ANALYSE DE TEMPS D'EXÉCUTION POUR DES APPLICATIONS TEMPS-RÉEL

CHRISTINE ROCHANGE

Plan

- Système temps-réel et temps d'exécution pire cas
- Comment évaluer le WCET d'une tâche ?
  - Mesures
  - Analyse statique
- Modélisation de l'architecture cible
  - pipeline
  - caches
  - prédicteur de branchement
- Prise en compte de l'environnement
  - événements asynchrones
  - processeur multithread
  - processeur multicoeur
- Conclusion

Système temps-réel

« A real-time system is defined as a system whose correctness of the system depends not only on the logical results of computations, but also on the time at which the results are produced »

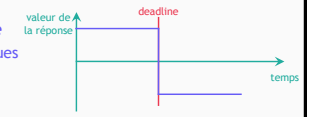
J. Stankovic (1988)





Niveau de criticité

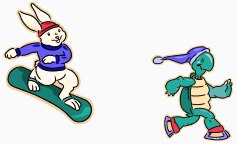
- **Temps réel strict ou dur**
  - il est interdit de rater une échéance
    - les conséquences peuvent être tragiques
  - exemples : commandes de vol, ABS



- **Temps réel mou**
  - il y a des contraintes temporelles mais ce n'est pas dramatique si elles ne sont pas satisfaites très ponctuellement
  - exemples :
    - vidéo à la demande, VoIP
  - conséquence possible : moindre qualité de service

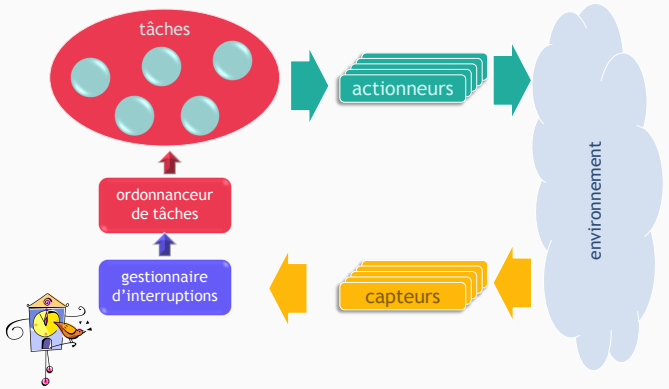
Malentendus fréquents*

- Système temps-réel = système rapide et performant
- Quasiment tous les problèmes du temps réel ont été résolus dans d'autres domaines de l'informatique …
- … et l'augmentation de la vitesse des processeurs va résoudre les autres

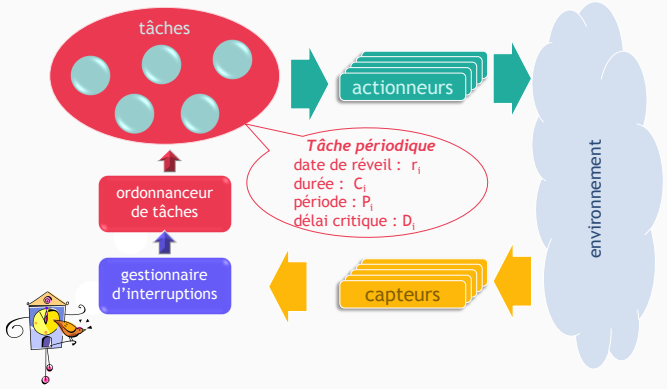


* J. Stankovic, « Misconceptions about real-time computing », IEEE Computer, 1988

