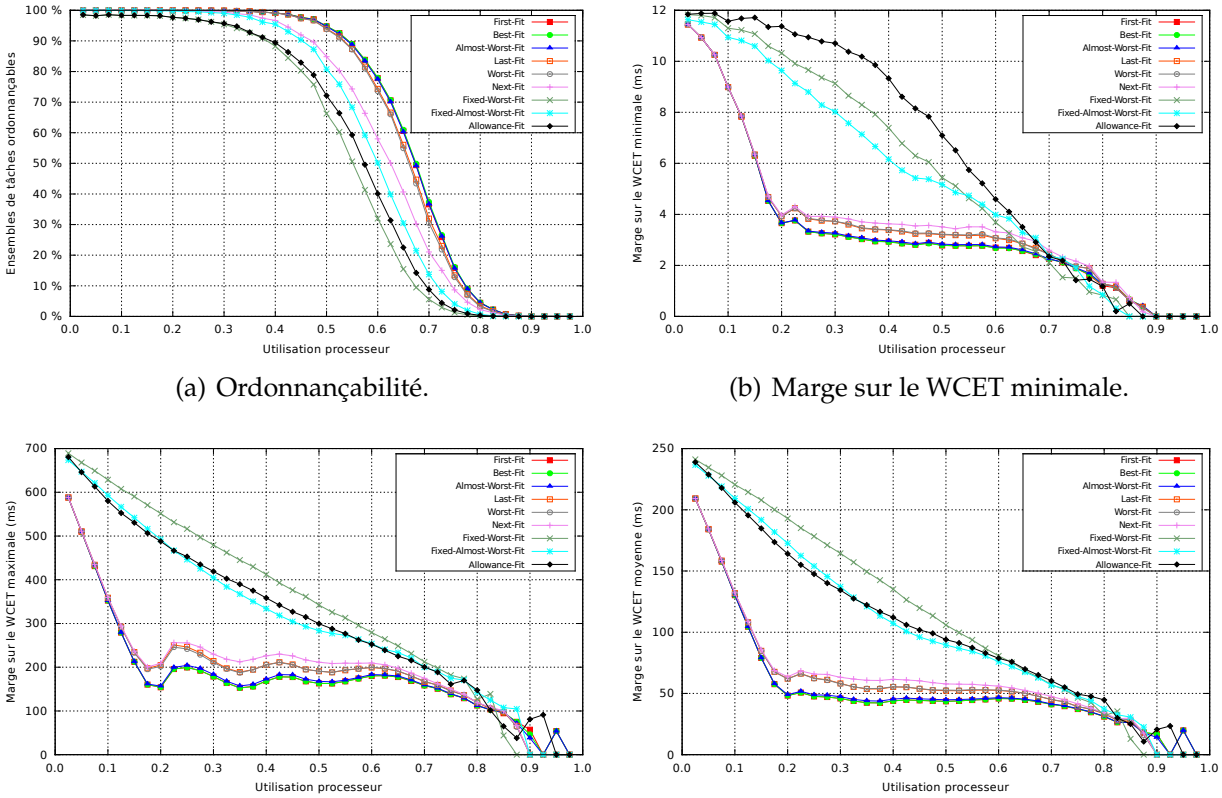
Algorithme 3.7: AF

L'algorithme 3.7 décrit le comportement général d'*Allowance-Fit (AF)*. La distinction entre AF^C et AF^f est faite par la fonction de calcul de marge (marge sur le WCET ou marge sur la période).

Le principe de cet algorithme est d'allouer les tâches sur le processeur où la marge minimale est la plus grande. Pour chaque processeur π_j (ligne 4-9), cette marge minimale est calculée (ligne 5) en considérant que la tâche τ_i est assignée sur π_j. La fonction `marge_minimale()` appelée avec la valeur j renvoit la valeur -1 si au moins une tâche n'est pas ordonnançable sur π_j. Si τ_i ne peut être assignée sur aucun processeur, l'algorithme échoue (ligne 6-8). Dans le cas contraire, le processeur choisit pour l'allocation de τ_i est celui ayant la plus grande valeur de marge minimale (ligne 10-11). Si toutes les tâches ont pu être allouées, l'algorithme termine avec succès (ligne 13).

