Université Polytechnique de Bobo Dioulasso
(UPB)

Ecole Supérieure d'Informatique
(ESI)

Mémoire de DEA en Informatique

Thème:

Etude pratique de la cyclicité des ordonnancements temps-réels dans un environnement multi-processeurs

Présenté par :
Malo Sadouanouan

Sous la direction de :
Dominique Geniet, université de Poitiers
Annie Choquet-Geniet, université de Poitiers

Année universitaire 2003-2004

Remerciements

Les travaux présentés dans ce mémoire ont été effectués au Centre De Calcul de Bobo Dioulasso dirigé par le Professeur Chantal Yvette Zoungrana-Kaboré que je remercie pour son accueil.

Je tiens à exprimer mes remerciements à Dominique Geniet et Annie Choquet-Geniet qui ont accepter de diriger mon travail de DEA pour toute leur disponibilités et leurs conseils tout au long de ce travail.

J'exprime mes vifs remerciements au Directeur de l'ESI , au Directeur Adjoint de l'ESI ainsi qu'au Directeur Adjoint du Centre De Calcul pour leur soutien tout au long de ce travail.

J'exprime un remerciement particulier:
- *au Dr Sado Traoré, Directeur Adjoint de l'ESI ,*
- *à Patrick Girard, directeur de la formation « Génie physiologique » à Poitiers,*
- *à Geneviève Jomier, professeur d'informatique à Paris-Dauphine,*
- *à Mme Martinez*

à qui je dois d'avoir eu un ordinateur à ma disposition pendant toute la durée de mon stage de recherche.

Mes remerciements s'étendent à tous mes collègues, mes amis et en particulier Traore Karim , Sanou Loe, Poda Pasteur, Poda Gabin, Compaoré Joëlle Anastasie, Savadogo Saïdou, Sirima Ouamar et toute sa famille, Ouattara Yacouba et Ouédraogo Salifou pour leur amitié.

Toute ma reconnaissance à l'ensemble du personnel du Centre De Calcul de Bobo Dioulasso pour leur soutien dans la réalisation de mon travail.

Finalement je tiens à remercier l'Université Polytechnique de Bobo Dioulasso de m'avoir accueilli.

Liste des abréviations

RM : Rate Monotonic
ID : Inverse Deadline
DM : Deadline Monotonic
ED : Earliest Deadline
LL : Least Laxity
PPCM : Plus Petit Commun Multiple
PGCD : Plus Grand Diviseur Commun

Liste des Tableaux

Liste des figures

Sommaire

I. Introduction

L'évolution technologique (calculateurs, réseaux de télécommunication) de ces dernières années a induit le grand développement d'applications temps réel. Une application temps réel est une application qui met en oeuvre un système informatique (appelé système de contrôle) dont le fonctionnement est assujetti à l'évolution dynamique de l'état de l'environnement (procédé) qui lui est connecté et dont il doit contrôler le comportement.

Le système de contrôle est donc soumis à des contraintes temporelles qui peuvent être de nature différente en fonction de l'application considérée : *dures* (par exemple contrôle de procédé) où la violation d'une contrainte est catastrophique, ou *molles* (par exemple application multimédia).

Il en découle que ces applications doivent non seulement être algorithmiquement correctes, mais en plus doivent être temporellement validées.

Le contexte de recherche dans lequel s'inscrit notre travail est celui de la validation temps réel.

Ce problème est bien connu dans le contexte des systèmes monoprocesseur, pour lesquels il existe un grand nombre de résultats.

Cependant, la taille grandissante des applications rend difficile leur validation sur des systèmes monoprocesseur. Il faut alors avoir recours à la parallélisation. Le concepteur de l'application doit se tourner vers des architectures multiprocesseurs. Dans ce contexte, il existe moins de résultats.

Notre travail s'inscrit dans le cadre de l'étude des problèmes d'ordonnancement temps réel des applications multiprocesseurs.

Il existe deux façons de concevoir un ordonnancement qui sont :

- déterminer un algorithme d'ordonnancement qui sera exécuté au cours de la vie de l'application, on parle d'approche en ligne. Cette approche ne peut aboutir car il ne peut exister d'algorithme capable d'ordonnancer correctement toute application pour laquelle il existe au moins un ordonnancement qui respecte toutes les contraintes temporelles [DER, MOK 1989];
- ou bien on pré calcule une séquence valide que l'on implante au niveau du distributeur, on parle alors d'approche hors ligne. Ce problème a trouvé une solution dans le contexte monoprocesseur, mais il reste ouvert dans le contexte multiprocesseurs. C'est cette approche que nous envisageons.

Dans une première partie nous vous présenterons les outils que nous avons développés pour nous permettre l'observation du phénomène et dans une seconde partie, nous présenterons les résultats obtenus ainsi que des suggestions.

II. Etat de l'art

Dans cette partie, nous nous sommes appuyé sur le livre « Ordonnancement temps réel, cours et exercices » de Cottet et al, 2000.

II.1 Systèmes temps réel et ordonnancement

II.1.1 Systèmes temps réel

Les systèmes temps réel sont des systèmes informatiques soumis à des contraintes temporelles en plus des contraintes de correction fonctionnelles usuelles.

On distingue trois grands types de systèmes temps réel, compte tenu de l'aspect « respect des contraintes temporelles » :

. Les systèmes Temps-Réel à contraintes strictes;

on parle de contraintes strictes quand on a des contraintes dont le non respect peut avoir des conséquences graves (catastrophes humaines, économiques ou écologiques).

. Les systèmes Temps-Réel à contraintes relatives;

on parle de contraintes relatives quand leur non respect est tolérable dans une certaine mesure.

. Les systèmes Temps-Réel à contraintes mixtes qui contiennent des contraintes temporelles strictes et relatives.

Un système temps réel doit donc permettre la réalisation des applications temps réel qui possèdent des contraintes temporelles strictes. Les applications temps réel sont structurées en tâches c'est à dire en un flot de contrôle.

II.1.1.1 Caractéristiques des tâches

Une application temps réel peut donc être représentée par un ensemble $t = \{t_1, ..., t_i, ..., t_n\}$ de n tâches. Une tâche $t_{i\ 1\leq i\leq n}$ est caractérisée par les quatre paramètres temporels :

. r_i, la date de la première activation de la tâche,

. C_i, la durée nécessaire à la tâche pour s'exécuter, lorsqu'elle dispose du processeur sans partage,

. D_i, le délai maximum pour son exécution,

. T_i, la période des activations de la tâche si c'est une tâche périodique.

Le modèle de tâches indique également d'autres contraintes liées aux instances d'une tâche. Les contraintes les plus courantes sont :

Tâches préemptibles ou non préemptibles : *Certaines tâches, une fois élues, ne doivent plus être arrêtées avant la fin de la requête. Elles sont non préemptibles. Au contraire, quand la tâche élue*

peut être arrêtée et remise à l'état prêt pour réquisitionner le processeur au profit d'une autre tâche, on dit que la tâche est préemptible.

Dépendance et indépendance des tâches : *Les tâches sont dites dépendantes lorsqu'il existe entre elles une relation de précédence ou qu'elles partagent d'autres ressources que le processeur. Elles sont dites indépendantes si elles n'ont entre elles ni une relation de précédence, ni de partage de ressource critique.*

Il existe également des mesures pour caractériser le comportement temporel. Ces mesures sont dérivées des contraintes et des caractéristiques temporelles. La mesure la plus utilisée est le *taux de charge* qui se définit comme suit :

Si la tâche t_i est périodique de période T_i, si chaque instance de la tâche nécessite un temps d'exécution C_i sur le processeur, son taux d'utilisation est $u_i=C_i/T_i$. On définit également le *taux d'utilisation processeur* pour un ensemble des tâches périodiques par $U=\sum_i u_i$.

II.1.1.2 Architecture physique et exécutif

II.1.1.2.1 Architecture physique

Un système temps réel repose sur un ou plusieurs processeurs pour effectuer les traitements (Figure 1). Il est décomposé en noeuds correspondant à un processeur et aux traitements sur ce processeur au cours de la vie du système. Les architectures matérielles pour le temps réel sont caractérisées par l'importance des entrées-sorties.

