Algorithme de prédictions statistiques du temps d'exécution basé sur la méthode de KPPV

GHAFFARI Fakhreddine *& ; AUGUIN Michel* ; ABID Mohamed** ; BENJEMAA Maher****

*Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis, les Algorithmes bat. Euclide, 2000, route des Lucioles BP 121, 06903 Sophia-Antipolis Cedex

**Unité de recherche GMS. Ecole Nationale d'Ingénieurs de Sfax, BPW 3038 Sfax Tunisie.

{ Ghaffari, auguin}@i3s.unice.fr ; {Maher.Benjemaa, mohamed.abid}@enis.rnu.tn

Résumé : Lors du partitionnement, en ligne, logiciel/matériel des tâches à temps d'exécution variable, l'estimation des temps d'exécution est un problème clé. Ce temps dépend de plusieurs paramètres qui caractérisent le jeu de données à traiter (taille de donnée, type de donnée…). Dans ce papier, nous présentons une méthode d'estimation du temps d'exécution des tâches, afin d'assurer un partitionnement logiciel/matériel d'une application sur une architecture hétérogène à reconfiguration dynamique. Notre technique d'estimation se base sur des observations antérieures enregistrées dans une base de donnée pour prédire les temps d'exécution de la nouvelle implémentation. *L'ajout d'un estimateur dans une approche du partitionnement en ligne permet d'adapter ce dernier aux caractéristiques des données traitées.*

Mots clés : Architecture hétérogène, temps d'exécution variable, estimation, K plus proche voisins, partitionnement en ligne.

1. Introduction

Dans une approche de conception des systèmes électroniques mixtes (logiciel/matériel), la phase du partitionnement donne une réponse à la question suivante : quelles sont les fonctionnalités à mettre sur les unités logicielles et quelles sont celles à mettre sur les unités matérielles?

Les approches classiques du partitionnement sont développés hors ligne et donnent un schéma d'exécution qui sera figé pour s'exécuter en permanence [15 ,16,17]. L'apparition des circuits reconfigurables avec leur flexibilité attrayante, a permis de réfléchir à des méthodes de partitionnement qui adapte en ligne l'architecture au traitement. Ce type d'approches est mieux adopté pour les applications à distribution variable de charge de calcul [18]. Quand-il s'agit d'une application contenant des tâches à temps d'exécution variable, l'outil de partitionnement a besoin des valeurs des temps d'exécution de chaque tâche. Pour une implémentation logicielle (RISC, DSP, ASIP…) on peut considérer les valeurs du temps d'exécution correspondant aux jeux de données à traiter.

Pour une implémentation matérielle (FPGA, ASIC …), il faut en plus du temps d'exécution, les nombres des ressources matérielles (CLB ou LAB) allouées pour chaque implémentation. Ces valeurs du temps d'exécution et le nombre des ressources allouées vont servir à la prise de décision de l'implémentation d'une tâche (l'implémentation logicielle ou une des implémentations matérielles).

Pour fournir en ligne ces valeurs du temps d'exécution à l'outil de partitionnement, des techniques d'estimation sont nécessaires pour prédire les valeurs correspondant à la prochaine itération.

Dans la littérature, il existe trois approches d'estimation du temps d'exécution d'une tâche : par analyse du code [13,14], par profiling [1,2,3,4,5,6,7,8] et par des prédictions statistiques [9,10,11,12].

Dans l'approche d'analyse du code, l'estimation du temps d'exécution est calculée après une analyse des instructions du code de la tâche. Cette méthode est limitée à un type spécifique du code ou bien à une architecture spéciale. Ce qui fait de cette technique très peu pratique surtout pour les architectures hétérogènes.

La technique de profiling vise à déterminer la composition d'une tâche en terme des instructions types dont les temps d'exécution sont connus à priori. L'inconvénient de cette méthode est de ne pas suivre les variations de jeu des données à traiter.

La troisième approche, qui est un algorithme de prédiction statistique, utilise les observations

antérieures pour prédire le temps d'exécution du futur. Au moment de l'exécution d'une tâche sur une unité de calcul, le temps d'exécution est mesuré et sa valeur sera ajoutée à l'ensemble des observations précédentes. La qualité des estimations produites par les algorithmes statistiques sera donc liée à la taille de l'ensemble des observations.