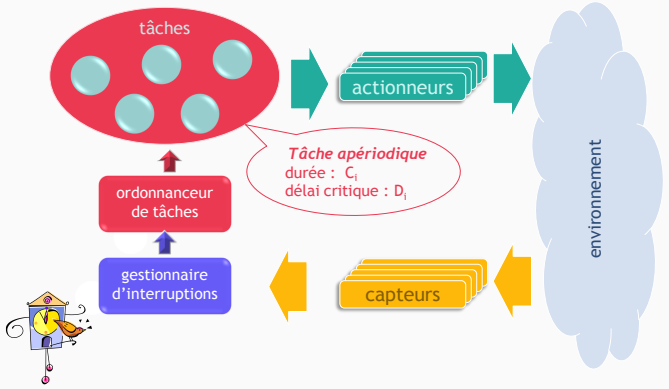
Système embarqué temps-réel



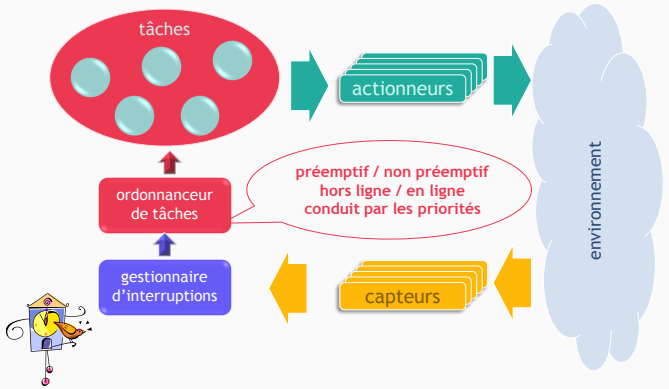
Système embarqué temps-réel



Système embarqué temps-réel



Système embarqué temps-réel



Temps d'exécution d'un programme

- **Exemple**

```
somme = 0;
nbpairs = 0;
for (i=0 ; i < N ; i++)
{
   somme += tab[i];
   if (tab[i] % 2 == 0)
      nbpairs ++;
}
```

```
        mov   r0,#0       @ r0:somme
        mov   r1,#0       @ r1:nbpairs
        mov   r2,#0       @ r2:i
        adr   r10,tab
        adr   r11,N
        ldr   r11,[r11] @ r11:N
for:    cmp   r2,r11
        bcs   fin
        ldr   r4,[r10],#4
        add   r0,r0,r4
if:     and   r3,r4,#1
        cmp   r3,#0
        bne   suite
        add   r1,r1,#1
suite:  add   r2,r2,#1
        b     for
fin:
```

Hypothèses :
- pas de pipeline
- chaque instruction dure 1 cycle

Temps d'exécution d'un programme

```
        mov   r0,#0
        mov   r1,#0
        mov   r2,#0
        adr   r10,tab
        adr   r11,N
        ldr   r11,[r11]
for:    cmp   r2,r11
        bcs   fin
        ldr   r4,[r10],#4
        add   r0,r0,r4
if:     and   r3,r4,#1
        cmp   r3,#0
        bne   suite
        add   r1,r1,#1
suite:  add   r2,r2,#1
        b     for
fin:
```

1 × 6 cycles

(N+1) × 2 cycles

N × 5 cycles

n × 1 cycle

N × 2 cycles

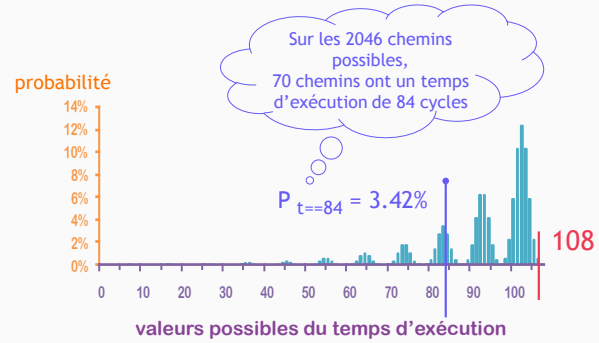
$T = 8 + 9 \times N + n$

si N = 5 et n = 2 :
T = 55
10 chemins possibles

si N = 8 et n = 4 :
T = 84
70 chemins possibles

si N = 10 et n = 10 :
T = 108

Temps d'exécution d'un programme



Sources de variabilité du temps d'exécution (1)

- **Les entrées du programme**
  - issues de capteurs
  - les données déterminent le chemin d'exécution par le biais des branchements conditionnels
  - exemples :

```
index = 0;
while (data_in[index] != 0) {
    /* traitement */
}
```

Combien d'itérations ?

```
if (température > T_MAX) {
    /* traitement anomalie */
}
else {
    /* traitement normal */
}
```

Quelle branche ?

Sources de variabilité du temps d'exécution (2)

- **Le matériel**
  - le temps d'exécution peut dépendre de l'état du processeur
    - exemple : contenu des mémoires caches

```
somme = 0;
for (i=0 ; i<100 ; i++){
    somme += tab[i];
}
```

Données dans le cache ?

    - mais aussi : état du pipeline, du prédicteur de branchement, …

Sources de variabilité du temps d'exécution (3)

- **L'environnement**
  - la tâche n'est pas toujours seule à s'exécuter sur le matériel
    - conflits d'accès au cache L2 partagé, au bus, à la mémoire, …

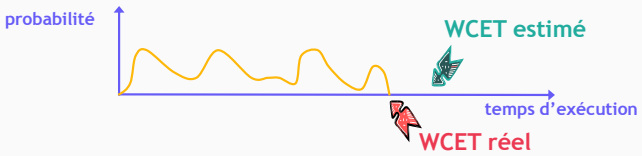
```
somme = 0;
for (i=0 ; i<100 ; i++){
    somme += tab[i];
}
```

Combien de temps pour accéder à la mémoire ?

  - certains événements asynchrones peuvent avoir un impact sur les latences
    - rafraîchissement mémoire
    - détection et correction d'erreurs (SEU)
    - DMA

Temps d'exécution pire cas

- **WCET = Worst-Case Execution Time**



- **Nécessité de :**
  - sûreté : pas de sous-estimation !
  - précision

Pourquoi évaluer le WCET ?

- **Une seule tâche**
  - vérification
    - est-ce que la tâche respecte les contraintes de temps ?
  - dimensionnement
    - le matériel est-il sur- ou sous-dimensionné ?
  - optimisation de code
    - quel est le chemin critique ?
- **Plusieurs tâches**
  - ordonnancement
    - comment ordonnancer les tâches pour que les contraintes temporelles soient respectées ?
  - test d'ordonnançabilité
    - les tâches peuvent-elles être ordonnancées de manière à respecter leurs échéances respectives ?

