


République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Akli Mohand Oulhadj de Bouira

Faculté des Sciences et des Sciences Appliquées

Département d'Informatique

Mémoire de Master

en Informatique

Spécialité : *ISIL*

Thème

Conception d'approche de régulation pour l'ordonnancement temps réel des tâches

---

Encadré par

— Dr. ABBAS Akli

Réalisé par

— KHITER ALI

— HELLAL CHAHINEZ

2019/2020

Remerciements

Nous tenons avant tout a remercier Dieu tout puissant de nous avoir donné la force et la volonté pour achever ce mo-deste travail.

Nous tenons à exprimer nos vifs remerciements envers notre encadreur Dr. ABBAS Akli, pour sa disponibilité, son encadrement, sa confiance et les conseils qu'il nous a généreusement prodigués.

Un grand merci a tout ceux qui nous a aides dans le département informatique et également a tous les enseignants pour tout le savoir qu'ils ont su nous transmettre durant ces dernières années. Et nous remercions tous ceux qui ont contribué de prés ou de loin a la réalisation de ce travail.

Dédicaces

je dédie ce modeste travail a :

Mes très chers parents pour m'avoir donné tout le soutien au long de mes études et m'avoir apporter un grand support moral lors de la rédaction de ce mémoire, qu'ils trouvent ici l'expression de mon profond amour.
Mon cher frère .
Ma cher sœur .
Mon frère binôme Ali et sa famille .

Toute ma famille je ne me permettrais surtout pas d'oublier mes très chères amies Ikram ,Rofaida, khadija ,Fatima ,Rabia .

Hellal chahinez.

Dédicaces

Du profond de mon cœur je dédie ce travail a tous ceux qui me sont chers. A mes chers parents,
Que ce travail soit l'expression de ma reconnaissance pour vos sacrifices consentis, votre soutient moral et matériel que vous n'avez cesse de prodiguer. Vous avez tout fait pour mon bonheur et ma réussite. Que dieu vous préserve en bonne santé et vous accorde une longue vie Vous étiez toujours présents pour m'aider et m'encourager.

A mes sœurs et tout ma famille.

khiter Ali.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

RM	Rate Monotonic
DM	Deadline Monotonic
LLF	Least Laxity First
EDF	Earliest Deadline First
FBB-FFD	Fisher, Baruah and Baker - First Fit Decreasing
RMNF	Rate Monotonic First-Fit
RMFF	Rate MonotonicNext-Fit
RMBF	Rate MonotonicBest-Fit
RMWF	Rate MonotonicWorst-Fit
DVFS	Dynamic Voltage and FrequencyScaling
DPM	Dynamic Power Management
CMOS	complementarymetal-oxide-semiconductor
EDeg	Earliest Deadline withenergyguarantee
FBA	Frame-BasedAlgorithm
LSA	LazySchedulingAlgorithm
EA-DVFS	Energy-aware DVFS Algorithm
HA-DVFS	Harvesting-aware DVFS algorithm
AS-DVFS	Adaptive Scheduling DVFS algorithm AS-DVFS
PID	Proportionnel-Intégral-Dérivé
WCET	Worst Case Execution Time
ARM	Advanced Risc Machines

Introduction générale

Motivations :

Au fil des dernières décennies, le développement des systèmes temps réel a représenté un défi qui a été relevé par les communautés dans différentes domaines, tel que le transport, la médecine, le multimédia, les téléphones mobiles, les consoles de jeux, ... etc. Les systèmes temps réel concernent des applications informatiques ayant un rôle de suivi ou de contrôle de procédé, dont le fonctionnement est assujetti à l'évolution dynamique de l'état de ce procédé. L'application temps réel est réactive, elle doit réagir au changement d'état du système contrôlé avec des contraintes temporelles précises.

Les systèmes embarqués sont autonome et interdisent les interventions humaines parce qu'ils sont inaccessibles ou déployés en trop grand nombre. Et dont l'alimentation est assurée par des batteries ou super condensateurs, mais dont la quantité d'énergie disponible limite encore fortement la durée de vie de ce type de système, ce qui signifie la fin de fonction de tels systèmes. Pour étendre la durée de ce système, l'utilisation des énergies renouvelables (solaire, éolienne, etc.) se révèle être une alternative de choix pour alimenter bon nombre de systèmes. Autrement dit, il s'agit de la récupération d'énergie. Cette approche est considérée très prometteuse. Elle permet de prolonger la durée de vie des batteries ou éliminées complètement.

Les solutions proposées dans ce domaine de recherche utilisent souvent la borne supérieure du temps d'exécution des tâches : le temps d'exécution dans le pire cas (ou WCET). Avec l'utilisation des WCET, il en résulte un sur-dimensionnement de l'ordonnancement ainsi que la sous-utilisation des ressources de traitement induisant un sur-dimensionnement coûteux, en prix du matériel, en poids et en énergie consommée, des capacités de ces

ressources. Dans ce contexte, le principe *régulation de l'ordonnancement* a pour avantage d'apporter, d'une part, une robustesse vis-à-vis des incertitudes des WCETs des tâches, et d'autre part, une adaptation à la variation de ces durées.

En plus, les systèmes embarqués actuels sont équipés de multiprocesseurs (multi-cœur), ce qui ajoute une contrainte supplémentaire à ces systèmes. Cette contrainte se matérialise dans le choix du processeur (ou du core) sur lequel vont s'exécutées les tâches. Ainsi dans le cas de l'ordonnancement temps réel multiprocesseur, deux approches principales se distinguent par leur aptitude à tolérer ou non les migrations entre processeurs. Le premier est l'ordonnancement par partitionnement, dans laquelle les migrations des tâches d'un processeur à un autre ne sont pas autorisées. La seconde est l'ordonnancement global, pour laquelle les migrations des tâches d'un processeur à un autre sont autorisées. Il existe aussi une autre approche d'ordonnancement dite de semi-partitionnement obtenue par combinaison de l'approche globale et par partitionnement.

Objectif de notre étude :

L'objectif de notre travail vise à étudier le problème d'ordonnancement temps réel multiprocesseur régulé des tâche sous contrainte d'énergie renouvelable. Ainsi notre étude vise d'une part à améliorer les performances et la qualité de contrôle du système et d'une autre part à éviter la décharge totale des batterie et donc de prolonger la durée de vie de système.

Contribution :

Dans notre contribution, nous avons proposé un régulateur flou d'ordonnancement temps réel multiprocesseur dans le contexte de système de récupération de l'énergie. Notre contribution vise à prendre en compte les incertitudes des durées d'exécution des tâches, d'énergie disponible dans la batterie afin d'étendre sa durée de vie et d'optimiser la qualité de contrôle du système.

Organisation du mémoire :

Notre mémoire est organisé comme suit :

Le chapitre 1 : Nous présenterons les systèmes temps réel et leurs architectures, la modélisation des taches temps réel, l'ordonnancement temps réel mono-processeur, multiprocesseur et les solutions proposées dans la littérature.

Le chapitre 2 : Ce chapitre donne un bref aperçu des différentes sources d'énergie renouvelable, des principales technologies de stockage d'énergie, la consommation énergétique dans les systèmes temps réel, les méthodes de réduction de la consommation énergétique, l'ordonnancement temps réel sous contraintes d'énergie renouvelable et leurs solutions proposées.

Le chapitre 3 : Nous présenterons les stratégies de commande des procédés, les paramètres et les critères de performances, la problématique d'ordonnancement basé sur WCET, l'ordonnancement temps réel régulé et un état de l'art des travaux importants sur l'ordonnancement régulé et des concepts de la logique floue.

Le chapitre 4 : Nous détaillerons notre contribution où nous proposons une conception d'un régulateur flou d'ordonnancement temps réel multiprocesseur, les différents modèles de la consommation d'énergie.

Le chapitre 5 : Nous décrirons les évaluations des solutions proposées et les résultats obtenus.

Chapitre 1

Présentation des systèmes temps réel

1.1 Introduction

Les systèmes embarqués temps réels sont de plus en plus présents dans le quotidien, on les trouve dans l'aéronautique, transport ferroviaire, l'automobile, l'électroménager ou le multimédia. On désigne du temps réel, toute application mettant en œuvre un système informatique dont le comportement (fonctionnement) est conditionné par l'évolution dynamique de l'état d'un environnement. Ces systèmes embarqués sont considérés comme étant des systèmes réactifs, du fait qu'ils gèrent des programmes dont les résultats sont fonction de données produites par l'environnement (le procédé à contrôler) auquel sont connectés. Ces systèmes temps réel embarqués sont de plus en plus réalisés avec des architectures multiprocesseur distribuées, c'est-à-dire comportant plusieurs processeurs.

1.2 Définition

1.2.1 Système temps réel

Les systèmes temps réel sont des systèmes numériques (informatique), qui implémentent une application où le respect des contraintes temporelles est la principale contrainte à satisfaire. Ces systèmes temps réel sont des systèmes réactifs continuellement aux stimuli venant de leur environnement considéré comme externe au système. Un tel système pour qu'il fonctionne correctement doit obligatoirement réagir à chacun des stimuli qu'il reçoit (par l'intermédiaire de capteurs, des entrées provenant de l'environnement), en effectuant un certain nombre d'opérations et produit, grâce à des actionneurs, des sorties utilisables

par l'environnement, appelées réactions. La figure 1.1 donne un aperçu des interactions qui existent entre procédé et contrôleur d'un système temps réel.

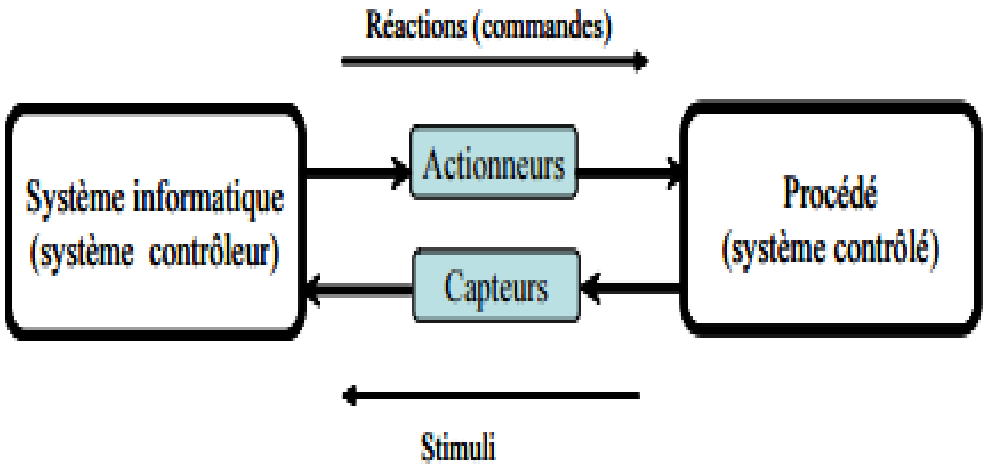

FIGURE 1.1 – Système temps réel

Ensuite La validité des systèmes temps réel ne dépend pas seulement de l'exactitude logique des calculs (exactitude des traitements) mais aussi de la date à laquelle le résultat est produit (Exactitude temporelle).[1]

1.2.2 Système contrôle-commande

Le système de contrôle consiste en une structure matérielle munie d'un ensemble de logiciels permettant d'agir sur le système contrôlé. La structure matérielle correspond à l'ensemble des ressources physiques (processeurs, cartes d'entrée/sortie, ...) nécessaires au bon fonctionnement du système [2]

les caractéristiques de système contrôle-commande :

- Efficacité de l'algorithme d'ordonnancement avec une complexité limitée.
- Respect des contraintes de temps (échéances).
- Prédictibilité (répétitivité des exécutions dans des contextes identiques).
- Capacité à supporter les surcharges.
- Possibilité de certification pour les applications de certains domaines comme l'avionique, l'automobile. [3]

1.2.3 Système embarqué

Un système embarqué est un dispositif qui inclut un composant programmable (microprocesseur ou microcontrôleur) mais qui n'est pas lui-même destiné à être un ordinateur à usage généraliste. Plus classiquement, un système embarqué est défini comme un système électronique/informatique autonome et ne possède pas des entrées/sorties standards et classiques comme clavier ou un écran d'ordinateur, souvent temps réel, spécialisé dans une tâche bien précise. Ses ressources sont généralement limitées. Cette limitation est généralement d'ordre spatial (encombrement réduit) et énergétique (consommation restreinte) [4].

Domaines d'application d'un Système Embarqué

- Télécommunications (transport de la parole, systèmes de commutation, ...)
- Domaine médical (assistance et supervision de malades, ...)
- Contrôle de différents équipements dans les voitures, bus, trains, avions, ...
- Contrôle et régulation de trafic en milieu urbain,
- Guidage de véhicules en milieu urbain,
- Industrie (contrôle/commande de procédés manufacturiers ou continus,...)
- Domaine militaire (suivi de trajectoires de missiles, ...)
- Aérospatial (suivi de satellites, simulation de vols, pilotage automatique, ...) [5]

Principales caractéristiques d'un Système Embarqué

- Encombrement mémoire (mémoire limitée, pas de disque en général)
- Consommation d'énergie (batterie : point faible des SE)
- Autonomie
- Mobilité
- Communication (attention : la communication affecte la batterie)
- Contraintes de temps réel.[5]

1.3 Architecture des systèmes temps réel

Un système temps réel est constitué d'une couche logicielle et d'une couche matérielle qui détermine la plateforme d'exécution de la couche logicielle. Dans la section suivent,

nous détaillons ces deux couches principale :

1.3.1 Architecture matérielle

Nous distinguons trois grandes catégories d'architecture matérielle pour les Systèmes Temps Réel en fonction de leur richesse en termes de nombre de cartes d'entrée/sortie, de mémoires, de processeurs et de la présence de réseaux (voir la figure 1.2) :[6] [7] [8]

❖ **Architecture monoprocesseur** Dans ce type de système, toutes les fonctions (tâches) de application sont exécutées par unique processeur partager donc les différentes applications partagent le temps processeur. Dans ce type il n'a pas de réseaux.

❖ **Architecture multiprocesseur** Dans ce cas, l'ensemble des fonctions du système temps réel sont réparties sur une architecture parallèle de certain nombre du processeurs possédant chacun une mémoire propre, ainsi que les architectures **multicoeurs** mutualisent leur mémoire suivant leur niveau de cache.

❖ **Architecture distribuée** Dans ce cas, les architectures multiprocesseurs ne partageant pas de mémoire centrale. Ces processeurs sont reliés entre eux par l'intermédiaire de réseaux permettant d'assurer les communications entre les différentes tâches. Une ferme d'ordinateurs est un exemple typique de cette architecture. La coopération se fait ici par communication par réseau.

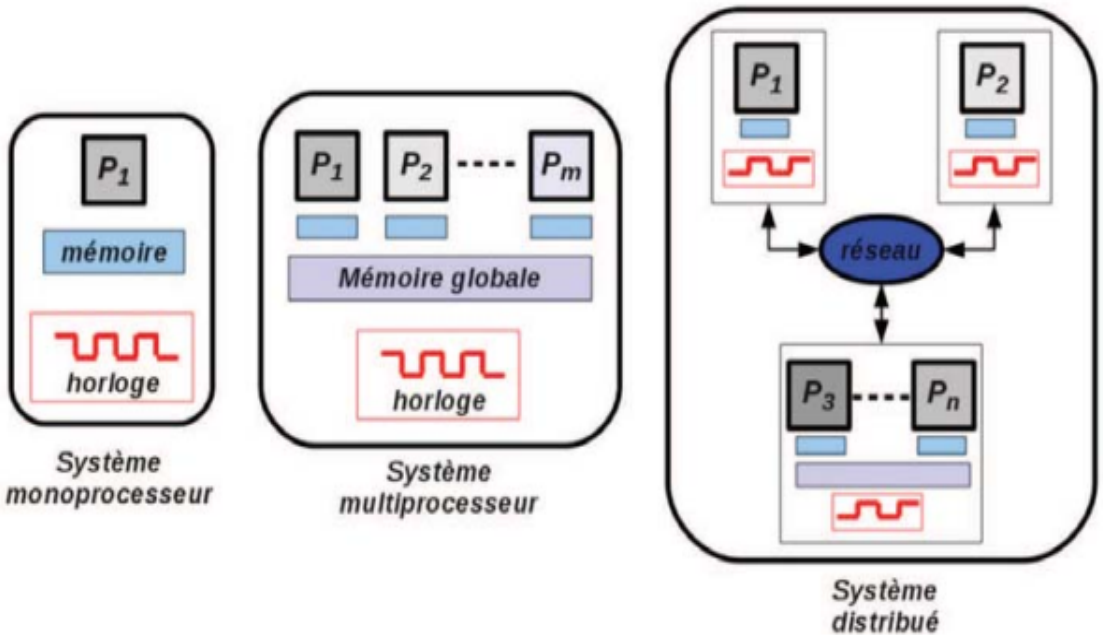

FIGURE 1.2 – Les différentes Architectures des système temps réel

1.3.2 Architecture logicielle

L'architecture logicielle d'un système temps réel se décompose en deux parties distinctes :

Exécutif temps réel : Est une partie de bas niveau faisant le lien avec la couche matérielle

Un programme applicatif : C'est une partie de haut niveau, correspondant aux fonctions permettant de contrôler le système temps réel.

➢ **Exécutif temps réel**

Nous pouvons maintenant décomposer l'architecture logicielle d'un système Temps réel en deux couches. La première consiste en une application concurrente composée d'un ensemble de tâches. Nous utilisons également le terme d'application multi tâche.La deuxième, de plus bas niveau, joue le rôle d'un système d'exploitation minimal chargé de faire le lien entre le procédé physique et l'application multitâches. Ce système d'exploitation, appelé exécutif Temps Réel, est de par la considération de l'asynchronisme, dirigé par les évènements, ceux-ci pouvant provenir de différentes sources :[7]

— Du procédé physique par l' intermédiare d'interruptions matérielles associées à chaque évènement.

— Du temps : chaque système est muni d'une horloge Temps Réel pouvant générer des interruptions.

— De l'application multitâches lorsque par exemple l'exécution d'une tâche est conditionnée par l'exécution d'autres tâches.

Exécutif temps réel possédé les services classiques d'un système d'exploitation, mais plus particulièrement :

— Gestion des tâches

— Gestion des ressources partagées

— Gestion des communications entre tâches

— Gestion du temps

— Gestion des interruptions et de la mémoire

➢ **Programme applicatif** Est la partie logicielle du système qui va exécuter les différentes fonctions nécessaires au contrôle du système, et plus particulièrement du procédé. Le programme applicatif est divisé en entités distinctes appelées tâches

ou processus, chacune ayant un rôle propre, comme par exemple : [8] Les tâches d'entrées/sorties et de traitement, de communications et aussi d'interactions entre les tâches, etc...

Un système temps réel peut être mis en place via un système d'exploitation ou non. Sans système d'exploitation, il est possible de disposer d'un exécutif (ou moniteur) temps réel, qui est une sorte de micronoyau. Cet exécutif est responsable de l'ordonnancement des tâches (ordre dans lequel elles s'exécutent), les tâches s'occupant de leur traitement propre. cette solution peut être adaptée à des système embarqués dont la fonctionnalité est très spécifique et peu soumise à modification. Dans des systèmes plus importants, il est agréable de disposer des facilités proposées par un système d'exploitation, tels qu'une interface utilisateur ou la section de fichier, par exemple. D'une manière générale, tout système d'exploitation doit supporter le temps réel. Il est en effet impossible de faire fonctionner une application temps réel strict sur un système d'exploitation standard. Ceci est notamment en partie dû aux facteurs nom prédictibles qui caractérisent les systèmes temps réel.[9] La figure 1.3 ci-dessous illustre la partie logicielle d'un système temps réel

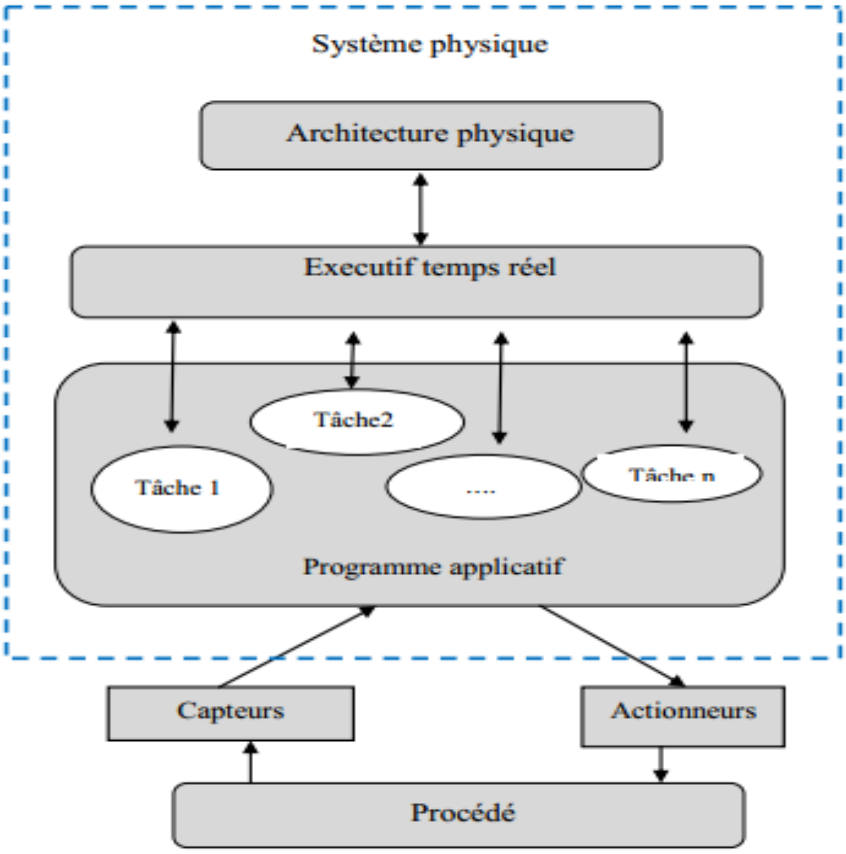

FIGURE 1.3 – Structure de l'architecture logicielle d'un système temps réel

1.4 Les caractéristiques des systèmes temps réel

Les systèmes temps réel et notamment à contraintes dures doivent répondre à trois critères fondamentaux : [10]

1.4.1 L'exactitude logique (exactitude des traitements)

Les mêmes entrées appliquées au système produisent les mêmes résultats.

1.4.2 L'exactitude temporelle

Les temps d'exécution de tâches est déterminé (les délais sont connus ou bornés et le retard est considéré comme une erreur qui peut entraîner de graves conséquences).

1.4.3 La fiabilité (aspect matériel)

Le système répond à des contraintes de disponibilité (matériel/logiciel)[6]

1.5 Classification des systèmes temps réel

On appelle échéance une contrainte temps à laquelle doit au plus tard se produire un événement. Les systèmes sont classifiés par rapport à la tolérance aux échéances. En fonction de la criticité des contraintes temporelles, on distingue essentiellement trois types de systèmes temps réel [11]

1.5.1 Un système temps réel dur (hard-real-time)

Où le non-respect des contraintes de temps peut conduire à des défaillances avec des conséquences pouvant être graves. Si l'échéance est dépassée, il y a faute. Exemples : contrôle aérien, contrôle d'une centrale nucléaire, etc. La figure 1.4 suivante illustre que une réponse dépassant son échéance est considéré comme réponse fautive.[5]

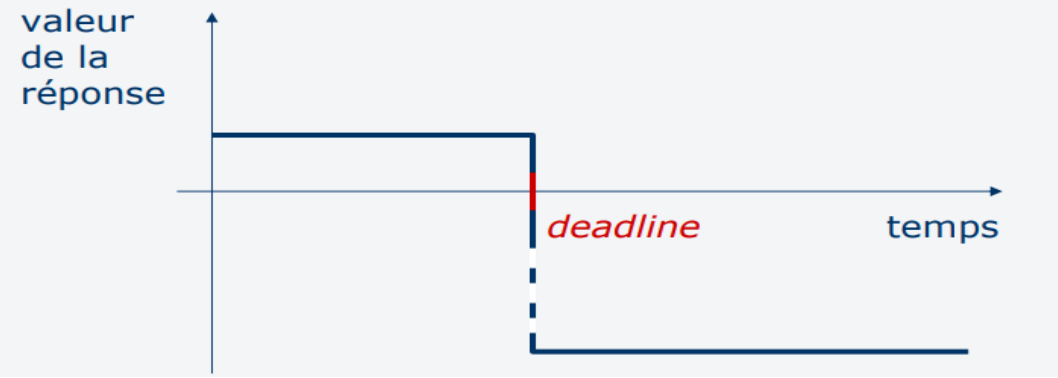

FIGURE 1.4 – Système temps réel dur

1.5.2 Un système temps réel souple (soft-real-time)

Où le dépassement des échéances est considéré comme une faute bénigne. Lorsqu'une échéance est dépassée, il n'y a pas faute ; le résultat peut être exploitable même s'il est fourni après l'échéance. Exemples : jeux vidéo, logiciel embarqué de votre téléphone, iPod, etc. La figure 1.5 suivante illustre qu'une réponse dépassant son échéance peut être acceptée à condition que la faute temporelle n'est pas grande.[5]

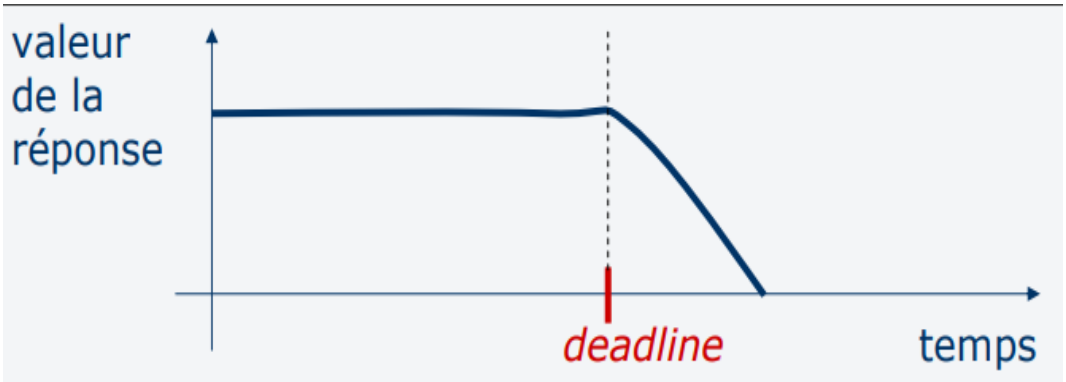

FIGURE 1.5 – Système temps réel souple

1.5.3 Un système temps réel ferme (firm-real-time)

Où le dépassement occasionnel des échéances est toléré. Il y a faute (bénigne) si l'échéance n'est pas respectée.

