Outline

1. Introduction au cours
   - Compilation et optimisations de codes
   - Des p'tites boucles, toujours des p'tites boucles
   - Exemples de spécificités architecturales
2. Pipeline logiciel
   - Sans contraintes de ressources
   - Compaction de boucles
   - Optimisations des durées de vie

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one "shadow".

Sequential code

```
L400 :
  ld[r26], r27
    nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
    nop
  sub.f r27, r25, r0
  bne L400
    nop
L399 :
```

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one “shadow”.

Sequential code

```
L400 :
  ld[r26], r27
    nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
    nop
  sub.f r27, r25, r0
  bne L400
    nop
L399 :
```

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one “shadow”.

Sequential code

```
L400 :
  ld[r26], r27
     nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
     nop
  sub.f r27, r25, r0
  bne L400
     nop
L399 :
```

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one “shadow”.

Sequential code

```
L400 :
  ld[r26], r27
     nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
     nop
  sub.f r27, r25, r0
  bne L400
     nop
L399 :
```

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one “shadow”.

Sequential code

```
L400 :
  ld[r26], r27
    nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
    nop
  sub.f r27, r25, r0
  bne L400
    nop
L399 :
```

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one "shadow".

Sequential code

```
L400 :
  ld[r26], r27
     nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
     nop
  sub.f r27, r25, r0
  bne L400
     nop
L399 :
```

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one “shadow”.

Sequential code

```
L400 :
  ld[r26], r27
     nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
     nop
  sub.f r27, r25, r0
  bne L400
     nop
L399 :
```

$8n$ cycles.

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one “shadow”.

Sequential code	Code compaction
L400 :	L400 :
ld[r26], r27	**ld**[r26], r27
nop	nop
add r27, 6740, r26	**add** r27, 6740, r26
ld 0x1A54[r27], r27	**ld** 0x1A54[r27], r27
nop	nop
sub.f r27, r25, r0	**sub.f** r27, r25, r0
bne L400	**bne** L400
nop	nop
L399 :	L399 :

$8n$ cycles.

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one “shadow”.

Sequential code

```
L400 :
  ld[r26], r27
    nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
    nop
  sub.f r27, r25, r0
  bne L400
    nop
L399 :
```

Code compaction

```
L400 :
  ld[r26], r27
    nop
  ld 0x1A54[r27], r27
  add r27, 6740, r26
    nop
  sub.f r27, r25, r0
  bne L400
    nop
L399 :
```

$8n$ cycles.

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one “shadow”.

Sequential code

```
L400 :
  ld[r26], r27
    nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
    nop
  sub.f r27, r25, r0
  bne L400
    nop
L399 :
```

$8n$ cycles.

Code compaction

```
L400 :
  ld[r26], r27
    nop
  ld 0x1A54[r27], r27
  add r27, 6740, r26
  sub.f r27, r25, r0
  bne L400
    nop
L399 :
```

$7n$ cycles.

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one "shadow".

Sequential code	Code compaction	Sofware pipelining
L400 : **ld**[r26], r27 nop **add** r27, 6740, r26 **ld** 0x1A54[r27], r27 nop **sub.f** r27, r25, r0 **bne** L400 nop L399 :	L400 : **ld**[r26], r27 nop **ld** 0x1A54[r27], r27 **add** r27, 6740, r26 **sub.f** r27, r25, r0 **bne** L400 nop L399 :	L400 : **ld**[r26], r27 nop **add** r27, 6740, r26 **ld** 0x1A54[r27], r27 nop **sub.f** r27, r25, r0 **bne** L400 nop L399 :
$8n$ cycles.	$7n$ cycles.	

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one “shadow”.

Sequential code

```
L400 :
  ld[r26], r27
    nop
  add r27, 6740, r26
  ld 0x1A54[r27], r27
    nop
  sub.f r27, r25, r0
  bne L400
    nop
L399 :
```

8*n* cycles.

Code compaction

```
L400 :
  ld[r26], r27
    nop
  ld 0x1A54[r27], r27
  add r27, 6740, r26
  sub.f r27, r25, r0
  bne L400
    nop
L399 :
```

7*n* cycles.