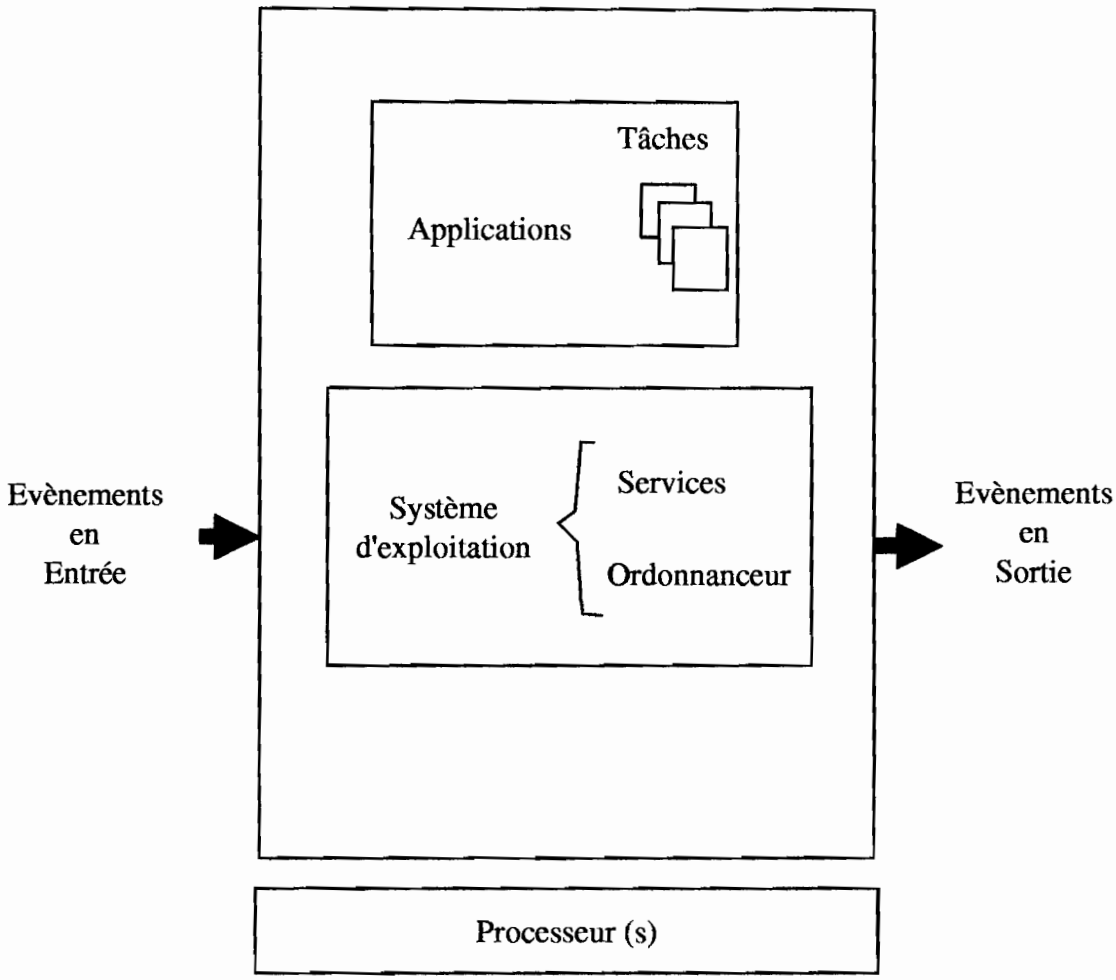

Figure 1 Schéma simplifié d'un système temps réel

II.1.1.2.2 Exécutif temps réel

Un *exécutif temps réel* est un système d'exploitation multitâches dans lequel la durée d'exécution de toute fonction système est documentée (ou bien au moins prédictible et connue). L'exécutif est chargé de faire le lien entre les applications, qui devraient être indépendantes de la plate-forme support, et l'architecture physique de la plate-forme support.

II.1.2 Ordonnancement temps réel

Le problème de l'ordonnancement temps réel est la définition d'une politique d'attribution du

processeur qui respecte les contraintes temporelles de l'application. Il faut donc définir une stratégie et ensuite prouver que les contraintes temporelles sont bien respectées. Ceci peut passer par la construction d'une séquence temporelle de l'ensemble des tâches à exécuter. Cette construction nécessite la définition d'un algorithme d'ordonnancement. De nombreuses études ont été faites sur ce sujet [Cottet et al, 2000], [Dertouzos et Mok, 89], [Yingfeng et Sang, 95], [Goossens et Devillers,1997], [Choquet-Geniet et al, 2000]. Mais avant la présentation de ces travaux nous allons donner quelques définitions:

Définition : La séquence d'ordonnancement délivrée est *valide* si toutes les tâches de la configuration respectent leurs contraintes.

Définition : Une configuration de tâches est *ordonnançable* dès qu'il existe au moins un algorithme capable de fournir une séquence valide pour cette configuration. Le *test d'ordonnançabilité* sert à vérifier qu'une configuration de tâches périodiques donnée soumise à un algorithme d'ordonnancement peut être ordonnancée selon une séquence valide.

Définition [Choquet-Geniet, 2003]: *Algorithme d'ordonnancement optimal*

Soit T une classe d'applications. Un algorithme d'ordonnancement est dit *optimal* (éventuellement parmi une sous-famille d'ordonnancements) pour les applications de la classe T si et seulement si quelle que soit l'application de la classe, soit l'algorithme l'ordonnance de manière correcte, soit aucun autre algorithme (de la sous famille) ne le pourra.

Selon le degré de connaissance que le système peut avoir des caractéristiques des tâches avant son démarrage et selon l'architecture physique du système plusieurs types d'algorithmes d'ordonnancement peuvent être employés : Les algorithmes peuvent être *en ligne* ou *hors ligne, préemptifs* ou *non préemptifs*, *centralisés* ou *répartis*.

Pour les algorithmes hors ligne, la construction de la séquence d'exécution est réalisée avant la mise en exploitation du système tandis que pour les algorithmes en ligne, la séquence peut être modifiée si une nouvelle tâche devient prête.

Un certains nombre de travaux ont permis d'obtenir des résultats pour l'ordonnancement temps réel de tâches périodiques ou apériodiques que ce soit avec préemption ou non. Parmi ces résultats on a le théorème suivant démontré par Labetoulle:

Théorème: *Pour qu'une configuration de tâches périodiques soit ordonnançable sur un processeur, il est nécessaire que le taux d'utilisation U du processeur soit inférieur ou égal à 1:* $U=\sum_i C_i/T_i \leq 1$

II.1.2.1 Ordonnancement des tâches indépendantes

Un grand nombre d'algorithmes d'ordonnancement temps réel est conduit par la notion de priorité qui les amène à ordonnancer dans la liste des tâches prêtes, la tâche qui possède la plus grande priorité. La priorité peut être fixe ou statique si elle est fixée à l'initialisation et si elle reste constante pendant toute la durée de vie de la tâche. Si la priorité évolue dans le temps, elle est dite dynamique. La notion d'optimalité est en générale réduite à la classe à laquelle appartient l'algorithme. Aussi on peut parler d'algorithme optimal dans la classe des algorithmes statiques ou dynamiques.

Les principaux algorithmes d'ordonnancement sont les suivants:

II.1.2.1.1 Algorithmes en ligne à priorités constantes

II.1.2.1.1.1 Rate Monotonic (RM)

C'est un algorithme qui est basé sur la priorité statique et qui est optimal parmi les algorithmes à priorités fixes pour les applications constituées de tâches indépendantes à échéances sur requête et à départs simultanés. Un tâche a une priorité fixe qui est inversement proportionnelle à sa période. L'algorithme a été introduit par [Liu et Layland, 1973] .

L'ordonnancement RM d'un ensemble de n tâches périodiques $t = \{t_1, ..., t_i, ..., t_n\}$ est faisable si le taux d'utilisation du processeur U vérifie la relation $U \leq n(2^{1/n}-1)$. Cette borne peut être dépassée et une étude montre qu'en moyenne la borne supérieure avoisine 88%

[Lehoczky et Ramos-Thuel, 1889].

II.1.2.1.1.2 Inverse Deadline (ID) ou Deadline Monotonic (DM)

La politique de cet algorithme attribue la priorité la plus forte à la tâche de plus petit délai critique c'est à dire le temps de réponse maximal toléré. Cet algorithme est optimal parmi les algorithmes à priorités fixes pour les applications constituées de tâches indépendantes à départs simultanés [Choquet-Geniet, 2003].

Les algorithmes RM et ID coïncident si les tâches sont à échéance sur requête d'après

[Choquet-Geniet, 2003] et une condition suffisante d'acceptabilité des tâches est $U \leq n(2^{1/n}-1)$.

Les tâches à échéance sur requête sont les tâches pour lesquelles D=T.

II.1.2.1.2 Algorithmes en ligne à priorités variables

II.1.2.1.2.1 Earlieste Deadline (ED)

C'est un algorithme basé sur les priorités dynamiques et qui est optimal pour les systèmes de tâches indépendantes. A tout instant, la priorité est inversement proportionnelle à la date de l'échéance. L'algorithme ED permet d'accepter un plus grand nombre de configurations de tâches que l'algorithme RM.