Test d'ordonnançabilité

- **Contexte**
  - Tâches périodiques (de périodes P_i) avec des échéances $D_i \leq P_i$
  - Ordonnancement par priorités fixes
    (D_i - le plus faible = le plus prioritaire)
  - Temps d'exécution pire cas : C_i
- **Condition nécessaire** $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$
- **Condition suffisante** $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n.(2^{\frac{1}{n}} - 1)$

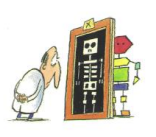
Plan

- Système temps-réel et temps d'exécution pire cas
- Comment évaluer le WCET d'une tâche ?
  - Mesures
  - Analyse statique
- Modélisation de l'architecture cible
  - pipeline
  - caches
  - prédicteur de branchement
- Prise en compte de l'environnement
  - événements asynchrones
  - processeur multithread
  - processeur multicoeur
- Conclusion

Comment évaluer le WCET ?

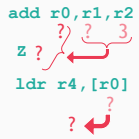
- **Mesures**
  - sur matériel
  - sur simulateur

- **Analyse statique**



Mesure du temps d'exécution pire cas (1)

- **Quel(s) jeu(x) d'entrées ?**
  - le jeu d'entrée qui conduit au temps le plus long
    - difficile à identifier dans le cas général
  - un ensemble de jeux d'entrées qui assurent une couverture totale des chemins
    - pas toujours identifiables
  - simulation symbolique
    - les données en entrées sont marquées « inconnues »
    - les valeurs inconnues sont propagées
    - lorsqu'un branchement dont la condition est inconnue est exécuté, les deux branches sont explorées



Mesure du temps d'exécution pire cas (2)

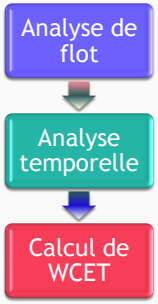
- **Difficultés**
  - la mesure/simulation de tous les chemins est trop longue en général
  - il peut être difficile d'initialiser l'état du matériel et de prendre en compte tous les états possibles
  - comment prendre en compte l'environnement ?
    - autres tâches, événements extérieurs, …
- **Solution : analyse statique**
  - objectif = prendre en compte :
    - toutes les entrées possibles
    - tous les contextes d'exécution possibles

Analyse statique du WCET

Quels sont les chemins possibles ?
- bornes de boucles ?
- chemins exclusifs ?

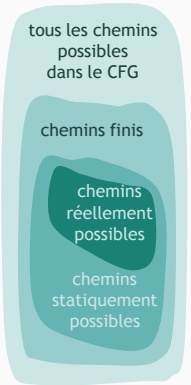
Temps d'exécution des chemins ?
ou de chemins partiels (blocs de base)

Chemin le plus long ?



Analyse de flot

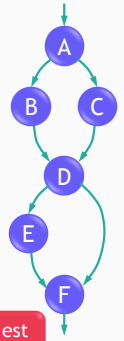
- **Objectif**
  - déterminer tous les chemins possibles
- **Résultats attendus**
  - quelles fonctions peuvent être appelées ?
  - combien de fois les boucles itèrent-elles ?
  - existe-il des dépendances entre structures du type si … alors … sinon … ?
- **Exigences**
  - les résultats doivent être sûrs
  - c'est mieux s'ils sont précis



Analyse de flot : exemple 1

```
int f(int x) {
  if (x < 5)
    x = x + 1;
  else
    x = x * 2;
  if (x > 10)
    x = 10;
  return(x);
}
```

A
B
C
D
E
F



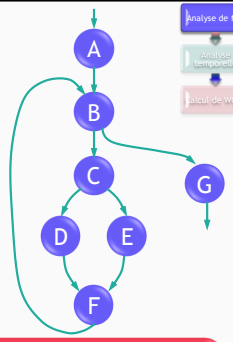
Informations de flot ?

le chemin A-B-C-D-E-F est impossible

Analyse de flot : exemple 2

```
int f(unsigned int x) {
  char i=0;
  while (i<100) {
    if ( i<20  || x<30 )
      x = x + 1;
    else
      x = x * 2;
    i++;
  }
  return(x)
}
```

A
B
C
D
E
F
G



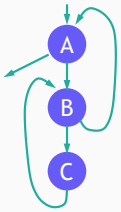
Informations de flot ?

- la boucle itère au plus 100 fois
- E ne peut pas être exécuté dans les 20 premières itérations
- D ne peut être exécuté que dans les 30 premières itérations

Analyse de flot : exemple 3

```
for (i=0 ; i<N ; i++)
  for (j=0 ; j<i ; j++) {
    … calcul C …
  }
```

A
B
C



Informations de flot ?

- la boucle *i* itère au plus *N* fois
- la boucle *j* itère au plus *N* fois
- le bloc C est exécuté au plus N.(N+1)/2 fois

Analyse de flot : exemple 4

```
for (i=0 ; i<20 ; i++)
{
  p2[i] = 0;
  prod = 1;
  while (prod < i)
    prod = prod << 1;
  if (prod == i)
    p2[i] = 1;
}
```

Informations de flot ?

- le nombre total d'itérations de la boucle while est 64
- la condition du if est vraie 4 fois

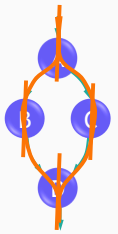
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
while	0	0	1	2	2	3	3	3	3	4	4	4	4	4	4	4	4	5	5	5
if	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

Extraction de flot

- **Méthodes d'analyse automatique**
  - à partir du code (source ou objet), sans intervention de l'utilisateur
  - différentes méthodes
    - de complexité différente
    - capables de gérer différents niveaux de complexité pour le programme analysé
- **Prise en compte d'annotations manuelles**
  - pour des codes trop complexes
  - fiabilité ?

