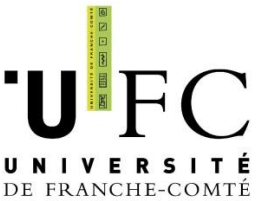

UNIVERSITÉ DE FRANCHE-COMTÉ

CENTRE DE TÉLÉ-ENSEIGNEMENT UNIVERSITAIRE

Master Informatique Avancé et Applications (I2A) option Recherche

Mémoire de fin d'Etudes

COMPARAISON D'HEURISTIQUES D'ORDONNANCEMENT DES TÂCHES SUR DES MACHINES PARALELLES

Etudiant :
Parfait Pascal Wangun

Encadrant de Recherche :
Laurent Philippe

1. Ière Partie : Introduction générale

a. Contexte et justification

Le développement fulgurant des réseaux informatiques a favorisé l'émergence des environnements de calcul distribués permettant l'exécution rapide des programmes informatiques. Ces environnements de calculs peuvent être répartis en deux catégories :

- Les environnements de calcul distribués homogènes ;
- et les environnements de calcul hétérogènes.

Contrairement aux environnements de calculs homogènes où les ressources de calcul sont toutes identiques, dans un environnement de calcul hétérogène les ressources de calcul présentent un certain nombre de différences. Ces différences sont entre autres, la puissance de calcul des processeurs, la capacité de la mémoire, la bande passante de transfert des informations, le nombre de processeurs etc... La problématique de l'ordonnancement des applications dans les systèmes de calcul hétérogène a jusqu'ici fait l'objet de nombreuses études dans la littérature, l'objectif étant de trouver une solution optimale à cette question d'ordonnancement. La recherche d'une solution optimale au problème d'ordonnancement des tâches sur des machines parallèles, a suscité le développement de nombreuses heuristiques dans le but de proposer à chaque fois une solution améliorée. En effet, il s'agit d'un problème NP-complet pour lequel il n'existe pas à priori une solution efficace.

b. Problématique

Il existe de nombreuses heuristiques développées dans la littérature, relatives à l'ordonnancement des tâches sur des machines parallèles. Il est fort de constater que les performances des algorithmes sont évalués afin d'identifiés celui qui offre les meilleures performances. Les données de comparaison ne sont pas toujours les même. D'un article à un autre l'évaluation ne prend pas toujours en compte les mêmes paramètres, valeurs et critères de comparaison. Comment définir un protocole expérimental qui permet de faire une comparaison équitable des algorithmes ?

Ce travail tire toute sa raison d'être sur la nécessité de proposer un protocole expérimental permettant de faire une comparaison équitable des algorithmes et identifier l'algorithme qui offre de meilleures performances.

c. Méthodologie

Les heuristiques présentées dans la littérature sont généralement mises en relief par rapport à celles déjà existantes. L'objectif des auteurs étant de mesurer les performances des algorithmes proposés par rapport à celles qui existent. En effet, le développement de

nouveaux algorithmes vise à proposer une solution améliorée par rapport à celles qui existent déjà. Il s'agit par exemple du cas de l'article [2] où l'auteur Hamid Arabnejad et al. procède à la selection des algoithmes avec lequel il voudrait faire la comparaison. Dans ce cas d'espèce, il s'agit de HEFT,HCP,HPS PETS et Lookahead. Après une brève description de ceux-ci, l'auteur propose un algorithme et procède ensuite à des études expérimentales pour effectuer la comparaison des performances. La comparaison peut être faite en générant de manière aléatoire des graphes ou alors en s'appuyant sur des cas réels de certaines applications. Dans de l'article [23], Haluk Topcuoglu et al. S'appuie les graphes de trois applications réels pour effectuer la comparaison. Il s'agit des applications de transformation de fourrier, le code dynamique moléculaire, et l'algorithme de l'élimination de Gauss. Les tests sont faits par le moyen de simulateur qui permet de tracer sur des courbes les performances des différents algorithmes.

Le travail que nous nous proposons de réaliser vise à faire une comparaison des heuristiques d'ordonnancement. Il est important d'entrée de jeu de rappeler que les systèmes de calcul parallèles permettent définir des architectures permettant de traiter des informations de manière simultanée, ainsi que de définir les algorithmes spécialisés pour celles-ci. Ces techniques ont pour but de réaliser un plus grand nombre d'opérations en un temps le plus court possible. La question de l''ordonnancement consiste donc à distribuer une application sur une machine parallèle de manière à obtenir de meilleures performances ou de traiter les problèmes plus larges. Il s'agit d'un problème NP-difficile pour lequel il n'existe pas de solution optimale. De nombreuses heuristiques ont dont été développées proposant les unes les autres des solutions approximatives et partielles.

Il est question au cours de cette étude de définir ce qu'est l'ordonnancement, de procéder à l'état de l'art en identifiant et en présentant quelques heuristiques existantes dans la littérature, ensuite nous ferons un bilan comparatif de ces heuristiques. Nous proposerons un protocole expérimental qui nous servira de base de comparaison et enfin nous procéderons à la simulation des de l'exécution des heuristique avant de procéder à l'analyse des données obtenues. Nous achèverons ce travail par discussion sur les résultats obtenus de l'analyse, ce qui nous permettra de tirer une conclusion.

Ce travaille vise donc à faire une étude exhaustive des algorithmes afin de les comparer leur performance sur différents types de DAG.