Test d'ordonnançabilité [Cottet et al, 2000] : pour des tâches à échéances sur requêtes, une condition nécessaire et suffisante d'ordonnancement est $\sum_i C_i/T_i \leq 1$. Pour des tâches quelconques, une condition suffisante d'ordonnancement est: $\sum_i C_i/D_i \leq 1$ et une condition nécessaire est la précédente. Ces relations sont données pour un ensemble $t = \{t_1, ..., t_i, ..., t_n\}$ de n tâches.

Mais là encore, quand les délais critiques sont inférieurs aux périodes, le problème de l'ordonnançabilité par ED est NP-difficile.

II.1.2.1.2.2 Least Laxity (LL)

La plus forte priorité est donnée à la tâche de plus faible laxité. La laxité d'une tâche à un instant t est égale à la durée restant jusqu'à l'échéance de la tâche moins la quantité de temps CPU qu'il reste à accomplir à la tâche.

D'après [Choquet-Geniet, 2003], LL présente les mêmes propriétés d'optimalité que ED, et le critère d'ordonnançabilité établi pour les systèmes de tâches à échéances sur requêtes fonctionne. Mais avec LL, on a plus de changement de contexte.

Il existe des algorithmes pour la prise en compte des tâches apériodiques, mais ceux-ci ne cadrent pas avec notre contexte de travail.

II.1.2.2 Ordonnancement des tâches dépendantes

Ce paragraphe ne cadre pas avec notre contexte, mais nous avons pensé qu'il est nécessaire de donner quelques résultats de travaux qui nous paraissent importants.

Il existe deux types de contraintes entre les tâches d'une application temps réel:

Le premier type est exprimé par une relation de précédence et le second par le partage de ressources.

Pour les tâches liées par une relation de précédence, deux algorithmes sont principalement utilisés notamment RM et ED. Dans le cadre de l'ordonnancement selon RM, on transforme hors ligne

l'ensemble des tâches liées par des contraintes de précédence en un ensemble de tâches indépendantes pour lequel le test d'acceptabilité de RM est valide. Dans le cas de ED, la transformation s'appuie sur le fait que l'algorithme d'ordonnancement est basé sur l'échéance. Ainsi, on doit modifier les échéances de telle façon qu'une tâche ait toujours une échéance inférieure à celle de ses successeurs.

Pour les tâches partageant des ressources critiques, des protocoles spécifiques ont été développés pour permettre d'effectuer une allocation dynamique des ressources. Ce sont :

le protocole de l'héritage de priorité, le *protocole de la priorité plafonnée* et le *protocole d'allocation de ressources basé sur une pile .*

On peut toujours se référer au livre « **Ordonnancement temps réel** » de Cottet et al, 2000 pour plus de détails.

II.1.2.3 Complexité de l'ordonnancement multiprocesseurs

Nous nous plaçons dans le contexte des systèmes multiprocesseurs à contrôle centralisé avec des processeurs identiques (même vitesse de traitement considérée unitaire).

Dans ce contexte le problème d'ordonnancement est bien plus complexe et il a été montré qu'aucun algorithme d'ordonnancement ne pouvait être optimal sans la connaissance préalable à la fois, des dates de réveil, des échéances et des durées d'exécution des tâches de la configuration fonctionnelle.

Le premier résultat important est un théorème décrivant l'absence d'optimalité des algorithmes en ligne :

Théorème[Sahni, 1979]: *il ne peut exister d'algorithme en ligne qui construise une séquence valide sur une configuration matérielle comportant m ($m \geq 2$) processeurs pour une configuration de tâches temps réel à date critique.*

Un second résultat donne une condition nécessaire d'ordonnançabilité:

Condition nécessaire: *La condition nécessaire d'ordonnançabilité faisant référence à la charge maximale U_j de chacun des processeurs ($U_j \leq 1$ avec j élément de $[1,m]$) est que:*

$U=\sum_i U_j=\sum_i u_i = \sum_i C_i/P_i \leq m$

Les applications, exécutées dans un environnement multiprocesseurs, peuvent conduire à des anomalies d'exécution lors de changements apparemment positifs de paramètres. Comme l'ont démontré [Cottet et al, 2000]:

Théorème[Graham, 1976]: *pour un problème donné, pour lequel il existe une solution valide d'ordonnancement, le changement de priorités des tâches, l'accroissement du nombre de processeurs, la réduction de la durée d'exécution d'une tâche ou de la diminution des contraintes de précédence peuvent conduire à une augmentation de la longueur de l'ordonnancement et par conséquence à des dépassements des échéances de certaines tâches.*

Un autre résultat important est la condition suffisante d'ordonnançabilité sachant que la condition nécessaire est bien satisfaite [Dertouzos et Mok, 1989]:

Condition suffisante: *Soit P' le PGCD des périodes P_i de la configuration, u_i le facteur d'utilisation du processeur par la tâche i et P'' le PGCD de P' et des produits $P'u_i$, une condition suffisante d'ordonnancement est que P'' soit un entier.*

La configuration a été restreinte à des tâches périodiques, indépendantes à échéance sur requête.

Dans les mêmes configurations, une condition nécessaire et suffisante d'ordonnançabilité est donnée :

Condition nécessaire et suffisante: *soit une configuration de tâches périodiques indépendantes à échéance sur requête, telle que $u_i \geq u_{i+1}$ pour i appartenant à [1,n-1], celle ci est ordonnançable sur m processeurs si et seulement si:*

$\text{Max}\{\max_{j \in [1,m]}\{(1/j)\sum_i u_i\},(1/m)\sum_i u_i\} \leq 1$.

Enfin la propriété suivante concernant les algorithmes « Earliest Deadline »et « Least Laxity » a été démontrée [Dertouzos et Nissanke, 1997]:

Propriété: *Toute configuration de tâches périodiques acceptée par l'algorithme ED sur une architecture multiprocesseurs est acceptée par l'algorithme LL.*

Ce qui montre réellement la complexité de l'ordonnancement dans un contexte multiprocesseurs est ce théorème qui dit:

Théorème: *Ordonnancer sans préemption de façon centralisée sur un système multiprocesseurs des tâches munies d'échéance et de date de réveil, qu'elles soient périodiques ou non, relèvent des problèmes NP-Complet.*

Les approches hors ligne

L'absence d'algorithmes polynomiaux optimaux dans le contexte multiprocesseurs même pour les tâches indépendantes a motivé les chercheurs à se pencher plus sur les approches hors ligne.

Celles ci sont de complexité au moins exponentielle lorsqu'elles sont exactes, car il s'agit de produire une séquence d'ordonnancement de longueur exponentielle lorsque les tâches considérées

sont périodiques.

Les approches hors ligne peuvent être classées en deux catégories :

- Les techniques exactes mises en oeuvre grâce aux techniques d'énumération ou de programmation linéaire en nombres entiers,
- Les techniques approchées mises en oeuvre grâce aux algorithmes sous optimaux ou aux algorithmes stochastiques.

Bien qu'elles paraissent moins flexibles, ces approches permettent d'avoir une séquence valide lorsque celle-ci existe. En effet, ce type d'ordonnancement se prête bien aux cas de systèmes complexes tels que les systèmes temps réel distribués et en plus, la garantie que le système respectera toutes les contraintes temporelles est une propriété facile à appréhender.

Ceci fait de cette solution celle qui paraît la plus rigoureuse et la plus stricte.

II.2 Etude de la cyclicité

II.2.1 Système de tâches indépendantes

Un premier résultat pour cette configuration de tâches est [Choquet-Geniet et Grolleau 2003]:

Définition de temps creux : Les temps creux acycliques sont les temps creux qui ne se reproduisent pas de manière périodique. Ils sont dus à la montée en charge du système.

On note t_c la date d'apparition du dernier temps creux acyclique.

Proposition : *Si le système de tâches est à charge maximale (U=1), le dernier temps creux acyclique apparaît au plus tard à la date r+P-1 où* $r=max\{r_i\}$ *et* $P=PPCM_{i=1..n}(P_i)$.

Un autre résultat est le suivant [Choquet-Geniet et Grolleau, 2003] :

Proposition : *Si le système de tâches est charge maximale, la séquence produite à l'instant* t_c+1 *est la même que celle produite à l'instant* t_c+P+1.

De ces deux propositions il en a découlé le corollaire suivant dans [Choquet-Geniet et Grolleau, 2003] :

Corollaire:

(1)*L'intervalle minimal d'ordonnançabilité est l'intervalle* $[0, t_c+P+1]$.