Sofware pipelining

```
  ld[r26], r27
    nop
  add r27, 6740, r26
L400 :
  ld 0x1A54[r27], r27
    nop
  sub.f r27, r25, r0
  bne L400
  ld[r26], r27
    nop
  add r27, 6740, r26
L399 :
```

Software pipelining : example for LANai 3.0

Optimize throughput, with dependence & resource constraints.

Ex : sequential, pipelined LANai3.0, load & branch = one "shadow".

Sequential code	Code compaction	Sofware pipelining
L400 : **ld**[r26], r27 nop **add** r27, 6740, r26 **ld** 0x1A54[r27], r27 nop **sub.f** r27, r25, r0 **bne** L400 nop L399 :	L400 : **ld**[r26], r27 nop **ld** 0x1A54[r27], r27 **add** r27, 6740, r26 **sub.f** r27, r25, r0 **bne** L400 nop L399 :	**ld**[r26], r27 nop **add** r27, 6740, r26 L400 : **ld** 0x1A54[r27], r27 **ld**[r26], r27 **sub.f** r27, r25, r0 **bne** L400 **add** r27, 6740, r26 L399 :
$8n$ cycles.	$7n$ cycles.	$3 + 5n$ cycles.

Modèle(s) de processeurs

- Unités parfois pipelinées, parfois non-pipelinées, parfois plus complexes.
- Unités spécifiques ou non.
- Durée d d'une opération: deux significations.
  - durée d'utilisation de la ressource: $d(u)$ pour une opération u.
  - délai entre deux opérations dépendantes: $d(e)$ pour un arc e.
- Distance de dépendances: $w(e)$ pour un arc e.
- Contraintes de registres.

☛ Modèle plus général = graphes avec délais sur sommets et arcs + tables d'utilisation des ressources (+ contraintes de registres).

Ordonnancement cyclique (pipeline logiciel)

Trouver un ordonnancement t tel que:

- $\forall\, e = (u, v)$, $\forall k \in \mathbb{N}$, $t(v, k) \geq t(u, k - w(e)) + d(u)$ (dépendances).
- Les contraintes de ressources sont satisfaites.
- $\lambda = \liminf_{k\to\infty} \dfrac{\max\{t(v,i) + d(v) \mid v \in V,\ i \leq k\}}{k}$ est minimal.

λ est le **temps de cycle moyen** (ou l'“initiation interval”). On cherchera des ordonnancements cycliques: $t(u, k) = \lambda k + \rho_u$. Alors période = temps de cycle moyen.

Bornes inférieures

Soit $T = \max\{t(v,i) + d(v) \mid v \in V,\ i \leq k\}$.

- Si on dispose de p ressources: $pT \geq k \sum_{v \in V} d(v)$ ☛ $\boxed{\lambda \geq \frac{\sum_v d(v)}{p}}$
- Soit C un circuit. $t(v,k) \geq t(v, k - w(C)) + d(C)$, puis $t(v,k) \geq t(v, k - \lfloor \frac{k}{w(C)} \rfloor w(C)) + \lfloor \frac{k}{w(C)} \rfloor d(C)$. Donc $T \geq \lfloor \frac{k}{w(C)} \rfloor d(C)$. On en déduit:

$$\frac{T}{k} \geq \left(\frac{k}{w(C)} - 1\right)\frac{d(C)}{k} = \left(1 - \frac{w(C)}{k}\right)\frac{d(C)}{w(C)} \quad ☛ \quad \boxed{\lambda \geq \frac{d(C)}{w(C)}}$$

Sans contrainte de ressources: ☛ $\boxed{\lambda_\infty \geq \max\{\lceil \frac{d(C)}{w(C)} \rceil \mid C \text{ circuit}\}}$

Ordonnancements cycliques

Ordonnancement cyclique de la forme $\sigma(u,k) = \lambda k + \rho_u$, $\lambda \in \mathbb{N}$, $\rho_u \in \mathbb{N}$.