2. IIème Partie : Présentation et Description de quelques algorithmes d'ordonnancement

a. Qu'est-ce l'ordonnancement des tâches?

L'ordonnancement est un processus consistant à affecter des tâches à des ressources de calcul pour traitement. L'environnement de traitement des tâches pouvant être homogène ou hétérogène. L'efficacité de l'exécution des applications parallèles dépend des méthodes utilisées pour ordonnancer les tâches. L'objectif étant d'attribuer des tâches aux processeurs, ordonner leur exécution de manière à satisfaire la relation de précédence des tâches et améliorer les performances. Ces performances sont définies par un ensemble de critères parmi lesquels le temps d'exécution, le makespan etc... La meilleure façon d'aborder le traitement/l'exécution d'une application de grande taille consiste à la diviser/partitionner en un ensemble de tâches en identifiant les relations de dépendance entre ces tâches. Celles-ci sont modélisées par un graphe de dépendance orienté sans circuit. Le graphe permet ainsi d'établir les relations d'interdépendance entre les tâches (quelle tâche doit-elle être exécutée avant quelle autre ?) et inclut par ailleurs les caractéristiques d'une application telles que : le temps d'exécution des tâches, la taille des données qui sont transférées entre les tâches, et l'interdépendance des tâches.

Le problème d'ordonnancement des tâches d'un graphe orienté sans circuit est NP-difficile aussi bien pour le cas des systèmes homogène que celui du cas des systèmes hétérogènes.

Etant donné l'importance de la question d'ordonnancement, elle a longuement été étudiée, et les travaux sur le sujet ont permis de classer les algorithmes d'ordonnancement plusieurs catégories : on peut par exemple citer les algorithmes d'ordonnancement de liste, les algorithmes de clustering, les algorithmes basés sur la duplication, et les algorithmes basé sur les méthodes de recherche aléatoires. Notre étude reposera essentiellement sur les algorithmes d'ordonnancement de type liste.

b. Généralités sur la problématique de l'ordonnancement des tâches sur des machines parallèles

Un système d'ordonnancement est un ensemble composé d'une application et un environnement de calcul cible. Le principe de l'ordonnancement consiste à partitionner une application en tâches de façon à pourvoir la modéliser par un graphe orienté sans cycle (DAG), $G = (V, E)$ où V est un ensemble de v tâches et E un ensemble de e extrémités entre les tâches. Chaque extrémité $(i, j) \in E$ représente les contraintes de précédence entre les tâches (la tâche ni doit s'achever avant la tâche nj). Une tâche d'entrée (entry task) est une tâche ne possédant aucun parent/prédécesseur), une tâche de sortie (exit task) est une tâche ne possédant aucun successeur/tâche fille. L'environnement de calcul

est un ensemble Q de q processeurs hétérogènes interconnectés. La communication inter processeur se fait sans contention.

La problématique d'ordonnancement des tâches consiste donc à affecter les tâches d'une application donnée sur des processeurs. Des critères permettent d'évaluer la performance parmi lesquels : le $makespan, le\ \ SLR$(Schedule length ratio), le temps d'exécution de l'agorithme.

$$Makespan\ =\ max\{AFT(n_{exit})\}$$

Le problème de l'ordonnancement se résume donc à la reduction du cout d'exécution d'une application en réduisant autant que possible le$makespan.$ De nombreuses variables et paramètres sont pris en compte dans le calcul du $makespan.$

Nome de variale	descriptif
V	Un ensemble de tâches
V_s	Un ensemble de tâche ordonnacées
V_{us}	Ensemble de tâches non encore ordonnancées
E	Ensemble de relations entre les tâches
$\|S\|$	Cardinalité de lensemble S
$n = \|V\|$	Nombre total des tâches
$m = \|P\|$	Nombre total des processeurs
t_i	Ième task
P_j	J ième proesseur
$e_{i,j} = (t_i, t_j)$	Une relation orienté partant de t_i vers t_j
$W_{i,j}$	temps d'exécution de la tâche i sur le processeur j
$C_{i,j}$	Temps de communication entre les processeurs i et j
$\gamma(t_i, t_j)$	Le temps pris pour transférer les données de t_i vers t_j
$X_{n\times m}$	Mtrice des couts d'exécution de dimension $n \times m$
$x_{i,j}$	Coût d'exécution des taches
x_i^{est}	Le coût estimatif de la tache t_i sur tout le processus
x_i^*	
$pred(t_i)$	Ensemble des predecesseurs immédiats à la tache t_i
$succ(t_i)$	Ensemble de successeurs immédiats à la tache t_i
ESF	Le temps de début au plus tôt d'une tâche
EFT	Le temps de fin au plus tôt d'une tâche
ASF	...
AFT	...

DA	Temps de disponibilié des données
PA	Temps de disponibilité du processeur
SL	Cout de l'ordonnancement
TL	
BL	

Tableau 1 paramètre de calcul

c. Les algorithmes d'Ordonnancement de Type Liste

Les algorithmes d'ordonnancement avec lesquels nous travaillons dans le cadre de cette étude sont ceux de type liste. Il s'agit des algorithmes qui fonctionnent en deux phases : la phase de priorisation des tâches et la phase d'ordonnancement. En effets, Ils déterminent pour un ordre de tâches, qui peut être donné par une liste, un ordonnancement correspondant. Dans un ordonnancement dit de liste, des ordres de priorités sont associés à chaque tâche et placés ensuite dans une liste ordonnée de façon décroissante des ordres de priorité. La tâche ayant une plus forte priorité sera ordonnancée avant celle de faible priorité. Dans le cas où deux tâches posséderaient la même priorité, une méthode spécifique peut être utilisée pour les départager ou alors le choix peut tout simplement se faire de manière aléatoire.