L'avantage de cette méthode réside dans ses aptitudes à résoudre le problème de dépendance du temps d'exécution aux caractéristiques des données (taille, type de donnée…) en fonction de la structure du code ou de l'unité de calcul.

Les fonctions sur lesquelles nous travaillons peuvent dépendre ou non d'une caractéristique de donnée à traiter que nous l'appellerons « paramètre de corrélation ».

La suite de ce papier est organisée comme suit : dans le deuxième paragraphe, nous détaillons notre approche de partitionnement en ligne, ensuite nous présentons une méthode de prédiction statistique du temps d'exécution dans le paragraphe 3. Le schéma d'exécution de l'approche est présenté dans un quatrième paragraphe suivi par les résultats des expérimentations et les interprétations associées pour finir par nos conclusions et perspectives.

2. Approche de partitionnement en ligne

La figure 1 ci-dessous récapitule les étapes de notre approche du partitionnement dynamique en ligne.

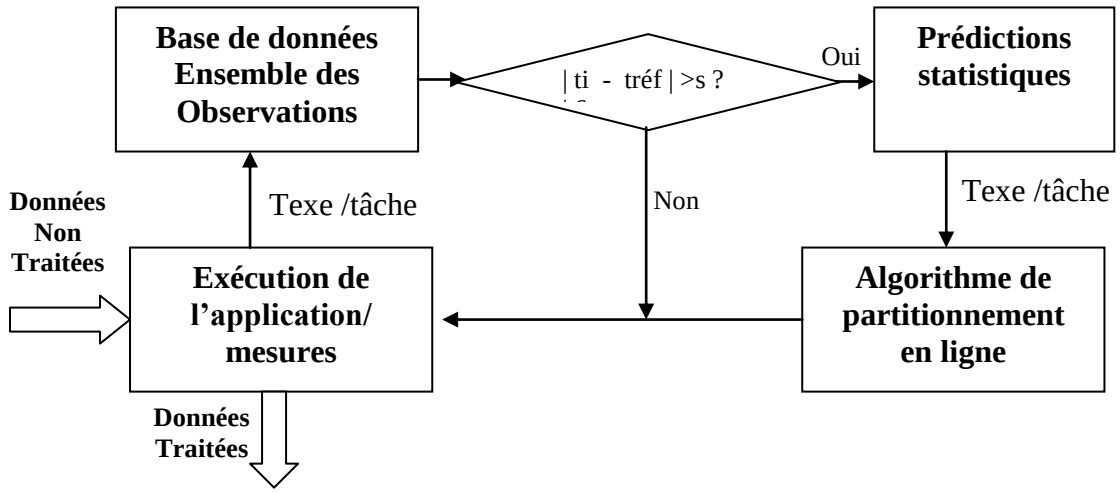

Figure 1 : Approche de partitionnement en ligne

L'algorithme de prédiction statistique prend en entrée les observations (mesures du temps d'exécution sur les images précédentes) et fournit en sortie une estimation du temps d'exécution des tâches sur chaque type d'implantation pour la prochaine itération de l'application. L'algorithme de partitionnement en ligne prend à chaque itération ces estimations et donne en sortie un ordonnancement et un schéma d'allocation de toutes les tâches sur les unités de l'architecture cible pour l'itération suivante.

A l'issue de chaque itération, l'ensemble des mesures des tâches de l'itération i réactualise la base de données qui contient l'ensemble des anciennes observations. Un test de comparaison est ensuite réalisé pour décider s'il est vraiment nécessaire d'effectuer un nouveau partitionnement ou bien les résultats du dernier partitionnement sont suffisants pour satisfaire les contraintes temps réel. La décision d'entamer de nouvelles estimations et un partitionnement est prise lorsque la différence entre les temps d'exécution de l'itération courante et les temps d'exécution qui ont provoqué le dernier partitionnement dépasse un certain seuil. Ce seuil est fixé par le concepteur pour chaque application.