Exemples : transactions en bourse : le temps réel ferme de l'un peut être le temps réel dur de l'autre. La figure 1.6 suivante montre que le dépassement de l'échéance n'arrête pas la tâche.[5]

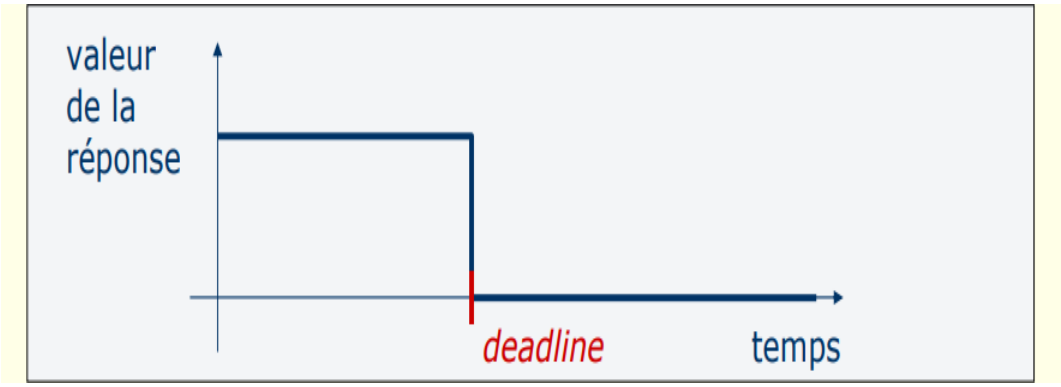

FIGURE 1.6 – Système temps réel ferme

1.6 Tâches temps réel

1.6.1 Définition

Nous appelons tâche temps réel une entité de base des programmes temps réel. Ce sont ces entités qui composent le programme d'un système temps réel. Les tâches temps réel peuvent être associées à des calculs, des alarmes, des traitements d'entrée/sortie, ...Les

tâches temps réel permettent ainsi de contrôler le procédé externe via un ensemble de capteurs et d'actionneurs intégrés au procédé. Elles utilisent les primitives de l'exécutif temps réel sur lequel elles sont exécutées pour communiquer entre elles, faire des acquisitions de données (depuis des capteurs du système contrôlé) ou encore actionner des commandes sur le système contrôlé [2]

1.6.2 Caractéristiques des tâches temps réel

Une tâche est un ensemble d'instructions destinées à être exécutées sur un processeur. La caractérisation d'une tâche peut varier d'un modèle d'ordonnancement à l'autre et suivant la nature de celle-ci. Parmi les paramètres temporels usuels définissant une tâche, nous citons :[12]

- ➢ **Date d'activation ri** C'est la date à laquelle la tâche τi peut commencer son exécution..
- ➢ **Durée d'exécution Ci (charge processeur)** C'est Le temps processeur requis par chaque instance de τi. Généralement, Ce paramètre est considéré comme le pire cas des temps d'exécution (WCET pour Worst Case Execution Time) sur le processeur ou elle va être exécuté. Le WCET représente une borne supérieure du temps d'exécution c'est à dire que la tâche peut se terminer plus tôt.
- ➢ **Période d'activation Ti** C'est La durée de temps fixe ou minimale entre deux activations de deux instances successives de τi.
- ➢ **Délai critique Di** C'est Le temps alloué à la tâche pour terminer complètement son exécution et le dépassement provoque une violation de la contrainte temporelle.

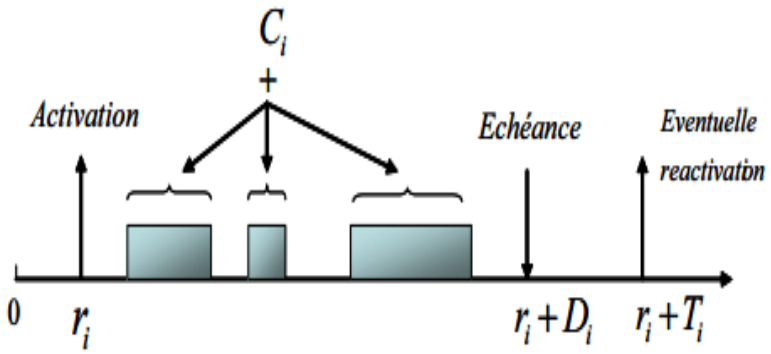

FIGURE 1.7 – Modèle usuelle d'une tâche temps réel.

1.7 Modélisations des tâches temps réel

Les tâches temps réel peuvent être périodiques, apériodiques ou sporadiques.

1.7.1 Modélisation des tâches périodiques

Les tâches périodiques représentent les tâches récurrentes dont les activations successives sont séparées par une période constante. Il existe dans la littérature temps réel plusieurs modèles de tâches périodiques. Le plus simple mais aussi le plus fondamental est celui de LIU et LAYLAND. Leur modèle représente chaque tâche par quatre paramètres temporels suivants : $\tau i = (ri\ ,Ci\ ,Di\ ,Ti\ )$ [2]

- **ri** : la date d'activation de la tâche τi
- **Ci** : désigne le temps d'exécution maximum de la tâche et τi sa période.
- **Di** : le délai critique alloué à la tâche pour se terminer après chacune de ces activations.
- **Ti** : la prériode d'activation de la tâche τi. C'est-à-dire qu'aprés chaque hi unités de temps à partir de ri, une occurrence (on dit aussi requête ou instance) aura lieu. La figure 1.8 illustre le rôle de chacun de ces paramètres.

D'après ces paramètres, on déduit les paramètres suivants :

- **Le facteur d'utilisation du processeur par une tâche τi est :**

$$Ui = Ci/Ti \tag{1.1}$$

- **Un facteur d'utilisation des tâches est :**

$$Ui = \sum_{i=1}^{n} Ci/Ti \tag{1.2}$$

- **La date de** Kéme **activation de la tach Ti est :**

$$r_{i,k} = r_i + (K-1) \times T_i \tag{1.3}$$

- **Le temps de réponse d'une tâche Ti :** R c'est le maximum des temps de réponse de ses requêtes :
-
- **Le temps de réponse d'une tâche Ti :** R c'est le maximum des temps de réponse de ses requêtes :

$$R = \max_{k \geq 1} R^K \tag{1.4}$$

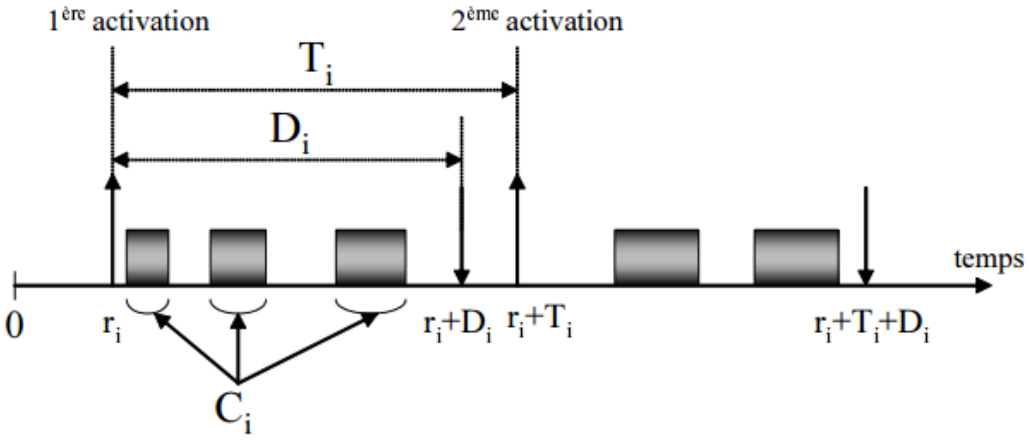

FIGURE 1.8 – Modélisation d'une tâche τi périodique..

Le paramètre ri permet de retarder la première exécution des tâches et n'impose pas ainsi aux tâches de commencer leur exécution au même moment à un instant $t = 0$. Ce paramètre est aussi appelé Date d'activation de la tâche. Deux tâches $\tau 1$ et $\tau 2$ sont à départ simultané si leurs dates d'activation sont égales (c.-à-d. r 1 = r 2).

1.7.2 Modélisation des tâches non périodiques

Les tâches non périodiques sont réparties en deux catégories. Les tâches sporadique et apériodique.

Les tâches sporadiques

Les tâches sporadiques sont des tâches récurrentes avec des contraintes strictes d'échéance. Une tâche sporadique τi est définie par trois paramètre suivent : $\tau = (C, D, T)$

- **C** est la durée d'exécution.
- **D** est le délai critique.
- **T** est le temps minimal entre deux requêtes successives de la tâche.

Le modèle peut être généralisé de la même façon que le modèle de tâche périodique.

Les tâches apériodiques

Une tâche apériodique a une date d'activation qui n'est connue qu'à l'activation et une certaine durée d'exécution. Par conséquent, deux paramètres suffisent pour représenter une tâche apériodique : $\tau = (r, C)$. Où r est sa date et C son temps d'exécution. Les

tâches apériodiques sont en général à contraintes souples car elles n'ont pas d'échéance avant la fin de leur exécution. Dans le cas contraire, un paramètre d'échéance d sera ajouté dans le modèle des tâches.

1.7.3 Dépendance entre tâches

La tâche peut dépendre d'autres tâches (sinon, la tâche est dite indépendante). La communication entre les tâches se fait via la mémoire partagée ou par message. L'instant de communication peut être limité par synchronisation. Dans ce cas, un ordre partiel prédéfini apparaît. Ces relations sont connues par avance. Ce sont les relations de précédence. Si τi précède τj, la relation est notée $\tau i \longrightarrow \tau j$. L'ensemble des relations est représenté par un graphe de précédence (voir la figure 1.9). Dans ce cas, la tâche $\tau 4$ ne peut s'exécuter que si les tâches $\tau 3$ et $\tau 2$ sont terminées. Et leur éligibilité est conditionnée par la terminaison de $\tau 1$. [6]

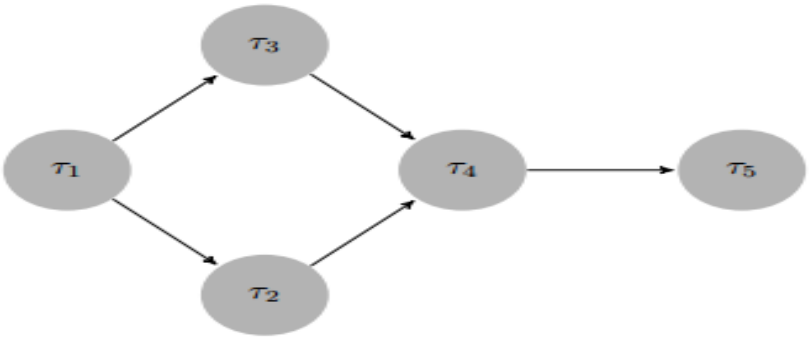

FIGURE 1.9 – La relation de précédence d'exécution entre les tâche par un graphe de précédence

1.7.4 Modélisation canonique des tâches temps réel

C'est un modèle qui regroupe les principaux paramètres temporels qui peuvent caractériser les tâches temps réel et guider l'ordonnancement temps réel. Les paramétrés spécifiques pour le modèle canonique des tâches sont présentes dans la figure 1.10
La signification des symboles est la suivante :

— **r** : sa date de réveil, c'est-à-dire le moment de déclenchement de la requête d'exécution ;

— **C** : sa durée d'exécution maximale quand elle dispose du processeur pour elle seule ;

— **R** : son délai critique, c'est-à-dire le délai maximal acceptable pour son exécution ;

— **P** : sa période lors qu'il s'agit d'une tâche périodique. [9]

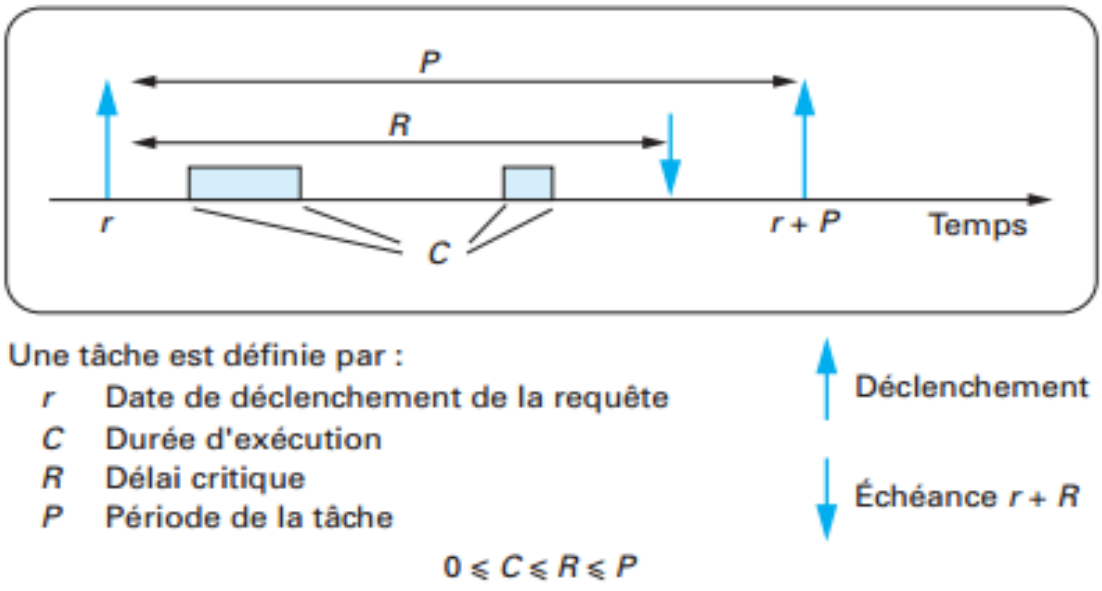

FIGURE 1.10 – Modèle canonique des tâches.

1.8 Ordonnancement temps réel

Du point de vue logiciel, une application temps réel est considéré comme un système multitâche. Le travail du système informatique consiste par conséquent à gérer l'exécution et la concurrence de l'ensemble tâches en optimisant l'occupation du processeur et en veillant à respecter les contraintes temporelles des tâches (délais, contraintes d'enchaînement). Toutes ces fonctions sont regroupées sous la notion **d'ordonnancement temps réel**. Si une contrainte temporelle n'est pas respectée, on parle d'une défaillance du système ou d'une faute temporelle.
Dans un système temps réel dur, chaque tâche est soumise à une échéance temporelle stricte. Le non-respect d'une contrainte temporelle n'est pas admissible à l'exécution.[2]

1.8.1 Définition d'un Ordonnancement

Un ordonnancement constitue une solution à la problématique d'ordonnancement. Il décrit l'exécution des tâches et l'allocation des ressources au cours du temps, et vise à satisfaire un ou plusieurs objectifs. Plus précisément, on parle de problème d'ordonnancement lorsqu'on doit déterminer comment agencer les exécutions des tâches sur chaque processeur. Un ordonnancement est faisable si toutes les contraintes sont respectées (temporelles et ressources).[13]

1.8.2 Algorithmes d'ordonnancement

Un algorithme d'ordonnancement est une méthode ou stratégie utilisée pour ordonnancer les processus (tâches), Le choix d'un algorithme d'ordonnancement dépend essentiellement du contexte défini par les différentes caractéristiques du système temps réel sur lequel il sera mis en œuvre. Nous citons dans cette partie, quelques propriétés :[9] [14]

- **Séquence valide** : Un algorithme d'ordonnancement délivre une séquence d'ordonnancement pour une configuration de tâches données. Cette séquence est valide si toutes les tâches de la configuration respectent leurs contraintes.
- **Configuration ordonnançable** : Une configuration de tâches est ordonnançable dès qu'il existe un algorithme au moins capable de fournir une séquence valide pour cette configuration.
- **Algorithme optimal** : Un algorithme est optimal s'il est capable de produire

une séquence valide pour toute configuration de tâche ordonnançable.

1.8.3 Les types d'Algorithmes d'ordonnancement

En général, les ordonnanceurs sont classifiés selon des caractéristiques du système sur lequel ils sont implantés. On indique ci-dessous quelques types d'ordonnanceurs [15]

Ordonnancements statique et dynamique

Un ordonnancement est **statique** (prédictif), si l'ensemble des informations nécessaires à sa résolution est fixé au départ (ensembles des tâches, des contraintes, des ressources, etc.). À l'inverse, un ordonnancement **dynamique** (réactif), si le plan prévisionnel n'est pas respecté et les objectifs sont modifiés,et nécessite donc une résolution d'une série de problèmes statiques et chaque étape doit débuter par une prise d'informations permettant d'actualiser le modèle à résoudre.

Ordonnancement préemptif et non préemptif

Selon les problèmes, les tâche peuvent être interrompue par une ou plusieurs tâches au cours de son exécution, on parle alors d'un ordonnancement préemptif. Par contre, si les tâches sont exécutées sans interruption jusqu'à son achèvement, l'ordonnancement est appelé non préemptif.

Ordonnancements actifs et semi-actifs

Dans un ordonnancement actif, aucune tâche ne peut commencer au plus tôt et que entraîne l'ordre relatif entre au moins deux tâches. Dans l'ordonnancement semi actif, on ne peut pas avancer une tâche sans modifier la séquence sur la ressource.

Ordonnancements admissibles

Un ordonnancement est dit admissible s'il respecte toutes les contraintes du problème (dates de début, dates de fin, précédence, contraintes de ressources, etc.).

Ordonnancement hors ligne et en ligne

Un ordonnancement hors-ligne signifie que la séquence fixe d'exécution des tâches à partir de tous les paramètres de celles-ci. Cette séquence est rangée dans une table et

exécutée en ligne par un automate (séquenceur), un ordonnancement est en ligne si la séquence d'exécution des tâches est établie dynamiquement par l'ordonnanceur au cours de la vie de l'application en fonction des événements qui surviennent. L'ordonnanceur choisit la prochaine tâche à élire en fonction d'un critère de priorité. [14]

1.9 Algorithmes d'ordonnancement monoprocesseur

1.9.1 Ordonnancement de tâches périodiques indépendantes

Nous plaçons ici dans le cadre simple de l'ordonnancement monoprocesseur non oisif à priorité, lorsque toutes les tâches sont indépendantes (pas de précédence, pas de ressource). Les ordonnancements à priorités les plus courants sont les suivants :

Algorithmes à priorité fixe (statique)

Pour ce type d'algorithmes, la priorité d'une tâche est fixée avant son exécution et elle est la même pour toutes ses activations. Les algorithmes RM et DM décrits ci-dessous sont de ce type.

— **Rate Monotonic (RM)** C'est L'algorithme a été introduit par LIU et LAYLAND en 1973 . qui est basé sur la priorité fixe pour :[11]

✔ les tâches périodiques.

✔ Les tâches indépendantes (aucune communication entre les tâches).

✔ L'échéance de la tâche est considérée comme la fin de la période.

✔ Les tâches sont préemptibles..

❖ **Condition d'ordonnançabilité**

Une condition suffisante d'ordonnançabilité pour n tâches est :

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \geq n(2^{1/n} - 1) \tag{1.5}$$

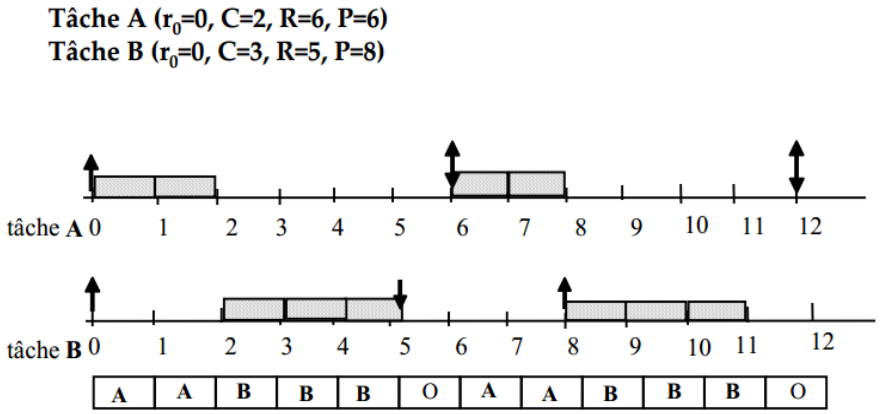

FIGURE 1.11 – Exemple d'ordonnancement RM.

— **Deadline Monotonic (DM)** L'algorithme Deadline Monotonic ou (DM) a été introduit par LEUNG et WHITEHEAD en 1982. Qui s'applique pour :

✔ les tâches périodiques

✔ Les tâches indépendantes (aucune communication entre les tâches).

✔ L'échéance de la tâche est inférieure ou égale à sa période. Les tâches sont préemptible.[16]

❖Condition d'ordonnançabilité

La condition suffisante d'ordonnançabilité de n tâches périodiques (triées par priorités décroissantes) $\tau_i, ..., \tau_i, ..., \tau_n$ avec DM basée sur la densité est :

$$U = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1 \tag{1.6}$$

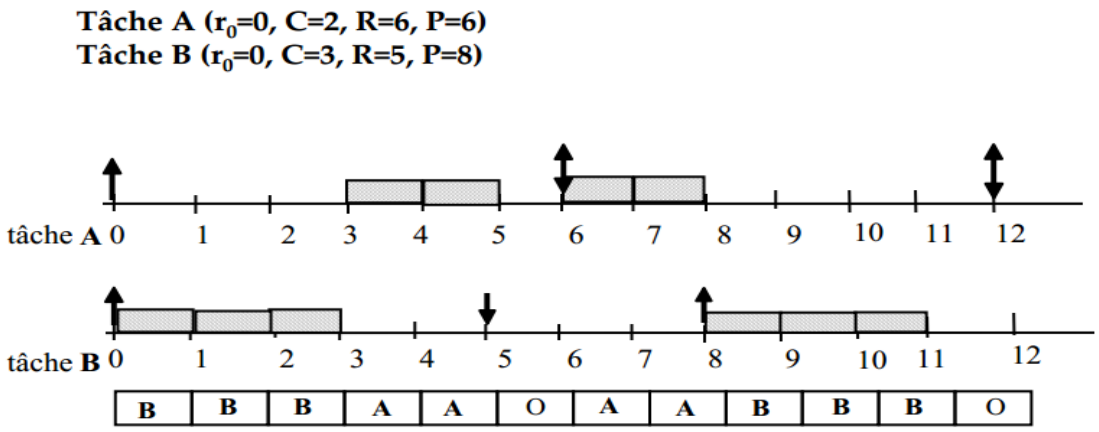

FIGURE 1.12 – Exemple d'ordonnancement DM.

Algorithmes à priorité variable (dynamique)

Pour cette catégorie d'algorithmes d'ordonnancement, la priorité d'une tâche est définie pour chaque activation de cette tâche

1. **Least Laxity First (LLF)**

   L'algorithme LLF a été introduit par Mok et Dertouzos.s'appuie sur un paramétré temporel dynamique dit Laxité et qui optimal pour l'ordonnancement de tâches indépendantes, et avec des échéances relatives inférieures ou égales aux périodes.[17]

   ❖ **Condition d'ordonnançabilité**

   Les conditions d'ordonnançabilité pour l'algorithme LLF sont les mêmes que pour EDF, c'est-à-dire que la condition nécessaire et suffisante d'ordonnançabilité dans le cas préemptif si $\forall i : D_i = T_i$ est :

   $$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{1.7}$$

2. **Earliest Deadline First (EDF)**

   Cet algorithme a été présenté par Jackson en 1955 [18] puis par Liu et Layland en 1973 [11]. C'est un algorithme qui attribue la priorité élevée à la tâche ayant l'échéance la plus proche ,et qui optimal relativement à la minimisation du retard maximal des tâches.

   ❖ **Condition d'ordonnançabilité**

   Pour un ensemble de tâches périodiques tel que $\forall i, (D_i = T_i)$, la condition nécessaire et suffisante de faisabilité est :

   $$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{1.8}$$

   Cette condition reste nécessaire et suffisante dans le cas où $\forall i, (D_i = T_i)$, n : Nombre de tâche

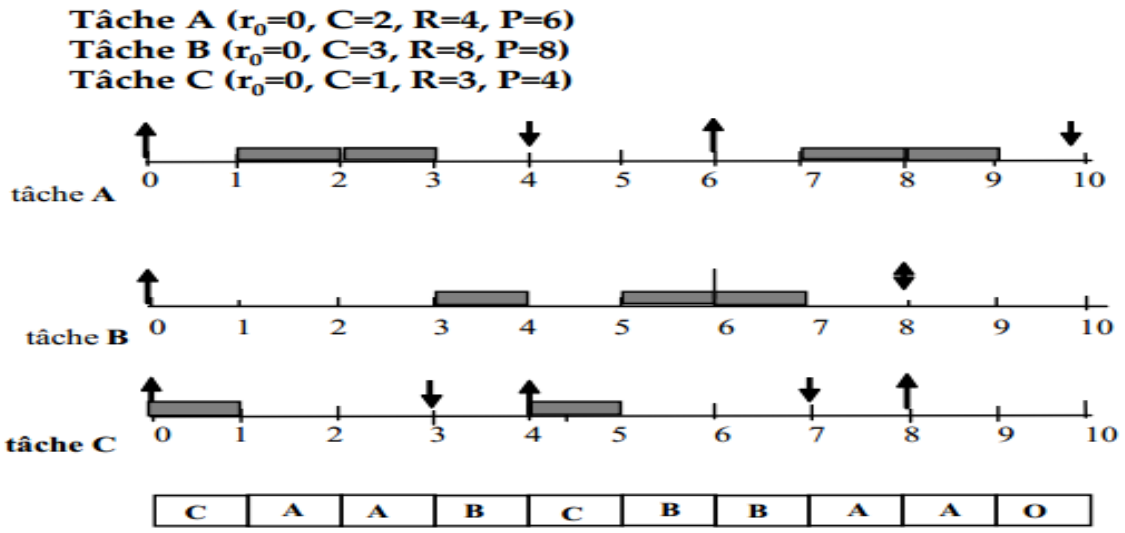

FIGURE 1.13 – Exemple d'ordonnancement EDF.