3.5.2 Protocole de simulation

Les résultats de simulation présentés dans cette section ont été produits par notre simulateur. Chaque simulation se compose de 10000 ensembles de tâches pour chaque valeur d'utilisation parmi $[0.025, 0.05, \ldots, 0.975] \times m$ où m est le nombre de processeurs. Une simulation revient donc à l'analyse de 390000 ensembles de tâches engendrés aléatoirement. Les ensembles de tâches sont constitués de 16 tâches à échéances contraintes et la plate-forme multiprocesseur est composée de 4 processeurs.



(a) Ordonnançabilité.

(b) Marge sur le WCET minimale.

(c) Marge sur le WCET maximale.

(d) Marge sur le WCET moyenne.

FIGURE 3.1 – Algorithmes de partitionnement avec utilisation décroissante, ordonnancement Deadline-Monotonic et analyse de temps de réponse.

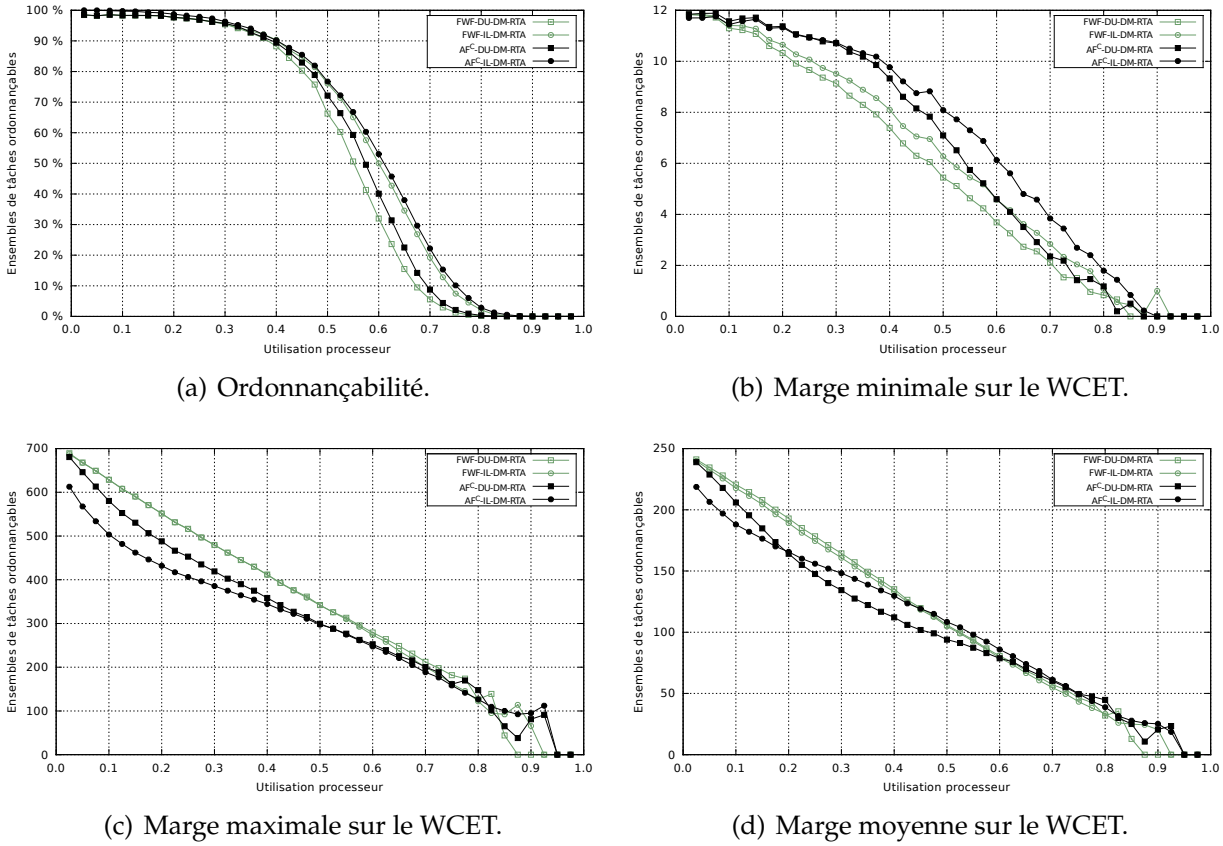
3.5.3 Marge sur le WCET

Dans cette section, nous appelons « marge » la marge sur le WCET. La figure 3.1 représente la comparaison d'un ensemble d'algorithmes de partitionnement. L'ordre sur les tâches, l'algorithme d'attribution des priorités, ainsi que le test d'ordonnançabilité sont les mêmes. Ainsi, pour tous les ensembles de tâches, les tâches sont triées dans l'ordre DU, les priorités sont attribuées suivant l'algorithme DM et l'acceptation d'une tâche sur un processeur relève d'une analyse de temps de réponse. Ainsi, ce sont les performances des algorithmes pour BIN-PACKING qui sont comparées et non pas les performances de conditions suffisantes d'ordonnançabilité. Dans la figure 3.1(a), nous comparons les performances en terme d'ordonnançabilité des algorithmes de partitionnement. Nos résultats sont concordants avec ceux que nous pouvons trouver dans la littérature, à savoir que les algorithmes FF et BF sont les plus performants en terme d'ordonnançabilité. Nous remarquons que l'algorithme AWF, qui, à la différence de FF et BF est très rarement cité dans la littérature, offre des performances en terme d'ordonnançabilité équivalentes. Les algorithmes LF et WF sont les algorithmes qui appliquent le comportement inverse des algorithmes FF et BF. Alors qu'avec FF c'est le premier processeur qui peut accepter une tâche qui est choisi, avec LF, c'est le dernier. Bien que LF et WF peuvent sembler être les algorithmes antagonistes de FF et BF, ils offrent des per-