3. Prédiction statistique du temps d'exécution

La méthode d'estimation envisagée dans ce papier est inspirée de celle de Iverson [12]. Pour chaque tâche, les temps d'exécution de chaque tâche sur chaque cible possible sont mesurés en fonction d'un

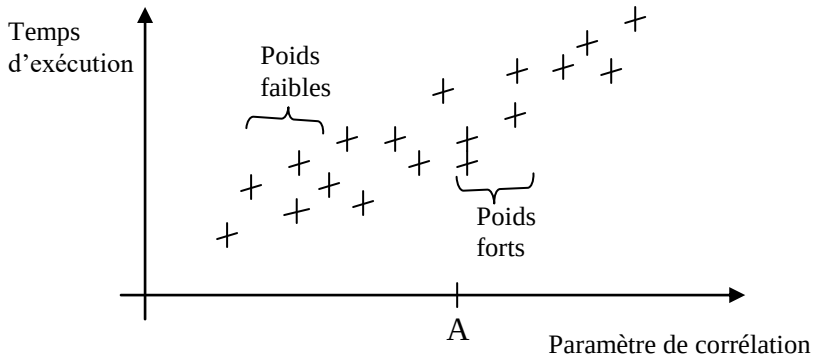

Figure 2 : méthode de K plus proches voisins (K-ppv)

paramètre de corrélation, ensuite introduits dans la base des données (voir figure 2). Le paramètre de corrélation correspond à la variable responsable de la fluctuation du temps d'exécution d'une tâche en fonction de la nature des images [9].

A chaque itération le temps d'exécution de chaque tâche est estimé par la formule suivante (1):

$$\mathbf{T = m(x) + e} \quad \textbf{(1)}$$

Où T est l'estimation du temps d'exécution, m(x) est le temps d'exécution qui dépend du paramètre de corrélation x (extrait de courbe de la figure 2) et e est un taux d'erreur (paramètre qui ne dépend pas de x).
La valeur de m(x) peut être calculée par la formule suivante (2).

$$m(x) = \frac{1}{n}\sum_{i=1}^{n} W_i(x) t_i \quad \textbf{(2)}$$

où n est le nombre de mesures disponibles, les coefficients w_i sont les poids affectés aux mesures et les t_i sont les valeurs des mesures.
Comme le montre la figure 2, en fixant une valeur au paramètre de corrélation (à A par exemple sur la figure 2), la valeur de l'estimation m(x) est obtenue par la somme pondérée à partir des voisins, la pondération étant fonction de la position de ce paramètre de corrélation. Des poids forts sont affectés aux voisins les plus proches et des poids de plus en plus faibles aux voisins les plus éloignés.
Pour des raisons de simplicité, nous calculons uniquement la variance de l'erreur e par la formule suivante (3):

$$\sigma^2(e) = \frac{1}{n}\sum_{i=1}^{n} W_i(x)(t_i - m(x))^2 \quad \textbf{(3)}$$

A partir du temps d'exécution ainsi estimé, nous considérons le temps d'exécution supérieur ou égal à celui qui a été utilisé pour choisir la version de la synthèse hors ligne de la tâche.

4. Schéma d'exécution de l'approche

La figure 3 montre un exemple d'ordonnancement possible pour une application contenant trois tâches et ciblant une architecture reconfigurable constituée par une unité de calcul matérielle par exemple un FPGA et une unité de calcul logicielle par exemple un processeur RISC. Nous présentons sur cette figure trois itérations successives avec des résultats du partitionnement différents.

Selon les étapes de notre approche : estimation, partitionnement en ligne, exécution, mesure, stockage des mesures dans la base de données… ; il faut que le total du temps nécessaire au partitionnement et le temps mis par l'algorithme d'estimation des temps d'exécution soit inférieur au temps d'exécution de l'application à partitionner. C'est-à-dire qu'à chaque itération il faut garantir un nouveau résultat de partitionnement avec les nouvelles estimations des temps d'exécutions. Ceci nous a conduit à allouer une unité reconfigurable pour l'algorithme de prédiction et l'algorithme de partitionnement en ligne. En effet, il faut pour ces deux algorithmes assurer d'une part que leur temps cumulé de

calcul soit inférieur au plus petit temps d'exécution d'une itération de l'application et d'autre part avoir une flexibilité de programmation pour tenir compte d'évolutions ou de modification dans la description sous forme de tâche de l'application. Pour l'exécution de l'application, nous ciblons une architecture constituée par un processeur connecté à une unité reconfigurable qui permet des reconfigurations partielles.