(2)*La date au plus tôt d'entrée dans le cycle est la date* t_c, *et la séquence produite dans l'intervalle* $[t_c+1,t_c+P+1)$ *est l'état cyclique du système.*

(3) $t_c<r+P$.

(4) *Si aucun temps creux acyclique n'apparaît dans l'intervalle* $[0,P)$ *alors* $t_c=-1$.

Il a été également prouvé que ces résultats peuvent s'étendre aux systèmes de tâches pour

lesquels U<1 [Choquet-Geniet et Grolleau, 2003].

Notre travail de DEA se positionne dans l'extension de ce travail au cas multiprocesseurs.

II.2.2 Système de tâches quelconques

Les résultats pour un système de tâches indépendantes ont été étendus à un systèmes de tâches quelconques. La difficulté se posait au niveau des tâches dépendantes liées par un contrainte de précédence. Mais cette difficulté a été surmontée en introduisant le notion de chaîne de blocage et en prouvant leur périodicité. [Choquet-Geniet et Grolleau, 2003].

II.2.3 Cyclicité en environnement multiprocesseurs

Comme nous l'avons dit précédemment le problème de cyclicité dans le contexte multiprocesseurs reste ouvert. Cependant quelques résultats existent dont le théorème suivant qui nous paraît intéressant:

Théorème: *Soit S un système de tâches simultanées du contexte* $|m|\ r_i=0, C_i, D_i, P_i$ *et soit* U_s *sa charge processeur. Toute séquence d'ordonnancement valide a exactement* $P(m-U_s)$ *temps creux dans l'intervalle* $[0..P[$ *où* $P=PPCM_{i=1..n}(P_i)$ *est la métapériode du système et m le nombre de processeurs.*

C'est le problème de cyclicité dans le contexte multiprocesseur que nous envisageons dans notre travail.

III. Description du travail effectué

III.1 Modélisation des tâches

III.1.1 Modèle temporel des tâches

Un système temps réel est composé d'un ensemble de tâches cycliques $t_{i=1..n}$ caractérisées par quatre paramètres temporels:

. r_i la date de la première activation,

. C_i la charge, durée de traitement nécessaire à chaque exécution,

. D_i le délai critique, temps donné à la tâche pour se terminer après chacune des activations,

. T_i période d'activation.

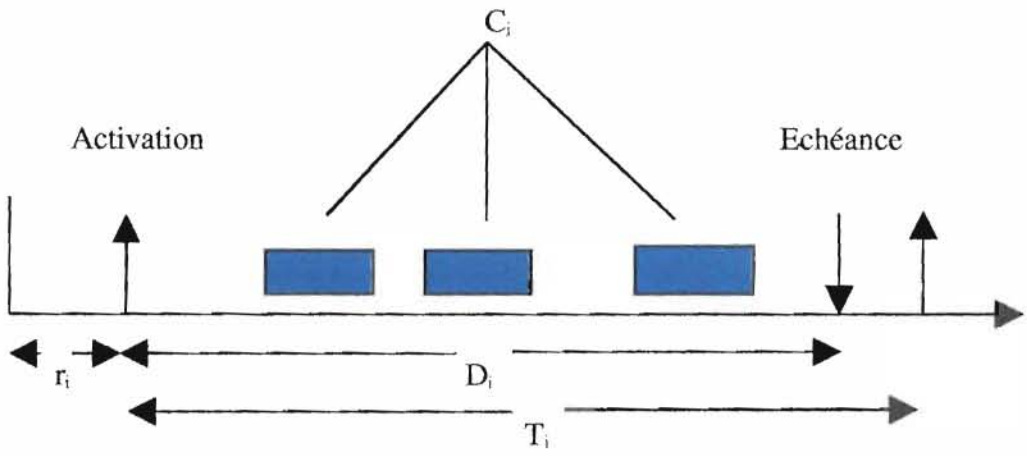

Figure 2: Paramètres temporels des tâches

Une tâche est donc caractérisée temporellement par les paramètres $< r_i, C_i, D_i, T_i >$.

Nous appelons $P=PPCM_{i=1..n}(T_i)$ la métapériode du système, et $r=max_{i=1..n}(r_i)$.

III.1.2 Hypothèses sur la configuration de tâches

Dans le cadre de notre travail nous nous restreindrons aux systèmes de tâches périodiques, indépendantes et à charge maximale c'est à dire $\sum_i C_i/T_i=2$ car dans le cadre de ce DEA nous sommes dans un contexte biprocesseurs et ceux-ci sont identiques. L'ordonnancement est global, les tâches préemptibles et leur migration possible en fin d'exécution ou en cours.

En plus de cela, nous ajoutons des contraintes liées aux processeurs notamment:

-un processeur ne peut traiter qu'une seule tâche à la fois,

-une tâche n'est traitée que par un seul processeur à tout instant: c'est l'hypothèse de non parallélisation.

III.2 Outils

Nous avons construit un simulateur qui pourra nous permettre d'observer le phénomène à étudier. Ce simulateur se compose : - d'un générateur aléatoire d'applications, constituées uniquement de tâches périodiques indépendantes et de charge maximale ; -d'un ordonnanceur d'applications, pour deux processeurs, mettant en oeuvre les principales stratégies connues : ED, LL, RM.

III.2.1 Le générateur aléatoire d'applications

Comme dit précédemment, une application temps réel est composée d'un ensemble $\{t_1,\dots, t_i, \dots, t_n\}$ de n tâches et une tâche t_i $1 \leq i \leq n$ est la donnée des paramètres temporels r_i, C_i, D_i, T_i. Engendrer une application revient donc à engendrer les paramètres temporels.

Pour disposer de données aléatoires dont on connaît précisément les lois, et pour lesquelles on peut piloter assez finement les paramètres, nous avons décidé que la date de la première activation r_i suive une *loi exponentielle* et les paramètres C_i, D_i, T_i suivent une *loi normale*. Nous avons également engendré les paramètres de ces lois pour ne pas avoir à faire des choix arbitraires.

L'engendrement doit être tel qu'on ait des tâches bien formées. Un tâche est *bien formée* si on a: $0 \leq C \leq D \leq T$ pour cette tâche.

Voici comment on a procédé :

T_i suit une loi normale de paramètres m, σ^2.

Pour tirer D_i on procède de la façon suivante: on tire une valeur aléatoire x suivant une loi normale de paramètres 0 et 1. On ne considère cette valeur x que si elle est comprise entre 0 et 1. la valeur de D_i est $x*T_i$. De cette façon on garantit le prédicat $0 \leq D_i \leq T_i$.

Pour tirer C_i, on procède de la même manière, mais on remplace D_i par C_i et T_i par D_i. On garantit donc $0 \leq C_i \leq D_i \leq T_i$.

Pour engendrer des tirages suivant une loi normale de paramètres 0 et 1, on peut considérer soit:

$X=\cos(2*\Pi*U_1)\sqrt{-2*\ln(U_2)}$ soit $Y=\sin(2*\Pi*U_1)\sqrt{-2*\ln(U_2)}$.

U_1 et U_2 représentent des variables aléatoires indépendantes de loi uniforme sur [0,1].

Dans le cadre de notre travail nous avons choisi $X=\cos(2*\Pi*U_1)\sqrt{-2*\ln(U_2)}$

La variable aléatoire $m+\sigma X$ suit un loi normale de paramètres m et ^{2}.

On engendre un tirage suivant une loi exponentielle de paramètre λ de la manière suivante:

$X=(-1/\lambda)\ln(U_1)$ où U_1 est une variable aléatoire de loi uniforme sur [0,1].

Pour engendrer une variable aléatoire de loi uniforme sur [0,1], nous avons utilisé la fonction *rand()* de la bibliothèque mathématique. Nous avons procédé de la manière suivante:

U=(float) rand()/(float)(RAND_MAX +1).

Pour engendrer les paramètres de la loi normale nous avons utilisé la fonction *rand()* tout en fixant une limite supérieure et une limite inférieure.

Il ne reste plus alors qu'à engendrer les tâches. Pour cela, on génère le nombre de tâches de manière aléatoire et on génère ensuite les paramètres des tâches. Les paramètres des tâches doivent être telles que la charge totale soit égale à deux.

Générer_Nombre_De_Taches;

while (Charge_Totale_Processeur!=2)

{

Générer_Paramètres_Taches ;

}

III.2.2 L'ordonnanceur d'application

L'objectif de l'ordonnanceur est de nous fournir une séquence d'ordonnancement sur laquelle portera notre étude. Il s'agit d'un ordonnanceur pour deux processeurs mettant en oeuvre les principales stratégies connues: ED, LL, RM.

