

Chapitre 4:

Plate-forme virtuelle et exploration architecturale:

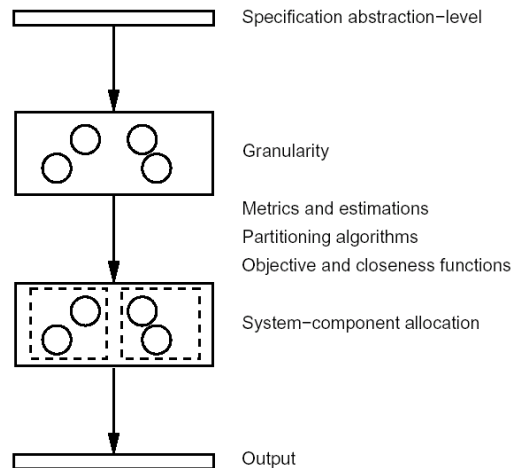
Partitionnement automatisée

Contenu

1. Introduction, flot et plate-forme virtuelle
2. Étude de cas d'une approche de partitionnement par plate-forme virtuelle
3. Estimateurs et analyse (monitoring)
4. **Partitionnement automatisée**
5. Partitionnement et ordonnancement

Les bases du partitionnement automatisé

- Le problème est séparé en 8 éléments distincts



INF8500 SE: Conception et vérification

5

#1 : Niveau d'abstraction des éléments à partitionner

- Le langage des spécification ne dit pas tout (Ex.: SystemC), il faut connaître le modèle de programmation (MOC)
- Le modèle donne déjà une partie de l'implantation
 - DFG et tâches : orientation logicielle
 - FSMD : orientation matérielle
 - CDFG : le plus utilisé pour le partitionnement
- On ajoutera aux spécifications des contraintes non fonctionnelles (e.g. timing).

INF8500 SE: Conception et vérification

4

#2 : Granularité

- Taille des objets de la spécification
- **Gros grains** : chaque objet contient beaucoup d'éléments de la spécification
 - Facile à comprendre pour l'humain, interaction manuelle possible car le nombre d'objet est manipulable
 - Ex.: Tâches, procédures, processus
- **Grains fins** : chaque objet contient peu d'information de la spécification
 - Plus d'objets
 - Plus de possibilités de partitionnement, donc plus grande la possibilité de trouver la solution optimale
 - Ex.: « Adder », registres, instructions
- La granularité idéale se situe au niveau intermédiaire, par exemple le bloc de base ou le *thread* dans le MTG

5

INF8500 SE: Conception et vérification

#3 : Allocation des composants du système

- Type de composants imaginables sur un partitionnement
 - ASIC ou circuits matériel dédiés
 - μ P, DSP
 - Mémoires
 - Bus
- L'allocation se fait manuellement, ou en parallèle avec des algorithmes de partitionnement automatique

6

INF8500 SE: Conception et vérification

#4 : Métriques et estimateurs

- Une métrique s'associe à une contrainte d'un système qui régit la justesse d'un partitionnement
- Technique d'utilisation des estimateurs
 - Partitionner de façon grossière puis calculer les coûts à l'aide d'outils : estimation!
- Les métriques sont multiples (liste non complète)
 - Coût monétaire
 - Temps d'exécution
 - Débit des communications
 - Trafic bus
 - Consommation de puissance
 - Aire de la puce
 - Nombre de broches
 - Testabilité et fiabilité
 - Taille du programme et des mémoires
 - Etc.

INF8500 SE: Conception et vérification

7

#5 : Fonctions de proximité ou de coût

- On évalue souvent un système à l'aide d'une combinaison de plusieurs métriques (plutôt qu'une seule)
- Il s'agit d'une **fonction de proximité** ou **fonction de coût** qui retourne une valeur spécifique: le **coût** du système
- Chaque métrique est associée à un poids représentant son importance relative

INF8500 SE: Conception et vérification

8

#5 : Fonctions de proximité ou de coût

- Par exemple, la fonction de proximité suivante permet de calculer le coût pour un système ayant des contraintes d'aire, de délai et de puissance:

$$Fct = k_1 \cdot F(aire, contraintes) + \\ k_2 \cdot F(délai, contraintes) + \\ k_3 \cdot F(puissance, contraintes)$$