formances qui sont assez proches. L'algorithme NF est moins performant car il bloque l'accès à un processeur dès lors qu'une tâche ne peut y être admise. Cette propriété réduit l'ordonnançabilité mais présente l'avantage d'empêcher les anomalies de partitionnement. Alors que les choix faits par BF, WF et AWF sont souvent les mêmes, ceux de F-WF et de F-AWF sont assez différents. En effet, BF, WF et AWF commencent le partitionnement en ne considérant qu'un seul processeur pour l'allocation des tâches. Si une allocation échoue, alors un processeur est ajouté tant que le nombre de processeurs ne dépasse pas celui de la plate-forme cible. En revanche, F-WF et F-AWF calculent un partitionnement directement sur un ensemble de processeurs correspondant au nombre de processeurs de la plate-forme considérée. Enfin, nous avons proposé un algorithme de partitionnement, nommé AF^C spécialement conçu pour maximiser la marge minimale des tâches. Bien qu'il soit plus complexe en terme de temps de calcul que F-WF, il offre de meilleures performances en terme d'ordonnançabilité.

Dans la figure 3.1(b), nous représentons la marge minimale (définie dans la section 2.3.1) obtenue à partir de chacun des algorithmes présentés précédemment. La marge minimale est la plus petite valeur de marge qui peut être ajoutée à une tâche. Elle nous donne un indicateur sur la robustesse aux dépassements de WCET du système. Nous montrons dans cette figure que l'algorithme AF^C offre la meilleure alternative pour maximiser la valeur de marge minimale pour des valeurs d'utilisation comprises dans l'intervalle $[0.025, 0.065] \times m$ où m est le nombre de processeurs. Pour des valeurs d'utilisation supérieures, les systèmes sont trop chargés pour pouvoir établir la prédominance d'un algorithme par rapport aux autres. La complexité d'AF^C est polynomiale en le nombre de tâches. Les algorithmes F-WF et F-AWF restent toutefois des alternatives intéressantes car moins complexes à implanter. Leur complexité, indépendamment du test d'ordonnançabilité, est linéaire en le nombre de tâches. Par contre, tous les algorithmes couramment utilisés dans la littératures, qui sont à l'origine des algorithmes fondés sur BIN-PACKING, ont des résultats bien en deçà des 3 algorithmes précédemment cités. Nous représentons dans la figure 3.1(c), et respectivement dans la figure 3.1(b), la marge maximale, et respectivement la marge moyenne, obtenue pour les ensembles de tâches testés. Les courbes représentées nous permettent de confirmer que les 3 meilleurs algorithmes de la figure 3.1(b) surpassent bien les autres en général, et pas seulement dans le cas de la marge minimale. Bien que AF^C soit prépondérant dans le cas de la marge minimale, ce n'est pas le cas pour la marge maximale et moyenne. Ce qui signifie que l'algorithme F-WF reste une bonne alternative pour fournir un partitionnement robuste aux dépassements de WCET.