Notre choix s'est porté sur un algorithme de liste. L'idée générale de cet algorithme est la mise à jour de deux listes, une liste de tâches et une liste des processeurs.

Dans notre contexte, le nombre de processeurs est fixé à deux et ils sont identiques; nous n'avons donc pas un problème de classement. Cependant pour les tâches, nous utilisons comme critère de classement leur priorité.

Une tâche est une structure de données contenant les éléments suivants:

-*Réveil* qui est la date de réveil de la tâche (r),

-*Périodes* qui est la période de la tâche (T),

-*Délai* qui est le délai de la tâche (D),

-*Durée* qui est la durée d'exécution de la tâche (C),

-*Etat* qui indique l'état de la tâches et

-*Charge* qui est le facteur d'utilisation du processeur par la tâche.

Un processeur est une structure de donnée contenant les éléments suivant:

-*Numéro* indiquant le numéro du processeur,

-*Etat* qui indique l'état du processeur,

-*Charge* qui indique la charge du processeur.

Pour localiser la date d'entrée dans le cycle nous avons défini la structure Tache_Enrichie qui est en fait la structure tâche à laquelle on a ajouté des éléments pour avoir les paramètres dynamiques de tâches. Ce sont:

-*Delai_Dyn*, qui indique le délai dynamique de la tâche,

-*Duree_Dyn*, qui est la durée dynamique de la tâche,

-*Num_Tache* qui est l'indice de la tache en question,

-*Num_Proc* qui est le numéro du processeur dans lequel la tâche a été placée,

-*Date_Placement* qui est la date de placement de la tâche dans le processeur concerné.

La liste de tâches est donc une liste de structures Tache. Cette liste n'est pas maintenue triée car il y a beaucoup de champs à trier lors de l'exécution du programme. Dans un soucis de rendre moins lourd l'exécution du programme, les champs qui ont besoin d'être manipulés (triés par exemple) sont extraits dans des tableaux et ensuite manipulés. Ceux qui ont besoin d'être maintenus triées le sont, notamment les tables de priorités. Cependant, les dates de réveils et les échéances sont régulièrement mises à jour directement sur la liste des tâches afin de simplifier certains calculs.

Nous avons ensuite la liste des tâches enrichies qui contient les paramètres de chaque tâche à chaque unité de temps lors de l'exécution du programme. C'est cette liste qui va ensuite nous permettre de localiser la date d'entrée dans le cycle.

Notre algorithme est le suivant:

Toutes les unités de temps durant le temps de simulation

{

On calcule l'ensemble Ready des tâches prêtes;

On met à jour la table de priorité;

Si Ready≠ ∅

On choisi les deux taches prêtes les plus prioritaires;

On les fait remplacer les tâches remplaçables;

}

III.2.2.1 Calcul de l'ensemble « Ready » des tâches prêtes

Une tâche t_i est prête à un instant t donné si $t \geq r_{ik}$ où r_{ik} est la k_{ieme} date de réveil de la tâche d'indice i et sa durée d'exécution résiduelle est strictement positive (la tâche n'est pas finie) et en plus la tâche ne possède pas le processeur. Pour calculer l'ensemble Ready il suffit donc de parcourir la liste des tâches et de sélectionner celles qui respectent la condition.

III.2.2.2 Calcul des priorités des tâches

On calcule les priorités selon les algorithmes RM, ED, LL.

III.2.2.2.1 Calcul des priorités selon RM

Selon RM la tâche la plus prioritaire est celle ayant la plus petite période.

Il suffit alors d'ordonner les tâches selon l'ordre croissant de leur période. Ainsi donc la tâche la plus prioritaire la première et la moins prioritaire est la dernière. On fait ce calcul une seule fois, raison pour laquelle on parle de priorités fixes.

III.2.2.2.2 Calcul des priorités selon ED

Selon ED, la tâche la plus prioritaire est celle ayant le délai critique le plus petit.

Pour avoir les tâches les plus prioritaires, nous ordonnons les tâches dans l'ordre croissant des délais critiques résiduels. Le calcul des délais résiduels se faisant à tout instant, le calcul des priorités doit aussi se faire à chaque unité de temps.

Si D(t) est le délai critique résiduel à la date t, nous avons:

D(t)=d-t où d est l'échéance de la tâche concernée.

III.2.2.2.3 Calcul des priorités selon LL

Selon LL, la tâche la plus prioritaire est celle ayant la plus petite laxité dynamique.

Pour avoir donc les tâches les plus prioritaires, nous calculons les laxités nominales résiduelles des tâches à chaque instant et ordonnons les tâches selon l'ordre croissant des laxités nominales résiduelles.

Si L(t) est la laxité est la laxité nominale résiduelle à la date t, nous avons:

L(t)=d-t-C(t) où d est l'échéance de la tâche en question et C(t) est sa durée d'exécution résiduelle.

III.2.2.3 Placement des tâches

Dans l'ensemble Ready on choisit les deux tâches les plus prioritaires.

On tente de placer la tâche la plus prioritaire d'abord. Si celle ci est placée on tente ensuite de placer la seconde.

Si on n'a qu'une seule tâche dans cet ensemble, c'est elle seulement qu'on tentera de placer.

On place une tâche si les conditions sont réunies pour qu'une autre libère le processeur. Pour qu'une tâche libère le processeur il faut que l'une au moins de ces conditions soit satisfaite :

. Sa priorité est inférieure à celle d'une des tâches élues,

. Le temps qui a été donné pour son exécution est terminé.

Si la priorité d'une tâche possédant le processeur est égale à la priorité d'une des tâches élue et sa durée d'exécution résiduelle non nulle, elle garde le processeur.

Si les conditions sont réunies pour qu'une tâche libère le processeur et l'ensemble Ready est vide, on parle alors de temps creux du processeur. Nous plaçons la tâche -1 à ce moment-là pour signifier qu'il s'agit d'un temps creux processeur.

III.3 Calcul de la date d'entrée en phase cyclique

Pour repérer la date d'entrée en phase cyclique, nous utilisons la liste des tâches enrichies.

Nous utilisons le principe suivant:

Il existe une date T0 à partir de laquelle, t T0 la condition (C) suivante est vérifiée:

(C): soit une configuration de n tâches. Pour toute tâche d'indice i, i étant quelconque,

{Duree_Dyn_i(t)=Duree_Dyn_i(t+P)}et{Delai_Dyn_i(t)=Delai_Dyn_i(t+P)}.

Duree_Dyn_i et la durée dynamique de la tâche d'indice i et Delai_Dyn_i est son délai dynamique.

P est l'hyperpériode qui vaut PPCM$\{T_i\}_{1\leq i\leq n}$.

La date T0, si elle existe est la date d'entrée en phase cyclique.

Il suffit donc de parcourir la liste des tâches enrichies pour rechercher cette date. On recherche la date pour chaque tâche t_i, et elle sera notée $T0_i$ pour la tâche d'indice i.

$T0=\max\{T0_i\}_{1\leq i\leq n}$.

Il arrive que cette date soit la même pour toutes les tâches et dans ces conditions on prend l'une d'entre elles pour T0.

III.4 Expérimentation

Dans cette partie nous vous présenterons quelques résultats de simulation avec notre outil.

Exemple de configuration de tâches donnée par le générateur de tâche : ce sont des tâches à échéances sur requête.

Indice	Ri	Ci	Di	Ti	Ui	Etat
0	5	6	11	11	0,55	0
1	0	6	11	11	0,55	0
2	0	6	11	11	0,55	0
3	3	4	11	11	0,36	0

Voici ce qu'indiquent les différentes colonnes:

Indice: désigne les indices des tâches,

Ri: les dates des réveil,

Ci: les durées d'exécution,

Di: les délais,

Ti: les périodes,

Ui: les facteurs d'utilisation du processeur,

Etat: les état des tâches.

Etat est un entier appartenant à l'ensemble {0,1,-1}.

Si la tâche est oisive, sont état prend la valeur -1.

Si la tâche est prête, son état prend la valeur 0.

Si la tâche est occupée, son état prend la valeur 1.

Ici nous vous présentons une séquence d'ordonnancement de la configuration précédente obtenue avec notre ordonnancer sur une durée de 100 unités de temps.

Nous ne pourrons pas vous présenter un diagramme de Gantt car le temps de simulation est assez long et par conséquent la présentation du diagramme n'a pas d'intérêt. Nous vous présenterons néanmoins la séquence obtenue sur chaque processeur.