Analyse temporelle

- **Objectif : éviter les redondances**
  - analyse de sous-chemins
    - instruction
    - bloc de base
    - séquence de blocs de base
  - exemple :
    - au lieu de $t_{A\text{-}B\text{-}D}$ et $t_{A\text{-}C\text{-}D}$
    - on évalue t_A, t_B, t_C, t_D

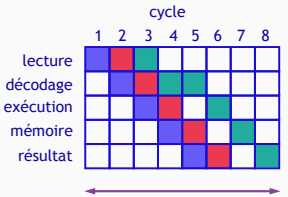


Temps d'exécution d'un bloc de base

- **Processeur sans pipeline**
  - somme des latences des instructions
    - cf. manuel utilisateur
- **Processeur avec pipeline**
  - prendre en compte le recouvrement entre instructions
    - plusieurs instructions dans des étages différents
    - calculer l'occupation des étages à chaque cycle
      - ➢ simulateur
      - ➢ table de réservation

Temps d'exécution d'un bloc de base

- **Table de réservation**

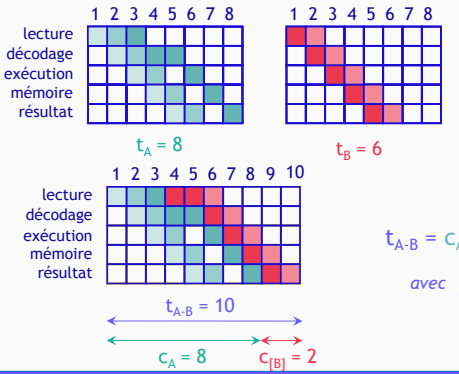


temps d'exécution = 8 cycles

bloc de base

```
add   r1,r15,#24
ldr   r2,[r1]
add   r3,r2,#5
```

Coût d'exécution d'un bloc de base



A

```
add   r1,r15,#24
ldr   r2,[r1]
add   r3,r2,#5
```

B

```
mov   r4,r3,LSL #2
sub   r4,r4,#1
```

$t_{A-B} = c_A + c_B$

avec $c_A = t_A = 8$

$c_{B/[A]} = t_{A-B} - t_A = 2$

Calcul de WCET

- **Deux approches**
  - A partir du code source
    - possibilité d'exploiter la structure algorithmique
      arbre syntaxique
    - pas toujours disponible
      - exemple : bibliothèques
  - A partir du code exécutable
    - peut être désassemblé
      graphe de contrôle de flot
    - pas d'informations directes sur la structure algorithmique

Calcul de WCET à partir de l'arbre syntaxique

- **Extended Timing Schema**
  - séquence :

```
S : S1 ; S2 ; … ; Sn
```

$$W(S) = \sum_{i=1}^{n} W(S_i)$$

  - conditionnelle :

```
S : if ( c ) then S1 else S2
```

$$W(S) = W(c) + \max(W(S_1), W(S_2))$$

  - boucle :

```
S : for (init ; tst ; incr) S1
```

$$W(S) = W(init) + N \times [W(test) + W(S) + W(incr)] + W(test)$$

Calcul de WCET à partir du CFG

- **Méthode IPET (Li et Malik, 1995)**
  - problème d'optimisation
    - maximiser le temps d'exécution en respectant un certain nombre de contraintes
    - programmation linéaire en nombres entiers
      (ILP = integer linear programming)
  - expression du temps d'exécution :

$$T = \sum x_{ij}\, c_{j/[i]}$$

temps d'exécution d'un chemin

$c_{j/[i]}$: coût du bloc j dans la séquence i-j

x_{ij} : nombre d'exécutions de l'arc i-j

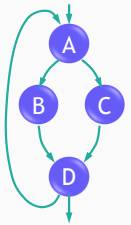
    - en maximisant T, on trouve le WCET

Calcul de WCET à partir du CFG

- **Contraintes**
  - structurelles
    - nb exécutions d'un bloc
      = nb exécutions des arcs entrants
      = nb exécutions des arcs sortants

$x_A = 1 + x_{DA}$ $\quad x_B = x_{AB} = x_{BD}$ $\quad x_D = x_{BD} + x_{CD}$

$x_A = x_{AB} + x_{AC}$ $\quad x_C = x_{AC} = x_{CD}$ $\quad x_D = 1 + x_{DA}$



  - de flot
    - bornes de boucles, …

$x_{DA} \leq N$

$x_{AB} = 0.5\ x_A$

Calcul de WCET à partir du CFG

- **Utilisation d'un outil de résolution ILP**

```
max: 8 + 3 xAB + 7 xAC + 0 xBD + 1 xCD + 6 xDA;

xA - xDA = 1;
xA - xAB - xAC = 0;
xB - xAB = 0;
xB - xBD = 0;
xC - xAC = 0;
xC - xCD = 0;
xD - xBD - xCD = 0
xD - xDA = 1;
xDA <= 9;
xAB - 0.5 xA = 0;
```

```
Value of objective function: 117

Actual values of the variables:
xA = 10
xB = 5
xC = 5
xD = 10
xAB = 5
xAC = 5
xBD = 5
xCD = 5
xDA = 9
```