Dépendances $\sigma(v, k + w(e)) \geq \sigma(u,k) + d(u)$, c'est-à-dire $\rho_v + \lambda w(e) \geq \rho_u + d(u)$. En sommant sur un circuit, on retrouve $\lambda w(C) \geq d(C)$.

Astuce à la Bellman-Ford Dans le graphe $G'_\lambda = (V, E, w')$ avec $w'(e) = d(u) - \lambda w(e)$ pour $e = (u,v)$, tous les circuits sont de poids négatif ou nul si et seulement si $\lambda w(C) \geq d(C)$.

Calcul de λ_∞: complexité $O(|V|(|V|+|E|)\log(|V|d_{\max})$

Pour λ fixé on construit le graphe $G'_\lambda = (V, E, w')$ avec
$w'(e) = d(u) - \lambda w(e)$ si $e = (u, v) \in E$.

Algorithme de Bellman-Ford Si G' a un circuit de poids strictement positif alors λ est trop petit, sinon λ est trop grand.

Recherche dichotomique dans l'intervalle $[0; \sum d(v)]$.

Pour λ_∞ On note ρ_v le plus long chemin arrivant en v dans G'_λ. On a alors $\rho_v \geq \rho_u + w'(e)$, c'est-à-dire:

$$\rho_v + \lambda w(e) \geq \rho_u + d(u)$$

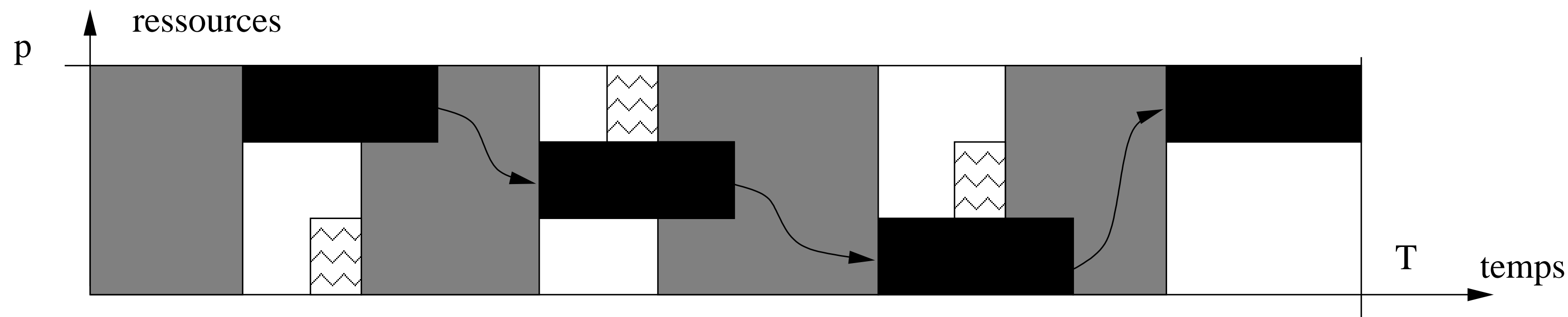
☛ l'ordonnancement cyclique $\sigma(u, k) = \rho_u + \lambda k$ est optimal!

Note calculer $\frac{d(C)}{w(C)}$ au lieu de $\lceil \frac{d(C)}{w(C)} \rceil$ est possible mais plus compliqué.

DAGs, "list scheduling" et borne de Graham

$T_l(p)$: temps d'exécution d'un ordonnancement de liste pour p ressources.

- Pour tout chemin $\mathcal{P}$, $T_l(p) \geq d(\mathcal{P})$ (durée du chemin).
- "Surface" de calcul: $pT_l(p) = \text{Idle time} + \text{ Travail } (= \sum_{v \in V} d(v))$.
- Il existe un chemin $\mathcal{P}_0$ tel que Idle time $\leq (p-1)d(\mathcal{P}_0)$.



Conséquence pour l'ordonnancement acyclique:

$pT_l(p) \leq (p-1)d(\mathcal{P}_0) + \sum_{v \in V} d(v)$ ☛ $\boxed{T_l(p) \leq (2 - \frac{1}{p})T_{\text{opt}}(p)}$

Borne de Graham pour l'ordonnancement de liste

$T_l(p)$: temps d'exécution d'un ordonnancement de liste pour p ressources.