L'ordonnancement de type liste se déroule en deux phases :

- Phase de calcul de priorité des taches
- La phase de sélection du processeur

La différence qui existe sur les nombreux heuristiques dont l'ordonnancent est basé sur les listes réside sur les méthodes de calcul des priorités et celle de sélection du processeur. Les critères majeurs pour attribuer les priorités au taches sont en général le top level ou $t-level$ (TL) et le bottom level ou $b-level$ (BL).

Le top level (t-level) d'une tâche est défini comme étant la longueur du plus long chemin (coût d'exécution + coût de communication) entre cette tâche et celle d'entrée. Le TL permet d'évaluer le paramètre EST (earliest start time) d'une tâche.

Le bottom level (b-level) est la longueur du chemin le plus long (coût d'exécution + coût de communication) partant de cette tache vers la tâche de sortie. Le BL est borné par le chemin critique du graphe. Lorsque deux taches sont ordonnancées sur le même processeur, le poids de l'extrémité reliant les deux tâches est égal à zéro, ce qui implique que le TL d'une tache change et donc peut être dynamique. Lorsqu'un algorithme utilise un TL qui change pendant le processus d'ordonnancement, cet algorithme est qualifié d'algorithme dynamique. Dans le ca s contraire si l'algorithme utilise un TL déterminé avant le démarrage du processus d'ordonnancement, cet algorithme est qualifié d'algorithme statique.

Les heuristiques qui feront l'objet de cet étude sont basés sur l'ordonnancement de type liste et cible principalement les environnements de calcul hétérogènes.

d. Analyse des différents protocoles expérimentaux

Dans ce chapitre, nous allons parcourir quelques algorithmes identifiés dans la littérature et procéder à une analyse de manière à procéder au un bilan comparatif des protocoles expérimentaux. Mais avant, nous allons faire une analyse des protocoles expérimentaux utilisés pour évaluer les performances des algorithmes. Les algorithmes qui feront l'objet de notre étude sont les suivants : HEFT et CPOP, PEFT, PETS.

En général, l'analyse des performances d'un algorithme d'ordonnancement de tâche donne lieu à la mise sur pied d'un protocole expérimental. Le but du protocole expérimental étant de définir le cadre qui permettra de faire les analyses, en identifiant les paramètres et critères d'analyse ainsi que les valeurs associées. Nous analyserons pour chaque protocole expérimental, les éléments fondamentaux qui le constituent :

- ✓ Génération des DAG
- ✓ Choix des paramètres
- ✓ Choix des valeurs
- ✓ Identification des paramètres de mesure des performances
- ✓ Analyse des résultats

1. Génération des DAG

Comme nous l'avons vu précédemment, les DAG permettent de modéliser les applications pour lesquelles l'ordonnancement doit être effectué. Ces applications sont définies sous forme de graphe de taches acyclique. Les DAG sont obtenus par génération aléatoire. Un générateur de DAG est donc un simulateur qui prend un certain nombre de paramètres d'entrées et génères les DAG en fonction des caractéristiques souhaitées. Les caractéristique d'un DAG pouvant être , la forme du DAG, le niveau d'hétérogénéité, le type de communication….

2. Choix des paramètres d'entrées

La génération aléatoire des graphes se fait par un générateur qui peut prendre en entrée un certain nombre de paramètres. Parmi ces paramètres, on peut citer:

- ✓ Le nombre de tâches
- ✓ Le nombre de processeurs
- ✓ Le facteur d'hétérogénéité (coefficient de variation des processeurs, coefficient de variation de la communication entre les taches)

- ✓ Le CCR (computeur communication ration). Ce paramètre lorsqu'on le fait varié permet de générer un DAG ou les communications sont intensives ou non.
- ✓ La forme du graphe (le nombre de niveau)
- ✓ Coefficient de variation des tâches par niveau

3. Le choix des valeurs

Une fois les paramètres de générations identifiés, il est important de définir une plage des valeurs qui seront utilisés. Pour chaque paramètre identifié on choisit une plage dans laquelle ces paramètres tirent leur valeur.

4. Identifications des paramètres d'évaluation des performances

Les graphes lorsqu'ils sont générés sont soumis à un simulateur. Le rôle du simulateur étant d'implémenter un algorithme et l'exécuter. L'exécution de l'algorithme donne lieu à un résultat. Ce résultat doit donc être analysé et comparer aux résultats des autres algorithmes. La comparaison des résultats prend en compte des critères d'évaluations. Ces critères sont définis par des paramètres bien identifiés qui sont :

SLR (schedule length ration) : Le paramètre permettant de mesurer les performances d'un algorithme c'est le makespan. Ce paramètre n'est rien d'autre que le temps d'exécution de la tâche de sortie. Compte tenu du fait que les DAG ont des caractéristiques différentes en fonction des valeurs des paramètres de génération de graphe, il est important de trouver une valeur normaliser permettant de mieux qualifier le coût de l'ordonnancement. Le SLR est donc une valeur moyenne qui sur plusieurs graphes de caractéristiques différentes permet de mesure le coût moyen d'exécution de l'algorithme.