Comparaison des deux méthodes

- **Calcul à partir de l'arbre syntaxique**
  - nécessite de connaître le code source
  - le code exécutable ne respecte pas forcément la structure algorithmique
  - calcul analytique, hiérarchique
  - le chemin le plus long est connu après recherche du WCET
- **Calcul à partir du CFG**
  - sur le code exécutable : permet de prendre en compte les optimisations
  - le temps de résolution peut être long
  - le chemin le plus long n'est pas directement identifié : on sait seulement quels blocs il contient

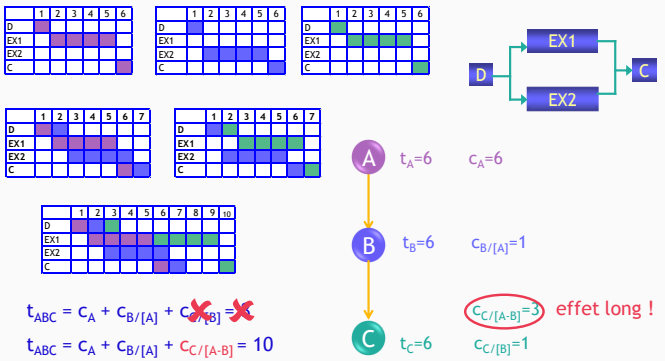
Plan

- Système temps-réel et temps d'exécution pire cas
- Comment évaluer le WCET d'une tâche ?
  - Mesures
  - Analyse statique
- Modélisation de l'architecture cible
  - pipeline
  - caches
  - prédicteur de branchement
- Prise en compte de l'environnement
  - événements asynchrones
  - processeur multithread
  - processeur multicoeur
- Conclusion

Analyse de l'architecture cible

- **En pratique, le calcul du temps d'exécution d'un bloc de base n'est pas si simple à cause :**
  - des interactions entre blocs de base
    - pipeline superscalaire
  - des mécanismes dynamiques
    - ordonnancement des instructions
    - exécution spéculative
  - des mécanismes qui exploitent l'historique :
    - caches
    - prédiction de branchement
  - ...

Interactions entre blocs de base distants

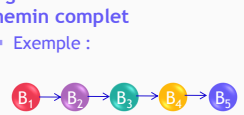


Origine des effets longs

- **Unités fonctionnelles à latences longues**
- **Parallélisme**
  - alignement des instructions
    - dans le pipeline
    - dans les unités fonctionnelles
- **Capacités limitées**
  - files d'instructions, tampon de réordonnancement, ...
  - nombre de branchements non résolus
- **???**

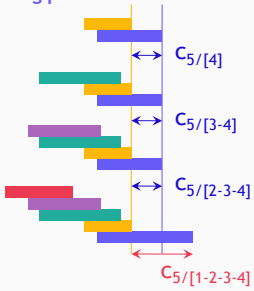
Caractéristiques des effets longs (1)

- **Leur longueur n'est pas bornable**
  - Engblom a montré comment un effet long peut concerner un chemin complet
    - Exemple :



$C_{5/[2-3-4]} = C_{5/[3-4]} = C_{5/[4]}$

$C_{5/[1-2-3-4]} \neq C_{5/[4]}$



Caractéristiques des effets longs (2)

- Exemple : mesure de toutes les séquences de 11 blocs ou moins pour un ensemble de benchmarks (processeur superscalaire d'ordre 4 – exécution non ordonnée)

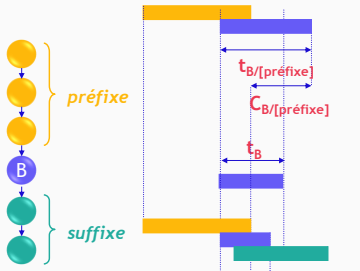
	longueur des effets longs (en nombre de blocs)								
programme	3	4	5	6	7	8	9	10	11
crc	0	26	26	0	0	12	3	0	0
jfdctint	0	4	2	4	1	1	1	2	0
ludcmp	0	15	15	5	2	4	3	1	0
fibcall	0	1	0	0	0	0	0	0	0
matmul	0	8	9	13	4	8	10	7	3
fft1	1	86	73	19	16	30	34	6	9
lms	3	31	19	3	18	12	0	18	12

Caractéristiques des effets longs (3)

- **Leur amplitude n'est pas bornable**
  - effets négatifs
    - peuvent être ignorés ⇒ risque de surestimation du WCET
  - effets positifs
    - ne doivent pas être ignorés
  - en pratique :
    - l'amplitude reste généralement limitée
    - mais on ne peut pas le prouver !

Caractéristiques des effets longs (4)

Ne pas oublier de tenir compte du suffixe en cas d'ordonnancement dynamique !



Comment évaluer le coût d'un bloc ?

- **Simulation symbolique** (Lundqvist et Stenström, 1999)
  - exploration (simulation) de tous les chemins en recherchant un point fixe
  - détermine tous les coûts possibles d'un bloc de base
- **Interprétation abstraite** (outil aiT d'AbsInt)
  - état abstrait du pipeline en entrée de chaque bloc = ensemble de tous les états possibles
  - analyse de l'effet de l'exécution de chaque bloc sur l'état abstrait du pipeline en entrée ⇨ état abstrait en sortie
  - recherche d'un point fixe

Comment évaluer le coût d'un bloc ?

