

ORCYAE

Ordonnancement cyclique pour architectures embarquées

Claire Hanen

Professeur à Paris X Nanterre

Membre du LIP6

Projet financé par Digiteo à partir d'octobre 2007

Thèse: Abir Benabid

Motivation



- Processeurs VLIW embarqués:
 - Abaisser la consommation d'énergie et de mémoire
 - Augmenter la performance des traitements multimédia
 - 90% du temps d'exécution généré par les boucles
- Ordonnancement cyclique d'instruction
 - ⇒ Efficacité des ordonnancements générés
- Génération de code embarquée.
 - ⇒ Algorithmes rapides

Objectifs scientifiques

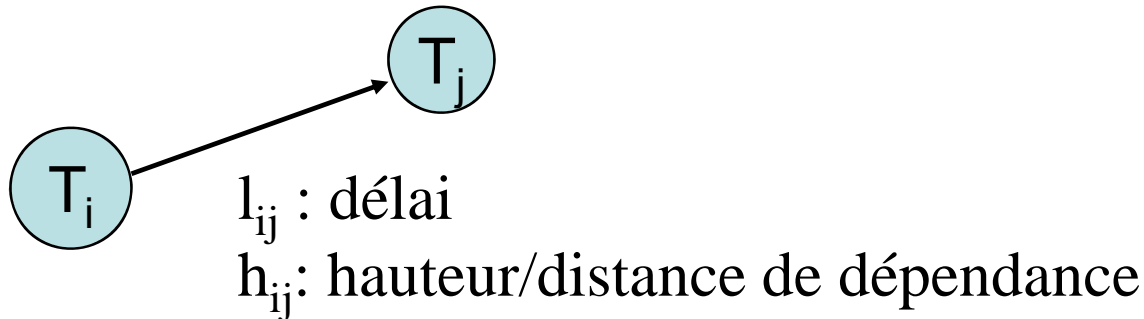
- Définir un modèle de problème cyclique adapté aux architectures VLIW actuelles.
- Définir des méthodes approchées et en évaluer les performances
 - Temps de calcul
 - Ratio cycle obtenu/cycle optimal
- Dans le pire des cas (garantie théorique)
- En pratique sur des benchmarks réels
 - ⇒ méthode exacte pour obtenir une solution optimale.

Partenaires extérieurs

- Collaboration avec projet RCPSP cyclique du GDR RO (Alix Munier-Kordon)
associés: LIP6, LAAS, STMicroelectronics.
- Collaboration avec B. de Dinechin, STMicroelectronics pour les jeux de données.

Le modèle retenu : tâches

- Graphe uniforme avec délais



$$\forall k > 0, s(T_i, k) + 1 + l_{ij} \leq s(T_j, k + h_{ij})$$

Le modèle retenu: les ressources

Tâches typées: k types de ressources ressource j ,
disponibilité m_j

Chaque tâche T_i utilise une unité de ressource d'un type unique u_j pendant une unité de temps.

RCPSP unitaire: une tâche peut utiliser plusieurs types de ressources simultanément (une unité de chaque type pendant une unité de temps)

RCPSP : une tâche peut utiliser plusieurs unités de ressources de plusieurs types.

```

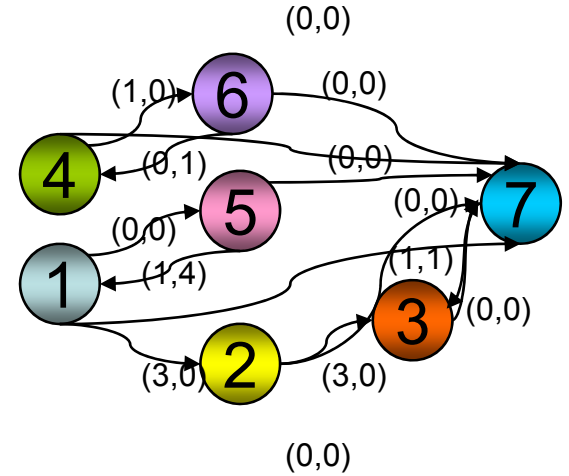
int
prod(int n, short a[ ], short b){
    int s=0, i;
    for (i=0; i<n; i++){
        s+=a[i]*b;
    }
    return s;
}