Dans la figure 3.1, nous avons comparé les algorithmes de placement des tâches sur les processeurs (algorithmes fondés sur BIN-PACKING). Nous nous intéressons maintenant à l'influence de l'ordre sur les tâches. Jusqu'à maintenant, nous considérions que les tâches étaient triées selon l'ordre DU. Cet ordre est parmi l'un des plus couramment utilisé, sûrement par analogie avec la taille des objets dans le problème BIN-PACKING.



(a) Ordonnançabilité.

(b) Marge minimale sur le WCET.

(c) Marge maximale sur le WCET.

(d) Marge moyenne sur le WCET.

FIGURE 3.2 – Algorithmes de partitionnement avec Deadline-Monotonic et analyse de temps de réponse. Comparaison entre utilisation décroissante et laxité croissante.

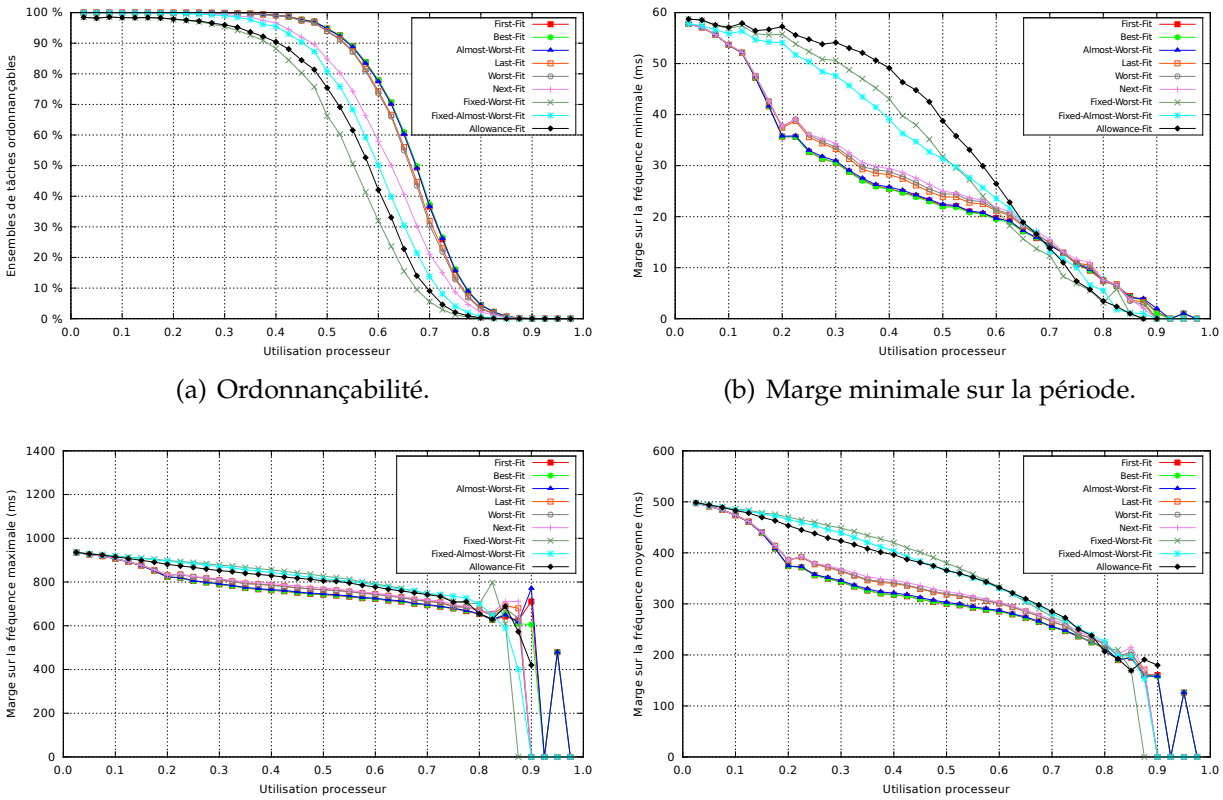
Dans le but de maximiser la marge minimale, nous proposons de trier les tâches par ordre de laxité croissante (*Increasing Laxity*) (IL). Nous définissons la laxité d'une tâche comme la durée entre la terminaison de son exécution et son échéance. Ainsi, une tâche τ_i considérée seule a comme valeur de laxité $L_i = D_i - C_i$. Cet ordre est intuitif car il conduit à placer en premier les tâches qui ont le moins de laxité.