- **Prise en compte d'un contexte pire-cas**
  - Li *et al.* (2004)
  - **Principe :**
    - calculer les dates de traitement des instructions dans les étages du pipeline en fonction du contexte
    - hypothèses optimistes et pessimistes sur le contexte
      → intervalles
    - coût *pire cas* d'un bloc =
      date de terminaison *au plus tard* de la dernière instruction du bloc
      –
      date de terminaison *au plus tôt* de l'instruction qui précède le bloc

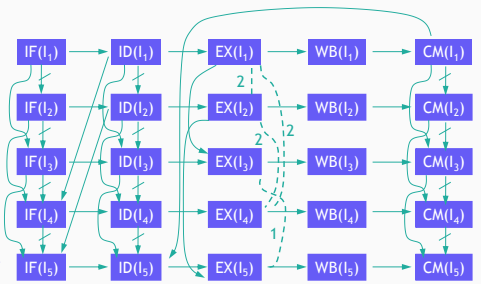
Graphes d'exécution (1)

```
I1  mult  r6,r10,4
I2  mult  r1,r10,r1
I3  sub   r6,r6,r2
I4  mult  r4,r8,r4
I5  add   r1,r1,r4
```

- **Exécution d'un bloc de base seul**

ordre pipeline
ordre programme
capacité iqueue
capacité ROB
dépendances
conflits possibles

hypothèses :
- file de chargement : 3 instructions
- ROB : 4 instructions
- 2 multiplieurs, 1 add.
- superscalaire 2 voies

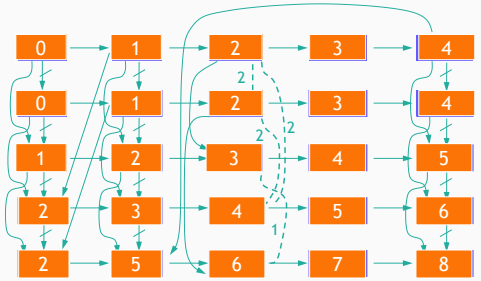


Graphes d'exécution (2)

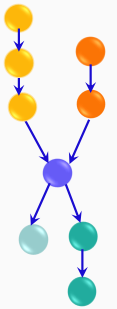
```
I1  mult  r6,r10,4
I2  mult  r1,r10,r1
I3  sub   r6,r6,r2
I4  mult  r4,r8,r4
I5  add   r1,r1,r4
```

- **Dates de démarrage**

hypothèse :
toutes les latences égales à 1 cycle

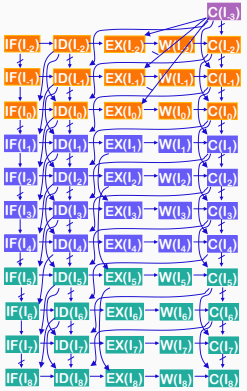


Graphes d'exécution (3)



4 contextes

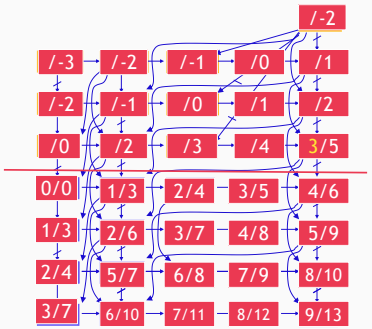
t=0



hypothèses :
- file de chargement : 2 instructions
- ROB : 2 instructions

Graphes d'exécution (4)

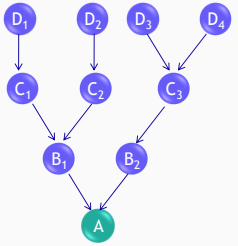
- **Dates de démarrage**
  - initialisation des dates au plus tard du prologue
  - calcul des dates au plus tôt et au plus tard
  - calcul de la date de fin au plus tard du dernier nœud du prologue
  - calcul du coût maximal
    - 10 cycles



Prise en compte des effets longs dans le calcul du WCET

- **Prise en compte du coût *pire cas***
  - coût tenant compte de tous les préfixes possibles
  - formulation IPET de base
    - contribution du bloc A au temps d'exécution :

$$x_{B1\text{-}A} \cdot c_{A/[B1]} + x_{B2\text{-}A} \cdot c_{A/[B2]}$$



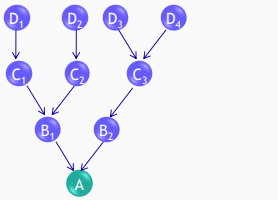
Prise en compte des effets longs dans le calcul du WCET

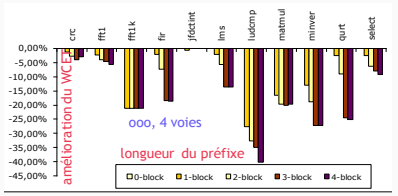
- **Prise en compte de coûts différenciés selon les préfixes**
  - contribution du bloc A au temps d'exécution :

$$x_{D1\text{-}C1\text{-}B1\text{-}A} \cdot c_{A/[D1\text{-}C1\text{-}B1]} + x_{D2\text{-}C2\text{-}B1\text{-}A} \cdot c_{A/[D2\text{-}C2\text{-}B1]} + x_{D3\text{-}C3\text{-}B2\text{-}A} \cdot c_{A/[D3\text{-}C3\text{-}B2]} + x_{D4\text{-}C3\text{-}B2\text{-}A} \cdot c_{A/[D4\text{-}C3\text{-}B2]}$$