- séquence obtenue sur le processeur 1:

1111113333-10211113333-10021113203-10002113220000002132

22000000132222000001322220000013222200000132222

- séquence obtenue sur le processeur 2:

22222000001122222000011122223300111122233 3-11111122333

-1111112233301111122333011111223330111112233301

L'ordonnanceur nous donnera aussi la date d'entrée dans une phase cyclique. Mais avant cela il nous donne le Plus Grand Commun Diviseur (PGCD) et le plus Plus Petit Commun Multiple (PPCM) des périodes de mon système de tâches.

Le PGCD des périodes est 11 et leur PPCM est 11

Ensuite le simulateur donne la date d'entrée dans le cycle pour chaque tâche (cela correspond à la vérification du prédicat donné précédemment dans la partie Outils. Le prédicat prend 1 quand il est vérifié).

Le prédicat est 1 pour la ligne 0 a la date 48

Le prédicat est 1 pour la ligne 1 a la date 0

Le prédicat est 1 pour la ligne 2 a la date 54

Le prédicat est 1 pour la ligne 3 a la date 14

La date d'entrée dans le cycle est le maximum des dates d'entrée dans le cycle pour chaque tâche.

La date d'entrée dans la phase cyclique vaut 54.

IV. Etude

Dans cette partie de notre étude nous allons présenter quelques résultats obtenus lors de notre travail. Pour toute notre étude, nous avons travaillé sur un ensemble de configurations de tâches que nous a donné notre générateur de tâches. Nous avons subdiviser ces tâches en plusieurs classes.

Pour chaque classe nous avons cherché la date d'entrée dans le cycle avec les trois algorithmes RM, ED et LL. Nous avons aussi retenu quelques paramètres qui nous seront utiles lors de notre étude.

Ces paramètres PPCM qui est le PPCM des périodes, N est qui le nombre de tâches d'une configuration de tâches, Rmin et Rmax qui sont respectivement les dates de réveil minimale et maximale d'une configuration de tâches. La colonne Numéro indique le numéro d'une configuration de tâches.

Voici résumé, la signification des différentes rubriques en colonne:

Numéro: Le numéro de la configuration de tâches,

RM: Date d'entrée dans le cycle sous l'ordonnancement RM,

ED: Date d'entrée dans le cycle sous l'ordonnancement ED,

LL: Date d'entrée dans le cycle sous l'ordonnancement LL,

PPCM: Méta-période de la configuration,

N: Nombre de tâches du systèmes de tâches,

Rmin: Les minimum des dates de réveil du système de tâches,

Rmax: Le maximum des dates de réveil du système de tâches.

- Les configurations de tâches ayant toutes une dates de réveil nulles

  Nous n'avons pas jugé ici nécessaire de présenter le tableau car toutes les dates d'entrée en phase cyclique son nulles quelle que soit l'algorithme utilisée.

  En effet, toutes les tâches sont exactement dans le même état aux instants 0 et $PPCM\{T_i\}_{1\leq i\leq n}$ c'est à dire qu'elles sont toutes activées à 0 et $PPCM\{T_i\}_{1\leq i\leq n}$

- Les configurations de tâches ayant une seule date de réveil non nulle :

Numéro	RM	ED	LL	PPCM	N	Rmin	Rmax
1	**4**	**4**	**4**	2	3	0	2
2	10	10	28	36	3	0	1
3	3	3	3	6	4	0	1
4	3	3	3	6	4	0	1
5	19	19	19	72	5	0	1
6	4	4	4	12	4	0	1
7	2	2	2	5	2	0	1
8	9	9	9	88	5	0	1
9	5	5	5	24	4	0	1
10	**3**	**3**	**3**	2	3	0	1
11	***5***	***5***	***5***	3	4	0	2
12	5	5	5	12	5	0	1
13	**15**	**15**	**15**	11	6	0	4
14	4	4	4	12	5	0	1
15	19	19	19	72	4	0	1
16	5	5	5	12	4	0	1
17	6	6	6	20	5	0	1
18	4	4	4	6	5	0	1
19	5	11	5	12	3	0	1
20	4	4	4	6	4	0	1
21	**3**	**3**	**3**	2	3	0	1
22	5	5	5	12	3	0	1
23	5	5	5	12	3	0	1

- Les configurations de tâches ayant deux dates de réveil non nulles :

Numéro	RM	ED	LL	PPCM	N	Rmin	Rmax
1	5	6	5	6	3	0	2
2	***19***	***19***	**10**	9	4	0	1
3	4	4	4	6	4	0	1
4	**8**	**8**	**8**	6	4	0	2
5	4	4	4	6	5	0	1
6	4	4	4	12	5	0	1
7	11	11	11	90	6	0	2
8	***15***	***15***	***15***	7	4	0	1
9	11	7	7	12	5	0	1
10	4	4	4	12	6	0	1
11	**3**	**3**	**3**	2	3	0	1
12	9	9	15	40	4	0	1
13	21	11	11	30	4	0	1
14	**8**	**8**	**8**	6	5	0	2
15	**13**	**13**	7	12	4	0	1
16	13	9	3	24	6	0	1
17	12	9	5	12	6	0	1
18	5	5	5	12	4	0	1
19	4	4	4	6	4	0	1
20	10	10	10	168	6	0	3

- Les configurations de tâches n'ayant aucune date de réveil nulle :

Numéro	RM	ED	LL	PPCM	N	Rmin	Rmax
1	***11***	**8**	**8**	5	5	1	3
2	**13**	7	7	12	4	1	1
3	**13**	7	9	12	5	1	1
4	17	21	15	56	6	1	1
5	22	22	13	210	6	1	2
6	13	13	11	30	7	1	1
7	29	15	15	42	5	1	2
8	**9**	**9**	**9**	8	4	1	1
9	**7**	**7**	**7**	6	4	1	1
10	**11**	**11**	**11**	4	4	2	7
11	***21***	**12**	**12**	10	5	1	2
12	***17***	28	9	56	5	1	1
13	**5**	**5**	**5**	4	3	1	1
14	**3**	**5**	**3**	2	3	1	1
15	**11**	***21***	**11**	10	5	1	1
16	**3**	**3**	**3**	2	4	1	1
17	4	5	4	6	3	1	1
18	8	17	8	21	3	1	2
19	**5**	**5**	**5**	4	3	1	1
20	**5**	**5**	**5**	**4**	4	1	1

- Les configurations de tâches à échéance sur requête :

Numéro	RM	ED	LL	PPCM	N	Rmin	Rmax
1	**7**	4	4	6	4	1	1
2	5	5	5	12	4	1	2
3	26	13	7	30	5	1	1
4	**7**	4	4	6	4	1	1
5	11	**26**	6	20	4	1	1
6	29	15	15	56	3	1	1
7	**11**	**11**	**6**	5	6	1	1
8	**9**	**6**	**6**	4	4	1	2
9	9	5	5	12	4	1	1
10	**13**	**13**	**13**	12	4	1	1
11	25	13	13	36	4	1	1
12	14	8	8	60	5	1	2
13	11	6	6	20	5	1	1
14	17	**62**	9	56	6	1	1
15	**10**	**10**	**10**	9	3	1	1
16	**5**	**5**	**3**	2	3	1	1
17	20	72	39	72	5	1	3
18	11	11	13	72	4	1	2
19	**7**	**7**	**4**	3	3	1	1
20	15	15	8	42	4	1	1

IV.1 Temps creux acycliques

Dans le cas monoprocesseur, il a été montré dans [Choquet-Geniet et Grolleau, 2003] que pour une configuration de tâches de charge maximale, le nombre de temps creux acyclique est bornée et mieux, que le dernier temps creux acyclique apparaît au plus tard à la date r+P-1 où $r=\max\{r_i\}_{1\leq i\leq n}$ et $P=PPCM\{T_i\}_{1\leq i\leq n}$ pour une configuration de n tâches.

Nous allons montrer ici que dans le cas multiprocesseurs, pour une configuration de tâches de charge maximale le nombre de temps creux acycliques demeure borné et sur quelques exemples montrer que la borne n'est plus la même dans le cas multiprocesseurs.

Enoncé: *Sur une architecture multiprocesseurs et pour une configuration de tâches de charge maximale, le nombre de temps creux acycliques est borné.*

Démonstration:

Soit une configuration de n tâches. Soit [0, r+k*T] un intervalle d'étude où :

$r=\max\{r_i\}_{1\leq i\leq n}$, $T=PPCM(T_i)_{1\leq i\leq n}$ et k un entier positif. Soit m le nombre de processeurs.

Partant du fait qu'un processeur peut exécuter jusqu'à N unités de temps CPU en N unités de temps,

les m processeurs peuvent exécuter au maximum m*N unités de temps CPU en N unités de temps.