- $F()$ est une fonction représentant la proximité de la contrainte par rapport à la réalité
- k est un coefficient servant à normaliser la courbe
- Par exemple, le but est d'avoir une fonction égale à 0
 - plus F est grand, plus la métrique viole la contrainte
 - si F vaut 0, la contrainte est respectée

9

INF8500 SE: Conception et vérification

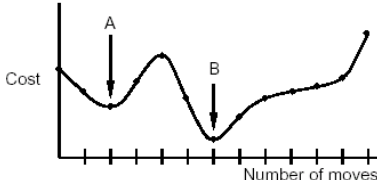
#6 : Algorithmes

- Des algorithmes peuvent chercher pour un partitionnement optimal, basé sur des contraintes spécifiques et des objets donnés
- Difficile d'implanter de tels algorithmes
 - Si n objets pour m composants architecturaux, m^n possibilités de partitionnement
- 2 classes d'algorithmes de partitionnement
 - Constructifs : groupent les objets entre eux pour créer des partitions (utilisent les métriques simples)
 - Itératifs : modifient les partitions existantes dans le but de les améliorer (utilise la fonction de proximité)

10

INF8500 SE: Conception et vérification

#6 : Algorithmes

- Les algorithmes itératifs tentent de minimiser le temps de calcul et fonctionnent généralement de la façon suivante :
 - Changer la partition d'un objet (*move*)
 - Estimer les résultats
 - Terminer si le résultat est acceptable
 - Sinon itérer à nouveau
 - Éviter les minimum locaux (A)
 - Trouver le minimum global (B)
- 
- The graph shows a wavy line representing 'Cost' on the y-axis and 'Number of moves' on the x-axis. There are two points labeled A and B with arrows pointing to them. Point A is at a local minimum, while point B is at a global minimum (the lowest point on the graph).
- Un algorithme qui arrête au premier minimum est nommé « glouton » ou « vorace » (*greedy*)
 - Un algorithme qui peut s'échapper d'un minimum local se nomme « grimpeur » (*hill-climbing*)

11

INF8500 SE: Conception et vérification

#7 : Résultat de sortie

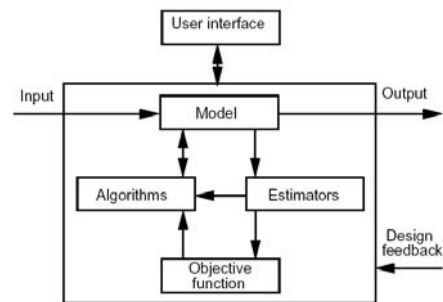
- Quels types d'objets connectés à quels composants ?
- Format et utilité
 - Graphiques à analyser pour un concepteur
 - Fichier à être utilisé par un outil de synthèse
 - Nouveau système complètement partitionné

12

INF8500 SE: Conception et vérification

#8 :Interaction des concepteurs

- Par des interfaces de type GUI pour aider le concepteur à déterminer plusieurs caractéristiques telles
 - Granularité
 - Fonctions de proximité
 - Algorithmes
 - Etc.



13

INF8500 SE: Conception et vérification

Principaux algorithmes de partitionnement

- Clustering
- Recuit simulé
- Recherche Tabu
- Group migration (min cut)
- Génétique
- Programmation en nombres entiers