- Speedup
- Numbers of occurrences of best quality schedule : le nuombre de fois que chaque algorithme produit un meilleur résultat
- Running time schedule : temps d'exécution de l'algorithme.
- Facteur d'accéllération(speedup)
- Efficacité(Efficiency)
- Nombre d'apparition de meilleure qualité d'ordonnancement
- Temps d'exécution de l'algorithme

5. Analyse des résultats

L'analyse des résultats est faite grâce aux courbes. Les courbes représente en général un nuage de point dont les coordonnées correspondant à la paire constituées des paramètres d'entrées de génération des graphes (CCR, nombre de tâches, nombre de processeurs) et les paramètres de mesure des performances (SLR,speedup, temps d'exécution…).

L'analyse de ces courbes permet de tirer les conclusions sur les performances des algorithmes.

3. IIIème Partie : Comparaison des heuristiques

Bilan comparatif des protocoles d'évaluation

Dans la deuxième partie de ce document nous avons fait une synthèse des protocoles expérimentaux des différents algorithmes soumis à notre étude. Il s'agit plus spécifiquement des algorithmes suivants : HEFT,PEFT,PETS,CPOP. Le bilan comparatif que nous nous proposons de réaliser a pour objectif d'analyser les protocoles expérimentaux des différents algorithmes afin d'identifier les paramètres ainsi que les valeurs utilisés. Les questions qui viennent à se poser sont les suivantes : Nos algorithmes ont-ils les mêmes bases d'évaluation ? Sinon, quelles sont les variations qui existent entre le choix des paramètres ainsi que les valeurs ?
Les réponses à ces questions nous permettront de proposer un protocole expérimental auquel nous soumettrons ces algorithmes.

Dans un premier temps nous allons identifier les différents algorithmes avec lesquels ils sont évalués. Le tableau suivant nous donne une correspondance entre les algorithme de notre étude et les différents autres algorithmes avec lesquels ils sont évalués :

I.1. Identification des algorithmes de comparaison

	PEFT	*MH*	*HEFT*	*CPOP*	*PETS*	*LMT*	*DLS*	*HCPT*	*HPS*	*LookAhead*
HEFT		√		√		√	√			
CPOP		√	√			√	√			
PEFT			√		√	√	√	√	√	√
PETS			√	√		√				

Tableau 2: Tableau de comparaison des heuristiques

A la lecture du tableau 2, nous pouvons remarquer que deux algorithmes reviennent dans la comparaison. Il s'agit de HEFT et LMT. En effet dans la littérature HEFT est présenté comme l'algorithme offrant les meilleures performances. PETS se positionne comme étant le premier algorithme dont les performances celles de HEFT. PEFT présente un algorithme offrant des performances bien meilleures que HEFT et PETS.

PEFT serait-il l'algorithme offrant les meilleures performances ? Ce travail par la suite sera l'occasion de répondre à cette question. Il est question de proposer un protocole expérimental qui servira de base d'évaluation afin de tirer les conclusions.

I.2. Paramètres de génération des DAG

Dans le tableau suivant nous énumérons la liste des paramètres de génération des DAG. Sur cette liste nous celles qui sont identifiés par chacun des algorithmes.

		PEFT	HEFT	CPOP	PETS	DLS
CCR	Taux de communication. Si la valeur est faible alors il s'agit d'un graphe offrant une communication intensive entre les tâches	√	√	√	√	√
v	Nombre de tâches	√	√	√	√	√
P	Nombre de processeurs	√	√	√	√	√
α	Forme du graphe du graphe si >1.0 graphe dense, si <1.0 graphe avec un dégrée de parallélisme faible	√	√	√		
out_degree	Dégrée sortant d'un noeud		√	√		
β	Facteur d'hétérogenéité	√	√	√		
$density$		√				
$regularity$		√				
$jump$		√				

Tableau 3: Tableau d'identification des paramètres de génération des DAG

On peut constater qu'il existe des paramètres spécifiques à certains algorithmes, tandis que qui interviennent dans tous les algorithmes. Parmi les paramètres constants, on peut citer : le nombre de tâches, le nombre de processeur, le CCR, le facteur d'hétérogénéité.

I.3. Choix des valeurs

Dans ce tableau nous allons répertorier les différentes valeurs associées aux paramètres de génération des graphes associés à chaque algorithme.

		PEFT	HEFT	CPOP	PETS	DLS
v	Nombre de tâches	[10,20, 30, 40, 50.,60, 70, 80, 90, 100,200 300.,400. 500]	[20, 40,60, 80, 100]	[20, 40,60, 80, 100]	[30, 40, 50, 60, 70, 80,. 90,100]	
CCR	Taux de communication . Si la valeur est faible alors il s'agit d'un graphe offrant une communication intensive entre les tâches	[0.1, 0.5,0.8,1,2, 5. 10]	[0.1, 0.5,1. 5. 10]	[0.1, 0.5,1. 5. 10]	[0:1,0:5, 1. 5. 10]	
P	Nombre de processeurs	[2. 4. 8. 16. 32]				
α	Forme du graphe du graphe si >1.0 graphe dense, si <1.0 graphe avec un dégrée de parallélisme faible	[1,2,3,4,5,v]	[0.5,1,2]	[0.5,1,2]	[0.5,1,2]	
out_degre	Dégrée sortant d'un noeud		{1,2,3,4,5}	{1,2,3,4,5}		
β	Facteur d'hétérogenéité	[0:1. 0:2. 0:5. 1. 2]	[0.1. 0.25. 0:5. 0.75. 1]	[0.1. 0.25. 0:5. 0.75. 1]	[1,2,3,4, 5]	
$density$						
$regularity$						
$jump$						

Tableau 4: Identification des valeurs

Ce que nous remarquons de ce tableau est que les valeurs des paramètres tels que α, β et CCR les plages dans lesquels les valeurs sont tirées appartiennent à la même plage.