Cela est dû au fait qu'on raisonne ici en terme de charge globale.

Le nombre minimal d'instances de la tâche d'indice i entre 0 et k*T est $[(k*T)/T_i]$,

a/b étant étant la division entière de a par b.

k*T et Ti étant des entiers et T un multiple de Ti, cette valeur vaut $k*(T/T_i)$.

La nombre quantité minimale de temps CPU consommée est donc :

$\sum_{1\leq i\leq n}[Ent((r-r_i)/T_i)+k*m*(T/T_i)]*C_i = \sum_{1\leq i\leq n}Ent[(r-r_i)/T_i]*C_i + m\sum_{1\leq i\leq n}k*(T/T_i)*C_i$

$=\sum_{1\leq i\leq n}Ent[(r-r_i)/T_i]*C_i + k*m*T.$

Le nombre maximum d'unités de temps d'inactivité processeur est égale à la somme des nombre de temps d'inactivité de chacun des processeurs et vaut

$[r+m*k*T- \sum_{1\leq i\leq n}Ent[(r-r_i)/T_i]*C_i + k*m*T]= r- \sum_{1\leq i\leq n}Ent[(r-r_i)/T_i]*C_i$.

Cette valeur ne dépendant pas de k, elle reste finie même si k est assez grand.

Par suite le nombre de temps creux est borné.

Sur architecture monoprocesseur, la borne supérieure est r+P-1. Mais dans notre contexte, le

tableau qui suit montre que cette borne est souvent dépassée.

Dans le tableau 1, nous avons mis en gras les configurations pour lesquelles la borne est dépassée. Les lignes indiquent le numéro de configuration et les colonnes indiquent la politique d'ordonnancement utilisée.

Numéro	RM	ED	LL	r+P-1
1	**10**	0	0	7
2	12	6	4	12
3	12	0	6	12
4	0	8	8	8
5	6	6	6	6
6	5	5	5	10
7	**20**	0	0	11
8	0	4	0	4
9	0	**4**	0	2
10	0	**20**	0	10
11	0	0	0	2
12	0	3	0	6
13	0	14	0	22
14	0	4	0	4
15	0	0	0	4

Tableau 1: Date d'apparition du dernier temps creux acyclique pour un échantillon de 15 configurations de tâches

IV.2 Dates d'entrée en phase cyclique

Sur architecture monoprocesseur, il a été montré dans [Choquet-Geniet et Grolleau, 2003] que la séquence devient cyclique après l'apparition du dernier temps creux acyclique. Cette propriété n'est plus respectée dans le cas multiprocesseur. En effet dans le tableau suivant (Tableau 2), nous vous présentons des configurations pour lesquelles la date d'entrée dans la phase cyclique est supérieure à r+P.

Numéro	RM	ED	LL	r+P
1	***11***	**8**	**8**	7
2	**13**	7	7	14
3	**13**	7	9	31
4	17	21	15	7
5	22	22	13	21
6	13	13	11	57
7	29	15	15	6
8	**9**	**9**	**9**	6
9	**7**	**7**	**7**	13
10	**11**	**11**	**11**	13
11	***21***	**12**	**12**	37
12	***17***	28	9	62
13	**5**	**5**	**5**	22
14	**3**	**5**	**3**	57
15	**11**	***21***	**11**	10
16	**3**	**3**	**3**	3
17	4	5	4	75
18	8	17	8	74
19	**5**	**5**	**5**	4
20	**5**	**5**	**5**	43

Tableau 2: Dates d'entrée dans la phase cyclique pour un un échantillon de 20 tâches.

Dans ce tableau les numéros de configuration de tâches sont en ligne et les politiques d'ordonnancement en colonne. Nous avons mis en gras les dates d'entrée dans le cycle supérieures ou égales à r+P.

Notre travail a pour but de voir comment évoluent les dates d'entrée dans le cycle et d'essayer de les caractériser analytiquement si possible. Pour cela nous avons essayé de voir s'il y a des corrélations entre la date d'entrée dans le cycle et les différent paramètres des tâches.

Tout d'abord, pour voir comment évoluent les dates d'entrée dans le cycle, nous avons subdivisé les tâches en quatre grandes classes. Ce sont:

- Les tâches dont toutes les dates de réveil sont nulles (Type0),
- Les tâches ayant une seule date de réveil non nulle (Type1),
- Les tâches ayant deux dates de réveil non nulles (Type2),
- Les tâches quelconques avec aucune date de réveil nulle (Type3) et
- Les tâches à échéance sur requête (Type4).

La tableau Tableau 3 et la figure 3 nous montre que quand on passe de Type0 à Type4 dans ce ordre, la date d'entrée dans le cycle croît en moyenne. Pour être sûr qu'il ne serait pas vide de sens de comparer les moyennes, nous avons calculé les coefficients de variation que nous présentons

dans le Tableau 4.

	Type0	Type1	Type2	Type3	Type4
RM	0	6,39	9,15	11,35	13,1
ED	0	6,65	8,15	11,3	15,55
LL	0	7,17	7,15	8,5	9,2

Tableau3: Evolution de la date d'entrée dans le cycle en fonction de la classe de tâche et de la politique d'ordonnancement.

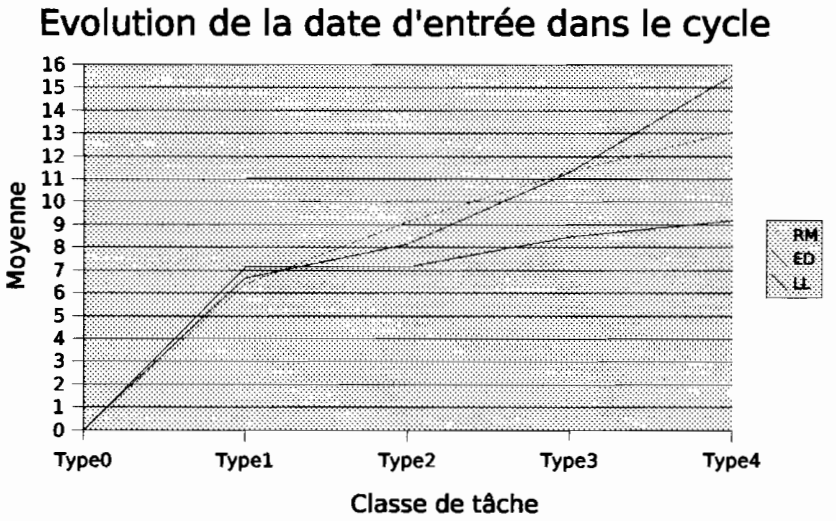

Figure 3 : Evolution de la date d'entrée dans le cycle en fonction de la classe de tâche et de la politique d'ordonnancement

	Type0	Type1	Type2	Type3	Type4
RM		0,76	0,57	0,62	0,53
ED		0,74	0,52	0,63	1,19
LL		0,92	0,53	0,43	0,85

Tableau 4 : Tableau des coefficients de variation.

D'une manière générale, les coefficients de variation sont inférieurs à 1 ce qui veut dire que la dispersion des dates d'entrée dans le cycle autour de la moyenne n'est pas grande. Utiliser les moyennes pour voir l'évolution des dates d'entrée dans le cycle n'est donc pas dénué de sens.

On voit sur la courbe que la date d'entrée dans le cycle croît selon que l'on passe de la classe Type0 à la classe Type4. On a donc l'impression que la date de réveil influe beaucoup sur la date d'entrée dans le cycle. Mais quand on voit les corrélations entre la date d'entrée dans le cycle et $r=\max\{r_i\}_{1\leq i\leq n}$ (Tableau5), à elle seule, la date de réveil n'influe pas sur la date d'entrée dans le cycle sauf pour les tâches à échéance sur requête où les coefficients sont assez élevés avec les politiques ED et LL.

Par contre si on voit les corrélations entre la date d'entrée dans le cycle et $r+PPCM\{T_i\}_{1\leq i\leq n}$ (Tableau6), on constate qu'il y a un lien. Ce lien lien est sans doute dû à $PPCM\{T_i\}_{1\leq i\leq n}$ ce qui permet de dire que la période influe sur la date d'entrée dans le cycle.

	Type0	Type1	Type2	Type3	Type4
RM		0,31	-0,05	0,17	0,01
ED		0,28	0,09	0,07	0,4
LL		0,19	0,22	0,29	0,64

Tableau 5 : Coefficients de corrélation entre la date d'entrée dans le cycle et $\max\{r_i\}_{1\leq i\leq n}$

	Type0	Type1	Type2	Type3	Type4
RM		0,77	0,17	0,57	0,61
ED		0,74	0,21	0,61	0,6
LL		0,65	0,37	0,51	0,66

Tableau 6 : Coefficients de corrélation entre la date d'entrée dans le cycle et $\max\{r_i\}_{1\leq i\leq n}+PPCM\{T_i\}_{1\leq i\leq n}$

Là non plus nous ne pouvons pas tirer une conclusion car il y a des tâches pour lesquelles la date d'entrée dans le cycle est très tardive.