14

INF8500 SE: Conception et vérification

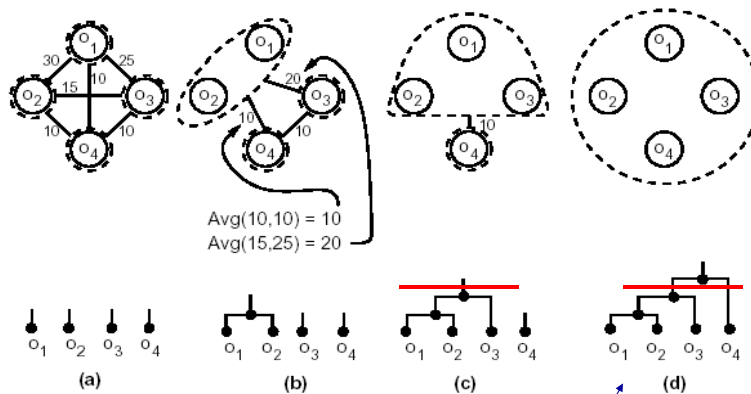
Algorithmes de base de partitionnement

- Définition formelle du problème
 - Une partition P est définie par un ensemble d'objets $O = \{o_1, o_2, \dots, o_n\}$ appartenant à un sous-ensemble H et un sous-ensemble S , avec
 - $H \subset P$
 - $S \subset P$
 - $H \cup S = P$
 - $H \cap S = \phi$

15

INF8500 SE: Conception et vérification

Clustering

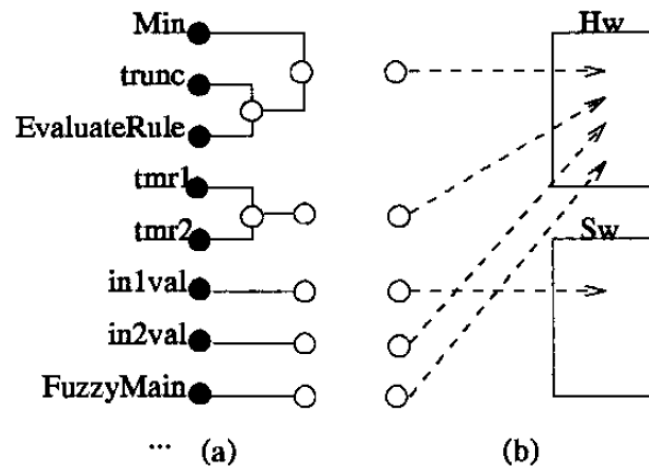


(a) Init (b) o_1 et o_2 regroupés, etc. o_1, o_2 et o_3 la partition!

16

INF8500 SE: Conception et vérification

Clustering (exemple)



INF8500 SE: Conception et vérification

17

Recuit simulé (*Simulated annealing*)

- Algorithme
 - Démarrer avec une partition aléatoire et une température
 - Décroître la température à chaque itération
 - Pour chaque température, générer un déplacement aléatoire
 - Accepter tous les déplacements qui améliorent le coût (meilleure partition)
 - Accepter quelques mauvais déplacements, surtout lorsque la température est élevée (au début)
- Idée :
 - À haute température, on déplace les blocs pour trouver le minimum global
 - À basse température, on raffine simplement le partitionnement

INF8500 SE: Conception et vérification

18

Recuit simulé (*Simulated annealing*)

```
temp = initial temperature
cost = Objct(P)
while not Frozen loop
  while not Equilibrium loop
    P_tentative = Move(P)
    cost_tentative = Objct(P_tentative)
    cost = cost_tentative - cost
    if (Accept(cost, temp) > Random(0, 1)) then
      P = P_tentative
      cost = cost_tentative
    end if
  end loop
  temp = DecreaseTemp(temp)
end loop
```

19

INF8500 SE: Conception et vérification

Recuit simulé (*Simulated annealing*)

- *cost* est le coût du partitionnement
- *cost_tentative* est le coût de l'exploration
- *cost* = *cost_tentative* - *cost* (négatif si meilleur)
- Équilibre après plusieurs itérations sans changement sur la partition
- Frozen détermine le nombre total d'itérations que l'algorithme devrait effectuer¹.