1.9.2 Ordonnancement des tâches indépendantes apériodiques

Traitement d'arrière-plan (background processing)

Cette méthode consiste à prendre en compte les tâches apériodiques et de les ordonnancer, c'est-à-dire lorsqu'il n'y a plus de tâches périodiques ou apériodiques acceptées précédemment à ordonnancer (le processeur est oisif). Si la charge (l'utilisation) des tâches périodiques est importante, l'ordonnanceur ne pourra pas consacrer beaucoup de temps aux tâches apériodiques et celles-ci risquent d'avoir un temps de réponse assez long. La figure 1.14 présente un exemple d'un tel ordonnancement sur un espace de temps égal à la période d'étude de la configuration périodique.[9]

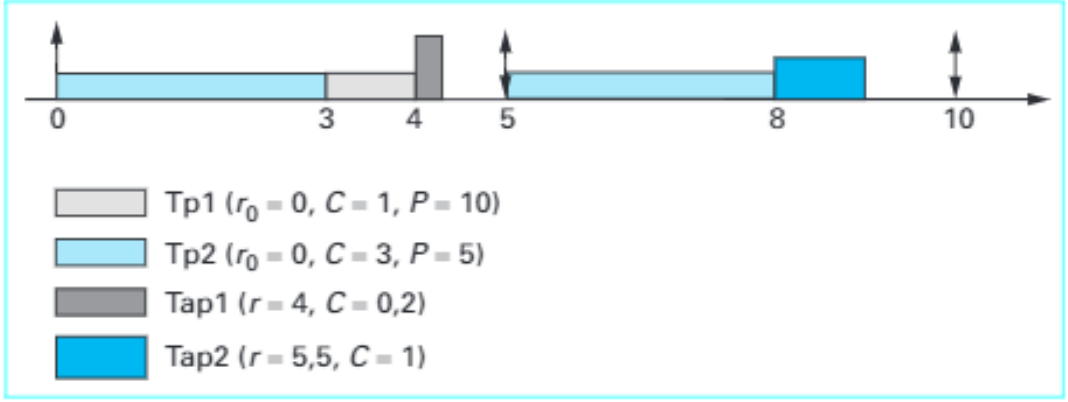

FIGURE 1.14 – Ordonnancement en arrière-plan.

Serveurs de tâches

L'ensemble des tâches périodiques est à échéance sur requête et est ordonnancé selon l'algorithme RM. Le serveur est une tâche périodique, généralement de la plus haute priorité, qui est créé pour ordonnancer les tâches apériodiques. On a plusieurs types de serveurs qui diffèrent par la manière dont ils répondent aux arrivées des tâches apériodiques :

— Traitement par scrutation (Polling).

— Serveur ajournable (Ajournable Server).

— Serveur sporadique (Sporadic Server).

1.9.3 Ordonnancement des tâches périodiques dépendantes

Ordonnancement de tâches liées par des contraintes de précédence

Un premier type de contraintes entre les tâches d'une application temps réel est représenté par la notion de précédence. On dit qu'il existe une contrainte de précédence entre la tâche Ti et la tâche Tj ou Tj précède Tj, si Tj doit attendre la fin d'exécution de Ti pour commencer sa propre exécution. Les précédences entre tâches d'une application peuvent être modélisées sous forme d'un graphe de précédence (voir le figure 1.15) tel qu'il existe un arc entre Ti et Tj si Ti précède Tj. L'objectif des algorithmes de prise en compte des relations de précédence est de transformer l'ensemble de tâches avec relations de précédence en un ensemble de tâches indépendantes, en intégrant explicitement ou implicitement les relations de précédence dans l'affectation de priorité. Ainsi, on obtient les règles de précédence suivantes :[9]

— Si Ti précède T_i, alors $r_j \geq r_i$;

— Si l'une des deux tâches est périodique, l'autre l'est obligatoirement et on a $P_i = P_j$; priorité (T_i) > priorité (T_j) dans le respect de la politique d'ordonnancement utilisée.

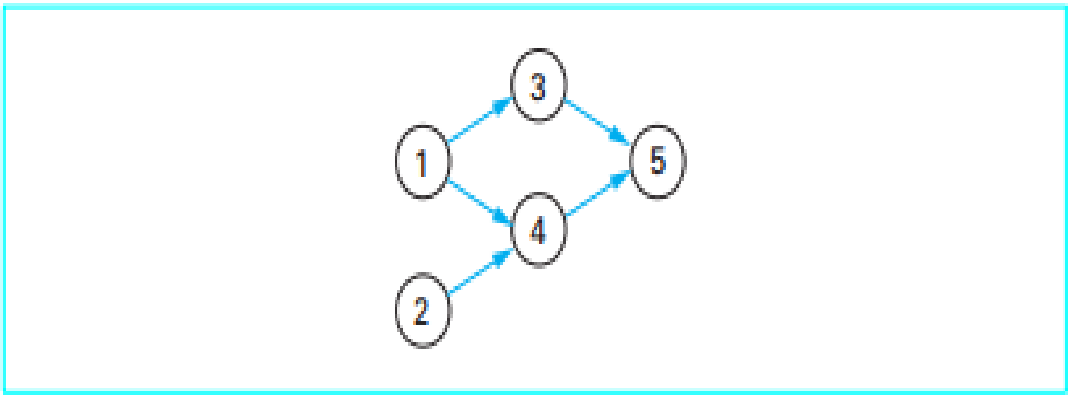

FIGURE 1.15 – Graphe de précédence.

Ordonnancement de tâches liées par des contraintes de ressources

L'ordonnancement de tâches avec contraintes de ressources pose différents problèmes ou phénomènes dans tout type de systèmes informatiques[9] :

—**L'inversion de priorité :** Ce phénomène au cours survient lorsqu'une tâche de haute priorité voulant accéder à la section critique associée à la ressource est retardée par des tâches de moindre priorité.

— **Des problèmes d'interblocage :**

Dans un système temps réel, la réservation et la préallocation des ressources sont des méthodes simples pour éviter ces problèmes,méthodes par lesquelles une tâche dispose dès le début de son exécution de toutes les ressources qui lui seront nécessaires au cours de celle-ci. Cependant, ces méthodes engendrent généralement un très faible taux d'utilisation des ressources ; aussi des protocoles spécifiques ont été développés pour permettre d'effectuer une allocation dynamique des ressources.

Afin de pallier à ces deux problèmes, des protocoles de gestion des ressources ont été mis au point pour fonctionner avec les algorithmes d'ordonnancement vus précédemment. On distingue principalement les techniques suivantes :

❖ **Le protocole à priorité héritée.** [19]

❖ **Le protocole à priorité plafond.** [19]

❖ **Le protocole d'allocation de la pile.** [20]

1.10 Ordonnancement multiprocesseur

Dans une architecture multiprocesseur, un algorithme d'ordonnancement est dit fiable si toutes les tâches de la configuration s'exécutent en respectant leur échéance. Cette définition, identique au cas monoprocesseur, s'étend avec les deux conditions suivantes :
— 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.

1.10.1 Problématique de l'Ordonnancement multiprocesseur

L'ordonnancement multiprocesseur et caractérisé par la présence de plusieurs processeur sur lesquels peuvent s'exécuter les tâche. Se pose alors les problèmes suivants :[21]

- 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 : un ressource ne doit pas être en mesure d'exécuter deux tâches en même temps qu'elle est exécutée sur des processeurs différents
- Le problème de l'ordonnancement des tâches : affectation des priorités.

1.10.2 Caractéristiques d'Ordonnancement multiprocesseur

Sur une plate-forme multiprocesseur, les applications disposent de plusieurs processeurs simultanément pour réaliser leur calcul. Il existe deux composants de plate-forme multiprocesseur : [22]
–**Plate-forme homogènes** sont constituées de plusieurs processeurs identiques ayant la même puissance de calcul
–**Plate-forme hétérogènes** sont constituées de plusieurs processeurs différent s'ayant par la même puissance de calcul

1.10.3 les Approches multiprocesseur

Pour résoudre le problème d'ordonnancement multiprocesseur. On distingue trois approches : partitionné pour lequel aucune migration n'est possible, par contre, globale pour

lequel aucune restriction ne porte sur les migrations, et semi-partitionné obtenue par combinaison entre stratégie globale et par partitionnement

➢ **Approche partitionnée :**

Dans un système partitionné, les tâches allouées aux processeurs ne sont pas autorisées à migrer d'un processeur à l'autre. Elles resteront sur le même processeur durant toute l'exécution du système. Une fois l'étape d'allocation terminée, les tâches sont ordonnancée à l'aide des algorithmes classique connus en environnement monoprocesseur, comme RM ou EDF. Pour les systèmes de tâches indépendantes, l'ordonnancement sur chaque processeur est totalement indépendant de ceux calculés sur les autres processeurs. Les résultats pour les tâches périodiques se généralisent donc, comme pour les systèmes monoprocesseur, aux tâches sporadiques.[23]

Le calcul du partitionnement des tâche pour allouer celles-ci les processeurs est une variante du célèbre problème d'optimisation combinatoire d'empaquetage qui est intensivement et utilisé comme problème canonique pour la mise au point de nouvelles techniques algorithme telles que les technique d'approximation ou d'augmentation de ressource. Le problème de partitionnement des tâches et NP-difficile, en conséquence il n'existe aucun algorithme polynomial pour calcul un partitionnement utilisant un nombre minimum de processeurs. Il est possible d'avoir de nombreuses variantes, en modifiant l'ordre de tri et / ou le critère d'acceptation :

— FBB-FFD (Fisher, Baruah and Baker - First Fit Decreasing)
— RMNF (Rate Monotonic First-Fit)
— RMFF (Rate MonotonicNext-Fit)
— RMBF (Rate MonotonicBest-Fit)
— RMWF (Rate MonotonicWorst-Fit)

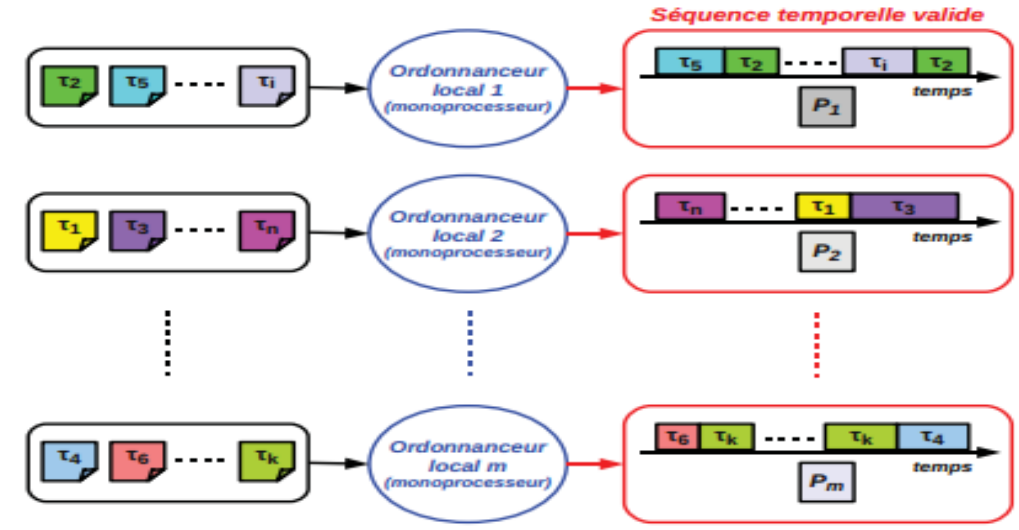

FIGURE 1.16 – Approche multiprocesseur partitionné.

➢ **Approche globale :**

Dans ce paradigme, aucune stratégie de placement (partition de tâches ou de travaux de tâches) n'est fixée a priori. Toutes les tâches sont dans la même file d'attente des tâches prêtes. Dans cette file d'attente les m tâches plus prioritaires sont sélectionnées sur les m processeurs de la plate-forme. Dans ce cas on autorise la migration (job-level migration), car celle-ci peut survenir à n'importe quel moment durant l'exécution des travaux. Donc une tâche peut commencer son exécution sur un processeur, être préemptée, puis reprendre sur un autre processeur. On parle alors d'ordonnancement global.

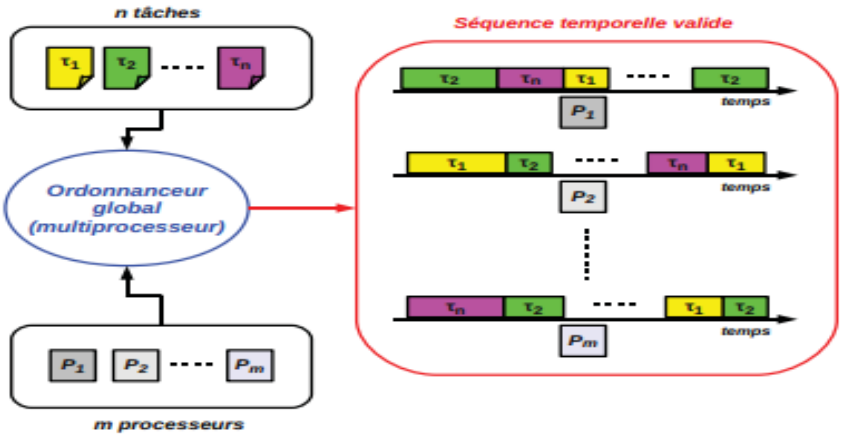

FIGURE 1.17 – Approche multiprocesseur globale.

➢ **Approche semi- partitionnée :**

Cette approche se base sur une approche partitionnée mais, lorsque le système n'est pas complètement partitionnable, certaines tâches sont autorisées à migrer. Il s'agit donc là d'introduire un peu de flexibilité dans les algorithmes partitionnés. On distingue deux sous-familles parmi ces algorithmes, on parle de restricted migration où la migration n'est possible qu'entre les travaux donc la migration est possible pendant l'exécution d'un travail. Par contre portioned scheduling, c'est-à-dire qu'une tâche est découpée en portion fixes, une portion est attribuée à un processeur et les portions sont exécutées à des dates garantissant l'absence de parallélisme en introduisant des échéances virtuelles fonction de l'échéance de la tâche.[24]

1.11 Conclusion :

Dans ce chapitre, nous avons proposé un état de l'art sur système temps réel (leurs définitions, caractéristiques, architecture, etc.). Par la suit, Nous avons évoqué les notions de base nécessaires à la comprendre du principe de l'ordonnancement temps réel et nous nous sommes intéressés à un système temps réel caractérisé par une architecture multi-processeur. Ainsi, nous avons constaté les approches d'ordonnancement multiprocesseur.

Chapitre 2

L'ordonnancement temps réel sous contrainte d'énergie

2.1 Introduction

L'ordonnancement en temps réel perceptif à l'énergie a fait l'objet de recherches intensives. La plupart des travaux visent à minimiser la consommation d'énergie ou à maximiser les performances du système telles que la durée de vie atteinte sous les contraintes énergétiques, les principaux travaux de recherche sur la minimisation de la consommation énergétique ont visé à exploiter la possibilité matérielle de changer les paramètres de fréquence et tension de courant à travers la proposition de nouvelles techniques appelées DVFS (Dynamic Voltage and Frequency Scaling) s'est particulièrement distinguée par sa grande efficacité à réduire la consommation processeur. L' autre connues sous le nom de DPM (Dynamic Power Management) qui visent à faire baisser la consommation d'énergie des circuits électroniques. Bien que les approches DPM et DVFS soient efficaces et souhaitables sur tout système embarqué, elles ne suffisent pas à satisfaire les contraintes spécifiques liées à la recharges de batterie. D'autre part, la grande majorité des applications, recharger ou remplacer une batterie se révèle délicat voire impossible. Par conséquent, l'avenir est à l'utilisation des sources d'énergie alternatives présentes dans l'environnement. Pourraient être exploitées afin de parvenir à une exploitation perpétuelle de ces systèmes : ceci est la récupération d'énergie. Cette approche permet d'étendre la durée de vie des batteries ou les éliminent complètement.

2.2 Stockage et récupération d'énergie

❖ **Système de récupération d'énergie :**

Le terme «récupération d'énergie» est un système qui récupère l'énergie ambiante et la stocke dans un réservoir d'énergie afin d'alimenter un appareil électronique, ce genre de système est composé de trois parties principales, (voir la figure 2.1) : [25]

— **Le récupérateur d'énergie (energy harvester) :** Le récupérateur d'énergie est la partie responsable de la collecte de l'énergie ambiante, il convertit différents types d'énergie environnementale (photoélectrique, mécanique, thermique, et des rayonnements électromagnétiques, etc.) en énergie électrique.

— **Unité de stockage d'énergie (energy storage unit) :** Stocke temporairement l'énergie collectée par le collecteur. Généralement, les batteries rechargeables ou les super-condensateurs sont utilisés pour cet effet en fonction des besoins de l'application.

—**Unité de calcul (le consommateur d'énergie) :** Est l'application embarquée ou la partie mission du système, elle est généralement composée de capteurs de données, d'une unité de traitement et d'un dispositif de transmission de données. La plupart des unités de calcul intégrées dans les systèmes de récupération d'énergie sont des systèmes en temps réel.

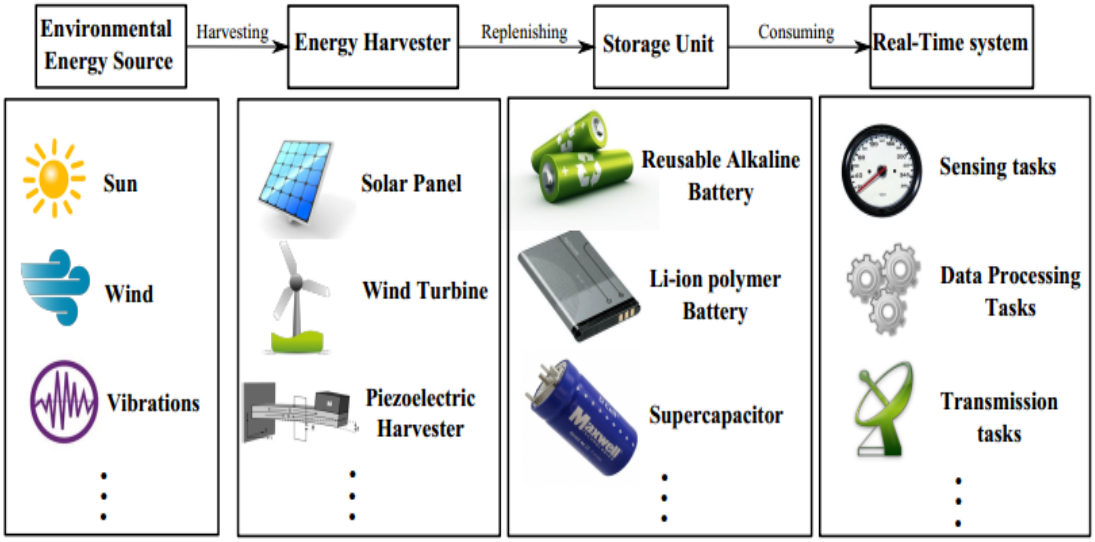

FIGURE 2.1 – Schéma d'un système de récupération d'énergie ambiante

❖ **Stockage d'énergie :** Nous considérons un support de stockage d'énergie idéal (super condensateur ou batterie) qui peut être chargé et déchargé avec des taux constants. Le niveau d'énergie dans le réservoir fluctue entre deux limites (voir la figure 2.2), le niveau minimal E_{min} qui assure la continuité de fonctionnement du système et le niveau maximal E_{max} qui ne peut être dépassé. La capacité effective du réservoir pour notre système temps réel notée C est la différence entre E_{max} et E_{min} ($C = E_{max} - E_{min}$).
Ces caractéristiques peuvent être satisfaites par les super-condensateurs qui permettent des taux de chargement et de déchargement constants et qui sont capables de supporter des cycles fréquents sans pour autant dégrader la capacité. Par souci de clarté, on peut considérer sans perte de généralité que $E_{min} = 0$ et que $C = E_{max}$. Le niveau de la batterie au temps t est noté $E(t)$.

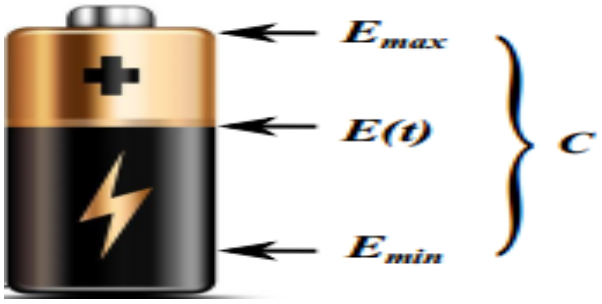

FIGURE 2.2 – Modèle de batterie rechargeable

2.2.1 Eléments de stockage d'énergie

Les matériaux de stockage d'énergie entrent dans plusieurs catégories (par exemple, le stockage sous forme d'énergie électrochimique et électromagnétique) selon ses mécanismes de fonctionnement. Avec la découverte de nouveaux matériaux et les progrès des techniques de fabrication, les processus de stockage d'énergie apparaissent dans un élément qui peut être une Batteries électrochimiques, pile à combustible, super-condensateur.

Batteries électrochimiques :

Le stockage de l'énergie dans les batteries électrochimiques est la technique la plus répandue pour les petites quantités d'énergie électrique. En fonction du type de batterie (plomb-acide, lithium-ion, etc.), différentes réactions chimiques sont provoquées à partir de l'électricité : Il s'agit de la phase de charge de la batterie. Selon la demande, les réactions

chimiques inversées produisent ensuite de l'électricité et déchargent le système. L'énergie électrique fournie par ces réactions électrochimiques est exprimé par en Watt Heure(Wh). L'inconvénient majeur est lié à leur durée de vie, limitée par les dégradations chimiques des réactions et leur coût .[26]

La pile à combustible :

Une pile à combustible est un générateur d'énergie électrique. Elle transforme directement l'énergie chimique du combustible(hydrogène, hydrocarbures, alcools, etc..) en énergie électrique. Via une réaction redox comme une pile classique.[27]

Super-condensateur :

Le super-condensateur se définit comme un condensateur électrochimique qui a une grande capacité de stockage d'énergie. Par rapport à Les batteries, qui caractérisées par une forte densité énergétique et une faible densité de puissance. Leur durée de vie en nombre de cycles charge/décharge est relativement limitée, donc le super-condensateur possède les qualités suivantes :[28]

— Le processus de charge/décharge est rapide ;
— Il peut délivrer des impulsions fréquentes d'énergie sans l'effet d'endommagement ;
— Sa vie est plus longue : il peut se charger/décharger jusqu'à des milliers fois ;
— La résistance interne est petite : son efficacité peut atteindre $84 - 95\%$;
— Il peut être chargé à n'importe quel niveau, sans effet mémoire ;
— La bande de température de fonctionnement est large, min.$-40^{\circ}C$;

2.2.2 Eléments de récupération d'énergie

❖Sources d'énergie renouvelables

Les sources d'énergie renouvelable sont principalement le soleil, le vent, la mer (courants marins, marées, vagues, gradient de température) et les cours d'eau, la biomasse, ainsi que la chaleur de la terre, hydraulique.
Ces sources renouvelables peuvent être converties en énergie sous diverses formes permettent de produire de l'énergie utile (électricité, chaleur, froid). Les énergies renouvelables sont des énergies de flux. Elles se régénèrent en permanence d'une valorisation qui ne restreint pas leur utilisation ultérieure. [29]

❖ **Energie solaire :**

L'énergie solaire repose sur l'exploitation directe de la lumière solaire. Le soleil est la source d'énergie la plus puissante et cette énergie est gratuite, les technologies sont réparties entre passives et actives. Les technologies actives transforment l'énergie solaire en une forme électrique ou thermique que nous pouvons utiliser directement. C'est le cas des cellules photovoltaïques qui transforment le rayonnement solaire en énergie électrique.

❖ **Energie éolienne :**

La force éolienne est connue et exploitée depuis des milliers d'années, comme les moulins à vent. Aujourd'hui, les éoliennes sont installées sur terre et en mer dans des endroits où le vent atteint une vitesse élevée et constante. Cette énergie pouvons exploiter à l'aide d'hélices spéciales qui emmagasinent le vent et de machines qui le transforment en énergie électrique.

❖ **Energie thermique :**

Est le processus de création d'énergie électrique à partir de la différence de température (gradients thermiques) à l'aide de générateurs d'énergie thermoélectrique (TEG). Ces thermos générateurs constitués d'un assemblage de jonctions, utilisent l'effet Seebeck pour convertir la différence de température entre deux milieux, en électricité [30]

❖ **Energie électromagnétique :**

Est basé sur la loi de Faradayen 1831 sur l'induction électromagnétique. L'aimant électromagnétique transforme l'énergie mécanique en énergie électrique à l'aide d'un système de masse à ressort inductif. Il déplace la tension en déplaçant une grande quantité de matériaux magnétiques à travers le champ magnétique généré par un aimant fixe. Plus précisément, la vibration magnétique reliée au ressort de la bobine modifie le flux magnétique à l'intérieur de la bobine et génère une tension enthousiaste.

2.3 L'ordonnancement temps réel et la consommation énergétique du processeur

2.3.1 Consommation énergétique d'un processeur :

Un processeur est un circuit intégré de la famille des CMOS (complementarymetal-oxide-semiconductor), un tel circuit répond aux équations génériques suivantes : [31]

L'énergie consommée dans un intervalle de temps [a, b] est par définition l'intégrale de la puissance dissipée :

$$E = \int_a^b P(t)dt \tag{2.1}$$

Où P(t) est la puissance dissipée à l'instant t. Cette puissance dissipée dans un circuit électroniques compose de la puissance statique et de la puissance dynamique :

$$P = P_{stat} + P_{dyn} \tag{2.2}$$

Où principale différence entre eux est que la puissance dynamique (P_{dyn}) dépend de la fréquence, tandis que la statique (P_{stat}) ne l'est pas. Dans les circuits CMOS la puissance dynamique représente 80-85 % de la puissance dissipée et, classiquement, on néglige la puissance statique. La puissance dissipée totale peut donc s'exprimer par :

$$P \approx P_{dyn} \sim \alpha f C V^2 \tag{2.3}$$

Où α est le nombre de transitions par cycle d'horloge, f est la fréquence de fonctionnement, C'est la capacité équivalente et V est la tension d'alimentation.
La fréquence de fonctionnement est donnée par :

$$f = \frac{(V - v_t)^\gamma}{V} \tag{2.4}$$

Où V_t est la tension de seuil qui est suffisamment petite par rapport à la tension d'alimentation et γ une constante. Donc la relation entre fréquence et tension devient : $f \sim v$

2.3.2 Technologie des processeurs

Nous distinguons deux classe de processeur en fonction de la possibilité ou non de changer la fréquence nominal de fonctionnement, l'efficacité des stratégies d'ordonnancement, sous contrainte d'énergie, sera naturellement dépendante de cette caractéristique du processeur.