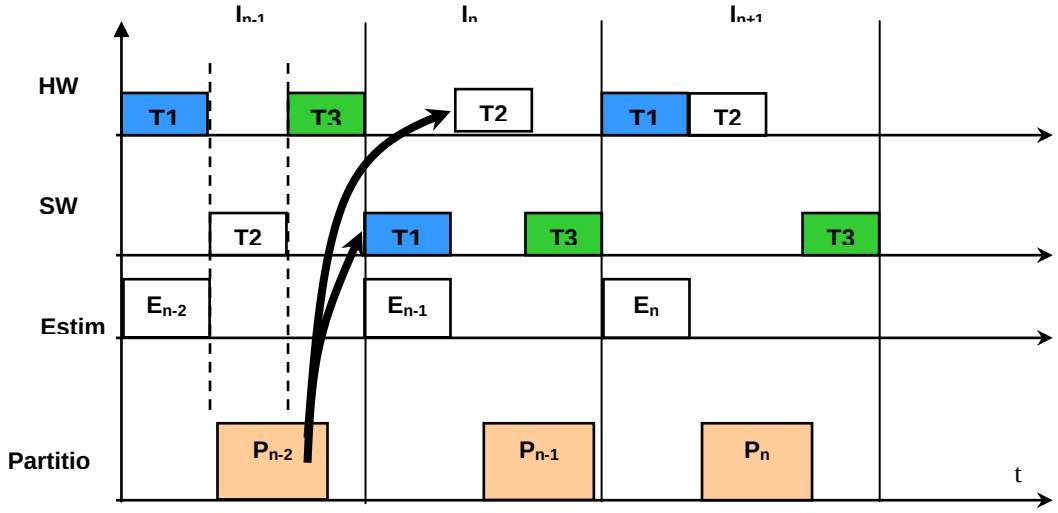

Figure3 : un exemple d'ordonnancement

5. Résultats et interprétations

Afin de calculer une estimation du temps d'exécution d'une tâche dont le temps d'exécution dépend des données, il faut calculer les distances entre la valeur de paramètre de corrélation et ses voisins (voir figure 4). Ces mesures des distances vont servir pour l'affectation des poids de plus en plus faible dès qu'on s'éloigne du point d'estimation.

$t_{i+1} = f(t_i, t_{i-1}, t_{i-2}\ldots, d_i, d_{i-1}\ldots)$

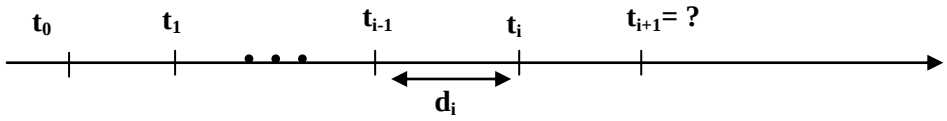

Figure 4 : principe d'estimation par régression

les paramètres de l'estimateur K plus proche voisins sont : les valeurs des poids à affecter aux observations antérieures, la fonction permettant d'estimer le temps d'exécution et enfin le nombre des voisins à considérer pour avoir des meilleurs résultats d'estimation.

Nous avons écrit un module qui utilise une bibliothèque KNN (the K Nearest Neighbours) pour calculer les distances séparant un point de ses voisins ensuite estimer le temps d'exécution de la prochaine itération par plusieurs fonctions (pour pouvoir comparer les résultats des différentes fonctions).

Il faut noter tout d'abord que la distance utilisée ici est la distance Euclidienne définie par l'équation 4 :

$$dist(p,q) = \left(\sum_{0<=i<d} (p_i - q_i)^2\right)^{1/2} \quad \textbf{(4)}$$

Nous avons travaillé sur une application qui consiste à effectuer la détection du mouvement sur un fond d'image fixe. Cette application contient des fonctions corrélées au contenue de l'image traitée.