- Pour tout chemin $\mathcal{P}$, $T_l(p) \geq d(\mathcal{P})$ (durée du chemin).
- "Surface" de calcul: $pT_l(p) = \text{Idle time} + \text{ Travail } (= \sum_{v \in V} d(v))$.
- Il existe un chemin $\mathcal{P}_0$ tel que Idle time $\leq (p-1)d(\mathcal{P}_0)$.

Conséquence pour l'ordonnancement acyclique

$pT_l(p) \leq (p-1)d(\mathcal{P}_0) + \sum_{v \in V} d(v)$ ☛ $\boxed{T_l(p) \leq (2 - \frac{1}{p})T_{\text{opt}}(p)}$

Conséquence sur la compaction de boucle

$\lambda = T_l(p) \leq \frac{p-1}{p} d(\mathcal{P}_0) + \underbrace{\frac{\sum_{v \in V} d(v)}{p}}_{\leq \lambda_p}$ ☛ $\boxed{\lambda \leq (1 - \frac{1}{p})d(\mathcal{P}_0) + \lambda_p}$

Différents types de stratégies de pipeline logiciel

Simple compaction de boucles avec expansion de scalaires. Très limité en présence de circuits (Ex : C2H, outil HLS d'Altera).

Iterative modulo scheduling B. Rau. Iterative modulo scheduling. Int. Journal of Parallel Programming, 24(1) :3-64, 1996. (Le "modulo scheduling" date de 1981, Rau & Glaeser)

Decomposed software pipelining Wang et Eisenbeis, PACT'93.

☛ Lien entre compaction de boucles, "decomposed software pipelining" et "retiming".
☛ Variantes pour le "stage scheduling" (réorganisation pour modifier les durées de vie)

Outline

1. Introduction au cours
   - Compilation et optimisations de codes
   - Des p'tites boucles, toujours des p'tites boucles
   - Exemples de spécificités architecturales
2. Pipeline logiciel
   - Sans contraintes de ressources
   - Compaction de boucles et retiming
   - Optimisations des durées de vie
3. Fusion de boucles
   - Intérêts et problèmes
   - Fusion de boucles simple : variantes
   - Fusion avec décalage

Choix du décalage

Idée double:

- Minimiser le chemin critique pour la compaction.

  ☛ Équivalent à minimiser la durée maximale d'un chemin n'ayant que des arcs de poids nul (**période d'horloge**). Algorithme de Leiserson et Saxe, complexité $O(VE\log(V))$.

- Minimiser le nombre de contraintes pour la compaction.

  ☛ Équivalent à miminiser le **nombre d'arcs de poids nul**. Variante de l'algorithme out-of-kilter, complexité $O(E^2)$ (version de base) ou $O(V^2E)$ sous contrainte de période d'horloge.

Retiming et circuits

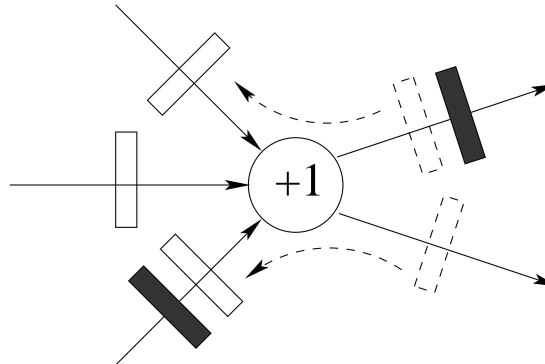
Correspondances Décalage = retiming. Graphe = circuit.

$w(e)$ = nombre de registres sur l'arc e. $d(u)$ = durée de l'opérateur u.

Période d'horloge $\Phi(G)$ = durée maximale d'un chemin sans registres.

Retiming Fonction $r\ :\ V \to \mathbb{Z}$ telle que, pour tout arc $e = (u, v)$,

$w_r(e) = w(e) + r(v) - r(u) \geq 0$ (nombre positif de registres).



But Trouver r tel que $\Phi(G_r)$ soit minimal.