Par contre nous constatons une variation des valeurs relatives au nombre de tâche. A la lecture des courbes d'évaluation des performances, nous remarquons pour HEFT par exemple, lorsque le nombre de tâches égale à 40, l'écart des performances entre les algorithmes en comparaison devient assez significative. Entre 40 et 100 tâche la courbe sui la même progression d'évolution tout en maintenant constant l'écart de performance avec les autres algorithmes. En ce qui concerne PEFT, nous remarquons l'évolution que la courbe de performance n'est pas constante et varie en fonction du nombre de tâches (confère la figure suivante). Nous constatons que ce n'est qu'à partir du nombre de tache égale à 400 que l'écart de variation entre les algorithmes devient constant.

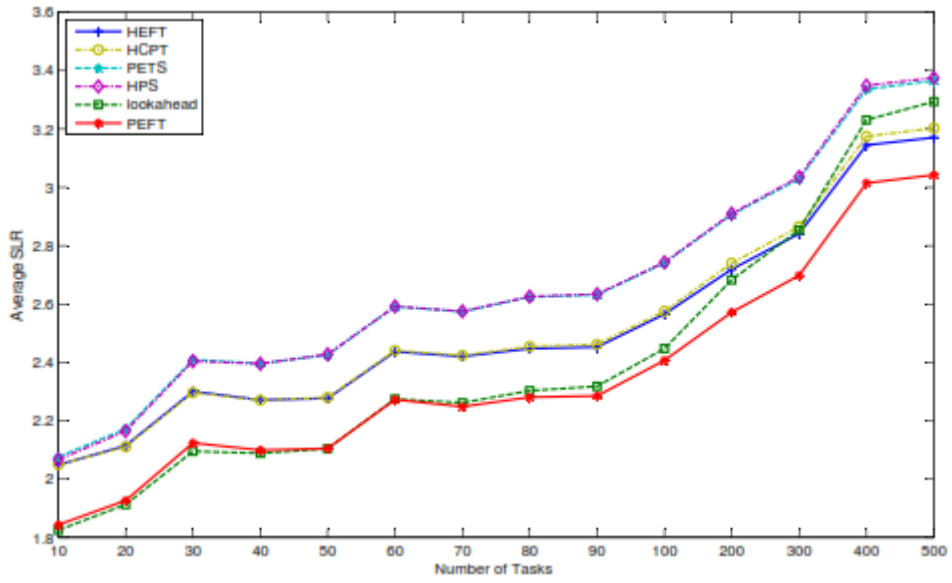

Figure 1: Diagramme extrait de l'algorithme PEFT tirer de l'article [2]

I.4. Paramètres de comparaison des graphes

La comparaison des performances tient compte d'un certain nombre de paramètres qui définissent les critères de comparaison entre les algorithmes. Le choix de ces paramètres est déterminant pour garantir la fiabilité ainsi que la qualité des résultats de comparaison.

Dans le tableau ci-dessous, nous avons identifié les paramètres utilisés dans chaque algorithme. Il ressort de ce tableau, des paramètres constants, utilisés dans tous les protocoles expérimentaux. Il s'agit entre autre du SLR, Excuting time,number of occurrences of better result.

Le SLR ou schedule length ration est un parameter permettant de mesurer la durée moyenne de l'ordonnancement. En faisant varier un ou plusieurs paramètre de génération des tâches la valeur du SLR permet d'évaluer la durée moyenne de l'ordonnancement. Ce paramètre est plus significative que le makespan.

		PEFT	HEFT	CPOP	PETS
SLR		√	√	√	
$efficiency$		√			
$slack$		√			
$speedup$		√	√	√	
$density$		√			
$regularity$		√			
$jump$		√			
$running\ time\ of\ algorithm$			√	√	
$number\ of\ occurences\ of\ better\ result$			√	√	

Tableau 5: Paramètres d'évaluation des heuristiques

D'un algorithme à un autre les métriques de comparaison peuvent varier. Les paramètres les plus significatives pour la comparaison et qui reviennent dans tous les articles sont les suivants ;

- ✓ SLR
- ✓ Speedup
- ✓ Temps d'execution de l'agorithme
- ✓ Nombre d'occurrences du meilleurs résultat

Proposition d'un protocole expérimental

Objectif

L'objectif de ce protocole expérimental est de proposer une base commune à tous les algorithmes pour l'évaluation des performances. Eu égard de ce qui précède, il est fort de constater dans chaque article, l'algorithme présenté est évalué sur la base des critères spécifiques. Il sera donc question de définir des critères de performance que nous allons appliquer à nos algorithmes.

En termes de critère de performance, nous faisons allusion ici aux paramètres de génération des DAG, les valeurs utilisées ainsi que le choix de paramètres de d'évaluation.