Par exemple le temps d'exécution de la fonction étiquetage des objets dépend du nombre des objets dans l'image à considérer. Dans la suite nous prenons l'exemple de la fonction ,construction de l'enveloppe englobant, qui dépend du nombre des objets dans l'image.
Soit la base de données suivante formée par 20 mesures du temps d'exécution (nombre d'objet, temps d'exécution):

(2, 3) (3, 3.26) (5, 5.21) (10, 8.23) (12, 11.82) (15, 13.76) (20, 16.38) (22, 17.5) (25, 20) (26, 21.17) (30, 23.77) (33, 25) (35, 27.08) (37, 27.98) (40, 30) (43, 32.79) (45, 34.09) (47, 35) (49, 36.89) (50, 37).

L'estimateur doit prévoir le comportement du temps d'exécution de la tâche pour la prochaine itération en se basant uniquement sur les observations précédentes c-à-d sur les mesures réelles effectuées précédemment. Il doit s'adapter aux différentes variations de la courbe du temps d'exécution en fonction du paramètre de corrélation.

Il est évident que les poids affectés aux anciennes mesures doivent être inversement proportionnels aux distances correspondants (distances séparant la dernière mesure de ses voisins).
Soit k le nombre des voisins qui entourent la dernière mesure de paramètre de corrélation. Nous avons choisi au début un estimateur qui calcule une quantité dépendant des voisins par pondération sur les temps d'exécutions et l'ajoute au temps d'exécution du voisin le plus proche (t_{min}).

$$Estim1 = (\frac{1}{k}\sum_{i=1}^{k}\frac{1}{dist(i)}t_i) + t_{\min} \quad \textbf{(5)}$$

Une variance de l'erreur est calculée comme suit :

$$err = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{dist(i)}(t_i - Estim1)^2 \quad \textbf{(6)}$$

Les estimations après calcul d'erreur consistent à ajouter l'erreur à la valeur de Estim1.
Un autre estimateur a était utilisé, il consiste à affecter des poids dont la somme égale à 1.

$$Estim2 = (\frac{1}{k}\sum_{i=1}^{k}\frac{dist(i)}{\sum_{i=1}^{k} dist(i)} * t_i) + t_{\min} \quad \textbf{(7)}$$

Des mesures sont effectuées pour le point (3, 3.26) voir figure 5. Nous constatons d'après ces mesures que l'estimateur 2 est le meilleur estimateur car il est le plus proche des mesures réelles.
Nous constatons aussi que plus la valeur de k est grande plus l'estimateur converge vers la dernière mesure effectuée.

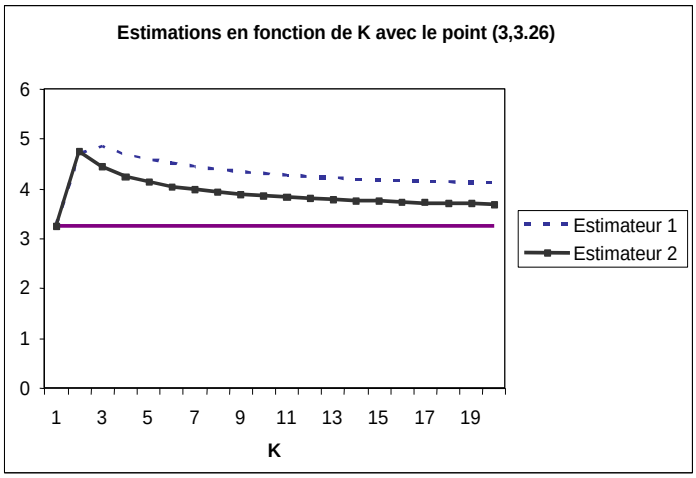

Figure 5 : comparaison entre estimateur 1 et estimateur 2

Fixons la valeur de K au maximum pour cette base de données (k = 20) et comparons l'allure de la courbe des mesures réelles avec les résultats fournis par l'estimateur Estim2.

La figure 6 montre bien que l'estimateur suive correctement le comportement de la courbe des valeurs mesurées.

.