  - générer de nouvelles contraintes dans la formulation IPET

$$x_{D1\text{-}C1\text{-}B1\text{-}A} \leq \ldots$$





Analyse du cache d'instructions

- **Quelle est latence de chargement d'une instruction ?**
  - dépend de sa présence ou non dans le cache
    - a-t-elle déjà été chargée ?
      - elle ou une de ses voisines (dans le même bloc)
    - a-t-elle été remplacée entre temps ?
- **Différentes méthodes d'analyse**
  - simulation statique
  - interprétation abstraite
  - modélisation ILP

Analyse du cache d'instructions par simulation statique

- **Deux étapes :**
  - calcul de l'état abstrait du cache en entrée et en sortie de chaque bloc de base
    - à partir du CFG
    - état abstrait du cache = ensemble de toutes les lignes qui peuvent être dans le cache à un certain point du code
  - catégorisation de chaque instruction
    - est-elle dans le cache ?
      - toujours, peut-être, jamais, …

Analyse du cache d'instructions par simulation statique

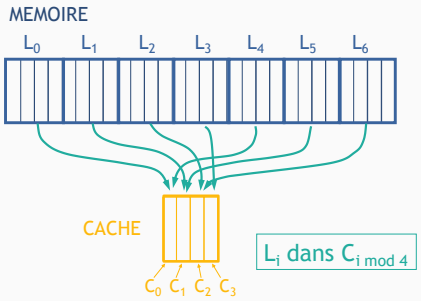
- **Analyse de l'état abstrait du cache**

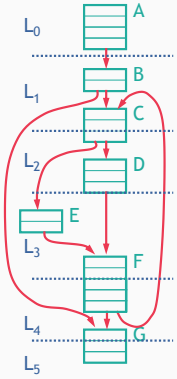
```
état_entrée(racine) = Ø ;

tantque les états en sortie évoluent faire
  pour chaque bloc B faire
    état_entrée(B) = Ø ;
    pour chaque prédecesseur P de B faire
      état_entrée(B) += état_sortie(P);
    finpour
    état_sortie(B) = état_entrée(B)
                   + lignes_lues(B)
                   - lignes_remplacées(B);
  finpour
fintantque
```


Analyse du cache d'instructions par simulation statique

- **Exemple**

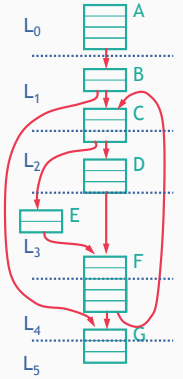




Analyse du cache d'instructions par simulation statique

- **Première passe**

	Etat en entrée				Etat en sortie				
	C_0	C_1	C_2	C_3	C_0	C_1	C_2	C_3	
A					L_0				☑
B	L_0				L_0	L_1			☑
C	L_0	L_1			L_0	L_1	L_2		☑
D	L_0	L_1	L_2		L_0	L_1	L_2		☑
E	L_0	L_1	L_2		L_0	L_1	L_2	L_3	☑
F	L_0	L_1	L_2	$\varnothing/L_3$	L_4	L_1	L_2	L_3	☑
G	L_0/L_4	L_1	$\varnothing/L_2$	$\varnothing/L_3$	L_4	L_5	$\varnothing/L_2$	$\varnothing/L_3$	☑

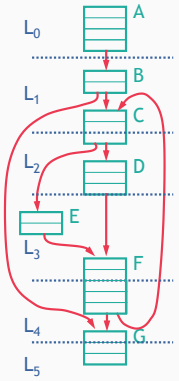


Analyse du cache d'instructions par simulation statique

- **Seconde passe**

	Etat en entrée				Etat en sortie			
	C_0	C_1	C_2	C_3	C_0	C_1	C_2	C_3
A					L_0			
B	L_0				L_0	L_1		
C	L_0/L_4	L_1	$\varnothing/L_2$	$\varnothing/L_3$	L_0/L_4	L_1	L_2	$\varnothing/L_3$
D	L_0/L_4	L_1	L_2	$\varnothing/L_3$	L_0/L_4	L_1	L_2	$\varnothing/L_3$
E	L_0/L_4	L_1	L_2	$\varnothing/L_3$	L_0/L_4	L_1	L_2	L_3
F	L_0/L_4	L_1	L_2	$\varnothing/L_3$	L_4	L_1	L_2	L_3
G	L_0/L_4	L_1	$\varnothing/L_2$	$\varnothing/L_3$	L_4	L_5	$\varnothing/L_2$	$\varnothing/L_3$