Méthodologie

Notre approche méthodologique s'appuie sur le bilan comparatif que nous avons effectué. La synthèse qui découle du bilan comparatif nous permet de proposer des paramètres et des valeurs nous permettant d'implémenter un simulateur capable de générer les DAG, exécuter les algorithmes et à l'exécution et générer des données de sortie. Les données de sorties seront ensuite analysées selon des critères bien définie

Plus précisément il s'agit de la construction d'un simulateur qui nous permettra d'une part de générer les graphes de tâches (DAG), d'autre part d'exécuter algorithmes, et enfin d'analyser les données générées.

Choix des technologies

Nous travaillons avec deux lanages : ptyhon et Java.

Les DAG sont générés par un générateur développé en python sous forme d'un fichier gml. Le générateur que nous utilisons dans le cadre de ce travail a été mis à notre disposition par notre encadreur et provient du travail de recherche d'un étudiant.

Les algorithmes sont implémentés en Java. Pour ce faire nous implémentons un parseur permettant de convertir les fichiers d'extension gml en graphe de tâches acyclique.

Les algorithmes sont par la suite implémentés en JAVA. Les résultats de l'exécution de ces algorithmes sont générés dans un fichier csv. Ces données sont ensuite transmises dans un programme python pour la génération des courbes de nuages de point.

Génération des DAG

Nous proposons un générateur de DAG qui implémenté en langage python. Ce générateur est le fruit du travail d'un étudiant et mis à notre disposition par notre encadreur. Le générateur prend en entrée les paramètres suivants :

Pour générer les DAGs nous utilisont un programme developpé avec le langage python. La fonction de génération prend en entrée les paramètres suivants : length, depth, filename, sdComp, sdComm, CCR, nbproc et génère en sorties un fichier GML représentant le DAG

- :param length: Length of the graph (number of nodes).
- :type length: int
- :param depth: Depth of the graph (number of levels)
- :type depth: int
- :param filename: Output filename
- :type filename: str
- :param sdComp: Standard Deviation of computations costs
- :type sdComp: float
- :param sdComm: Standard Deviation of communications costs
- :type sdComm: float
- :param CCR: Communications to Computations Ratio
- :type CCR: float
- :param nbproc: Number of processors used when generating the graph
- :type nbproc: int
- :return: Generated and converted graph

De façon précise, ces paramètres sont :

- ✓ Nombre de tâches : n
- ✓ CCR (Computer to communication ration)
- ✓ Nombre de processeurs
- ✓ Le coefficient de variationt des processeurs qui nous permettra de définir le dégré d'homogenéité ou d'hétérogénéités des processeur
- ✓ Coefficient de variation de la communication ente processeur
- ✓ Le nombre de niveaux du DAG

Ces paramètres nous permettent de définir des DAG de différentes topologies cette variation de la structure des DAG nous permettra d'enrichir nos analyses afin de produire un résultat pertinent.

Le coefficient de variation des processeurs permet de définir le niveau d'homogenéité des processeurs. Lors que cette valeur est petite, cela signifie que les processeurs sont presque homogènes et lors que cette valeur est grande, celà traduit la forte hétérogénéité entre les Processeurs.

Le CCR est un paramètre qui lorsqu'on le fait varier permet de définir soit un DAG pour lequel la communication entre tâche est intensive ou alors un DAG représentant des calculs intensifs au niveau des processeurs. Lorsque le CCR présente une valeur faible ce là signifie que l'application présente des calculs intenses. Dans le cas contraire, si cette valeur est élevée, cela signifie que la communication entre les tâches de l'application est intense.

En faisant varier ces paramètres, nous obtenons des DAG générés dans un fichier au fomat **gml**. Le fichier gml est par la suite traduit par un parseur implémenté en langage java pour générer le DAG sous forme de graphe acyclique.

Choix des valeurs

Nous avons remarqué dans le chapitre relatif au bilan de comparaison que les protocoles expérimentaux étudiés situent le nombre de tâche dans la plage allant de 10 à 500 tâches. Entre 40 et 100 tâches la courbe de progression du SLR par exemple demeure constante. Nous nous proposons d'aller au-delà de 100 tâches pour avoir plus de précision sur la qualité des résultats.

Concernant les autres paramètres, tous les articles définissent la même plage de valeur. Le choix des valeurs pour l'analyse des performances correspond donc à :

length= [10; 20; 30; 40; 50; 60; 70; 80; 90; 100; 200 300; 400; 500],

- CCR =[0:1; 0:5; 0:8; 1; 2; 5; 10]
- sdComp = [0:1; 0:2; 0:5; 1; 2]
- nbproc = [2; 4; 8; 16; 32]
- sdComm = [0.1,0.25,0.5,0.75,1]
- level = [1,2,3,4,5,length]
-

Métriques de mesure des performances

De nombreux paramètres permettent l'évaluation de la performance d'un Heuristique. Le paramètre principal est le makespan. Ce paramètre permet de mesurer le coût d'exécution de la tâche de sortie. Tant il est vrai que ce paramètre est très important dans la mesure des performances des algorithmes, il demeure vrai qu'il n'est pas suffisant pour analyser les performances de plusieurs heuristiques. Etant donné que nous travaillons avec des DAG de différentes caractéristiques et topologies, il est important de faire appel à de nouveau paramètre ou métrique d'analyse. Il est donc question d'évaluer la moyenne des performances obtenues par chaque heuristique pour chaque topologie.