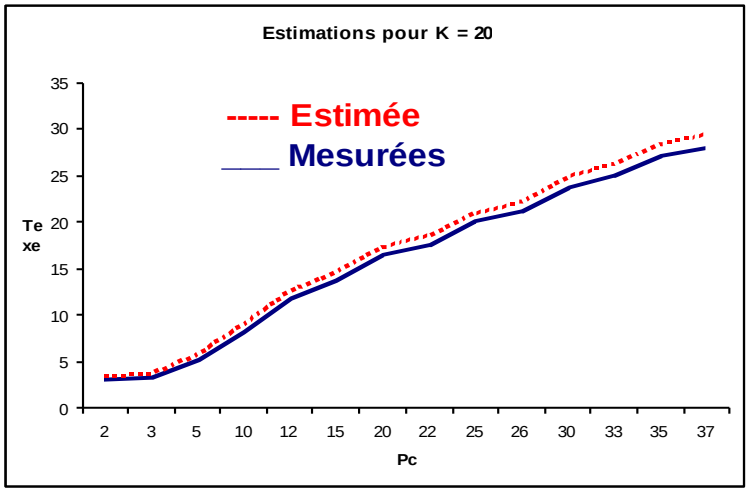

Figure 6 : comparaison entre mesures et estimations

Maintenant travaillons sur un cas plus réel : à chaque itération on incrémente la base de données par la valeur mesurée. La valeur de k égale à la taille de la base de données.
En choisissant toujours l'estimateur 2, nous supposons que le point (25, 20) va se répéter plusieurs fois avant de passer aux autres valeurs.
La figure 7 nous montre les résultats des estimations. Nous remarquons bien l'adaptation des estimations pour suivre le comportement des valeurs mesurées.

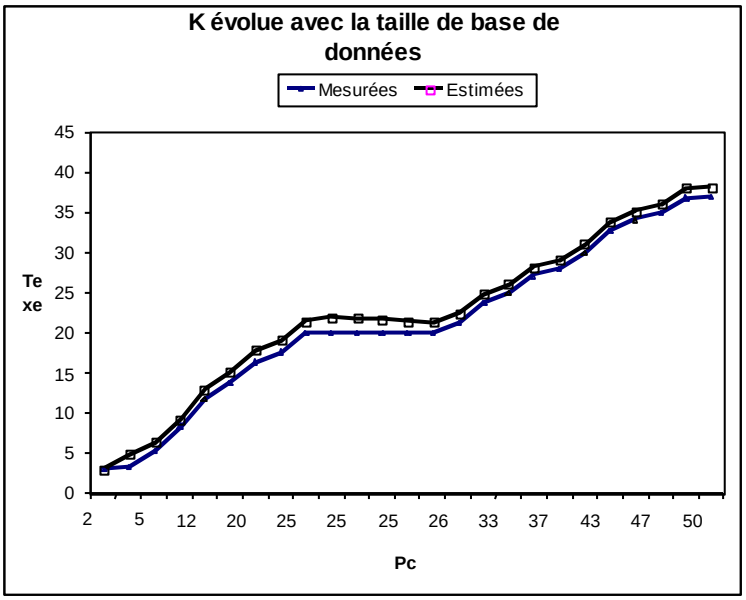

Figure 7: poursuite des estimations (avec k variable) à la courbe des mesures

6. **Conclusion**

Dans ce papier, nous avons présenté une approche de prédiction statistique du temps d'exécution des tâches basée sur la méthode de K plus proches voisins. Cette méthode d'estimation constitue une étape indispensable dans l'approche du partitionnement dynamique en ligne. Nous adoptons cette dernière stratégie pour adapter l'architecture au traitement des applications à temps d'exécution variable.

Actuellement, nous essayons d'expérimenter notre approche de partitionnement sur une application de traitement d'image « la détection de mouvement sur un fond d'image fixe ».

Il s'agit d'un projet ambitieux et novateur qui soulève de nombreuses questions. Les solutions architecturales sont envisageables mais elles restent à être précisées. De plus, l'efficacité d'une telle approche de partitionnement par rapport à des solutions traditionnelles doit être prouvée.