```

```

prod:
LDH A = 0, B
MUL C = D, A
ADD E = E, C
ADD F = F, 1
ADD G = G, 2
CMPNE H = I, F
BRF H, prod

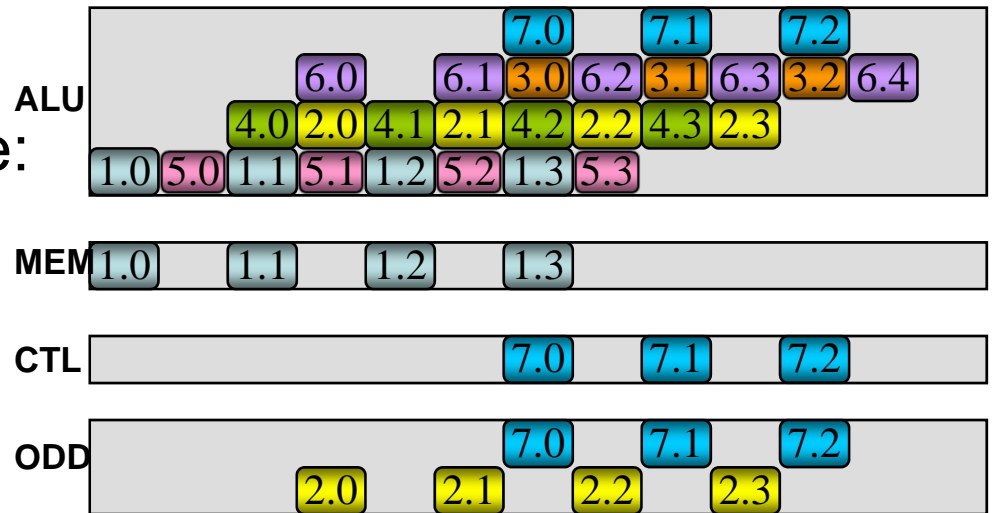
```



λ



Ordonnancement périodique:
 $S(T_i, k) = s(T_i, 0) + \lambda k$



Minimiser λ
 Temps de cycle

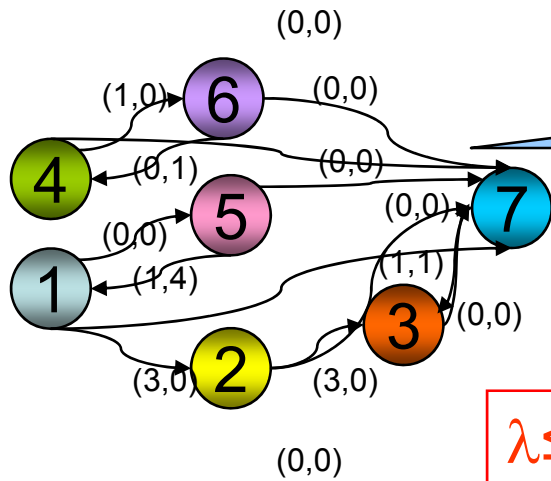
Travaux antérieurs en cyclique

<i>Problèmes</i>	<i>Complexité</i>	<i>Méthodes exactes</i>	<i>Méthodes approchées</i>
Sans ressources	P	Algorithmes de chemins	
Machines parallèles sans délais.	P si G acyclique, UET ? Pour UET, $m=2$ NP-difficiles cas général	PLNE	Algorithmes d'insertion Approche DSP Garantie $2-1/m$
Machines typées sans délais.	NP-difficiles sauf si G acyclique	Méthode arborescente spécifique si 1 machine par type/ PLNE	

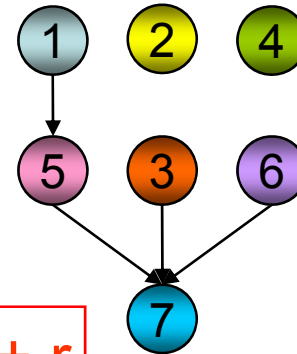
L'approche D_{ecomposed} S_{oftware} P_{ipelining}

- Gasperoni, Scwhiegelsohn, Eisenbeis, Vivien, Darte

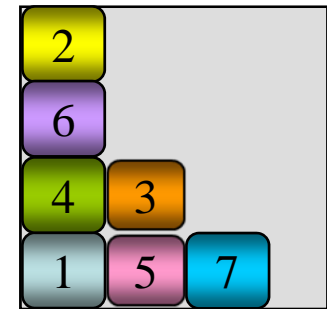
Retiming



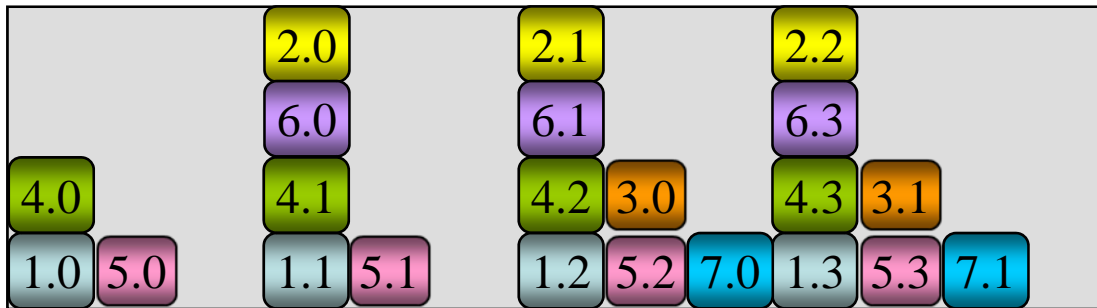
$$\lambda \leq (2 - 1/m) \lambda^{\text{opt}} + r$$