Les critères de base sur lesquels nous allons nous appuyer dans notre étude sont les suivants :

- SLR
- temps d'exécution
- speedup

Analyse des performances

Pour analyser les performances de nos algorithmes, nous générons des courbes sous forme de nuage de point. Ces courbes en fonctions des caractéristiques du graphe nous permettent de visualiser les performances.

Nous analysons le comportement des algorithmes dans les conditions suivantes :

- ✓ évolution du SLR en fonction du nombre de tâche.
- ✓ Evolution du SLR en fonction du CCR
- ✓ Evolution du SLR en fonction du facteur d'hétérogénéité

Nous avons générer è DAG dont le nombre tâches correspondait à la plage suivante [30,40,50,60,70,80,90], la génération de ces graphes nous a permis grâce à notre simulateur d'exécuter nos deux algorithmes. Les résultats obtenus ont été gérés dans un fichier csv. Ce fichier a ensuite été interprété en langage python pour générer la courbe suivante :

Nous utilisons en python la librairie matplotlib pour la génération de ces courbes.

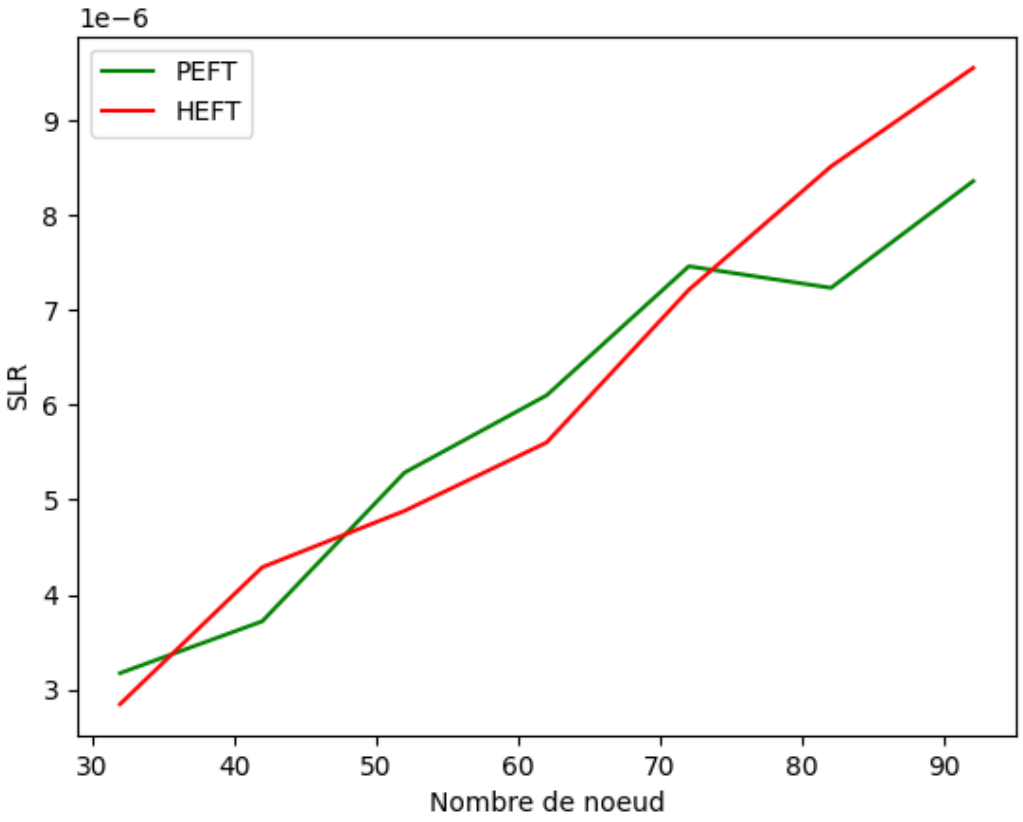

Figure 2: Courbe d'évaluation des performances en fonction du nombre de tache et du SLR

D’après la figure 2, nous pouvons remarquer que le nombre de tâche influence sur la qualité des performances des algorithmes. Au-delà de 70 tâches nous remarquons une amélioration considérable des performances de PEFT. Ceci nous amènerait à conclure que le choix du nombre de tâche est crucial dans la performance des algorithmes.

Nous n’avons malheureusement pas pu générer des graphes de plus de 100 tâches. Au-delà de 100 tâches notre programme n’a malheureusement pas pu s’exécuter.

Il convient donc de dire à la lecture de cette courbe que l’algorithme PEFT devient plus performant lors que le nombre de tâches dans le graphe devient de plus en plus grand. C’est certainement la raison pour laquelle dans l’article *[2]* et contrairement aux autres articles les valeurs de tâches varient dans la plage allant de 30 à 500 tâches. L’article *[3]* propose des taches dont les valeurs appartiennent à la plage allant de 30 à 100 tâches.