Processeurs à vitesse constante et mode veille :

Les processeurs à vitesse constante opèrent à leur fréquence d'horloge et leur tension d'alimentation nominales et consomment donc la même quantité d'énergie à l'exécution.

Le plus souvent, ces processeurs possèdent au minimum deux modes de fonctionnement, le mode actif et le mode veille dans lequel aucune instruction n'est exécutée avec une consommation énergétique grandement réduite. Les techniques qui visent à sélectionner au mieux les modes de fonctionnement des ressources sont connues sous le terme Dynamic Power Management (DPM).

Processeurs à vitesse variable :

D'autres types de processeurs sont conçus pour l'économie d'énergie. Ces processeurs permettent de varier la tension d'alimentation et donc la fréquence de fonctionnement. Les stratégies d'adaptation dynamique de la tension sont connues sous le terme de Dynamic Voltage FrequancyScaling (DVFS). La technologie actuelle des processeurs implique nécessairement un nombre de fréquences fini. Parmi les processeurs à fréquence variable, on peut distinguer ceux qui permettent un changement de fréquence pendant l'exécution d'une application et ceux qui ne le permettent pas (nécessité de réinitialisation, temps de changement de fréquence trop importants, etc.)

2.3.3 Approches à visée de minimisation de la consommation énergétique

Pour étendre l'autonomie d'un système embarqué, et donc réduire sa consommation, tout en gardant ces performances optimales, il existe deux méthodes. La première méthode a entrainé de nombreuses recherches dans le domaine des batteries, il est toujours difficile d'augmenter la capacité des batteries sans en augmenter le poids, volume et le cout (on parle d'augmenter la quantité d'énergie embarquée). Dans ce dernier méthode, on parle d'un vise à diminuer la consommation énergétique du système, qui complémentaire à la première. Cette méthode utilise des techniques regroupées en trois catégories.

–Les techniques matérielles, qui visent à concevoir des composants consommant le minimum d'énergie.

–Les techniques logicielles, qui consistent à modifier le logiciel pour diminuer le coût énergétique de son exécution.

–Les techniques hybrides, qui consistent à faire collaborer le logiciel et le matériel. Dans ce cas, il existe deux façons pour réduire la consommation d'énergie des processeurs se basant sur cette collaboration. DPM et DVFS qui nous présentons dans section ci-dessous

avec des explications. [32]

✔La méthode DPM (Dynamic Power Management) :

La gestion dynamique de la consommation (DPM) qui permet de gérer dynamiquement l'activité du système en faisant des commutations d'un état de veille à un état actif et vice versa. En effet, les techniques DPM permettent de réduire la consommation de puissance du système sans dégrader considérablement les performances en basculant en mode veille lorsque qu'il n'y a aucune tâche à exécuter et de rebasculer en mode actif quand le processeur est sollicité. Ces techniques utilisent des processeurs qui disposent d'une fonction de mise en veille. Par conséquent, le processeur est éteint temporairement lorsque cela s'avère intéressant ou nécessaire. Dans ce cas, il ne consommera que peu ou pas d'énergie (appelée énergie statique) dans cet état de veille.[33]

✔La méthode DVFS (Dynamic Voltage and FrequencySelection) :

Variation de tension et de la fréquence du processeur appelée (DVFS) permettent de changer la fréquence du processeur quand cela est nécessaire. Ainsi, ces méthodes utilisent des processeurs conçus pour réduire l'énergie utilisée en variant la tension d'alimentation et donc la fréquence de fonctionnement. Par conséquent, si la fréquence du processeur est réduite, le job en exécution mettra plus de temps à s'exécuter.

2.4 Approches et algorithmes d'ordonnancement dans les systèmes récupérateurs d'énergie

Dans la littérature on identifie principalement deux approches pour répondre au besoin temps réel des systèmes récupérateur d'énergie. La première consiste la technique de gestion dynamique de la consommation appelée DPM (Dynamic Power Management, en anglais). La deuxième consiste à la technique d'adaptation dynamique de la tension d'alimentation et de la fréquence du processeur (DVFS). Cela passe par une sélection judicieuse des moments d'exécution et ceux de rechargement ou des périodes d'activité et celles d'inactivité afin de ne jamais tomber à court d'énergie et faire au mieux pour respecter toutes les échéances. Nous nous intéresseront en explorant les différents algorithmes proposés dans la littérature des systèmes temps réel récupérateurs d'énergie.

2.4.1 Les algorithmes basés sur la technique DPM

Dans cette sous-section nous présentons les algorithmes de la technique DPM.

EDeg (Earliest Deadline with energy guarantee) :

EDeg est proposé un algorithme de planification en temps réel appelé (Earliest Deadline withenergyguarantee) dans [34]. Chaque tâche est caractérisée par une consommation d'énergie en plus de son timing traditionnel paramètres. Selon EDeg, le processeur exécute des tâches dès que possible selon la règle EDF. Cependant, le système commence à exécuter une tâche uniquement si la soi-disant énergie lâche est positif et la batterie n'est pas vide. L'énergie lâche permet nous de quantifier l'énergie consommée par les emplois futurs et de les empêcher de violer leurs délais à cause de l'énergie pénurie. En outre, le système arrête son activité tant que le temps de relâchement est positif et la batterie n'est pas complètement rechargée. Les problèmes clés de cet algorithme sont de bien prédire la production d'énergie et la mesure du niveau d'énergie actuel la batterie. [35]

FBA (Frame-Based Algorithm) :

FBA est le premier algorithme proposé pour les systèmes récupérateurs d'énergie ambiante par (Allavena et Mossé). [25] Il est spécifique au modèle de tâches dit (Frame-Based, en anglais) où toutes les tâches partagent la même période et la même échéance. L'idée de l'algorithme est de diviser les tâches en deux groupes : Les tâches dites consommatrices qui ont un taux de consommation supérieur à celui du rechargement, et les tâches dites génératrices qui ont un taux de consommation inférieur ou égal. L'algorithme ordonnance alors en continue les tâches consommatrice pour vider la batterie puis les génératrices pour la recharger. Si l'énergie générée par les tâches génératrices n'est pas suffisante, une période de rechargement suffisamment longue est ajoutée au début du cycle. La même séquence d'ordonnancement est répétée à chaque période.

LSA (Lazy Scheduling Algorithm)

LSA a été introduit en 2006 par (Moser et al). De l'Institut Polytechnique de Zurich [36]. C'est un algorithme d'ordonnancement en ligne. II permet d'ordonnancer les tâches périodiques ou apériodiques critiques, c'est-à-dire munies d'une échéance stricte. A l'arrivée d'une tâche, l'ordonnanceur LSA lui calcule une date dite de démarrage, notée si à

partir de laquelle la tâche commencera à s'exécuter sur le processeur en utilisant sa puissance de consommation maximum E_{max} pendant toute son exécution. La détermination de cette date est faite de telle sorte que le processeur ne commence à l'exécuter ni trop tôt, ni trop tard. Entre la date d'arrivée et la date de démarrage. Si le processeur est laissé volontairement en veille pour permettre au réservoir de se charger. Cette intervalle de temps de recharge est calculé tel que, lorsque la tâche commencera à s'exécuter, le processeur disposera de suffisamment d'énergie pour fonctionner de façon continue jusqu'à la date d'échéance de la tâche. Cet algorithme a été présenté comme étant optimal mais uniquement dans le cas où les tâches consomment l'énergie avec le même taux et que le temps d'exécution des tâches varie en fonction du taux de consommation appliqué .

2.4.2 Les algorithmes basés sur la technique DVFS

Dans cette sous-section nous présentons les algorithmes de la technique DVFS

Energy-aware DVFS Algorithm (EA-DVFS) :

L'algorithme EA-DVFS a été proposé dans (Liu et al.2008) .[37] Cette L'algorithme proposé ralentit l'exécution de la tâche si le système n'a pas suffisamment d'énergie disponible ; sinon, les tâches sont exécutées à pleine vitesse. Les principales lacunes de ce travail sont :

— «**L'énergie disponible suffisante**» est définie sur la base d'une seule tâche courante. Tant que le temps de fonctionnement restant du système à pleine vitesse est plus que le délai relatif de la tâche,le système considère alors qu'il a suffisamment d'énergie. Cependant,il peut ne rester que 1% d'énergie dans le stockage d'énergie tandis que le système peut fonctionner à pleine vitesse pendant plus délai relatif d'une tâche. Ensuite, les programmes de l'algorithme EA-DVFS la tâche à pleine vitesse. Ce n'est pas le comportement souhaité.

— Lorsque les tâches sont planifiées et que les tensions de fonctionnement sont sélectionnées, l'algorithme EA-DVFS ne considère qu'une seule tâche au lieu de tenant compte de toutes les tâches dans la file d'attente des tâches prêtes. Il en résulte que les tâches ne sont pas pleinement exploitées pour réaliser des économies d'énergie

.

Harvesting-aware DVFS algorithm (HA-DVFS) :

L'algorithme HA-DVFS (Harvesting-awareDynamic Voltage and FrequencySelectionAlgorithm) est proposé dans (Liu et al. 2012) [30] pour améliorer encore les performances du système et l'efficacité énergétique des EA-DVFS et AS-DVFS. En particulier, les objectifs principaux d'HA-DVFS sont de maintenir la vitesse d'exécution des tâches toujours à la valeur la plus basse possible et d'éviter de gaspiller l'énergie récoltée. Sur la base de la prévision du taux de récupération d'énergie dans un avenir proche, HA-DVFS planifie les tâches et ajuste la vitesse et la charge de travail du système pour éviter les débordements d'énergie. Trois techniques différentes de prévision des séries chronologiques, à savoir l'analyse de régression, la moyenne mobile et le lissage exponentiel, sont utilisées pour prédire l'énergie récoltée. Semblable à AS-DVFS, HA-DVFS dissocie les contraintes d'énergie et les contraintes de temps pour réduire la complexité de l'algorithme de programmation.

Adaptive Scheduling DVFS algorithm (AS-DVFS) :

L'algorithme AS-DVFS (Adaptive Schedulingand DVFS algorithm). AS-DVFS ajuste de manière adaptative la tension et la fréquence de fonctionnement d'un processeur de nœud dans la mesure du possible tout en respectant les contraintes de temps et d'énergie. Le but d'AS-DVFS est d'atteindre une efficacité énergétique à l'échelle du système en planifiant et en exécutant les tâches à la vitesse la plus basse possible et en allouant la charge de travail au processeur aussi uniformément que possible. De plus, il dissocie les contraintes de temps et d'énergie, en les traitants séparément

L'algorithme EDfbs-eg

L'algorithme EDfbs-eg (Earliest Deadline Feedback Scheduling with energy guarantee) est proposé dans [38]. Il s'agit de l'application du principe de l'ordonnancement temps réel régulé au contexte des systèmes récupérateurs d'énergie environnante. D'autre algorithmes ont été proposé dans la literature comme FS-EH [39] et FSCE-EH [40]

2.5 Conclusion

Il existe actuellement plusieurs travaux qui ont été proposé ces dernières années pour réduire massivement la consommation d'énergie avec des technologies innovantes. A tra-

vers le réglage de la vitesse du processeur, soit à travers hors tension l'ensemble ou une partie des circuits électroniques. Malgré ces améliorations technologiques, prendre en compte conjointement les aspects temporels des applications temps réel et les aspects énergétiques des systèmes autonomes alimentés par une source d'énergie renouvelable.

Chapitre 3

L'Ordonnancement temps réel régulé

3.1 Introduction

L'automatique et l'informatique temps réel sont depuis longtemps associés dans des systèmes de commande dans le but de contrôler un procédé et assurer un haut degré d'efficacité et d'offrir une multitude d'avantages comme l'automatisation de tous les processus. Le principe général de commande des procédés est connu sous le terme de commande en boucle fermée ou commande par rétroaction, il existe plusieurs méthodes de contrôleur (Contrôleur PID, Commande Prédictive, Commande robuste...) parmi ces méthode, on a la commande floue qui permet de faire le lien entre modélisation numérique et la modélisation symbolique.

3.2 Stratégies de commande des procédés

Le système temps réel est associé dans des systèmes de commande dans le but de contrôler un processus pour l'amener dans un état conforme aux besoins de l'utilisateur. Il peut être utile en régulation, pour maintenir à une valeur constante la sortie du procédé contrôlé, par exemple la vitesse d'un véhicule asservie par un régulateur de vitesse. Il peut être utilisé pour poursuivre un objectif variant dans le temps, exemple l'asservissement de l'orientation d'une antenne radar pour maintenir l'objet observé dans la zone d'activités de l'antenne. Ainsi, pour réaliser cette commande, deux approches de stratégies sont envisageables

— Commande par boucle ouverte

— Commande par boucle fermée

3.2.1 Commande par boucle ouverte

La commande par boucle ouverte ne peut être mise en œuvre qui si l'on connait la loi régissant le fonctionnement du processus, en d'autres termes, il est nécessaire de connaitre la corrélation entre la valeur mesurée et la grandeur réglant. On dit que le système est en boucle ouverte lorsque la commande et élaborée sans connaitre les grandeurs de sortie, les inconvénients imposer par le processus, c'est qu'il n'a aucun moyen de contrôler, à fortiori de compenser les erreurs, les accidents qui peuvent intervenir à l'intérieur de la boucle,les dérives et le système en boucle ne compense pas les signaux de perturbation .[41]

3.2.2 Commande par boucle fermée

Un contrôle par boucle fermée (voir la figure 3.1) est une forme de contrôle d'un système qui intègre la réaction. Les actionneurs sont envoyée par la commande, exerce une influence sur la grandeur réglée, de sorte à la maintenir dans des directives fixées, même en présence de perturbation issues de l'extérieure. La rétroaction atténue considérablement l'effet d'une perturbation externe sur le système bouclé selon sa transmittance caractéristique d'un filtre. Autrement, la rétroaction permet de rendre le système stable.

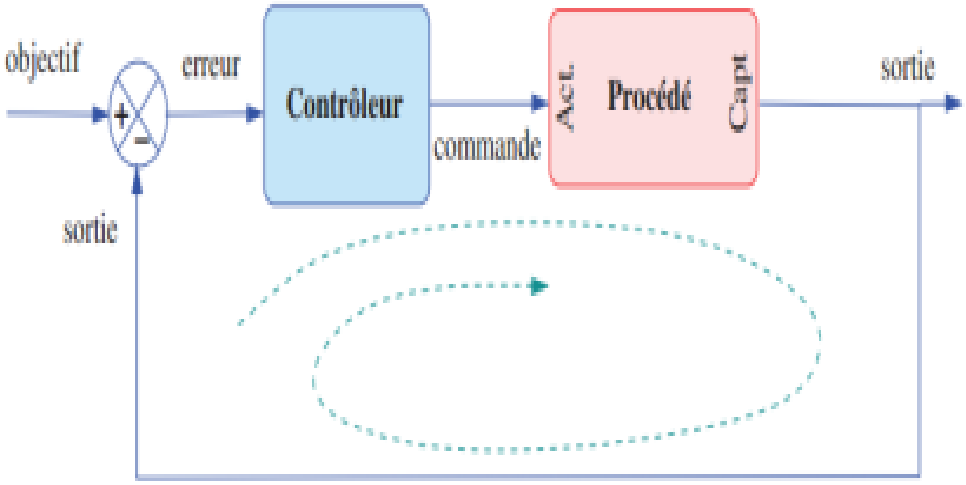

FIGURE 3.1 – Principe de commande par rétroaction

3.3 Paramètres et méthodes d'évaluation des systèmes de commande

3.3.1 Paramètres d'un système de commande

La structure d'un système de commande est représentée sur la figure 3.2 le système à commander, le capteur et l'actionneur qui est connecté au contrôleur. L'exécution du contrôleur est composée de 3 phases : L'acquisition des mesures, le calcul de la nouvelle commande et la mise à jour de l'actionneur.[42]

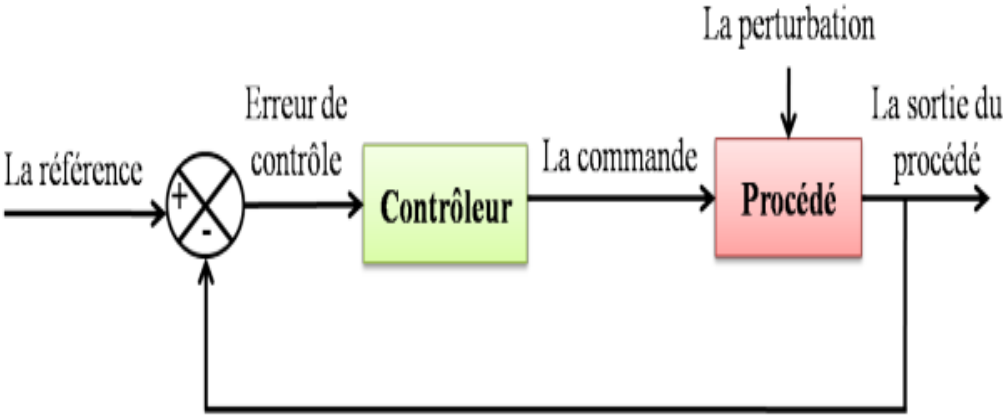

FIGURE 3.2 – La structure d'un système de commande

Les paramètres essentiels du système de commande sont représentés sur la figure 3.2 . Ces paramètres sont :

- La référence $x(t)$: représente les consignes de l'opérateur au système de contrôle.
- La sortie du procédé $y(t)$: est une caractéristique mesurable du procédé.
- Erreur de contrôle e(t) : est la différence entre la référence $x(t)$ et la sortie $y(t)$, Noté : $e(k) = x(t) - y(t)$.
- La commande $u(k)$; c'est un paramètre qui influe sur le comportement du procédé .
- La perturbation : est une modification qui affecte la manière dont la commande influence la sortie du procédé

3.3.2 Les critères de performances du système de commande

Les critères de performance d'un système de commande sont basés soit sur la valeur de la commande et de la sortie du procédé. Cependant, il existe D'autres métriques sont utilisées pour mesurer la performance du contrôle, on a recours à différentes caractéristiques de la sortie du processus contrôlé. Celles-ci sont typiquement définies dans le domaine temporel sur la réponse. La figure 3.3 ci-dessus illustre ces caractéristiques sur la réponse indicielle d'un processus.

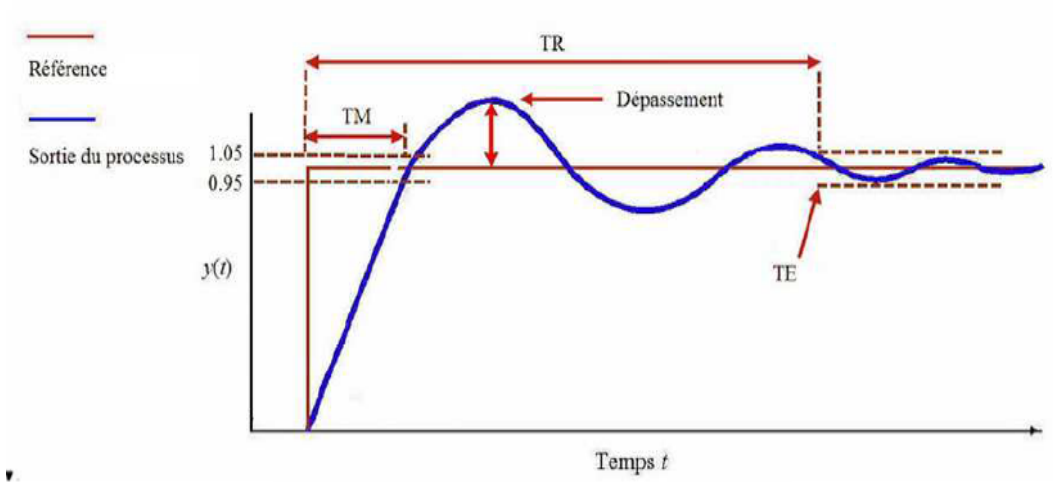

FIGURE 3.3 – Caractéristiques temporelles de la réponse d'un processus

M : Dépassement, exprime à quelle mesure la sortie $y(t)$ va dépasser sa référence $x(t)$ avant de l'atteindre. Ce dépassement est calculé comme suit :

$$M = \max_{t \geq 0} \frac{y(t) - x}{x} \times 100\% \tag{3.1}$$

TR : Temps de réponse, en fonction de la précision exigée, le temps de réponse du procédé est le temps au bout duquel la sortie du processus pénètre dans le couloir de plus ou moins 5% de la référence sans en sortir.

TM : Temps de montée, c'est le temps nécessaire à la sortie du processus pour atteindre un certain pourcentage de sa référence (par exemple atteindre 95%). Notons qu'on définit également, dans certains ouvrages, comme le temps où la sortie passe de 10% à 90% de la référence.

TE : Temps d'établissement, c'est temps au-delà duquel l'écart entre $y(t)$ et la référence $x(t)$ est borné par une certaine valeur, c-à-d tel que $y(t)$ ne sorte plus d'un certain couloir centré sur la référence. Cet écart est exprimé en %.

3.4 Les méthodes d'évaluation du système de commande

Les méthodes d'évaluation des systèmes de commande sont classées en trois catégories :[43]

Simulation Dépassement (M) : Tant le système temps réel que les procédés asservis sont simulés par logiciel, par exemple à l'aide de l'outil TrueTime.

Hardware-in-the-Loop : Le système de commande est implémenté sur un vrai système temps réel sur une plate-forme matérielle ; les procédés sont toutefois simulés au sein de ce dernier.

Implantation : L'ensemble est implanté dans une vraie application de régulation embarquée. La très grande majorité des approches étudiées n'ont été évaluées que par simulation.

3.5 Problématique d'ordonnancement basé sur WCET

Actuellement, les systèmes de commande, comme les commandes de vols ou de freinages, sont généralement considérés, dès la phase de conception, comme des systèmes temps-réel dur où les tâches logicielles, devront s'exécuter de manière strictement périodique. Ainsi, des créneaux de temps préfixés sont alloués aux tâches de contrôle, et les dépassements d'échéances et le phénomène de gigue sont interdits. On considère que toute déviation par rapport cet ordonnancement idéal est une panne du système. La connaissance supposée sure des WCET est utilisée pour dimensionner le système. Cependant, la perte en temps processeur est d'autant plus importante que le WCET s'écarte de la valeur moyenne des temps d'exécution observés. En effet, les hautes performances des nœuds de calculs modernes sont dues à l'introduction de mécanismes architecturaux tels que les pipelines, les mécanismes d'exécution spéculatifs et l'utilisation des mémoires caches à plusieurs niveaux qui réduisent le déterminisme temporel des processeurs. L'estimation du WCET devient de plus en plus difficile, et l'étalement de la distribution des temps d'exécution rend l'approche pire cas encore plus conservatrice. La distribution du temps d'exécution, (voir la figure 3.4) pour un processeur embarqué montre qu'une exécution proche du WCET est un événement plutôt rare et qu'un ordonnancement basé sur une occupation moyenne du CPU plutôt que sur le pire cas donnerait des gains substantiels en termes de dimension-

nement des équipements embarqués.[44]

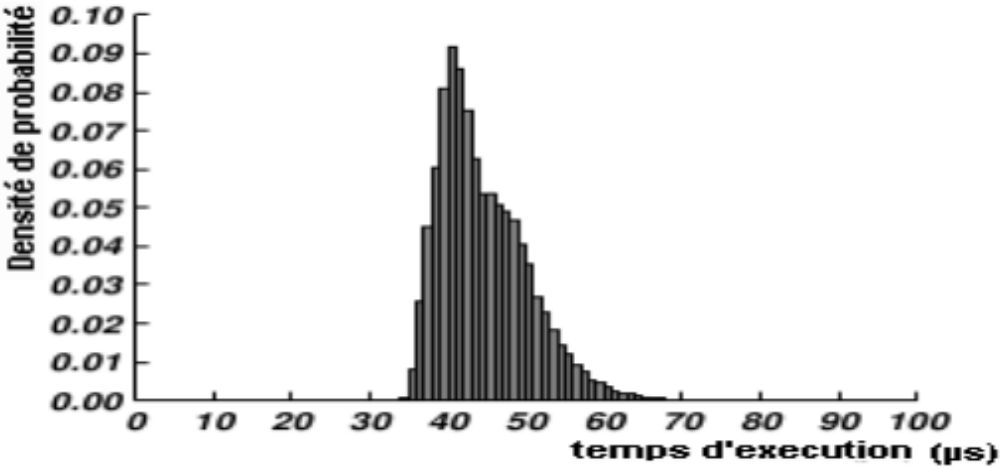

FIGURE 3.4 – La Distribution caractéristique du temps d'exécution

La validité de l'hypothèse habituelle de temps réel (dur) mérite d'être réexaminée soigneusement en tenant compte de la robustesse des systèmes de commande par rétroaction.

3.6 Ordonnancement temps réel régulé

3.6.1 Définitions

Ordonnancement temps réel régulé consiste à adapter en temps réel l'ordonnancement grâce a appliquer le principe de régulation a boucle fermée. Dans ce cas, l'ordonnanceur avec le système de tâches sont vus comme étant des procédés à contrôler. L'objectif est d'ajuster les paramètres d'ordonnancement du système en fonction de certaines mesures. Cette approche permet d'optimiser les performances de la régulation.[45]

3.6.2 Les éléments constitutifs d'une boucle de régulation d'ordonnancement temps réel

Comme Pour construire un régulateur d'ordonnancement à partir d'objectifs spécifiés, il faut définir quelles sont les variables à mesurer, ses sorties (actionneurs) et quelles sont les lois de commande (contrôleurs) utilisables sur le procédé pour agir d'une manière efficace sur celui-ci. Dans ce qui suit on va détailler les éléments qui peuvent être utilisés.