1. Wangtong et al., « Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign », p. 432.

20

INF8500 SE: Conception et vérification

Recuit simulé (*Simulated annealing*)

- $Accept(cost, temp) = min(1, e^{-\frac{cost}{temp}})$
- La fonction de coût est déterminée grâce aux estimateurs (métriques)
- Les minimum locaux sont évités
- L'algorithme fournit de bons résultats, mais il est très lent

21

INF8500 SE: Conception et vérification

Exemple de recuit simulé (Cosyma)

- Soit $\Delta c(b)$, l'accélération (temps d'exécution) d'un bloc de base b
$$\Delta c(b) = w \cdot (t_{HW}(b) - t_{SW}(b) + t_{com}(Z) - t_{com}(Z \cup b)) \cdot It(b)$$
- w est un facteur de contrôle du recuit simulé (poids)
- $t_{hw}(b)$ est le temps estimé pour l'exécution de b sur un copro matériel
- $t_{sw}(b)$ est le temps estimé pour l'exécution de b sur un processeur RISC
- $t_{com}(Z)$ est le temps estimé des communications processeur-coprocasseur
- Z est l'ensemble des blocs de base sur le coprocasseur
- $It(b)$ est le nombre d'itérations sur b

22

INF8500 SE: Conception et vérification

Exemple de recuit simulé (Cosyma)

- $t_{sw}(b)$ est estimé à partir du code source compilé ou par profilage (statique ou dynamique)
- $t_{hw}(b)$ est estimé à partir de la synthèse
- $T_{com}(Z \cup b)$ est estimé à partir des blocs matériels adjacents

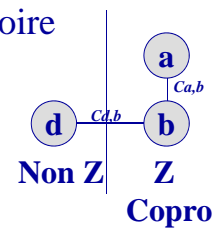
23

INF8500 SE: Conception et vérification

Exemple de recuit simulé (Cosyma)

$$t_{com}(Z \cup b) = t_{com}(Z) - \left(\sum_{a \in Z} C_{a,b} - \sum_{d \in U \setminus Z} C_{d,b} \right) \cdot t_{trans}$$

- $C_{a,b}$ sont tous les coûts reliés aux communications entre a et b
- t_{trans} est le temps de transfert via la mémoire partagé (fixe)



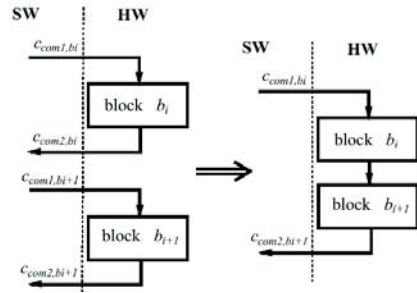
24

INF8500 SE: Conception et vérification

Exemple de recuit simulé (Cosyma)

- Détection des blocs adjacents dépendants

$$C_j \neq \sum \{c_{b_i} | b_i \in P_j\}$$



$$C = c_{b_i} + c_{b_{i+1}} - (c_{b_i,com2} + c_{b_{i+1},com1}).$$

- w est choisit en fonction du temps d'exécution logiciel estimé T_S et le temps d'exécution visé (contrainte) T_C

$$w = \text{sign}(T_C - T_S) e^{\frac{T_C - T_S}{T}}$$

25

INF8500 SE: Conception et vérification

Recherche Tabu

- Contrairement à l'approche de recuit simulé, la recherche Tabu contrôle les déplacements de blocs de manière intelligente plutôt qu'avec le hasard
- Deux composantes: stratégies **d'intensification** et de **diversification**
- La stratégie d'intensification repose sur la possibilité de modifier les règles de décision pour encourager les combinaisons de mouvement ou solutions qui se sont avérées avantageuses dans le passé. En somme ces solutions dites *délite*, doivent être conservées dans l'ordre pour qu'on puisse examiner leurs voisins immédiats, on peut constater que cette stratégie est étroitement liée à la gestion de la mémoire nommée *recency-based memory*

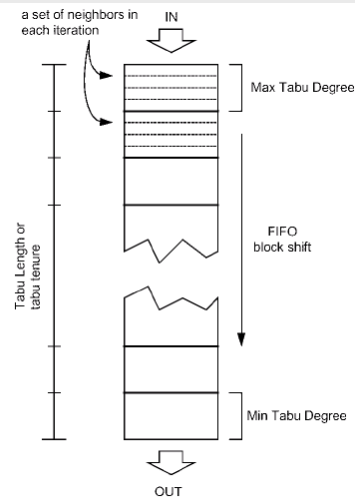
26

INF8500 SE: Conception et vérification

Recherche Tabu

Typiquement une profondeur (degree) de 7 à 20 ou encore de $.5N^{1/2}$ à $2N^{1/2}$ où N est fonction de la dimension du Problème.