Dans la figure 3.2, nous comparons l'ordre DU avec l'ordre IL pour les algorithmes F-WF et AF^C. Dans la figure 3.2(a), nous représentons l'ordonnançabilité des ensembles de tâches obtenue à partir ces algorithmes. Nous voyons donc que le fait de prendre les tâches dans l'ordre IL améliore légèrement l'ordonnançabilité sans ajouter aucune complexité à l'algorithme de partitionnement.

Dans la figure 3.2(b), nous voyons que l'ordre IL, améliore les performances de l'algorithme AF^C en terme de maximisation de la marge minimale. Cette amélioration est encore plus flagrante pour l'algorithme F-WF. Par contre, dans les figures 3.2(c) et 3.2(d), la marge maximale et moyenne qui peut être ajoutée aux tâches est inférieure avec l'ordre IL. Ce résultat est dû au fait que maximiser la marge minimale tend à réduire la marge maximale et moyenne. Notre objectif étant de garantir la plus grande valeur de marge pour toutes les tâches de l'ensemble de tâches, l'ordre IL apparaît être un choix intéressant. Couplé à l'algorithme AF^C, il permet ainsi d'obtenir une bonne

robustesse aux dépassements de WCET.

3.5.4 Marge sur la période



(a) Ordonnançabilité.

(b) Marge minimale sur la période.

(c) Marge maximale sur la période.

(d) Marge moyenne sur la période.

FIGURE 3.3 – Algorithmes de partitionnement avec utilisation décroissante, ordonnancement Deadline-Monotonic et analyse de temps de réponse.

Dans cette section, nous appelons « marge » la marge sur la période. Nous proposons également l'algorithme AF^f dans le but de maximiser la marge sur la période. La différence entre AF^C et AF^f est l'algorithme de calcul de marge utilisé, à savoir le calcul de marge sur le WCET pour AF^C et le calcul de marge sur la période pour AF^f. Nous l'avons comparé à l'ensemble des algorithmes auxquels nous avions comparé AF^C pour la marge sur le WCET. Dans la figure 3.3(a), nous montrons que AF^f a les mêmes performances en terme d'ordonnançabilité que AF^C. Concernant notre métrique principale, à savoir la marge minimale, nous montrons dans la figure 3.3(b) que AF^f pour les périodes est le meilleur algorithme pour une utilisation allant jusqu'à 65%. Pour une utilisation supérieure, la laxité des tâches n'est plus suffisante pour faire ressortir un algorithme prédominant. Nous remarquons qu'il est possible de faire le parallèle entre la marge minimale sur le WCET et la marge minimale. En effet, les résultats pour ces deux métriques sont sensiblement équivalents. Dans les figures 3.3(c) et 3.3(d), nous montrons les résultats concernant les valeurs maximales et moyennes de marge. Au-

cun des algorithmes ne se démarque vraiment, même si les algorithmes F-WF, F-AWF et AF^f restent les meilleurs algorithmes. Encore une fois, notre métrique la plus significative étant la marge minimale, l'algorithme AF^f reste le choix le plus intéressant pour maximiser la marge minimale.

3.6 Conclusion

Dans ce chapitre, nous nous sommes intéressés à la classe d'ordonnancement (FTP-FTII) dans le cas d'un ordonnancement hors-ligne. Cette classe d'ordonnancement est plus connue dans la littérature sous la dénomination d'ordonnancement par partitionnement. Nous avons considéré un modèle de tâches sporadiques à échéances contraintes ne partageant pas de ressource. Nous avons montré par une étude basée sur la simulation que l'algorithme AF^C-IL-DM est un choix intéressant pour maximiser la marge minimale sur le WCET. Nous avons aussi établi le parallèle avec la marge minimale sur la période et nous avons montré que les algorithmes AF^C et AF^f sont les candidats les plus efficaces pour maximiser la marge minimale. Ce travail a donné lieu aux publications [FMG09, FMG10a].