3.6.3 Consignes

Le régulateur agira sur base d'une grandeur de référence. La consigne utilisée dans l'ensemble des approches étudiées dans ce travail est l'utilisation désirée Uref (Utilisation désirée du processeur) du système (ou plus exactement, l'utilisation désirée des tâches sur laquelle le gestionnaire peut agir), et elle sera bien sûr choisie inférieure à 1 de sorte à ne pas pousser expressément le système en surcharge. Il faut également tenir compte des temps d'exécution du gestionnaire qui doit aussi occuper le processeur. [43] En général dans les approches étudiées, la consigne est choisie selon l'ordonnanceur en aval. En effet, il est en pratique désiré, plutôt que de seulement éviter des surcharges, d'également éviter au mieux les violations d'échéances. Par exemple, on se base sur les bornes supérieures des conditions d'ordonnancement existantes en aval comme 100% dans le cas d'EDF et 69% dans le cas de RM pour choisir la référence

3.6.4 Capteurs et mesures :

Les capteurs sont des éléments, transformant la grandeur à mesurer en un signal,représentatif de l'information originelle. L'état de la ressource d'exécution peut être obtenu au travers de différentes mesures.[46]

- **Charige CPU :** Un des buts principaux d'un régulateur d'ordonnancement est de gérer la charge de calcul d'une ressource d'exécution informatique, il est donc presque toujours nécessaire d'évaluer la charge de travail de la ressource.
- **Le dépassement d'échéances :** Sont des évènements faciles à détecter. Ils peuvent cependant être utilisés en complément de l'estimation de charge.
- **La laxité des tâches** (temps restant entre la fin d'exécution d'une instance et l'instant d'activation suivant) peut aussi être envisagée comme indicateur de charge du système.
- **La charge de la batterie** ou l'énergie disponible dans l'unité de stockage

3.6.5 Actionneurs et commandes :

Les actionneurs ou les commandes sont les paramètres de l'ordonnancement, agissant sur l'exécution des tâches considérées, y compris celle concernant la gestion du processeur. Ainsi, le régulateur d'ordonnancement pourrait manipuler les caractéristiques suivantes :

- **Les périodes des tâches :** La charge CPU induite par n tâches de durée ci et de période hi est donnée par $U = \sum_{i=1}^{n} \frac{c_i}{h_i}$ on voit immédiatement que les périodes sont des actionneurs efficaces pour agir sur la charge globale de calcul.
- **La priorité des tâches :** L'ordre des priorités n'affecte pas la charge de calcul mais l'entrelacement de calcul, et donc les latences mesure / commande. Les priorités doivent aussi refléter l'urgence et l'importance relative des composants sur la performance du contrôleur.
- **La tension et la fréquence du processeur :** Les actionneurs la variation de la fréquence du processeur peut être utilisée comme actionneur, à travers la méthode DVFS, pour modifier la charge du processeur ou réduire sa consommation.

3.6.6 Lois de commande

La loi de commande, c'est à dire l'algorithme qui calcule la nouvelle commande en fonction des mesures et des références, est conçue par l'usage d'une des nombreuses méthodes de synthèse de contrôleur suivante :

- ❖ **Contrôleur PID (Proportionnel-Intégral-Dérivé) :** Le contrôleur PID (pour Proportionnel - Intégral - Dérivé) est un type de pilotage à rétroaction et il est très largement utilisé dans de nombreuses applications industrielles, vu sa simplicité, et son adaptation aux systèmes du type SISO(Single-Input-Single-Output)[42] .
- ❖ **Commande Prédictive :** La commande prédictive a joué un rôle très important dans le domaine de contrôle de processus, elle est basée sur l'utilisation d'un modèle pour prédire le comportement futur du système sur un horizon de temps fini. Une séquence optimale des signaux de commande sur l'horizon de prédiction est obtenue par la minimisation d'un certain coût, le premier signal de la séquence de commande est transmis au processus et l'opération entière de « prédiction-optimisation » est répétée à chaque période d'échantillonnage [47].
- ❖ **Commande robuste :** La commande robuste est une première technique de commande de l'automatique traitant la difficulté d'obtention d'un modèle exact du procédé. Dans la synthèse de la loi de commande sont pris en compte un modèle nominal du procédé à contrôler mais aussi les incertitudes paramétriques liées au modèle. La structure du contrôleur robuste est finalement composée d'une partie «nominale» mais aussi de termes additionnels permettant de compenser au mieux

les incertitudes liées au modèle. [47]

3.7 Commande floue

La logique floue est une théorie connue depuis que le professeur Lotfi A.Zadeh a introduit le concept des sous-ensembles flous en 1965. Par la suite, en 1974, Mamdani introduisait la commande floue pour la régulation de processus industriel [48]. Dans les années 80s, la commande floue connait un essor considérable au Japon, notamment grâce aux travaux de Sugeno [49].
La logique floue permet de faire le lien entre la modélisation numérique et la modélisation symbolique, ce qui a permis des développements industriels spectaculaires à partir des algorithmes très simples de traduction de connaissances symboliques en entités numériques et inversement. Elle présente en effet l'avantage d'utiliser des règles linguistiques simples permettant de traduire facilement le savoir-faire d'un expert pour répondre à une problématique spécifique, « C'est de prendre en compte les états intermédiaires entre le tout et le rien » [50].

3.8 Les concepts

Le concept de la théorie des sous-ensembles flous, s'appuie sur la notion de degré d'appartenance d'un élément à un sous-ensemble flou. Tandis que les ensembles traditionnels sont caractérisés par une fonction d'appartenance notée c , (également appelée fonction caractéristique) définie sur $0, 1$, les sous-ensembles flous sont, eux, caractérisés par une fonction d'appartenance notée μ définie sur $[0, 1]$. En d'autres termes, dans le langage ensembliste classique, un élément appartient ou n'appartient pas à un ensemble tandis qu'un élément appartient à un sous-ensemble flou avec un certain degré (éventuellement nul). En résumé, pour un sous-ensemble A défini sur un univers de discours U, on peut écrire :
A Sous-ensemble classique : fonction caractéristique $C_A : U \rightarrow 0, 1$
A Sous-ensemble flou : fonction d'appartenance $U \rightarrow [0, 1]$

Nous allons maintenant définir un certain nombre de termes propres au domaine de la logique floue auxquels nous pourrons nous référer [51] .

3.8.1 Sous-ensemble flou

Le concept de sous-ensemble flou a été introduit pour éviter les passages brusques d'une classe à une autre (de la classe noire à la classe blanche par exemple) et autoriser des éléments à n'appartenir complètement ni à l'une ni à l'autre (à être gris, par exemple) ou encore à appartenir partiellement à chacune (avec un fort degré à la classe noire et un faible degré à la classe blanche dans le cas du gris foncé). Nous venons de voir ce que l'on entend par sous-ensemble ou, d'un point de vue formel. Un sous-ensemble où A sur un univers de discours U, est représenté comme dans la figure 3.5 à travers sa fonction caractéristique . Il peut également être décrit par un certain nombre de caractéristiques comme :

• **Son support :**

$$Support(A) = x \in U/\mu_A(x) \neq 0 \tag{3.2}$$

• **Sa hauteur :**

$$Hauteur(A) = \sup U_A(x) \tag{3.3}$$

• **Noyau :**

$$Noyau(A) = x \in U/\mu_A(x) = 0 \tag{3.4}$$

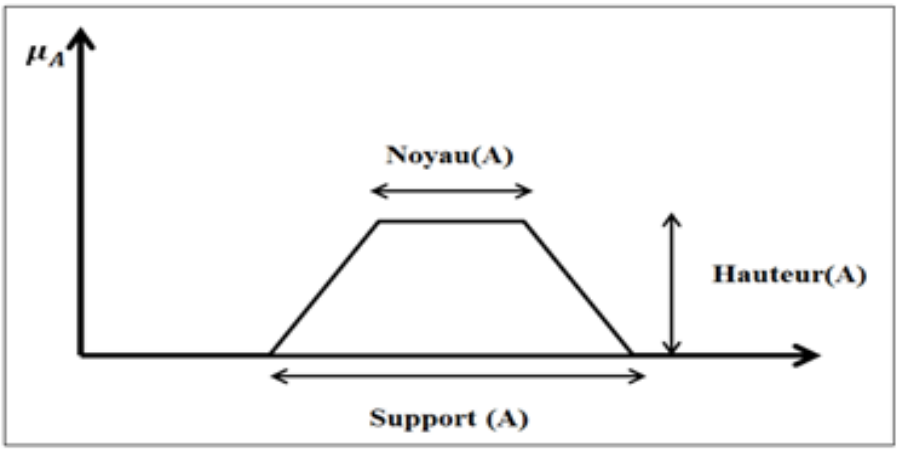

FIGURE 3.5 – Représentation d'un sou ensemble floue et principales caractéristiques

La notion de sous-ensemble flou permet de traiter :

• Des catégories aux limites mal définies (comme « centre-ville » ou « ancien »),

• Des situations intermédiaires entre le tout et le rien (« presque noir »),

• De passage progressif d'une propriété à une autre (de « proche » à « éloigné » selon la distance),

• Des valeurs approximatives (« environ 2 km »),

• Des classes en évitant l'utilisation arbitraire de limites rigides (il est difficile de dire qu'une maison située à 200 m de la plage en est proche, mais qu'à 210 m d'elle en est éloignée).

3.8.2 Variable linguistique

Une variable linguistique sert à modéliser les connaissances imprécises ou vagues sur une variable dont la valeur précise peut être inconnue. Elle est définie par un triplé $(V; U; Tv)$ où V représente une variable classique (âge, température,. . .) définie sur l'univers de discours U. T_v est l'ensemble des instanciations possible de la variable V : Il s'agit de sous-ensembles flous repérés par leur label A_i : On écrit ainsi $T_v = A_i, A_2.....A_n$ Graphiquement,une variable linguistique peut être représente comme dans la figure 3.6

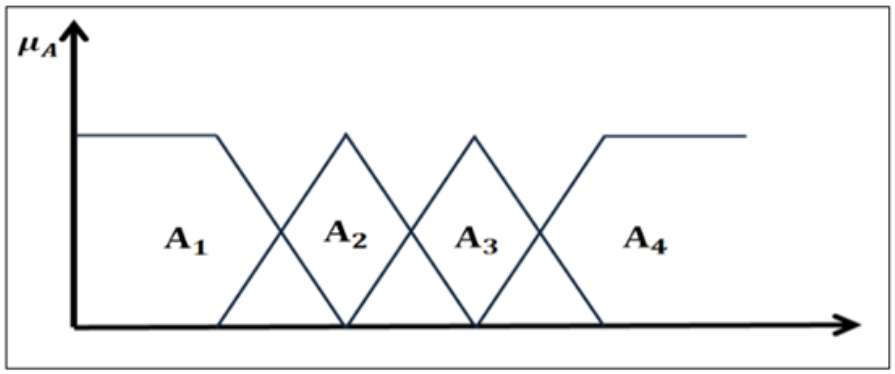

FIGURE 3.6 – Représentation d'une Variable linguistique

3.8.3 Opérateurs de la logique floue

Les variables linguistiques sont liées entre elles au niveau des règles d'inférence par des opérateurs ET ou OU. On définit les opérateurs de la logique floue qui interviennent sur les fonctions d'appartenance représentant les variables linguistiques.

❖ **Opérateur OU (union)**

La fonction d'appartenance $\mu_{A\cup B}$ de deux ensembles A et B est définie pour tout $\mu \in u$

➢ Mamdani :

$$\mu_{A\cup B}(u) = \max\{\mu_A(u), \mu_B(u)\} \tag{3.5}$$

➢ Sugeno :

$$\mu_{A\cup B}(u) = \mu_A(u) + \mu_B(u) - \mu_A * \mu_B \tag{3.6}$$

❖ **Opérateur ET (l'intersection)**

La fonction d'appartenance $\mu_{A\cap B}$ de deux ensembles A et B est définie pour tout $\mu \in u$

➢ Mamdani :

$$\mu_{A\cap B}(u) = \min\{\mu_A(u), \mu_B(u)\} \tag{3.7}$$

➢ Sugeno :

$$\mu_{A\cap B}(u) = \mu_A * \mu_B \tag{3.8}$$

❖ **Opérateurs NON (complémentation)**

La fonction d'appartenance μ_A d'un ensemble A est définit, pour tout $\mu \in u$

$$\mu_{A'} = 1 - \mu_A \tag{3.9}$$

3.8.4 fonction d'appartenance

On présente les variables linguistiques par leurs fonctions d'appartenances. Alors à chaque sous-ensemble flou est associé une fonction d'appartenance ou est la variable linguistique. Tel que, à chaque point est associé une valeur précise de qui désigne, le degré d'appartenance de à La fonction d'appartenance peut étre présentée par plusieurs forme(riangulaire,trapézoidale , gussienne). On peut définir d'autres formes de fonctions d'appartenance, mais dans le réglage par logique flous, les formes déjà citées et illustrées sur la figure 3.7 sont largement suffisantes pour délimiter les ensembles flous, on définit ensuite (a) Fonction triangulaire (b) Fonction trapézoïdale. (c) Fonction gaussienne.

❖**Fonction triangulaire** [52]

Elle est définie par trois paramaitres a, b, c , qui déterminent les cordonnées des trois sommets.

$$\mu_x = \max\left(\min\left(\frac{x-a}{b-a}\right), \frac{c-x}{c-d}\right) \tag{3.10}$$

❖**Fonctions trapézoidale**[52] Elle est définie par quatre paramètre a, b, c, d

$$\mu_A = \max\left(\min\left(\frac{x-a}{b-a}\right), 1, \frac{c-x}{c-d}, 0\right) \tag{3.11}$$

❖**Fonction gussienne**[52] Elle est définie par deux paramètre σ, m

$$\nu(x) = \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right) \tag{3.12}$$

❖**Fonction sigmoidale**[52]

$$\nu(x) = \frac{1}{1 + \exp(-a - (x - c))} \tag{3.13}$$

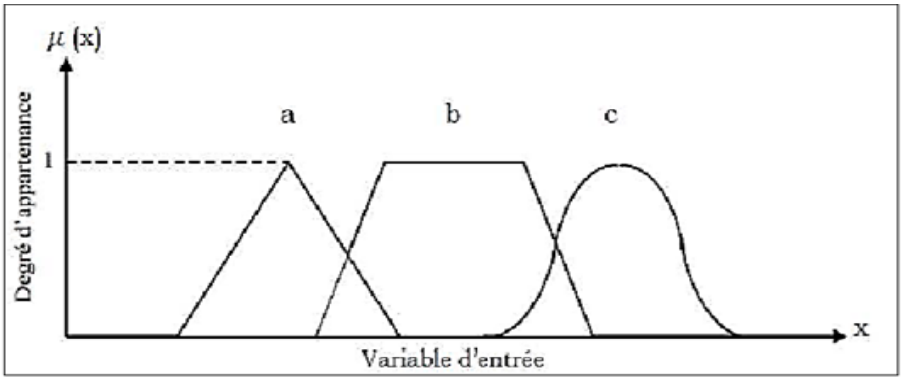

FIGURE 3.7 – Exemple de fonctions d'appartenance.

3.9 Structure de base d'un contrôleur flou

Le schéma synoptique général d'un contrôleur flou est représenté dans la figure 3.8 , [53]

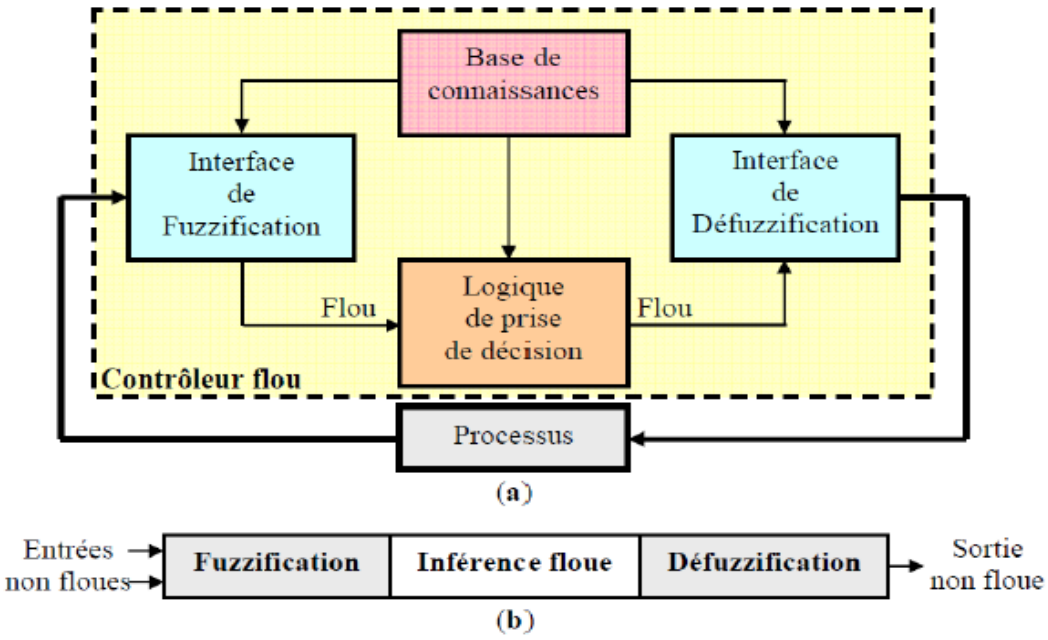

FIGURE 3.8 – a) : Schéma synoptique d'un contrôleur flou

b) : configuration d'un contrôleur flou.

3.9.1 Base de connaissances

La base de connaissances comprend une connaissance du domaine d'application et les but du contrôle prévu. Elle est composée de :

1. Une base de données fournissant les informations nécessaires pour les fonctions de normalisation[54] ;

2. La base des règles constitue un ensemble d'expressions linguistiques structurées autour d'une connaissance dexpert, et représentée sous forme de règle.

3.9.2 Inférence Fuzzification

La fuzzification est l'opération qui consiste à affecter pour chaque entrée physique, un degré d'appartenance à chaque sous-ensemble flous. Autrement dit, c'est l'opération qui permet le passage du numérique (grandeures physiques) au symbolique (variables floues). Pour illustrer le mécanisme de la fuzzification, nous donnerons un exemple en fixant comme valeur d'entrée $e_K = 0.45$. Le résultat de la fuzzification sera présenté sur la figure 3.9 on remarque que pour cette erreur correspond les ensemble flous PP et PM avec les degrés d'appartenances$\mu_{pp}(e_K) = 0.75$ et $\mu_{pM}(e_K) = 0.25$.

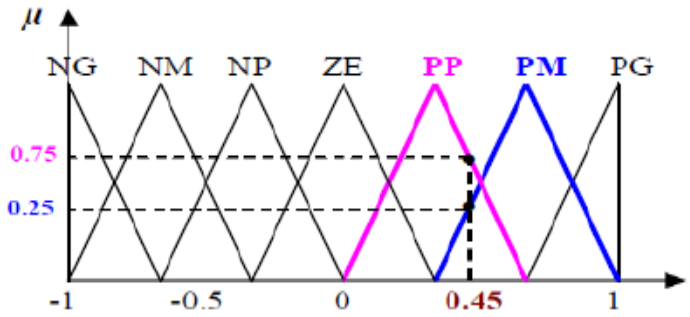

FIGURE 3.9 – exemple de fuzzification

3.9.3 Règles d'inférence floue :

Il existe plusieurs façons peuvent être utilisées pour décrire les règles d'inférence : linguistiquement, symboliquement ou bien par matrice d'inférence. Les trois méthodes dinférence les plus usuelles sont : Max-Produit, Somme-Produit et Max-Min (Implication de Mamdani), cette dernière méthode est la plus utilisée à cause de sa simplicité, elle

réalise l'opérateur "ET" par la fonction "Min", la conclusion "ALORS" de chaque règle par la fonction "Min" et la liaison entre toutes les règles (opérateur "OU") par la fonction Max [55].

3.9.4 Defuzzification :

Il existe Plusieurs stratégies de defuzzification, les plus utilisées sont [55] :

➢ **Méthode du maximum :**

Comme son nom l'indique, la commande en sortie est égale à la commande ayant la fonction d'appartenance maximale. La méthode du maximum simple, facile et rapide mais elle introduit des ambiguïtés et une discontinuité de la sortie (parfois on trouve deux valeurs maximales).

➢ **Méthode de la moyenne du maximum :**

Pour lesquelles la fonction d'appartenance issue de l'inférence est-elle considère, comme valeur de sortie, la moyenne de toutes les valeurs maximale

➢ **Méthode du centre de gravité :** C'est La méthode de defuzzification la plus courante . L'abscisse du centre de gravité peut être déterminée en utilisant la forme générale :

$$y = \frac{\sum_{L=1}^{M} v' \mu_{B'}(\nu')}{\sum_{L}^{M} \frac{\mu_{B'}(\nu')}{\sigma'^2}} \tag{3.14}$$

Ou ν' désigne le centre de gravité de la fonction d'appartenance de l'ensemble flou B' et de défuzzificateur évalue premièrement $\mu_{B'}(\nu')$ et σ' est une mesure de support de la fonction d'appartenance pour la Gieme règle pour la I^{ieme} les fonctions d'appartenance gaussienne σ' est l'écart type.

3.9.5 Logique de prise de décision

C'est un mécanisme de décision, il permet à partir d'un fait observé de la base des règles floues une décision en exploitant le raisonnement approximatif. Dans les inférences de régulateur par logique floue interviennent les opérateurs ET et OU. L'opérateur ET s'applique aux variables à l'intérieur d'une règle tandis que l'opérateur OU lie les différentes règles.

3.9.6 Différents types de régulateurs flous

1. **Régulateur de type Mamdani :**

   Dans la plupart des applications reportées dans la littérature, un contrôleur de ce type est conçu pour réguler, asservir une variable de sortie d'un procédé, soit uniquement à partir de l'erreur e (consigne moins mesure), soit à partir de l'erreur et de sa variation Δe [56]. En 1974, E.H Mamdani avait présenté, pour la première fois, la technique de réglage par logique floue. Celle-ci consiste à déterminer un ensemble de règles qui maîtrise le comportement dynamique du système à commander. L'obtention de ces règles est facile auprès des experts qui connaissent bien le système. Il avait utilisé des règles à prémisses et conclusions symboliques, l'inférence $(\max, \min)$, et la defuzzification par centre de gravité.

2. **Régulateur de type Sugeno :** Dans les régulateurs de ce type, les conclusions des règles ne sont pas symboliques (i.e. représentées par des sous-ensembles flous) mais une fonction des entrées :

   $b^i = f(x_i, K, x_n)$ **les prémisses étant symbolique**

   ou : $f(\ldots)$ **Est généralement une fonction polynomiale**

   la sortie du régulateur est donnée par :

$$y = \frac{\sum_{i=1}^{n} b^i * a_i(x)}{\sum_{j=1}^{n} a_j} \tag{3.15}$$

   ou : a_i **Sont les valeur de l'appartenance de chaque règle pour** $i = 1 \ldots n$

3.10 Les travaux sur l'ordonnancement temps réel régulé(le feedback scheduling)

Nous présentons maintenant quelques travaux importants sur l'ordonnancement régulé.

Buttazzo et al. (1998) où un modèle de tâche élastique pour les tâches périodiques sont présentées. La sensibilité relative des tâches au rééchelonnement est exprimée en termes de coefficients d'élasticité. Une analyse d'ordonnancement du système sous ordonnancement EDF est donnée. Les stratégies d'ajustement des attributs de tâches sont

également présentées dans (Nakajima,1998 ; Kuo et Mok, 1991 ; Kosugi et al. 1994 ; Nakajima et Tezuka, 1994 ; Lee et al. 1996).[57]

Stankovic et al. (1999) Présenter un algorithme d'ordonnancement, le FC-EDF, qui utilise explicitement la rétroaction en combinaison avec l'ordonnancement EDF. Un contrôleur PID régule le taux d'échec d'échéance pour un ensemble de tâches douces en temps réel avec des temps d'exécution variables, en ajustant l'utilisation de CPU demandée. On suppose que les tâches peuvent modifier leur consommation de processeur en exécutant différentes versions du même algorithme. Un contrôleur d'admission est utilisé pour s'adapter à des changements plus importants de la charge de travail. Dans (Lu et al. 2000), la même approche est étendue. Un contrôleur PID supplémentaire est ajouté qui contrôle à la place l'utilisation du CPU.

Cervin et al. (2002) Ont proposé une régulation de l'ordonnancement de plusieurs lois de commande. La période des lois de commande est modifiée en ligne en réponse à la variation de la charge processeur. La conception du régulateur repose sur les travaux de (Eker et al, 2000). Leurs résultats montrent qu'un choix convenable des périodes nominales (fréquences nominales) et un simple rééchelonnement des périodes nominales en respectant les contraintes d'utilisation du processeur permettent d'avoir des périodes d'échantillonnage optimales et par là de bonne performance totale des contrôleurs. [45]

Cervin (2003) Dans l'ordonnancement régulé de Cervin, la durée d'exécution est elle-même la mesure utilisée dans la boucle. Un filtre passe bas avec un facteur d'oubli noté λ dans l'intervalle [0, 1] est ainsi utilisé pour estimer la prochaine durée d'exécution sur laquelle on doit se baser pour calculer le facteur d'utilisation du processeur. l'estimation de la durée d'exécution de chaque tâche laquelle le régulateur d'ordonnancement peut agir est comme suit :

$$\hat{C}\iota(0) = hi, 0$$

$$\hat{C}\iota(0) = \lambda * (k-1) + (1-\lambda) * Ci \tag{3.16}$$

Où les ci sont les durées d'exécutions réelles des tâches et $\hat{C}_i(0)$ est l'initialisation arbitraire de l'estimateur de chaque tâche à sa période nominale.Le paramètre lambda α permet de lisser l'estimateur dans le temps : S'il est proche de 1, l'estimateur prendra plus de temps pour s'adapter à un changement soutenu du temps d'exécution de la tâche, tandis que s'il est faible, il sera très sensible à des changements transitoires.[45] Ainsi, lorsqu'une instance k de la tâche ti termine l'exécution, elle renvoie sa durée d'exécution ci qui sera

utilisée pour estimer la prochaine durée d'exécution $\hat{C}_i^k$ puis Le facteur d'utilisation $\hat{U}$:

$$\hat{U} = \sum_{i=1}^{n} \frac{\hat{C}i}{hnom, i} \tag{3.17}$$

Pour contourner la surcharge processeur Cervin utilise un rééchelonnement des périodes nominales avec un paramètre $\alpha > 1$ (appelé facteur de rééchelonnement) calculé selon la borne de Lu et Layland pour EDF et selon la borne de Leung et all pour DM :

$$\alpha = \hat{U}/ref \tag{3.18}$$

3.11 Conclusion

Nous avons présenté dans ce chapitre que l'objectif principal d'un système de commande est de contrôler un processus pour l'amener dans un état adéquat aux désirs de l'utilisateur , et nous avons étudié L'ordonnancement temps réel régulé, s'agit de l'application du principe de régulation à boucle de rétroaction (boucle fermée) sur des systèmes informatiques dans le but d'améliorer leurs performances pendant leur exécution. Ensuite on a défini les notions générales de la logique floue.
Enfin,La majorité des travaux sur l'ordonnancement régulé. Ces travaux ont pour unique objectif la régulation de paramètres d'ordonnancement, plus exactement la charge processeur ou le nombre d'échéances ratées .