On appelle ça aussi la liste Tabu



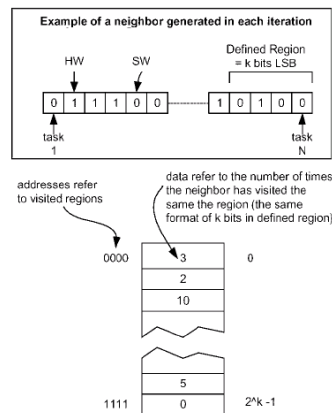
Recency-based memory Structure

7

INF8500 SE: Conception et vérification

Recherche Tabu

On maintient tous les détails sur une région. Soit N le nombre d'objets fonctionnels donc 2^N partitions, on s'intéressera à un sous-ensemble k de N .



Frequency-based memory Structure

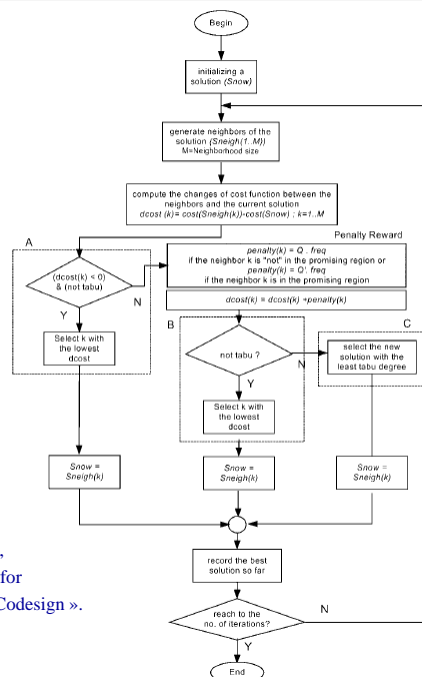
28

INF8500 SE: Conception et vérification

Recherche Tabu

- La stratégie de diversification va quant à elle encourager la progression de la recherche vers des régions encore inexplorées, ce qui va permettre de produire des solutions qui vont différer de plusieurs façons significatives des solutions déjà parcourues. On utilise ici un autre type de mémoire: *frequency-based memory*

Recherche Tabu



Pour plus de détails, voir dans Wangtong et al.,
« Comparing Three Heuristic Search Methods for
Functional Partitioning in Hardware-Software Codesign ».

Recherche Tabu

Avec le test de A , deux situations peuvent survenir :

1) À partir de la solution courante, on va tout d'abord explorer des solutions voisines (par exemple en déplaçant un objet du logiciel au matériel ou vice versa). Pour ces solutions voisines ($k = 1$ à M), on calcule un coût $dcost(k)$. Si ce $dcost(k)$ est plus petit que le meilleur $dcost$ obtenu à ce jour et que k n'est pas dans la liste Tabu on conserve la solution (permet la diversification).

31

INF8500 SE: Conception et vérification

Recherche Tabu

2) Si $dcost(k)$ est plus grand que le meilleur $dcost$ obtenu à ce jour ou si k est dans la liste Tabu on va donner une pénalité à k ($penalty(k)$) qui est ajouté à $dcost(k)$. Cette pénalité est fonction de la fréquence d'apparition d'un certain sous-voisinage de k (k bits LSB). Plus ce sous-voisinage est apparu souvent plus la pénalité est grande (permet la diversification). Par contre cette dernière est moins grande si k est apparu au cours des Kq dernières itérations, ce qu'on appelle *promising region* et qui permet l'intensification. Finalement, on sélectionne le plus petit $dcost(k)$ pour les k qui ne sont pas dans la liste Tabu et pour ceux qui le sont on choisit le k le plus profond dans la queue.

Finalement, on détermine le meilleur $dcost$ parmi 1) et 2) qui devient alors *Snow*.