7. Références

[1] G. Stitt, R. Lysecky, F. Vahid. Dynamic Hardware/Software Partitioning: A First Approach. Design Automation Conference (DAC),pp 250-255, June 2003

[2] R. Freund. Optimal selection theory for super concurrency. In Proceedings of the 1989 Supercomputing Conference, pages 13–17. IEEE Computer Society Press, 1989.

[3] A. Khokhar, V. Prasanna, M. Shaaban, and C.-L.Wang. Heterogeneous supercomputing: Problems and issues. In Proc. of the 1992 Workshop on Heterogeneous Processing, pages 3–12. IEEE Computer Society Press, March. 1992.

[4] A. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C.-L. Wang. Heterogeneous computing: Challenges and opportunities. IEEE Computer, 26(6):18–27, June 1993.

[5] D. Pease, A. Ghafoor, I. Ahmad, D. L. Andrews, K. Foudil-Bey, T. E. Karpinski, M. A. Mikki, and M. Zerrouki. PAWS: A performance evaluation tool for parallel computing systems. IEEE Computer, 24(1):18–29, January. 1991.

[6] H. J. Siegel. Heterogeneous computing. Annual Research Summary 5.92, http://ece.www.ecn.purdue.edu/ Researchsummary/Section5/sec5 92.html, 1994.

[7] J. Yang, I. Ahmad, and A. Ghafoor. Estimation of execution times on heterogeneous supercomputer architectures. In the 1993 Inter. Conf. on Parallel Processing, volume 1, pages 219–226. CRC Press, Aug. 1993.

[8] T. Yang and A. Gerasoulis. DSC: Scheduling tasks on an unbounded number of processors. IEEE Trans. Parallel and Distributed Systems, 5(6):951–967, Sept. 1994.

[9] M. V. Devarakonda and R. K. Iyer. Predictability of process resource usage: A measurement-based study on UNIX. IEEE Trans. Software Engineering, 15(12):1579–1586, Dec. 1989.

[10] M. A. Iverson, F. ¨ Ozg¨uner, and G. Follen. Run-time statistical estimation of task execution times for heterogeneous distributed computing. In Proc. of the 1996 High Performance Distributed Computing Conference, pages 263–270, Syracuse, NY, Aug. 1996.

[11] T. Kidd, D. Hensgen, L. Moore, R. Freund, D. Charley, M. Halderman, and M. Janakiraman. Studies in the useful predictability of programs in a distributed and homogeneous environment. The Smartnet Home Page (http://papaya.nosc.mil:80/SmartNet/), 1995.

[12] Michael A. Iverson, Füsun Özgüner, Lee C. Potter, Statistical Prediction of Task Execution Times Through Analytic Benchmarking for Scheduling in a Heterogeneous Environment, Eighth Heterogeneous Computing Workshop April 12 - 12, 1999 San Juan, Puerto Rico.

[13] B. Reistad and D. K. Gifford. “Static dependent costs for estimating execution time”. In Proc. of the 1994 ACM Conference on LISP and functional programming, pages 65–78. ACM Press, June 1994.

[14] Frank Vahid and Daniel D.Gajski. « Incremental Hardware Estimation During Hardware/Software Functional Partitioning » IEEE Transactions on very large scale integration (VLSI) Systems. 3, No. 3, September 1995. pp 459-464.

[15] Jörg Henkel, Rolf Ernst, “A Hardware/Software Partitioner Using a Dynamically Determined Granularity”, DAC 1997, pp 691-696.

[16] A. Jantsch, P. Ellervee, J. Oeberg et. al., Hardware/Software Partitioning and Minimizing Memory Interface Traffic, IEEE/ACM Proc. of The European Conference on Design Automation (EuroDAC) 1994, pp. 220–225, 1994.

[17] Z. Peng, and K. Kuchcinki, "An Algorithm for Partitioning of Application Specific Systems", Proc. European Design & Test Conference (EDAC-ETC-EuroASIC), IEEE CS Press, pp. 316-321, February 1993.

[18] F.GHAFFARI, M.AUGUIN, M.BENJEMAA. Etude du partitionnement logiciel/matériel d’applications à distribution variable de charge de calcul. Renpar’14 /ASF/SYMPA pp. 334-338 Hammamet, TUNISIE 10 – 13 avril 2002