Chapitre 4

Conception de régulateur flou d'ordonnancement temps réel multiprocesseur

4.1 Introduction

La conception d'un ordonnancement se base sur le temps d'exécution dans le pire cas (noté WCET). Cependant, dans les cas d'activités à durées variables, l'ordonnancement basé sur WCET induit un sur-dimensionnement du processeur qui peut être important. Une estimation au plus juste de cette borne (c.-à-d. des WCET) est difficile et par facilité ou par prudence on la surestime. Ceci induit un sur-dimensionnement de l'ordonnancement et une sous-utilisation du processeur. Le principe de la commande par rétroaction (ou de régulation) a pour avantage de permettre le rejet des perturbations et de réduire la sensibilité aux incertitudes. Parmi les lois de commande les contrôleurs flous permettent de piloter des systèmes complexes et difficilement mobilisable.

Ainsi, dans ce chapitre se focalise sur le problème d'ordonnancement temps réel régulé multiprocesseur dans les systèmes de récupération d'énergie. Nous avant proposée comme solution un régulateur flou dont les entrées sont la charge de processeur et la charge batterie, alors que la sortie est le facteur de vitesse. Notre contribution vise d'une part à améliorer les performances et la qualité de contrôle du système et d'une autre part à éviter la décharge totale des batterie et donc prolonger la durée de vie de système.

4.2 Hypothèses

Nous proposons dans cette étude de la modélisation les hypothèses suivantes :

— L'ensemble des taches à ordonnancer ainsi leurs caractéristiques sont connus ;

— Nous estimons que le système est alimente par un système de récupération de l'énergie ambiante.

— Les tâches à ordonnancer sont périodiques.

4.3 Modèles de la consommation d'énergie

4.3.1 Modèle théorique

Un processeur est un circuit intégré de la famille des CMOS(complementarymetal-oxide-semiconductor), un tel circuit répond aux équations génériques suivantes : [31]

L'énergie consommée dans un intervalle de temps [a, b] est par définition l'intégrale de la puissance dissipée :

$$E = \int_a^b P(t)dt \tag{4.1}$$

Où P(t) est la puissance dissipée à l'instant t. Cette puissance dissipée dans un circuit électroniques compose de la puissance statique et de la puissance dynamique :

$$P = P_{stat} + P_{dyn} \tag{4.2}$$

Où principale différence entre eux est que la puissance dynamique (P_{dyn}) dépend de la fréquence, tandis que la statique (P_{stat}) ne l'est pas. Dans les circuits CMOS la puissance dynamique représente 80-85 % de la puissance dissipée et, classiquement, on néglige la puissance statique. La puissance dissipée totale peut donc s'exprimer par :

$$P \approx P_{dyn} \sim \alpha f C V^2 \tag{4.3}$$

Où α est le nombre de transitions par cycle d'horloge, f est la fréquence de fonctionnement, C'est la capacité équivalente et V est la tension d'alimentation.
La fréquence de fonctionnement est donnée par :

$$f = \frac{(V - v_t)^\gamma}{V} \tag{4.4}$$

Où V_t est la tension de seuil qui est suffisamment petite par rapport à la tension d'alimentation et γ une constante. Donc la relation entre fréquence et tension devient : $f \sim v$

4.3.2 Modèle de système multi-cœur

Ce modèle se base sur utilisation de big core et LITTLE core qui est une technologie de microprocesseur multi-cœur conçue par ARM, utilisant simultanément un cœur à très faible consommation (LITTLE) comme cœur principal et plusieurs cœurs plus puissants (big), s'activant en cas de demande importante en puissance de calcul. Ceci permet d'économiser d'énergie par rapport au cas où plusieurs processeurs de même puissance sont mis en parallèle, tout en conservant une forte capacité de calcul. D'autre part, le but à utiliser des processeurs multi-coeurs, qui peuvent mieux répondre aux besoins de l'informatique dynamique moderne et réduire la consommation d'énergie.
Notre étude est basée sur le processeur multi-cœur. Ces principales caractéristiques sont données dans le tableau ci-dessous

Big core		Little core	
F(MH3)	Power(w)	F(MH3)	Power(w)
800	0.2721	200	0.0346
900	0.3177	300	0.0356
1000	0.3724	400	0.0377
1100	0.4367	500	0.0414
1200	0.5114	600	0.0472
1300	0.5969	700	0.0557
1400	0.6938	800	0.0675
1500	0.8027	900	0.0832
1600	0.9240	1000	0.1034
		1100	0.1288
		1200	0.1600

TABLE 4.1 – Les paramètres Big core et Little core

Le modèle qui décris la puissance consommée sous forme d'une fonction polynomiale de la fréquence du processeur f présenté dans [58] où la puissance est écrite comme suit :

$$p(f) = k * f^{\alpha} + \beta \tag{4.5}$$

La figure 4.1 montre le différence de consommation d'énergie dans big core et LITTLE core en fonction des fréquences (des valeurs discret) données dans tableau précédant.

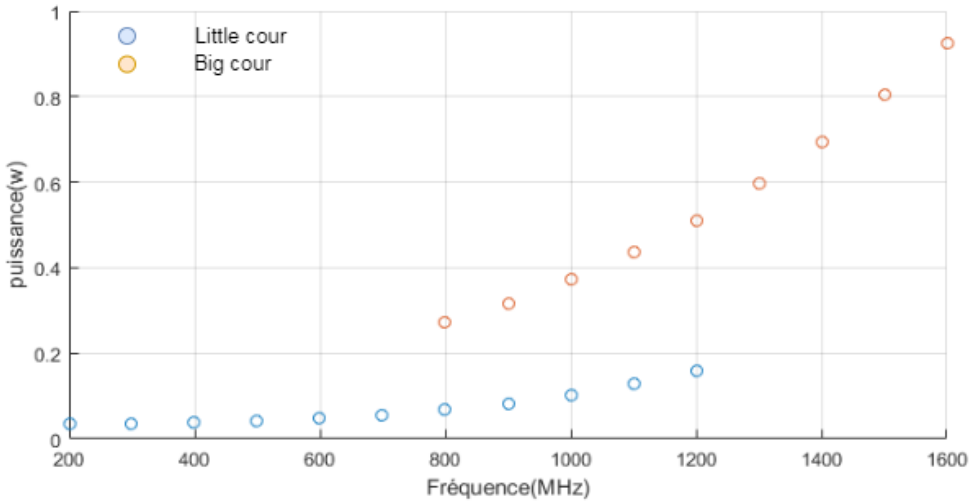

FIGURE 4.1 – La fonction de la puissance consommée

4.3.3 Modélisation de la source d'énergie

Supposons que l'énergie de l'environnement, telle que l'énergie solaire, soit récupérée et convertie en énergie électrique pour compléter la batterie du système embarqué. Le comportement de la source d'énergie solaire est modélisé comme suit :

$$P_s(t) = |0.9 \times R(t) \times \cos(\frac{t}{0.7\pi}) \times \cos(\frac{t}{1\pi})| \tag{4.6}$$

Ou $R(t)$ c'est une variable aléatoire uniforme distribué entre 0 et 1, tandis que Les valeurs de P_s sont limitée à une valeur maximale égale à P_s, $max = 0,9$

Comme montrer sur la figure 4.2, la courbe de puissance obtenue $P_s(t)$ simule des périodes similaires à celles obtenues avec des cellules solaires dans un environnement extérieur. La puissance récupérée $P_s(t)$ est la puissance nette alimentant batterie. L'énergie totale récupérée dans un intervalle $[t1; t2]$, notée $Es(t1; t2)$ se calcule donc de la façon suivante :

$$E_s = \int_{t_1}^{t_2} P_s(t)dt \tag{4.7}$$

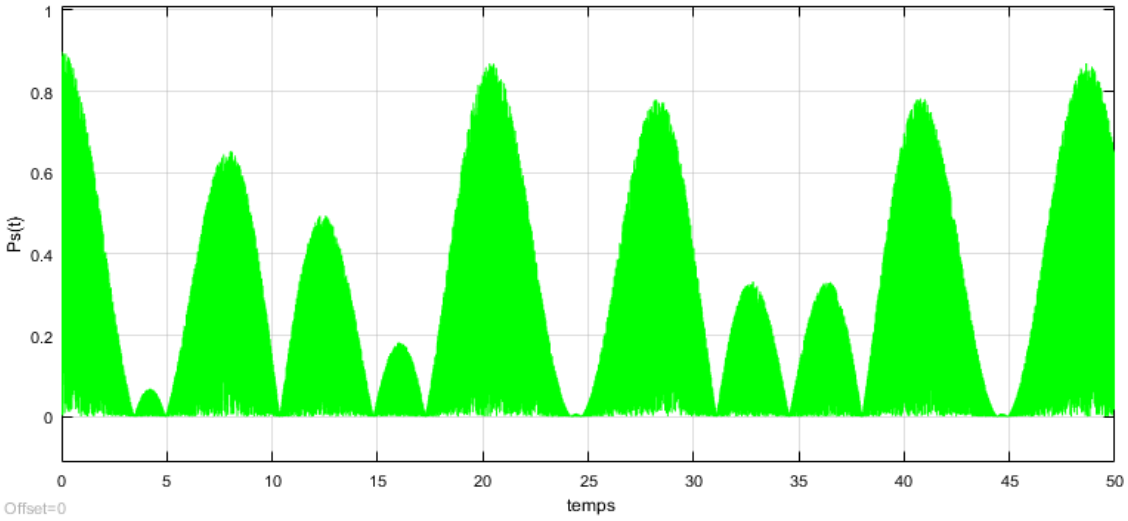

FIGURE 4.2 – La courbe de la puissance P_s(t)

4.3.4 Modélisation du réservoir d'énergie

Le système considéré utilise une unité de stockage de l'énergie d'une capacité nominale (E, exprimé en Joules ou Watt-heure). Le niveau d'énergie, noté $El(t)$ à un moment donné t, le niveau de l'énergie stockée oscille entre deux seuils E_{min} et E_{max} donnant une capacité de l'énergie disponible pour le processeur égale à $E = E_{max} - E_{min}$. Si l'unité de stockage de l'énergie est complètement déchargée. Simplement le niveau d'énergie égal à E_{min}, dans ce cas aucune tâche ne peut être exécuté, et le processeur doit s'arrêter et restera inactif. D'autre part, si l'unité de stockage de l'énergie est complètement remplie, le niveau d'énergie égal à E_{max}, et nous continuons à la charger, l'énergie est gaspillée. Pour réduire le gaspillage et assurer au même temps une qualité de contrôle des tâches, lorsque la batterie est "presque pleine" il serait utile d'exécuter les tâches avec la vitesse maximale du processeur.

4.4 Conception du régulateur flou d'ordonnancement

Dans cette section, nous décrivons la conception de l'ordonnanceur flou. Il se compose de quatre éléments principaux : l'interface floue, la base de règles, mécanisme de raisonnement et interface de défuzzification. L'entrée du régulateur flou est l'utilisation instantanée du processeur et le niveau d'énergie définis par la formule (4.1), et la sortie est le fréquence.

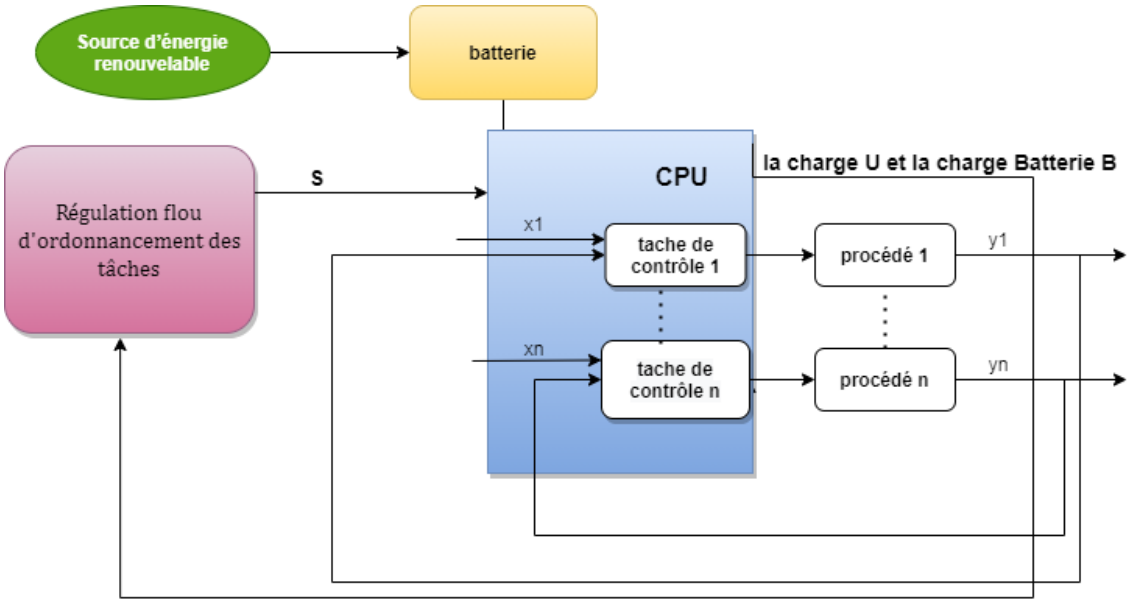

FIGURE 4.3 – Schéma du régulateur flou d'ordonnancement

Remarque :la tâches de Régulation flou d'ordonnancement s'exécute dans **CPU**

4.4.1 Structure du régulateur flou d'ordonnancement

La description ci-dessus montre un régulateur à logique floue bidimensionnelle, c'est-à-dire que les variables linguistiques d'entrée sont la charge de processeur représentée par U, la charge batterie qui représente B, et sa variable linguistiques de sortie est représentée par la fréquence S. Soit $U \in [0, 1]$ et $B[0, 2.5]$ et $S[0, 1]$. L'intervalle $[0, 2.5]$ est la différence entre le niveau maximale et minimale de la batterie.

4.4.2 Les valeurs linguistiques des entrées et des sorties

L'ensemble des valeurs linguistiques pour les deux variables linguistiques U et S est défini par $\{VVS, VS, S, SM, M, HM, B, VB, VVB\}$ correspondant aux neuf fréquences discrètes du multiprocesseur biglittel Arm où VVS signifie VeryVery Small, VS : Very Small, S : Small, SM : Small Medium, M : Medium, HM : High Medium , B : Big, VB : VeryBig , VVB : VeryVeryBig, Pour la variable linguistique B, les valeurs linguistiques sont EMin qui représentant Énergie-Minimale et EMax représente Énergie-Maximale. Supposons que la batterie soit dans l'état d'énergie maximal, si le niveau de d'énergie est supérieur à 50% de sa charge d'énergie, sinon elle est dans l'état d'énergie minimum.

4.4.3 Les fonctions d'appartenances des valeurs linguistique d'entrées et de sorties

Les fonctions d'appartenance, notées MFs (MembershipFunctions, en anglais) utilisées. Pour notre étude, nous avons choisi le types de fonctions d'appartenance sous forme **trapézoïdales** elle est définie par quatre paramètre $\{a, b, c, d\}$

$$\mu_A = \max\left(\min\left(\frac{x-a}{b-a}\right), 1, \frac{c-x}{c-d}, 0\right) \tag{4.8}$$

Dans cette contribution, pour les valeurs linguistiques des variables entrées et des sorties sont représentées sur les figures suivantes :

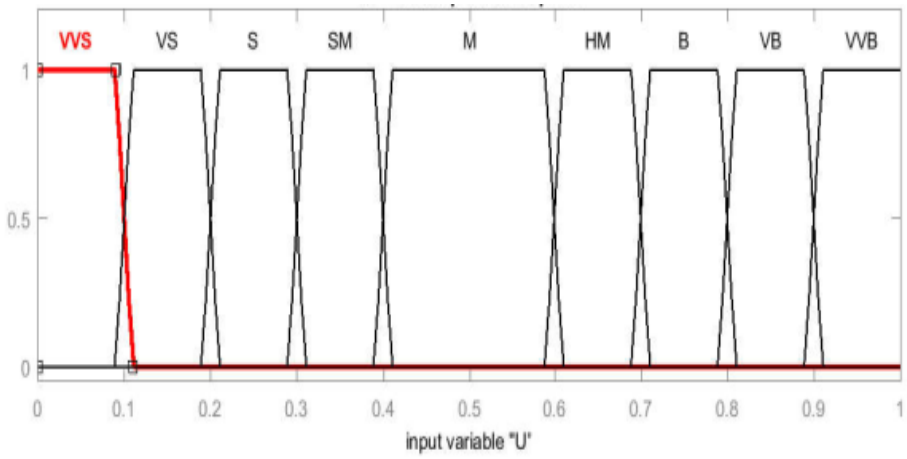

FIGURE 4.4 – La fonctions d'appartenance de valeur linguistique d'entrées la charge "U"

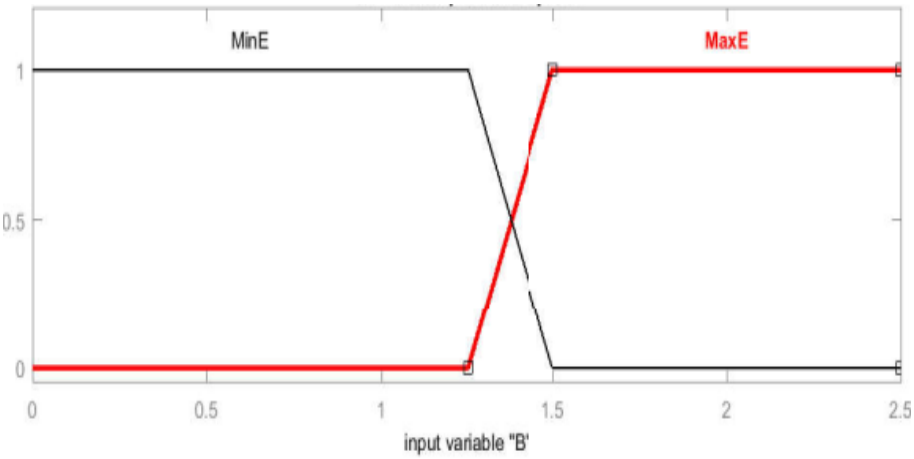

FIGURE 4.5 – La fonctions d'appartenance de valeur linguistique d'entrées la charge batterie "B"

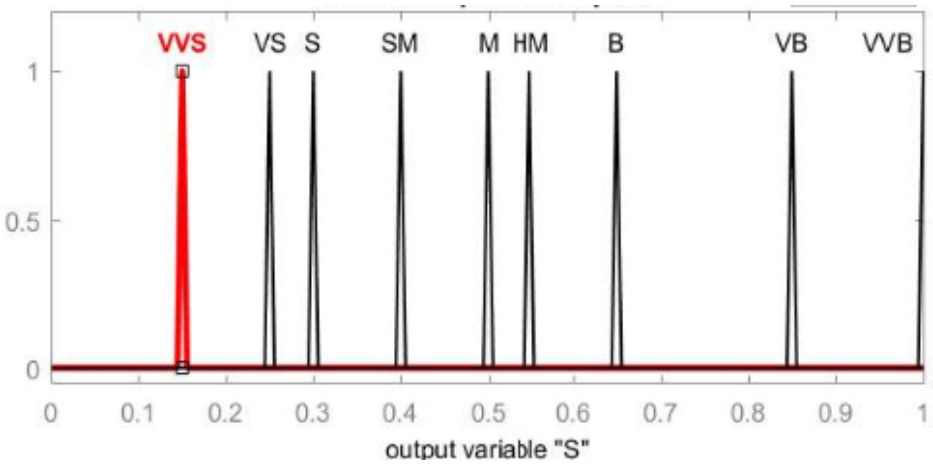

FIGURE 4.6 – La fonctions d'appartenance de valeur linguistique de sorties "S"

4.4.4 Les règles de commande floue et le mécanisme d'inférence

L'ensemble des règles de décision constituent la "base de règles" qui caractérise notre stratégie pour contrôler le processus dynamique étudié, c'est-à-dire que le système de contrôle est l'organisation est une matrice (voir le tableau 4.2 ci-dessous). La méthode d'inférence adoptée est basée sur le mécanisme d'implication de Mamdani pour obtenir des valeurs précises(crisp values, en anglais) des actions de contrôle flou présumées. Dans notre étude, nous avons utilisé le opérateur "ET" (l'intersection) de la logique floue qui intervienne sur les fonctions d'appartenance et représente les variables linguistiques comme :

$$\mu_{A\cap B}(u) = \min\{\mu_A(u), \mu_B(u)\} \tag{4.9}$$

Aussi nous avons choisi la technique de défuzzification dite : La plus grande valeur (absolue) maximale notée LOM (largest value (absolute) of maximum), qui produit le point où la fonction d'appartenance de l'action de la commande floue atteint une valeur absolue maximale. Nous définissons les deux core (big and little) **core big** $\in \{HM, B, VB, VVB\}$ ET **core Little** $\in \{VVS, VS, S, SM\}$

S		U								
		VVS	VS	S	SM	M	HM	B	VB	VVB
B	MaxE	VS	S	SM	M	HM	B	VB	VVB	VVB
	MinE	VVS	VS	S	SM	M	HM	B	VB	VVB

TABLE 4.2 – La table de décision

La surface d'entrées-sorties du système d'inférence flou qui correspond au tableau est représentée sur la figure 4.7. Cette figure montre la mise en correspondance entre les entrées (à savoir U et B) et la sortie S conçues par la table de décision. Nous pouvons remarquer que lorsque la batterie contient un minimum d'énergie, le régulateur flou renvoi la fréquence S qui est juste au-dessus ou égale à la charge du processeur. Ceci est pour favoriser la charge de la batterie. Si la batterie comporte un maximum d'énergie, la fréquence S retourné est deux fois supérieur à la charge instantanée du processeur. Cela permet de réduire le risque de manquer des échéances, ce qui fournit ensuite une **bonne qualité du contrôle**.

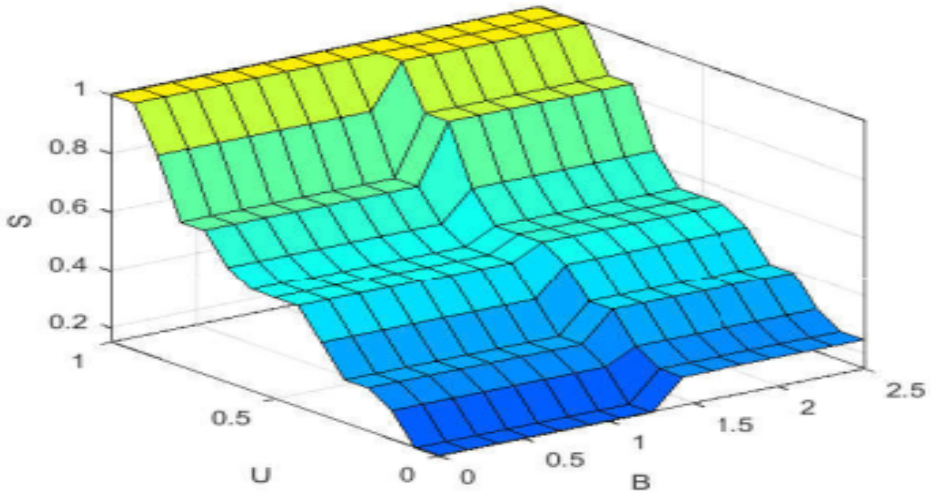

FIGURE 4.7 – Entrées sortie du régulateur flou d'ordonnancement

4.4.5 Exemple d'utilisation de régulateur flux

Nous présentons ici un exemple montrant la variation de facture de vitesse dans le cas d'utilisé multiprocesseur avec le core Little noté faible et le core Big noté fort telle que $Little \in [0, 0.5]$ et $Big \in [0.55, 1]$:

— Facteur de vitesse lorsque nous considérons $U = 0.15$ (VS) et $B = 2.03$ $(MaxE)$ la figure 4.8 montre la facture de vitesse $S = 0.3$ du processeur Little.

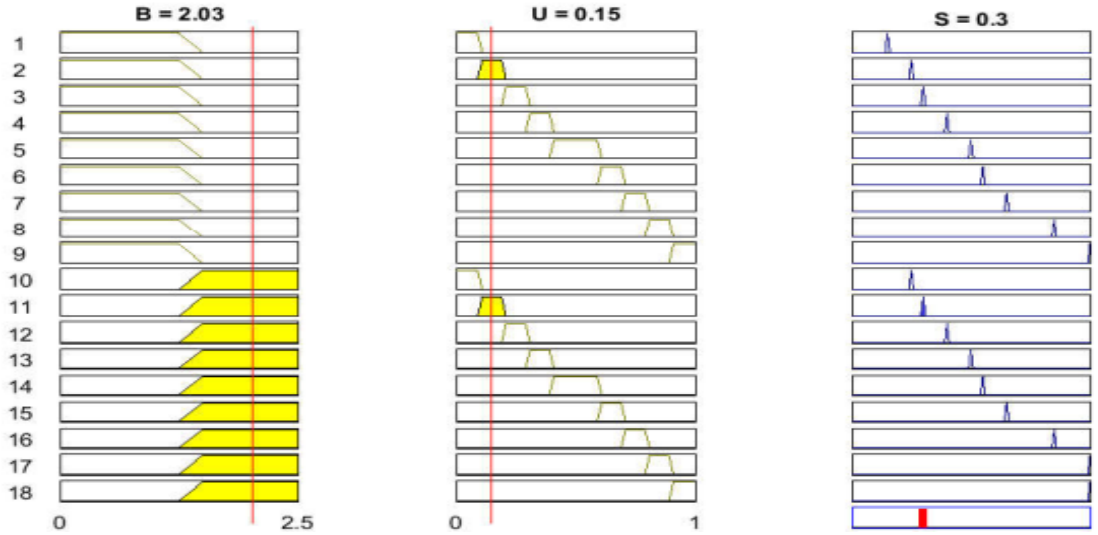

FIGURE 4.8 – Inférence floue avec des fonctions d'appartenance dans le premier cas

— Facteur de vitesse lorsque nous considérons $U = 0.75$ (VB) et $B = 0.47$ (MinE) la figure 4.9 montre la facture de vitesse $S = 0.65$ du processeur Big .