32

INF8500 SE: Conception et vérification

Algorithmes génétiques

- Un ensemble de partitions est appelé une génération
- Trois types intéressants d'algorithmes
 - Un algorithme de sélection qui conserve une partition et qui la copie dans une autre génération
 - Un algorithme de croisement qui, à partir de deux partitions A et B, sélectionne un trait fort de A et le copie dans B (et ainsi créer une nouvelle génération)
 - Un algorithme de mutation qui correspond au déplacement aléatoire d'objets dans une partition

33

INF8500 SE: Conception et vérification

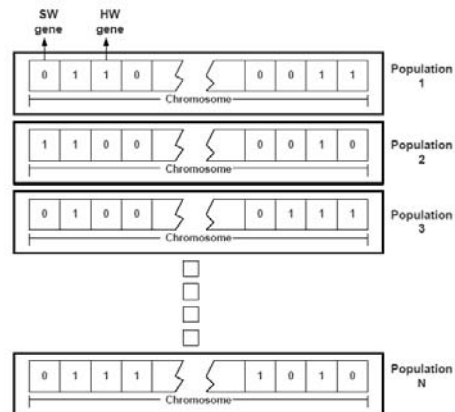
Algorithmes génétiques

- À partir d'une population initiale, un degré de ressemblance (*fitness*) est calculé pour chaque membre et on peut ainsi faire la *sélection* pour la prochaine génération.
- Un parent est une partition nommé chromosome composé de n tâches nommés gènes.
- Un gène encodé à 0 signifie que la tâche correspondante est logicielle alors qu'un gène encodé à 1 veut dire que la tâche correspondante est matérielle.

34

INF8500 SE: Conception et vérification

Algorithmes génétiques



Populations, chromosomes et gènes

35

INF8500 SE: Conception et vérification

Algorithmes génétiques

- On obtient un enfant de la prochaine génération par un croisement de parents
- La mutation évite de stagner dans un minimum local

36

INF8500 SE: Conception et vérification

Algorithmes génétiques

- Voici la fonction de ressemblance F_i peut être définie de la façon suivante¹:

$$F_i = \max(C_i) - C_i$$

1. Wangtong et al., « Comparing Three Heuristic Search Methods for Functional Partitioning in Hardware-Software Codesign », pp. 430 et 431.

37

INF8500 SE: Conception et vérification

Algorithmes génétiques

Avec

$$C_i = \begin{cases} K_1 A_i + \phi B_i & \text{if } A_i > 0 \\ \phi B_i & \text{if } A_i \leq 0 \end{cases}$$

where C_i = cost of member i in the population.

K_1, ϕ = constants.

$A_i = (HWarea_i - A_{constraint})$

= the amount of hardware area exceeding the area constraint.

B_i = processing time of member i in the population.

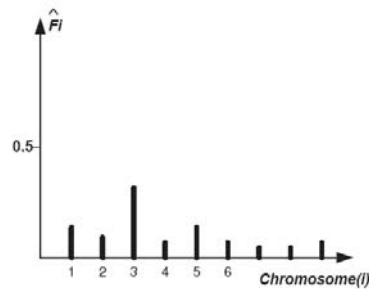
38

INF8500 SE: Conception et vérification

Algorithmes génétiques

On peut normalisé:

$$\hat{F}_i = \frac{F_i}{\sum_{\text{all } i's} F_i}$$



39

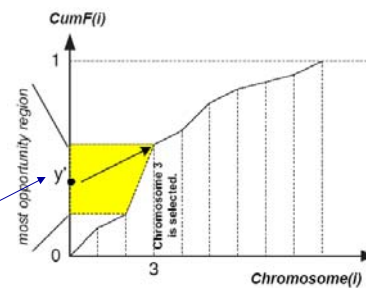
INF8500 SE: Conception et vérification

Algorithmes génétiques

- Finalement on se sert de la cumulation qui va permettre la sélection d'un parent à partir de la génération d'un nombre aléatoire:

$$CumF_i = \sum_{i=1}^N \hat{F}_i$$

Les probabilités sont fortes pour tomber souvent dans la zone y' et ainsi obtenir 3



40

INF8500 SE: Conception et vérification

Algorithmes génétiques