Nous présentons dans Tableau 7 quelques unes des configurations:

	Numéro tâche	r	C	D	T
Config 1	1	9	3	16	16
	2	1	3	4	4
	3	3	9	16	16
	4	5	1	2	2
Config 2	1	2	4	9	9
	2	0	4	9	9
	3	6	2	3	3
	4	5	4	9	9
Config 3	1	0	12	13	13
	2	1	3	13	13
	3	1	3	13	13
	4	7	8	13	13

Tableau 7 : Exemples de configurations de tâches

Pour la configuration *config1*, la date d'entrée dans le cycle est: pour RM 35, ED 25 et LL 56

Pour la configuration *config2*, la date d'entrée dans le cycle est: pour RM 14, ED 35 et LL 35

Pour la configuration *config3*, la date d'entrée dans le cycle est: pour RM 33, ED 77 et LL 64

La nécessité de voir alors les corrélations entre la date d'entrée dans le cycle et les autres paramètres notamment C et D des tâches s'imposent à nous pour essayer de comprendre comment évoluent les dates d'entrée en phase cyclique.

Nous présentons dans les tableaux Tableau 8 et Tableau 9 respectivement le coefficients de corrélation entre les date d'entrée dans la phase cyclique et le maximum des délais ainsi que le coefficient de corrélation entre les dates d'entrée dans la phase cyclique et le maximum des durées d'exécution.

	Type0	Type1	Type2	Type3	Type4
RM		0,33	0,79	0,66	0,75
ED		0,33	0,84	0,82	0,3
LL		0,21	0,69	0,68	0,53

Tableau 8 : Coefficient de corrélation entre la date d'entrée en cycle et le Délai

	Type0	Type1	Type2	Type3	Type4
RM		0,43	0,71	0,47	0,7
ED		0,42	0,83	0,67	0,39
LL		0,31	0,66	0,57	0,53

Tableau 9 : Coefficient de corrélation entre la date d'entrée en cycle et la Durée d'exécution

Sur ces tableaux, on peut observer qu'il y a une corrélation entre la date d'entrée dans le cycle et la durée d'exécution ainsi qu'avec le délai pour : les taches ayant seulement deux dates de réveil non nulles, les tâches n'ayant aucune date de réveil nulle et les tâches à échéance sur requête. Cependant, avec la politique ED et en ce qui concerne les tâches à échéance sur requête, la valeur du coefficient de corrélation indique qu'il n'y a pas de corrélation.

Pour les tâches ayant seulement une date de réveil non nulle, il n' y a pas corrélation corrélation entre la date d'entrée dans le cycle et ces paramètres. Pour celles ci il y a plutôt corrélation avec $\max\{r_i\}_{1\leq i\leq n} + \text{PPCM}\{T_i\}_{1\leq i\leq n}$.

Pris individuellement, les paramètres influent sur la date d'entrée dans le cycle mais pas de la même manière. Les paramètres n'influent pas de la même manière non plus selon la politique d'ordonnancement utilisée.

Nous avons constaté que d'une manière générale, la politique ED crée l'exception à chaque constat.

Cela est sans doute dû au fait qu'avec cette politique, il y a trop de variation de la date d'entrée dans le cycle comme le montre d'ailleurs le coefficient de variation de ces dates.

Une combinaison des différents paramètres nous permettrait de voir comment évolue les dates d'entrée dans le cycle.

Il serait aussi intéressant d'étudier individuellement chaque paramètres et de voir comment évolue la date d'entrée dans la phase cyclique en fonction d'eux.

IV.3 Conclusion de la partie étude

Nous avons montré dans cette partie que le nombre de temps creux acyclique reste borné sans toutefois donner une borne. Il serait intéressant de donner cette borne.

Sur des exemples nous avons constaté aussi que les paramètres des tâches influent sur la date date d'entrée dans la phase cyclique et que cette influence n'est pas la même selon la politique d'ordonnancement utilisée.

Nous n'avons pas encore caractérisé de manière analytique comment évoluent ces dates dates d'entrée dans le cycle, mais si on arrivait à le faire, cela permettrait de donner une borne la date d'entrée dans le cycle.

De façon globale, nous avons montré sur des exemples que ce qui est un acquis en monoprocesseur ne l'est plus sur une architecture multiprocesseurs.

V. Conclusions

La plupart des résultats vrais sur architecture monoprocesseur ne le sont plus sur architecture multiprocesseurs ce qui rend ce domaine de recherche encore ouvert. Parmi ces résultats, on peut citer les résultats sur la date d'apparition du dernier temps creux acyclique ainsi que la date d'entrée dans le cycle.

Nous avons montré que le nombre de temps creux acycliques reste borné sur architecture multiprocesseurs sans toutefois donner une borne.

Nous avons aussi montré sur des exemples de séquences d'ordonnancement que la date d'entrée dans le cycle est corrélée avec les paramètres des tâches. Mais les paramètres avec lesquels il y a corrélation ne sont pas les mêmes selon que l'on passe d'un algorithme d'ordonnancement à un autre.

Il est aussi ressorti de notre étude que les dates d'entrée en phase cyclique sont plus dispersées avec la politique d'ordonnancement ED qu'avec RM et LL.

Une étude plus approfondie sur les types de corrélation ainsi qu'une bonne combinaison des paramètres permettrait de caractériser analytiquement ces dates d'entrée en phase cyclique.

Il serait aussi intéressant de borner la date d'apparition du dernier temps creux acyclique et de voir comment évolue la date d'entrée dans le cycle avec cette borne.

L'ordonnancement hors-ligne sur architecture multiprocesseurs reste donc un domaine à approfondir car pour le moment, il n'existe que des résultats d'impossibilité et des limites théoriques.

Bibliographie

[Chquet-Geniet et Grolleau, 2003] Annie Choquet-Geniet, Ammanuel Grolleau, **Minimal schedulability interval for real-time systèmes of periodic tasks with offsets**, Elsevier B.V, 2003.

[Choquet-Geniet 2003] Annie Choque-Geniet, **Panorama de l'ordonnancement temps reél monoprocesseur**, Ecole d'été Temps Réel 2003.

[Choquet-Geniet et al, 2000] Annie Choquet-Geniet, Emmanuel Grolleau, Francis Cottet, **Etude hors ligne d'une application temps réel à contraintes strictes**, 2000.

[Cottet et al, 2000] F.Cottet, J.Delacroix, C.Kaiser, Z.Mammeri, **Ordonnancement temps réel -cours et exercices corrigés**, Hermès Science Publications, Paris, 2000.

[Dertouzos et Mok, 89] Michael L. Dertouzos et Aloysius KA-LAU MOK, **Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks**, IEEE Transactions on software engineering, Vol. 5, NO. 12, December 1989.

Emmanuel Grolleau, **Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseurs**, Thèse, Novembre 1999.

F.Cottet, **Les systèmes informatiques temps réel embarqués,** Cours ENSMA A3 et DEA T3IA 2003/2004.

[Gossens et Devillers, 1997] J.Goossen and R.Devillers, **The Non-Optimality of the Monotonic Priority Assignments for Hard Real-Time Offset Free Systems**.

[Graham, 1976] Graham R. **« Bounds on the performance of scheduling algorithms »**, In Computer and Job Shop Scheduling Theory, John Wiley and Sons, 1976.

[Lehoczky et Ramos-Thuel, 1989] Lehoczky J.P et Ramos-Thuel S., **« The Rate Monotonic scheduling algorithm : exact characterization and average case behavor »**, Proceedings of Real-Time Systems Symposium, 1989.

[Liu et Layland, 1973] C.L. Liu et J.W.Layland, **« Scheduling algorithms for multiprogamming in a hard real-time environment »**, Journal of the ACM, 1973.

[Mok et Dertouzos, 1978] Mok A.K.L., and DERTOUZOS M.L., **Multiprocessor scheduling in real-time environment**, 1978.

[Sahni, 1979] Sahni S.K, **Preemptive scheduling with due dates, Operational Research**, Vol.27, 1979.

[Yingfeng et Sang, 1995] Yingfeng Oh and Sang H.Son, **Allocating Fixed-Periodic Tasks on Multiprocessor Systems**, Kluwer Academic Publisher, No 3, Novembre 1995.