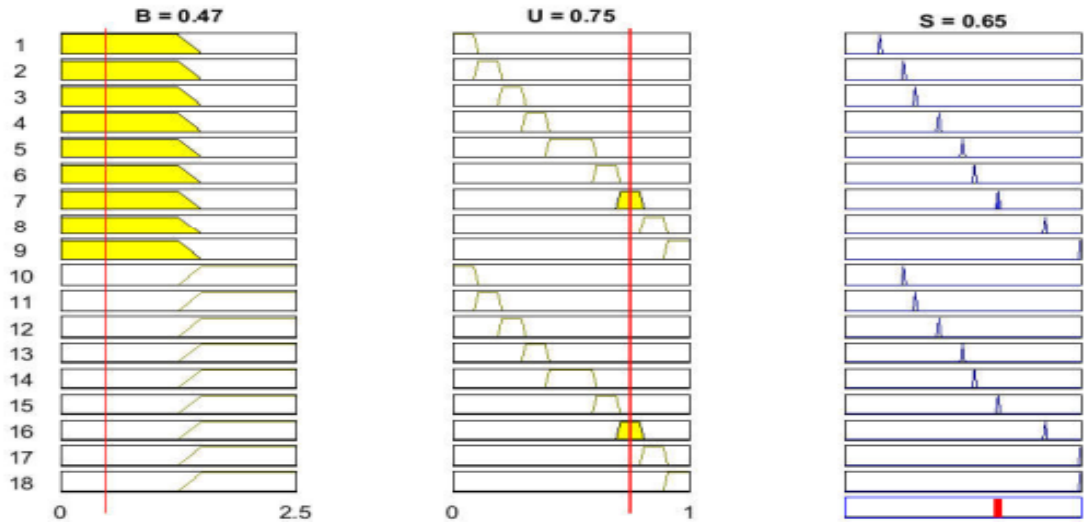

FIGURE 4.9 – Inférence floue avec des fonctions d'appartenance dans le deuxième cas

— Facteur de vitesse lorsque nous considérons $U = 0.105$ (VVS) et$B = 0.31$ $(MinE)$ la figure 4.10 montre la facture de vitesse $S = 0.25$(VS) du processeur Little.

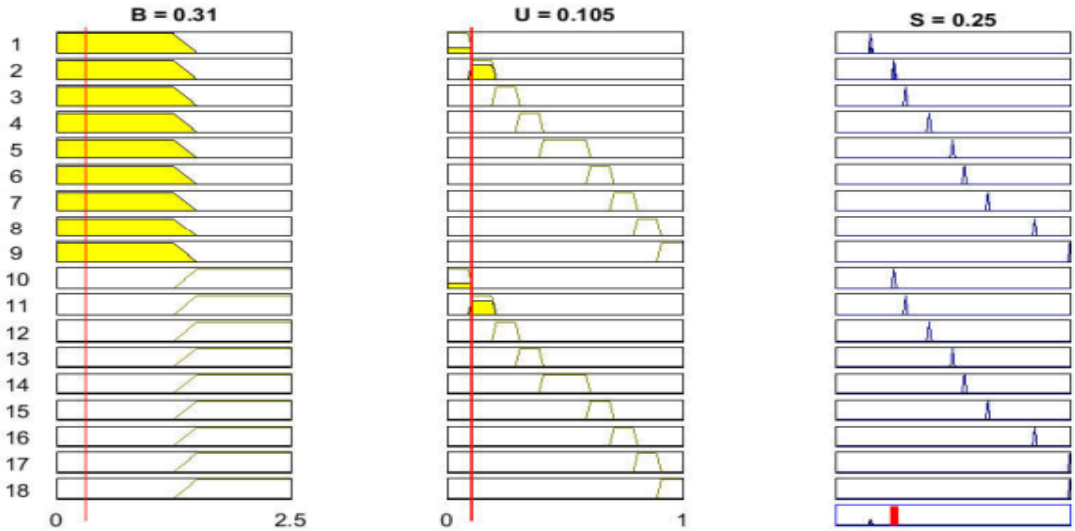

FIGURE 4.10 – Inférence floue avec des fonctions d'appartenance dans le troisième cas

4.5 Conclusion

Dans ce chapitre, nous avons proposé un régulateur flou d'ordonnancement temps réel multiprocesseur dans le contexte de système de récupération de l'énergie. Notre contribution a travers ce régulateur vise à prendre en compte la charge du processeur et énergie disponible dans la batterie pour calculer la fréquence utile d'étendre sa durée de vie et d'optimiser la qualité de contrôle du système.

Dans le chapitre suivant nous détaillerons les évaluations effectuées et les résultats obtenus.

Chapitre 5

Implémentation et Résultats

5.1 Introduction

Dans ce chapitre, nous détaillerons les simulations effectuées pour évaluer la solution que nous avons présenté précédemment. Nous introduisons d'abord l'environnement de simulation. Ensuite, nous donnerons les résultats obtenus et l'analyse effectuée.

5.2 Environnement de simulation

5.2.1 TrueTime

Nous avons utilisé l'outil TrueTime pour implémenter notre proposition. TrueTime est un simulateur sur Matlab/Simulink développé depuis 1999 à l'Université de Lund, et dont l'objectif est de faciliter la modélisation de régulateurs embarqués et distribués. [59] La figure 5.1 ci-dessous présente les différents objets de cette bibliothèque à outils.

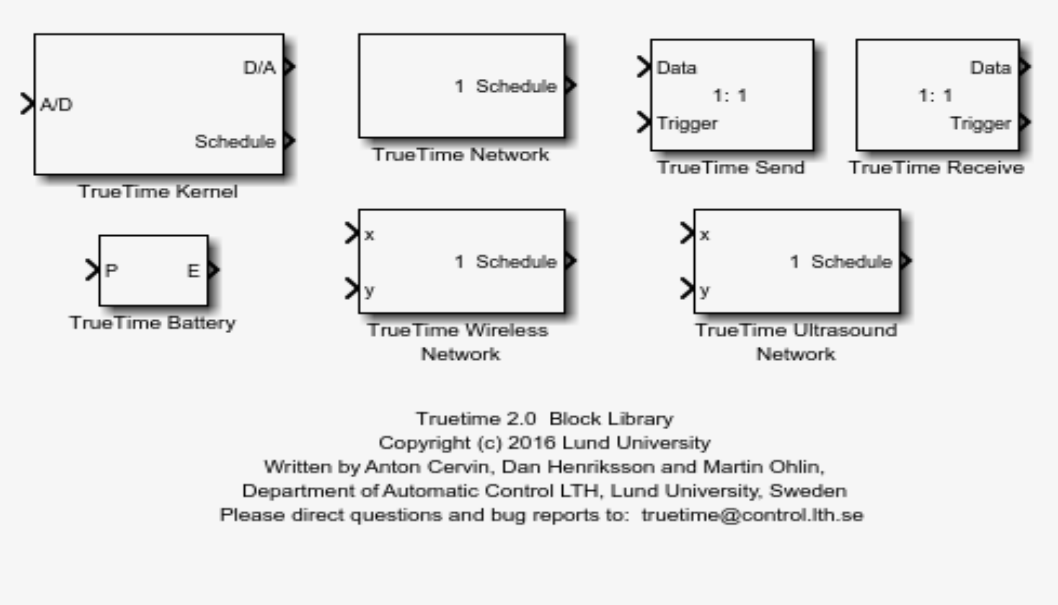

FIGURE 5.1 – Bibliothèque de TrueTime

Les blocs TrueTime sont connectés à des blocs Simulink ordinaires pour former un système de contrôle en temps réel, la caractéristique principale de TrueTime est la possibilité deco-simulation de l'interaction entre les dynamiques continues du monde réel et l'architecture informatique sous forme d'exécution de tâches. Cet outil fournit une librairie open source dédiée au domaine de temps réel et composé des composants suivants :

- ➢ Un noyau temps réel
- ➢ Réseau TrueTime
- ➢ Nœud émetteur et un nœud récepteur
- ➢ Réseau sans fil
- ➢ Réseau ultrason
- ➢ Batterie

Nous considérons un système de contrôle embarqué composé de trois boucles de contrôle indépendantes (voir la Figure 5.2). Chaque procédé est contrôlé à l'aide d'un algorithme PID, qui a été conçu et proposé dans [60]. La fonction de transfert pour chaque procédé est : $G(s) = 1000/(s2+s)$. Le signal de référence est un signal carré avec une fréquence $1Hz$ et une amplitude de 1 unité. Nous avons implémenté un générateur aléatoire des durées d'exécution des tâches ci selon la loi de Weibull donnée ci-après :$C_i = -\log(1-R)^{1/K} \times \beta$ avec$k = 3$, $\beta = 0.0015$, et R une variable aléatoire uniformément distribué entre 0 et 1. Nous choisissons ces paramètres afin que le maximum des ci soit inférieur ou égal à

WCETi

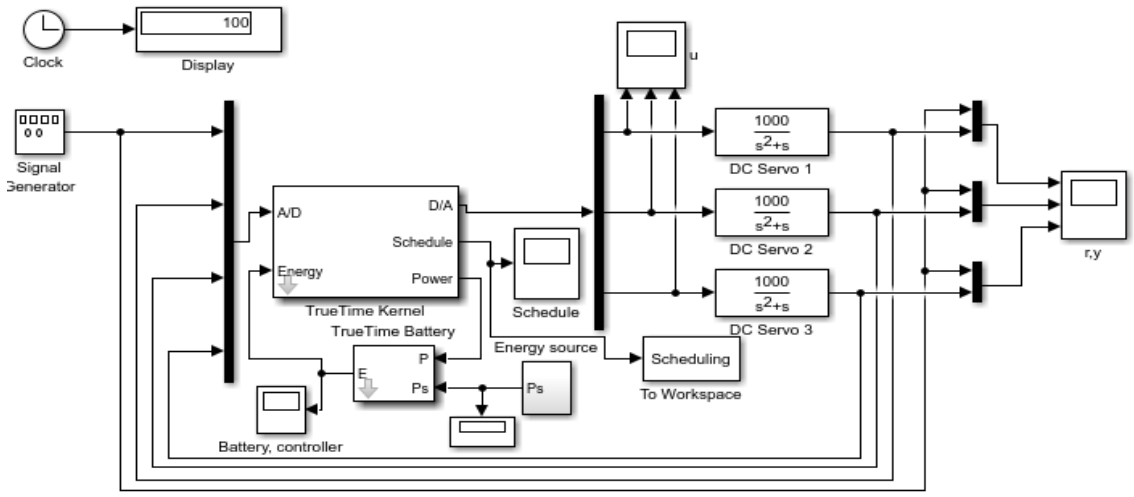

FIGURE 5.2 – Schéma du système de contrôle

5.3 Résultats

5.3.1 Ordonnancement régulé des tâches

Nous considérons un ensemble de trois tâches $T = t_i - 1 \leq i \leq 3$ ou $ti = (Ci, Di, Pi)$ avec une tâche FBS (régulateur) qui s'exécuté sur multiprocesseur(big.little). La figure 5.3 montre que la tâche t1, est la plus prioritaire que t2 et t3 et le changement des couleurs montre la migration des tâches d'un processeur à un autre.

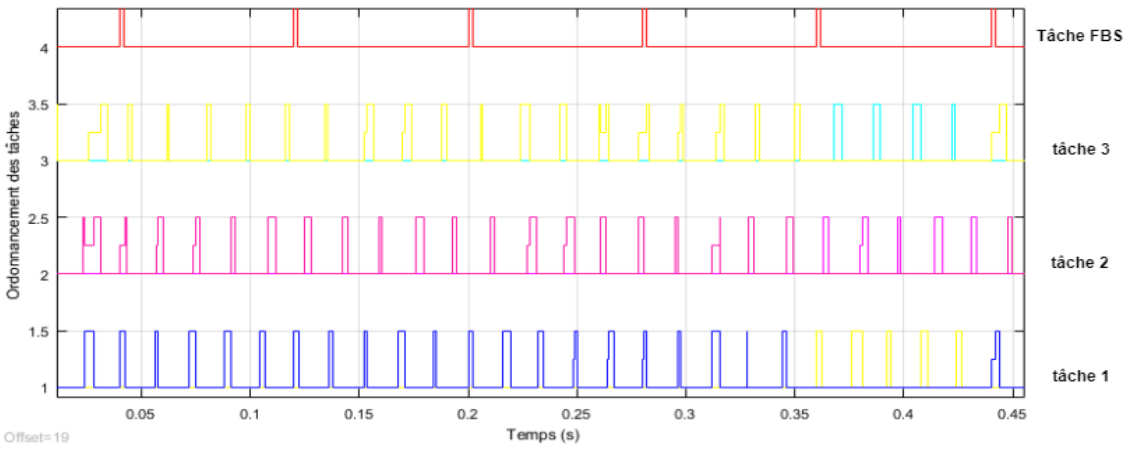

FIGURE 5.3 – Ordonnancement régule des tâche sous leur FBS

— La figure 5.4 montre l'ordonnancement des tâche sans régulateur flou (FBS) qui s'exécuté sur processeur Big (on remarqué que l'ordonnancement s'arrête un certain temps (18.45 s), vue l'épuisement d'énergie)

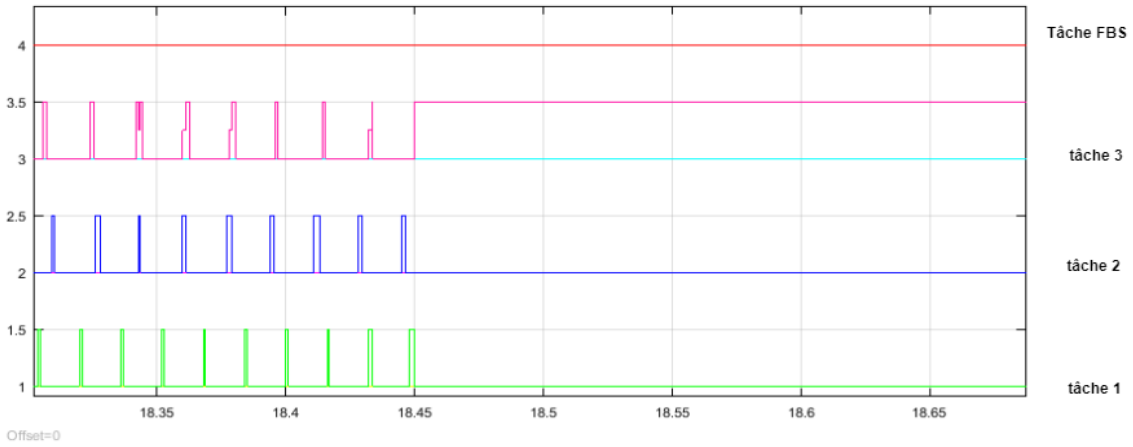

FIGURE 5.4 – Ordonnancement régule des tâche sans FBS

5.3.2 La consommation d'énergie :

La figure 5.5 montre que l'énergie disponible sans la tâche FBS dans le cas d'utilisation **core Big**, la batterie se décharge après un certain temps(18.45 s). Ceci cause l'arrête de l'exécution et la déstabilisation des procédés contrôlés, car le processeur fonctionne avec une vitesse maximales et ceci même si avec le core Big, l'erreur de contrôle est la plus faible ($e_{big} = 4.1293e \times 10^4$)

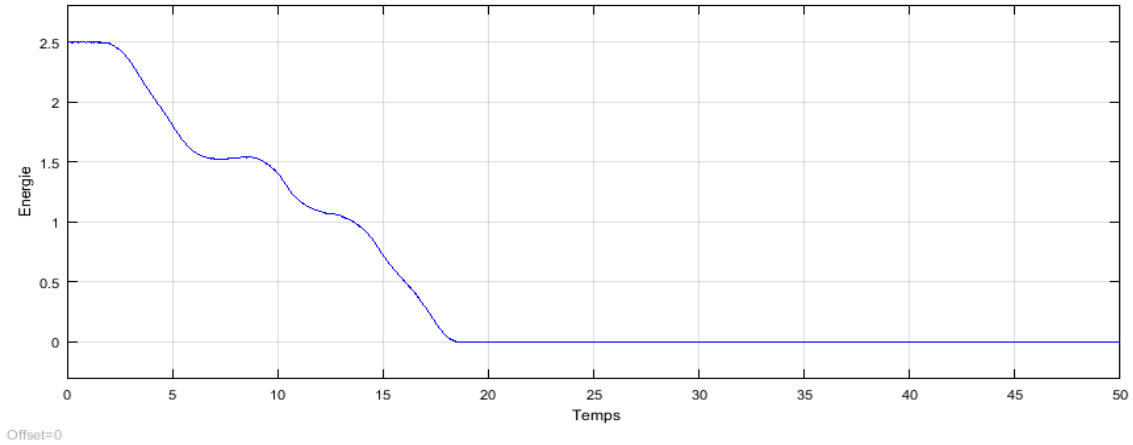

FIGURE 5.5 – Énergie disponible dans le cas core Big

La figure 5.6 montre que l'utilisation d'un régulateur flou (FBS) permet d'optimiser l'énergie disponible avec une faible consommation d'énergie et stabilisation des procédés contrôlés. Donc FBS dans ce cas protège contre la décharge de la batterie. Nous précision ici que l'érreur de controle dans ce cas est $e_{fbs} = 4.5496e \times 10^4$ qui est inférieure à celle obtenue sans FBS avec le core Little

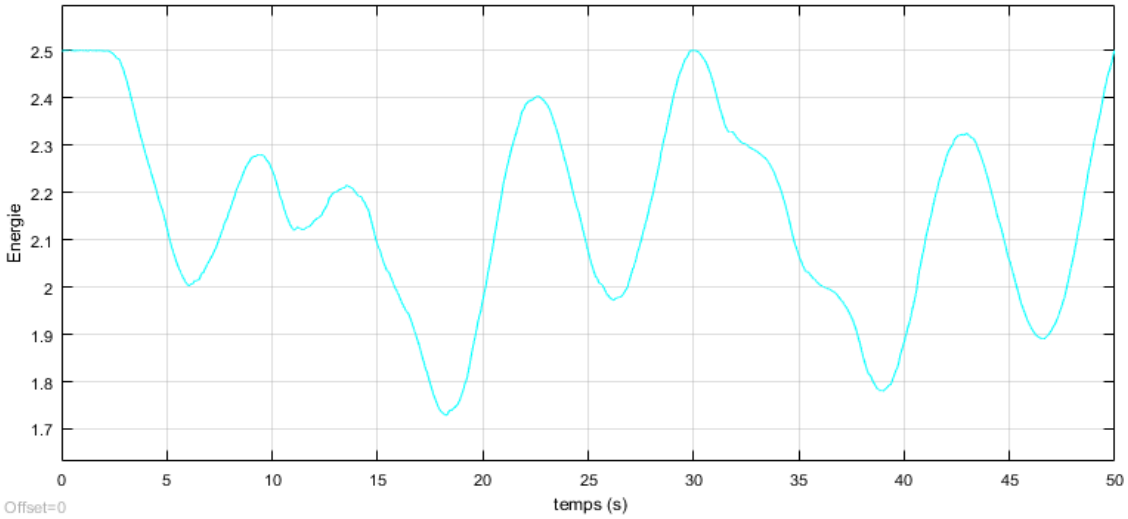

FIGURE 5.6 – Énergie disponible avec FBS

La figure 5.7 montre que l'énergie disponible sans régulateur flou dans le cas d'utilisation unique du **core Little**, la batterie ne se décharge pas. Cependant, l'utilisation de core Little induit une erreur de contrôle très élevée ($e_{little} = 4.9927e \times 10^4$))

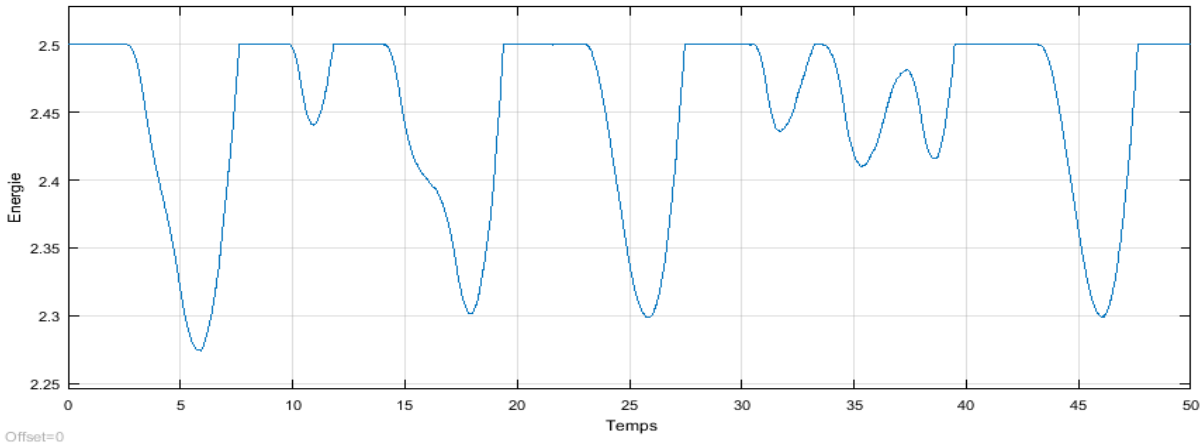

FIGURE 5.7 – Énergie disponible dans le cas core Little

5.3.3 La commande de moteur(U)

La figure 5.8 montre la commande (U) sans FBS dans le cas d'utilisation du **core Big** en cours de fonctionner jusqu'un certain temps(18.45 s) déstabilisé, à cause épuisement d'énergie

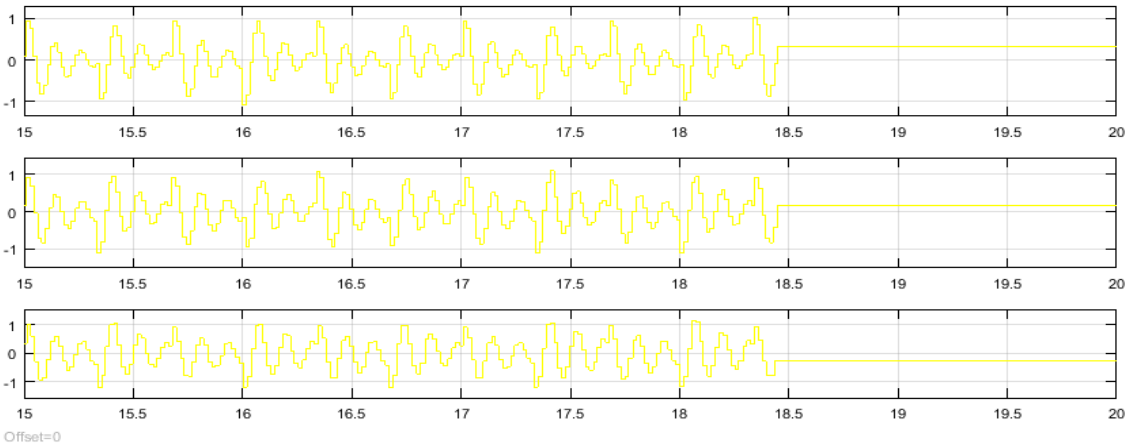

FIGURE 5.8 – la commande de moteur(U) sons FBS

la commande de moteur(U) avec FBS fonctionne et stabilise (voir la figure 5.9 ci-dessous), Nous remarquons que la commande ne s'arrête pas au temps (18.45 s).

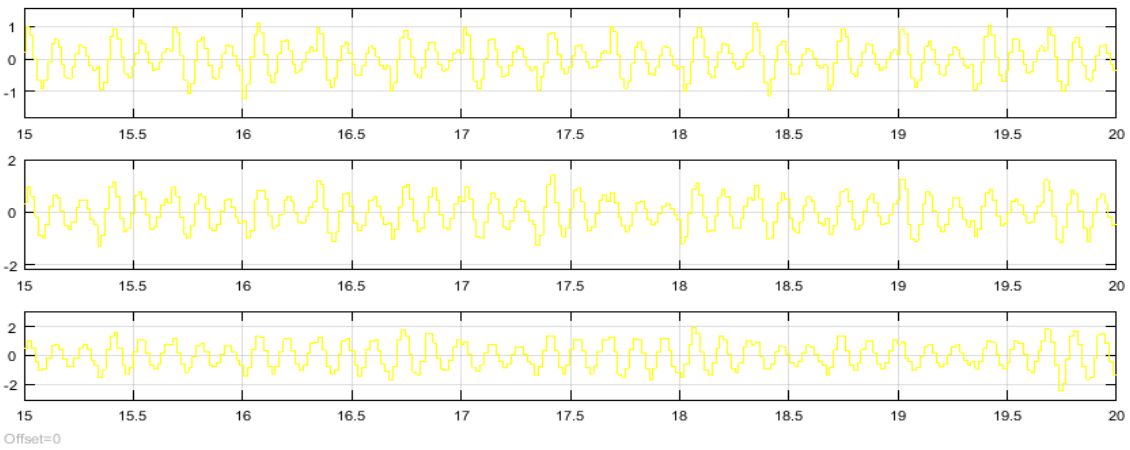

FIGURE 5.9 – la commande de moteur(U) avec FBS

5.3.4 Qualité de Contrôle

Nous considérons la performance de régulation de trois tâche, les traits jeune y représentent la consigne fournie par un générateur de signaux à laquelle les régulateurs doivent faire tendres le signal de sortie des moteur (procédés), représentés par les traits rose, des servomoteurs asservis.