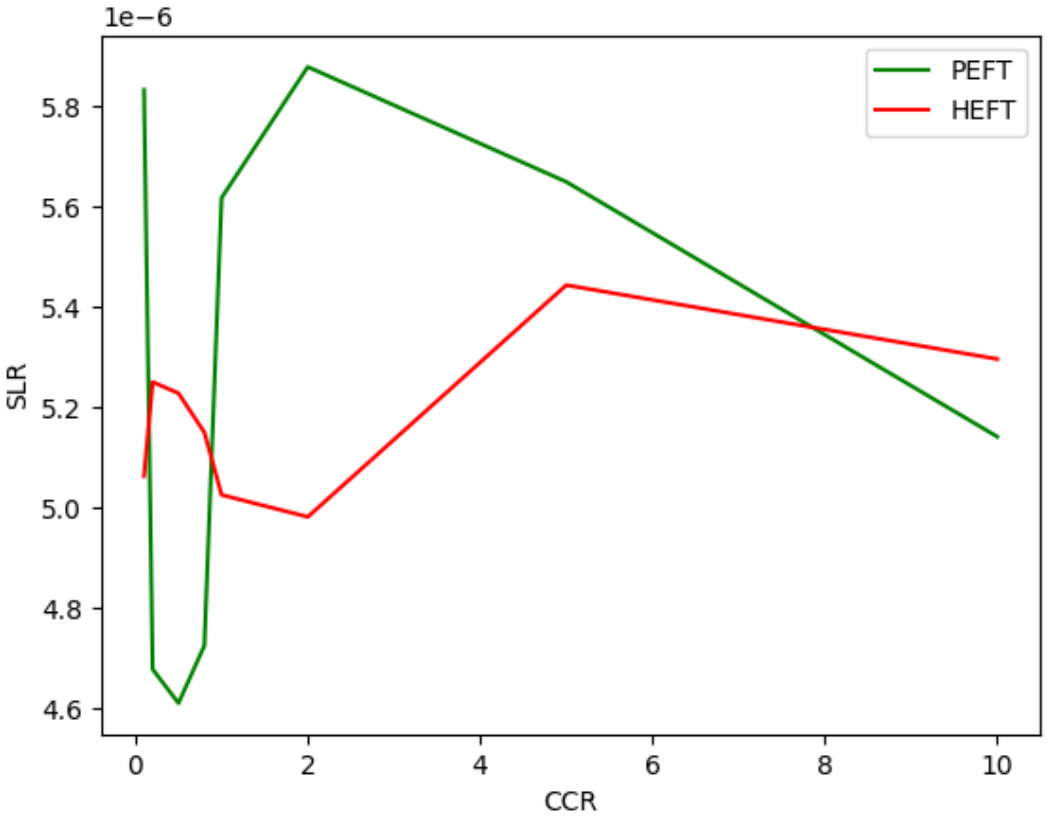

Figure 3: Courbe d'évaluation de performance en fonction du CCR et du SLR

La figure 3 illustre les performances en fonction de la variation du CCR.A titre de rappel, un valeur faible du CCR, le graphe révèle une communication intensive entre les tâche. Nous avons généré des graphes de 50 tâches en faisant varier la valeur du CCR dans la plage suivante [0.1,0.2,1,2,4,6,8,10]. Il en découle de ce graphe que lorsque la valeur du CCR est faible PEFT offre une bien meilleure performance.

4. Conclusion

Pendant cette étude nous avons lu avec beaucoup d'intérêt les articles relatifs aux algorithmes d'ordonnancement des tâches dans les systèmes parallèles. Notre «étude s'est principalement focalisée sur les articles décrivant les algorithmes CPOP, PETS, HEFT, et PEFT. Chacun de ces articles présente de façon claire les performances des algorithmes étudiés. Le calcul de performance a été évalué en utilisant aussi bien les graphes de tâches générés de façon aléatoire que celles découlant des applications réels. Les critères de performances étaient les suivant :

- Le temps moyen d'exécution de l'algorithme ;
- Le facteur d'accélération ;
- le nombre d'occurrence d'une meilleure qualité d'ordonnancement ;
- le temps d'exécution de l'algorithme.

A la lecture du protocole expérimental que nous avons proposé, il est fort de constater que ces articles n'évaluent pas les performances des algorithmes proposé sur la base des critères communs. Proposer un protocole expérimental nous a permis de proposer une base d'évaluation commune à chaque algorithme. Ces critères reposent sur les paramètres ainsi que les valeurs de génération des graphes de tâches, les paramètres d'évaluation ainsi que la mise sur pied d'un simulateur pour étudier les performances de ces algorithmes.

Tant il est vrai que nous n'avons pas considéré tous cas de figure qui nous aurait pu tirer avec autorité une conclusion, nous pouvons à regard du travail qui a été fait confirmer qu'à bien d'égards l'algorithme PEFT offre une bien meilleure performance que le HEFT. Ce qui nous permet de confirmer l'affirmation tirée de l'article [2] selon laquelle PEFT offre des performances bien meilleures que HEFT.

Les résultats démontrent à suffisance que sur la base des critères de performance cités plus haut, PEFT est l'algorithme qui réalise une meilleure performance. D'où le classement suivant :

{PEFT, PETS, HEFT, DLS} suivant l'ordre ascendante des meilleures performances.

Référence

[1] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3), pp. 260-274, 2002.

[2]H. Arabnejad and J. Barbosa. List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table. *IEEE Transactions on Parallel and Distributed Systems*, 99, 2014.

[3]E. Ilavarasan and P. Thambidurai. Low complexity performance effective task schedu-ling algorithm for heterogeneous computing environments. *Journal of Computer Science*, 3(2) :94–103, 2007.

[4]G.C. Sih and E.A. Lee. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architecture. *IEEE Transactions on Parallel and Distributed Systems*, 4(2), pp. 175-187, 1993.