List algorithm



Pattern of scheduling



DSP pour RCPSP unitaire et délais

- Mise en évidence d'un problème non cyclique sous-jacent après retiming:

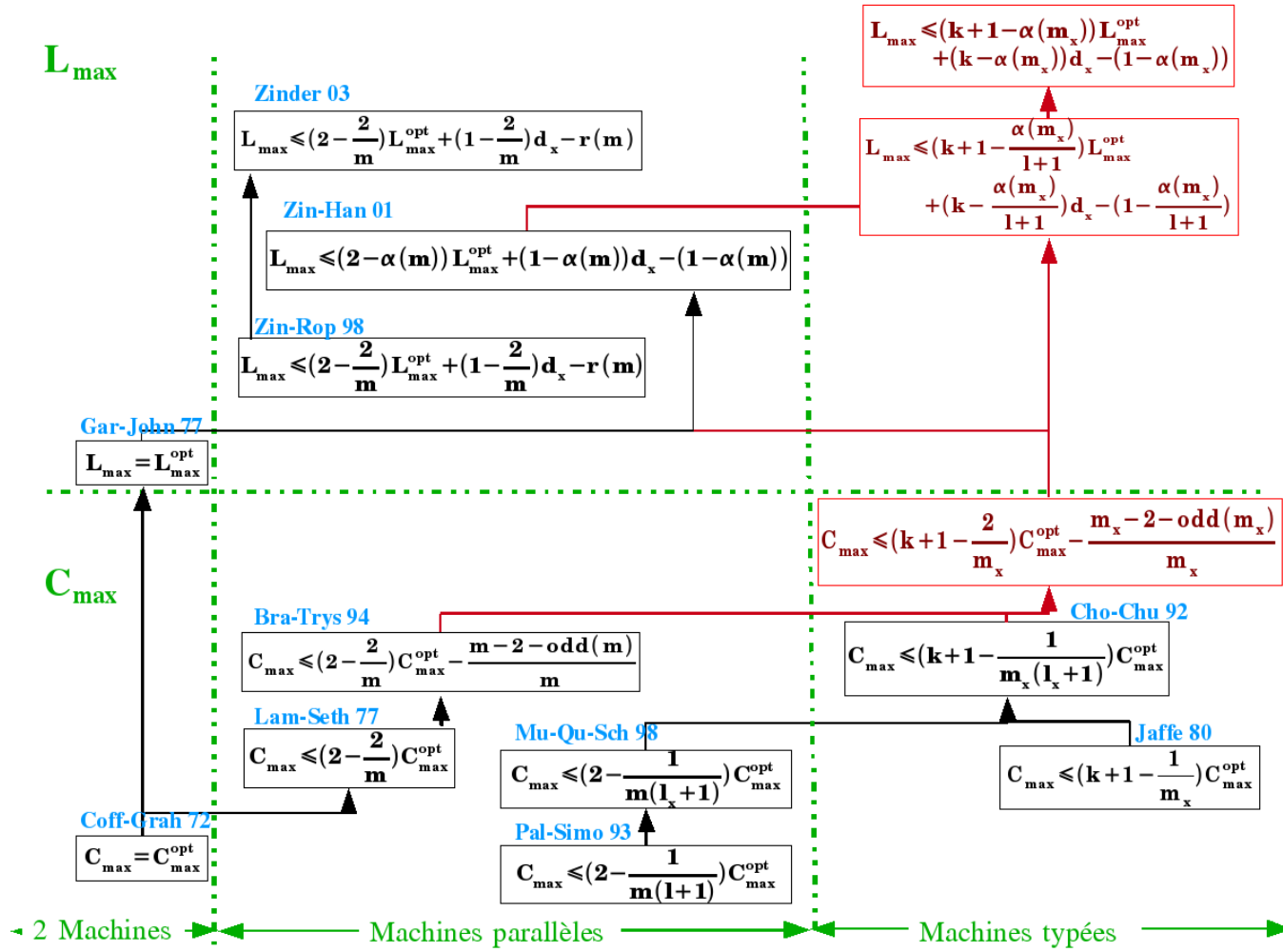
T_i -> latence q_i ;

minimiser $G_{\max} = \max(s(T_i) + 1 + q_i)$

En utilisant résultat de Chou(1992) :

$$\lambda \leq (k + 1 - (1/M \cdot (\delta + 1))) \lambda^{\text{opt}} + (1 - (1/M \cdot (\delta + 1))) \delta$$

Ordonnancement non cyclique



Perspectives

- Trouver un retiming qui donne une meilleure garantie.
 - Vers la solution du problème à 2 machines?
- Proposer des alternatives à DSP
- Expérimenter les algorithmes approchés sur les jeux de données issu du compilateur ST200
- Définir une méthode arborescente basée sur DSP pour obtenir la solution optimale.
- Généraliser au cas RCPSP unitaire et RCPSP