Dans le cas de la performance de régulation avec FBS, la figure 5.10 montre stabilisation de système de contrôle

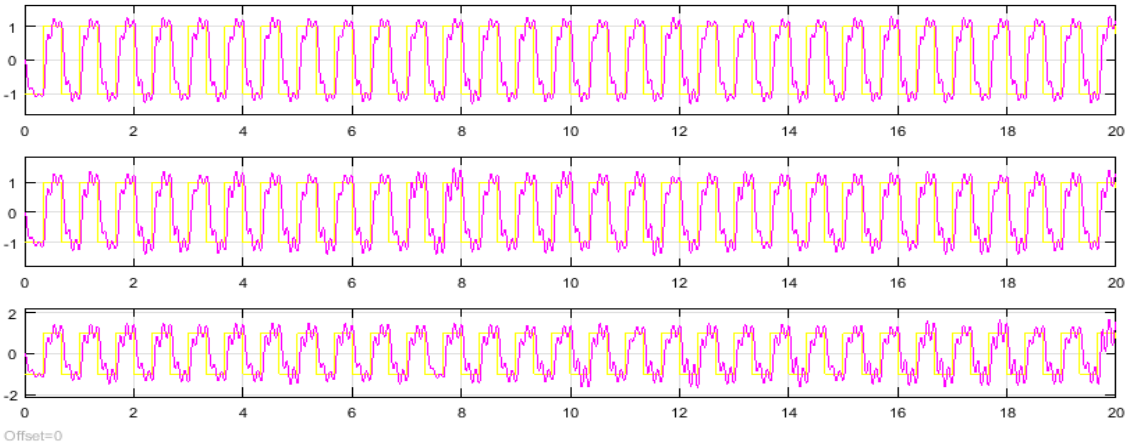

FIGURE 5.10 – Performance de contrôle des trois tâches avec FBS

La performance de régulation sans FBS dans le cas d'utilisation **core Big** où la figure 5.11 montre déstabilisation de système de contrôle et l'erreur de contrôle est très grande, en précession dans la figure 5.12 montre que à l'instant $t = 18.45s$ le système contrôlé est déstabilisé.

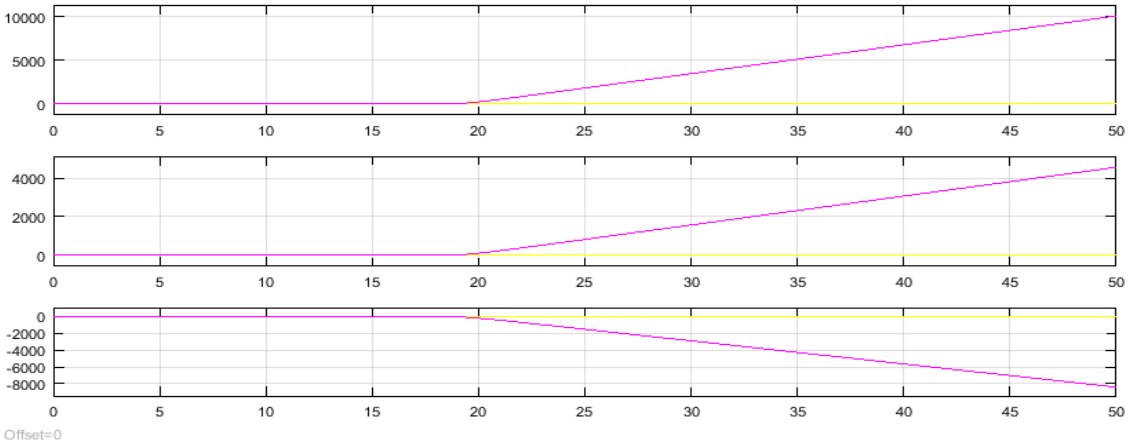

FIGURE 5.11 – Performance de contrôle des trois tâches dans Big core

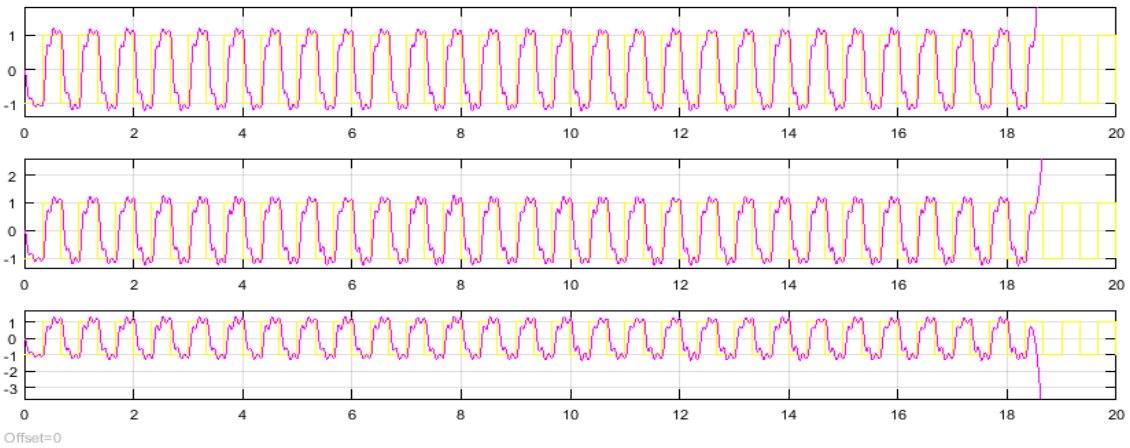

FIGURE 5.12 – Performance de contrôle des trois tâches dans Big core

Dans le cas de la performance de régulation sans FBS avec le cas d'utilisation **core Little** , la figure 5.13 montre stabilisation de système de contrôle

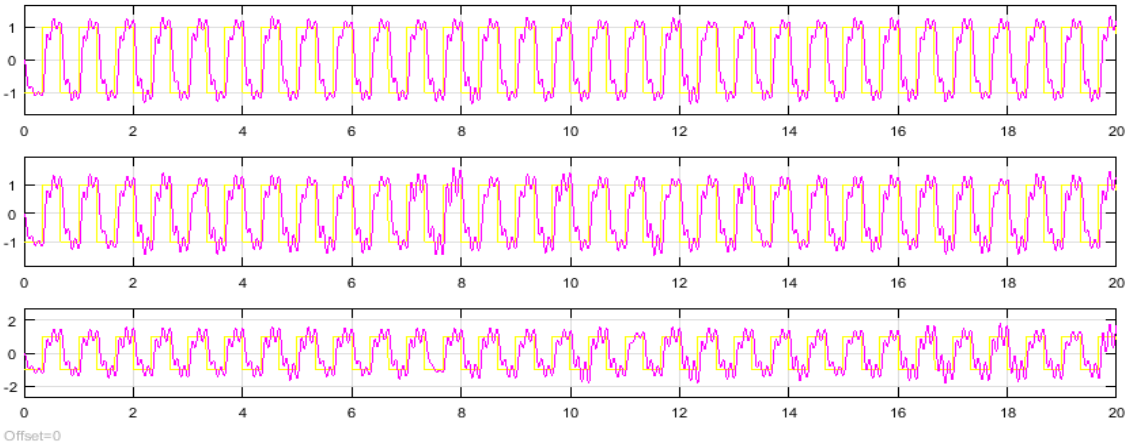

FIGURE 5.13 – Performance de contrôle des trois tâches dans Little core

Pour mesurer la qualité de contrôle, on définie l'erreur de contrôle e(t). Elle est la valeur absolue de la différence entre la référence $x(t)$ et la sortie $y(t)$ du procédé contrôlé, noté : $e(k) = |x(t) - y(t)|$

— Dans la cas d'utilisation little core on a trouvé comme mesure d'erreur la valeur

$$e_{little} = 4.9927e \times 10^4$$

On remarque que la valeur d'erreur très grand car on à utilisé une vitesse minimale de processeur et les commande ne fonctionné pas bien

— Dans la cas d'utilisation Big core on a trouvé comme mesure d'erreur la valeur

$$e_{big} = 4.1293e \times 10^4$$

On remarque que la valeur d'erreur inférieure que little core car on à utilisé une vitesse maximal de processeur mais ça cause le problème d'épuisement d'énergie.

— Dans la cas d'utilisation FBS on a trouvé comme mesure d'erreur la valeur

$$e_{fbs} = 4.5496e \times 10^4$$

On remarque que la valeur d'erreur de FBS est entre little et Big. D'après nos études, on déduit que utilisation de FBS si la meilleur façon.

5.3.5 Facteur de vitesse

Nous présentons les résultats des simulations montrant la variation de la vitesse du processeur(big.little). Dans cette figure 5.14 montre la basculement des fréquence choisies par le régulateur flou.

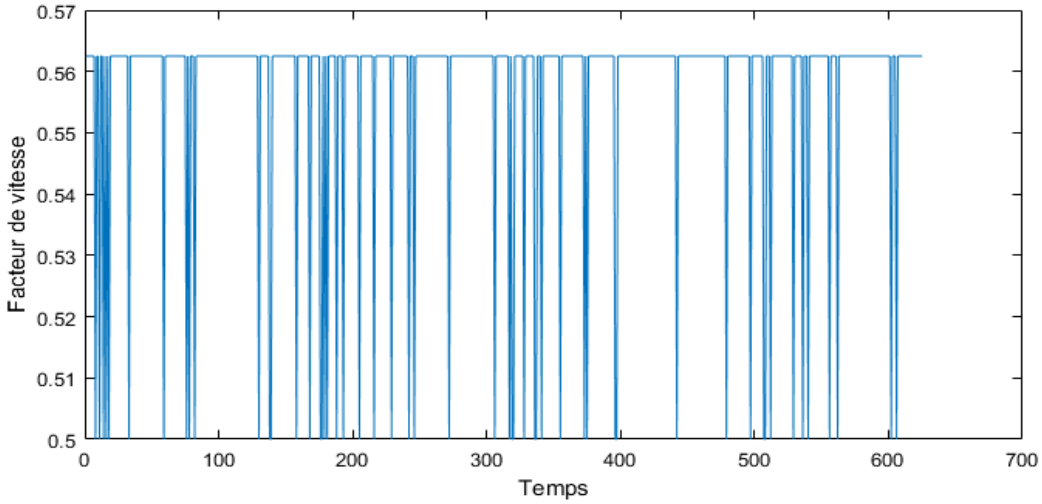

FIGURE 5.14 – facteur de vitesse par fréquence utilisé

La figure 5.15 montre la basculement entre processeurs dans le cas d'utilisation du régulateur flou.

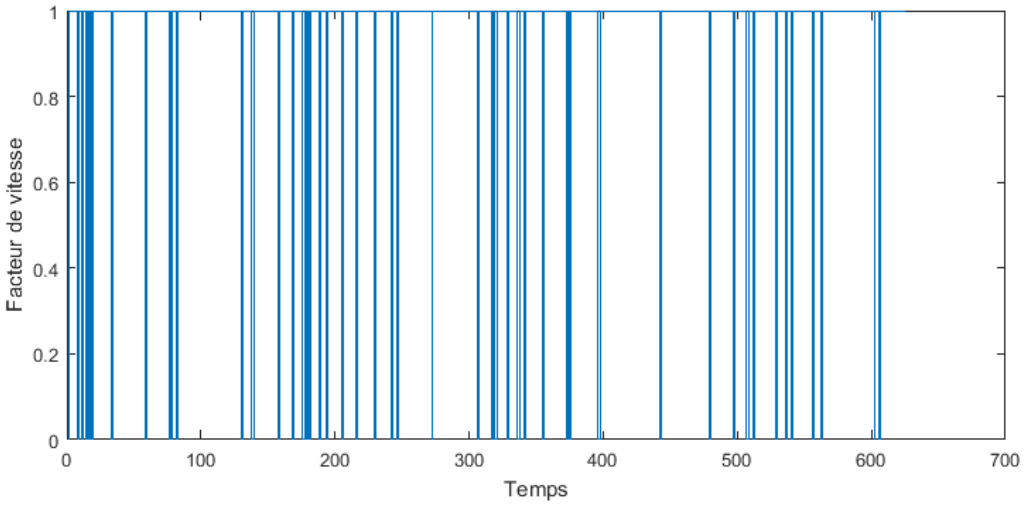

FIGURE 5.15 – facteur de vitesse par processeur utilisé

5.4 Conclusion

Dans ce chapitre, nous avons présenté l'environnement de simulation TrueTime. Par la suite, nous avons présenté les différents simulations et résultats obtenus. La solution que nous avons proposé, à travers le régulateur flou, minimisation la consommation d'énergie et d'optimiser la qualité de contrôle du système.

Conclusion générale

L'objectif de ce mémoire est l'étude du problème d'ordonnancement temps réel régulé multiprocesseur des tâches périodiques sous contraintes d'énergie renouvelable.
Pour cette étude, nous avons consacré une partie de notre travail à l'état de l'art, qui constitué de trois chapitres.

☞ Dans un premier chapitre, introduction sur les systèmes temps réel, dans laquelle nous avons introduit le problème de l'ordonnancement temps réel, ainsi que la modélisation des tâches temps réel. Ensuite, nous avons exposé les différentes algorithmes d'ordonnancement monoprocesseur et les approches multiprocesseur existant dans la littérature traitant les problèmes d'ordonnancement temps réel. Cette introduction nous permet d'avoir énorme connaissance dans le domaine de l'ordonnancement temps réel sous contraintes afin de mieux comprendre notre étude.

☞ Dans un deuxième chapitre, nous avons présenté le problème de l'ordonnancement temps réel sous les contraintes de consommation d'énergie du processeur. Nous avons constaté que les solutions proposées sous contraintes énergétiques ne prennent pas en considération l'ordonnancement régulier des tâches périodiques dans le cas de multiprocesseur. Nous notons que les techniques proposées dans ces solutions considèrent les pires durées d'exécution (WCET) pour les tâches, entraînant une sous-utilisation des ressources de traitement et de communication et des sur dimensionnements coûteux, notamment en consommation d'énergie.

☞ Dans un troisième chapitre, nous avons présenté les stratégies de commande des procédés. Ensuite, nous avons donné les paramètres, les critères de performances et les méthodes d'évaluation des systèmes de commande. Puis, nous avons abordé la problématique d'ordonnancement basé sur WCET. Nous avons parlé, de l'ordonnancement temps réel

régulé et des travaux proposées dans l'ordonnancement temps réel régulé.

☞ Dans un quatrième chapitre, nous nous sommes intéressé à la régulation floue d'ordonnancement temps réel multi-processeur pour les systèmes de récupération d l'énergie. Dans ce contexte, nous avons présenté les différents modèles du système considéré. Puis, nous avons décris la conception de régulateur flou. Il se compose de quatre éléments principaux : l'interface floue, Base de règles, mécanisme de raisonnement et interface de défuzzification.

☞ Dans un cinquième chapitre " Evaluations et Résultats ", nous avons décrit TrueTime. Puis, après la présentation modèle Simulink proposé en guise avec TrueTime, nous avons commenté les résultats obtenus de la simulation. Ces résultats obtenus montrent que les solutions proposées optimisent d'énergie et la qualité de contrôle de système.

Bibliographie

[1] O. Kermia, Ordonnancement temps réel multiprocesseur de tâches non préemptives avec contraintes de précédence, de périodicité stricte et de latence, These de Doctorat en science. 5

[2] P. Richard, F. Ridouard, Ordonnancement temps réel monoprocesseur. 5, 13, 14, 18

[3] F. Cottet, E. Grolleau, Systèmes temps réel de contrôle-commande : conception et implémentation, Dunod, 2005. 5

[4] [link].
URL https://stephanelegrand.files.wordpress.com/2013/09/memoire_sysembed.pdf 6

[5] M. Zoubir, Conception de systèmes temps réel, Ph.D. thesis, Université Toulouse (2004-2008). 6, 11, 12

[6] F. Many, Combinaison des aspects temps réel et sûreté de fonctionnement pour la conception des plateformes avioniques, Ph.D. thesis, Toulouse, ISAE (2013). 7, 10, 16

[7] S. Pailler, Analyse hors ligne d'ordonnançabilité d'applications temps réels comportant des tâches conditionnelles et sporadiques, Ph.D. thesis, Poitiers (2006). 7, 8

[8] Y. Khadidja, L'apport des outils de l'intelligence artificielle dans les systèmes temps réel : Ordonnancement des tâches, Ph.D. thesis, Université d'Oran (2013). 7, 9

[9] F. Cottet, N. DELACROIX, C. Kaiser, Z. Mammeri, Ordonnancement temps réel ordonnancement centralisé, Techniques de l'ingénieur. Informatique industrielle 3 (S8055) (1999) S8055–1. 9, 17, 18, 23, 24, 25

[10] I. Demeure, C. Bonnet, Introduction aux systèmes temps réel. 10

[11] C. L. Liu, J. W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, Journal of the ACM (JACM) 20 (1) (1973) 46–61. 11, 20, 22

[12] A. Rahni, Contributions à la validation d'ordonnancement temps réel en présence de transactions sous priorités fixes et edf, Ph.D. thesis (2008). 13

[13] P. Lopez, Approche par contraintes des problèmes d'ordonnancement et d'affectation : structures temporelles et mécanismes de propagation, Ph.D. thesis, Institut National Polytechnique de Toulouse-INPT (2003). 18

[14] S. Bouzefrane, Ordonnancement centralisé de tâches temps réel ordonnancement centralisé de tâches temps réel. 18, 20

[15] N. E. MOUHOUB, Algorithmes de construction de graphes dans les problèmes d'ordonnancement de projet, Ph.D. thesis (2014). 19

[16] J. Y.-T. Leung, J. Whitehead, On the complexity of fixed-priority scheduling of periodic, real-time tasks, Performance evaluation 2 (4) (1982) 237–250. 21

[17] A. K.-L. Mok, Fundamental design problems of distributed systems for the hard-real-time environment, Ph.D. thesis, Massachusetts Institute of Technology (1983). 22

[18] J. R. Jackson, Scheduling a production line to minimize maximum tardiness, management science research project. 22

[19] L. Sha, R. Rajkumar, J. P. Lehoczky, Priority inheritance protocols : An approach to real-time synchronization, IEEE Transactions on computers 39 (9) (1990) 1175–1185. 25

[20] T. P. Baker, Stack-based scheduling of realtime processes, Real-Time Systems 3 (1) (1991) 67–99. 25

[21] F. Dorin, Contributions à l'ordonnancement et l'analyse des systèmes temps réel critiques, Ph.D. thesis (2010). 26

[22] J.-B. Chaudron, Architecture de simulation distribuée temps-réel, Ph.D. thesis, Toulouse, ISAE (2012). 26

[23] M. Chetto, Ordonnancement dans les systèmes temps réel, ISTE Group, 2014. 27

[24] M. Chéramy, P.-E. Hladik, A.-M. Déplanche, Algorithmes pour l'ordonnancement temps réel multiprocesseur. 29

[25] Y. Chandarli, Gestion de l'énergie renouvelable et ordonnancement temps réel dans les systèmes embarqués, Ph.D. thesis (2014). 31, 38

[26] L. Bridier, Modélisation et optimisation d'un système de stockage couplé à une production électrique renouvelable intermittente, Ph.D. thesis (2016). 33

[27] Z. Zhang, Modélisation mécanique des interfaces multi-contacts dans une pile à combustible, Ph.D. thesis, Evry-Val d'Essonne (2010). 33

[28] M. Chetto, A. Queudet, Systèmes temps réel autonomes en énergie, Vol. 2, ISTE Group, 2017. 33

[29] [link].
URL www.fichier-pdf.fr/2017/06/20/sources-d-energie-renouvelable 33

[30] S. Basagni, M. Y. Naderi, C. Petrioli, D. Spenza, M. Conti, S. Giordano, I. Stojmenovic, Wireless sensor networks with energy harvesting., Mobile ad hoc networking 1 (2013) 701–736. 34, 40

[31] N. Navet, B. Gaujal, Ordonnancement temps réel et minimisation de la consommation d'énergie (2006). 34, 61

[32] F. Parain, M. Banâtre, G. Cabillic, T. Higuera, V. Issarny, J.-P. Lesot, Techniques de réduction de la consommation dans les systèmes embarqués temps-réel. 37

[33] M. Abdallah, Ordonnancement temps réel pour l'optimisation de la qualité de service dans les systèmes autonomes en énergie, Ph.D. thesis (2014). 37

[34] H. E. Ghor, M. Chetto, R. H. Chehade, A real-time scheduling framework for embedded systems with environmental energy harvesting, Computers & Electrical Engineering 37 (4) (2011) 498–510. 38

[35] M. Abdallah, M. Chetto, A. Queudet, Scheduling with quality of service requirements in real-time energy harvesting sensors, in : 2012 IEEE International Conference on Green Computing and Communications, IEEE, 2012, pp. 644–646. 38

[36] H. Zhang, Gestion de l'énergie renouvelable et ordonnancement temps réel dans les systèmes embarqués, Ph.D. thesis, Nantes (2012). 38

[37] S. Liu, Q. Wu, Q. Qiu, An adaptive scheduling and voltage/frequency selection algorithm for real-time energy harvesting systems, in : Proceedings of the 46th Annual Design Automation Conference, 2009, pp. 782–787. 39

[38] A. ABBAS, M. LOUDINI, W.-K. HIDOUCI, Energy and quality of control constraints in real-time scheduling of synchronous hybrid tasks, Journal of Control Engineering and Applied Informatics 15 (4) (2013) 86–96. 40

[39] A. Abbas, E. Grolleau, M. Loudini, D. Mehdi, A real-time feedback scheduler for environmental energy harvesting, in : 3rd International Conference on Systems and Control, IEEE, 2013, pp. 1013–1019. 40

[40] A. Abbas, E. Grolleau, M. Loudini, W.-K. Hidouci, A real-time feedback scheduler based on control error for environmental energy harvesting systems, in : 2015 International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), IEEE, 2015, pp. 1–9. 40

[41] Y. El Afou, Contribution au contrôle des paramètres climatiques sous serre, Ph.D. thesis, Lille 1 (2014). 43

[42] A. ABBAS, Application de techniques de commande par rétroaction pour l'ordonnancement temps réel des tâches dans les systemes industriels, Ph.D. thesis, Ecole Nationale Supérieure d'informatique (2016). 44, 49

[43] L. Markus, Feedback scheduling, Ph.D. thesis, UNIVERSITAIRE EUROPÉEN BRUXELLES WALLONIE (2010). 46, 48

[44] P. Andrianiaina, Commande robuste avec relâchement des contraintes temps-réel, Ph.D. thesis (2012). 47

[45] R. HAMID, Ordonnancement temps réel des tâches dans un système industriel par intégration d'une loi de régulation automatique, Ph.D. thesis, Université Bouira (2013). 47, 58

[46] D. Simon, Conception conjointe commande/ordonnancement et ordonnancement régulé. 48

[47] T. Amieur, Commande des systèmes non linéaires par mode glissant flou, Ph.D. thesis, Université Mohamed Khider-Biskra (2009). 49, 50

[48] B. Kadmiry, D. Driankov, A fuzzy flight controller combining linguistic and model-based fuzzy control, Fuzzy Sets and Systems 146 (3) (2004) 313–347. 50

[49] B. Kadmiry, D. Driankov, A fuzzy gain-scheduler for the attitude control of an unmanned helicopter, IEEE Transactions on fuzzy systems 12 (4) (2004) 502–515. 50

[50] S. K. Tso, Y. Fung, Methodological development of fuzzy-logic controllers from multivariable linear control, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 27 (3) (1997) 566–572. 50

[51] B.-D. Liu, C.-Y. Chen, J.-Y. Tsao, Design of adaptive fuzzy logic controller based on linguistic-hedge concepts and genetic algorithms, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 31 (1) (2001) 32–53. 50

[52] L. Gacôgne, Eléments de logique floue, Hermès, 1997. 53, 54

[53] D. Ran, B. Ryan, P. Power, Using fuzzy logic toward intelligent system (1994). 54

[54] M. Nezar, Diagnostic des associations convertisseurs statique-machines asynchrones en utilisant les techniques de l'intélligence artificielle, Ph.D. thesis, Université de Batna 2-Mustafa Ben Boulaid (2006). 55

[55] H. Hellendoorn, Closure properties of the compositional rule of inference, Fuzzy sets and systems 35 (2) (1990) 163–183. 56

[56] B. Islam, N. Ahmed, D. Bhatti, S. Khan, Controller design using fuzzy logic for a twin rotor mimo system, in : 7th International Multi Topic Conference, 2003. INMIC 2003., IEEE, 2003, pp. 264–268. 57

[57] A. Cervin, J. Eker, B. Bernhardsson, K.-E. Årzén, Feedback–feedforward scheduling of control tasks, Real-Time Systems 23 (1-2) (2002) 25–53. 58

[58] I. S. Divya Pathak, Houman Homayoun, Work load scheduling for multi core systems with under-provisioned power delivery, RIGHTSLINK canada. 63

[59] M. O. Anton Cervin, Dan Henriksson, TRUETIME 2.0 beta—Reference Manual. 71

[60] [link].
URL http://www.control.lth.se/truetime/ 72

Abstract

In our work we have dealt with the problem of multiprocessor regulated real-time scheduling under constraints of renewable energy and critical resources in the system Regulated real-time scheduling consists in applying the principle of closed-loop regulation (feedback control) on computer systems with the aim of improving their performance during their execution. We have highlighted the details of our approach relating to the inclusion of a fuzzy logic control in order to adjust the parameters of the tasks characterized by the variation of their run times over time.
The solution we proposed in our contribution Fuzzy multi-processor (Big core and Little core) real-time scheduling regulation for energy recovery systems on a variable speed multiprocessor powered by a battery which is recharged by a renewable energy (solar energy). the specific objectives in our study duration of tasks, control error and minimization of energy consumption and to optimize the quality of control of the system.

Key words : multiprocessor scheduling, Embedded system, Real time scheduling, Regulated real time scheduling, Periodic and aperiodic tasks, Consumption reduction, Energy recovery, Fuzzy logic, Big.Little .

Résumé

Dans notre travail nous avons traite le problème d'ordonnancement temps réel régule multiprocesseur sous contraintes d'énergie renouvelable et de ressources critiques dans le système L'ordonnancement temps réel régule consiste a appliquer le principe de régulation a boucle fermée (commande par rétroaction) sur des systèmes informatiques dans le but d'améliorer leurs performances pendant leur exécution .Nous avons mis en évidence les détails de notre approche ayant trait à l'inclusion d'un contrôle logique floue afin d'ajuster les paramètres des tâches caractérisées par la variation de leurs durées d'exécution dans le temps.
La solution que nous avons proposée dans notre contribution Régulation floue d'ordonnancement temps réel multi-processeur(Big core et Little core) pour les systèmes de récupération de l'énergie sur un multiprocesseur a vitesse variable alimente par une batterie qui est rechargée par une énergie renouvelable(énergie solaire). L'objectifs propres dans note étude durées d'exécution des tâches,l'erreur de contrôle et minimisation la

consommation d'énergie et d'optimiser la qualité de contrôle du système.

Mots clés :ordonnancement multiprocesseur , Système embarqué , Ordonnancement temps réel , Ordonnancement temps réel régulé , Taches périodiques et apériodiques,Réduction de consommation , Récupération de l'énergie ,Logique floue, Big.Little .