☑ ☑ ☑

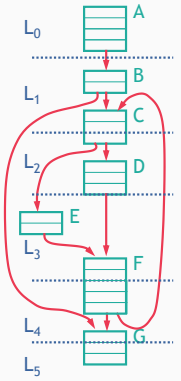


Analyse du cache d'instructions par simulation statique

- **Troisième passe**

	Etat en entrée				Etat en sortie			
	C_0	C_1	C_2	C_3	C_0	C_1	C_2	C_3
A					L_0			
B	L_0				L_0	L_1		
C	L_0/L_4	L_1	$\varnothing/L_2$	$\varnothing/L_3$	L_0/L_4	L_1	L_2	$\varnothing/L_3$
D	L_0/L_4	L_1	L_2	$\varnothing/L_3$	L_0/L_4	L_1	L_2	$\varnothing/L_3$
E	L_0/L_4	L_1	L_2	$\varnothing/L_3$	L_0/L_4	L_1	L_2	L_3
F	L_0/L_4	L_1	L_2	$\varnothing/L_3$	L_4	L_1	L_2	L_3
G	L_0/L_4	L_1	$\varnothing/L_2$	$\varnothing/L_3$	L_4	L_5	$\varnothing/L_2$	$\varnothing/L_3$

pas de changement



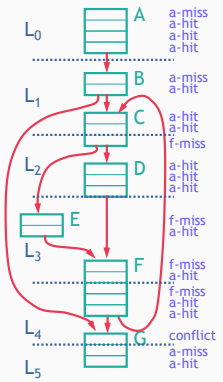
Analyse du cache d'instructions par simulation statique

always-miss	• 1ère instruction dans la ligne de cache • la ligne de cache n'est pas dans l'état abstrait en entrée
always-hit	• pas la 1ère instruction dans la ligne • 1ère instruction dans la ligne ET ligne seule dans l'état abstrait du cache en entrée
first-miss	• 1ère instruction dans la ligne de cache • la ligne est dans l'état abstrait du cache en entrée • il y a au moins une autre ligne dans l'état abstrait du cache en entrée • les autres lignes dans l'état abstrait du cache en entrée ne sont pas dans l'état abstrait du cache en sortie de la boucle • toutes les autres instructions de la ligne sont always-hit
conflict	• tous les autres cas

Analyse du cache d'instructions par simulation statique

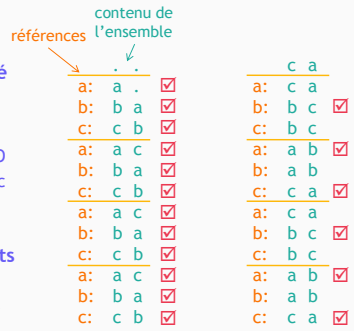
- **Catégorisation des instructions**

	Etat en entrée				Etat en sortie			
	C_0	C_1	C_2	C_3	C_0	C_1	C_2	C_3
A					L_0			
B	L_0				L_0	L_1		
C	L_0/L_4	L_1	$\varnothing/L_2$	$\varnothing/L_3$	L_0/L_4	L_1	L_2	$\varnothing/L_3$
D	L_0/L_4	L_1	L_2	$\varnothing/L_3$	L_0/L_4	L_1	L_2	$\varnothing/L_3$
E	L_0/L_4	L_1	L_2	$\varnothing/L_3$	L_0/L_4	L_1	L_2	L_3
F	L_0/L_4	L_1	L_2	$\varnothing/L_3$	L_4	L_1	L_2	L_3
G	L_0/L_4	L_1	$\varnothing/L_2$	$\varnothing/L_3$	L_4	L_5	$\varnothing/L_2$	$\varnothing/L_3$



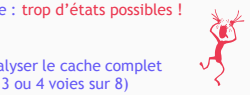
Analyse de caches associatifs par ensembles

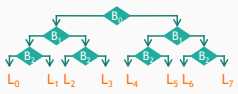
- **Effets dominos**
  - source de variabilité
  - exemple :
    - cache 2 voies
    - remplacement FIFO
    - 3 références a, b, c dans le même ensemble
  - le nombre de défauts varie :
    - selon les itérations
    - selon l'état initial



Analyse de caches associatifs par ensembles

- **Autres politiques de remplacement**
  - pseudo-round-robin
    - à chaque remplacement de ligne, un compteur est incrémenté (modulo la taille de l'ensemble) et désigne la prochaine ligne à remplacer
    - problème : il n'y a souvent qu'un seul compteur commun à tous les ensembles !
    - analyse du cache : trop d'états possibles !
  - pseudo LRU
    - impossible d'analyser le cache complet (ex : seulement 3 ou 4 voies sur 8) (Heckmann *et al.* 2003)
    - effets dominos (Berg 2006)





mise à jour

L_0:	$B_0 \leftarrow 1$;	$B_1 \leftarrow 1$;	$B_3 \leftarrow 1$;
L_1:	$B_0 \leftarrow 1$;	$B_1 \leftarrow 1$;	$B_3 \leftarrow 0$;
L_2:	$B_0 \leftarrow 1$;	$B_1 \leftarrow 0$;	$B_4 \leftarrow 1$;
L_3:	$B_0 \leftarrow 1$;	$B_1 \leftarrow 0$;	$B_4 \leftarrow 0$;
L_4:	$B_0 \leftarrow 1$;	$B_2 \leftarrow 1$;	$B_5 \leftarrow 1$;
L_5:	$B_0 \leftarrow 0$;	$B_2 \leftarrow 1$;	$B_5 \leftarrow 0$;
L_6:	$B_0 \leftarrow 0$;	$B_2 \leftarrow 0$;	$B_6 \leftarrow 1$;
L_7:	$B_0 \leftarrow 0$;	$B_2 \leftarrow 0$;	$B_6 \leftarrow 0$;

Prise en compte des résultats de l'analyse du cache

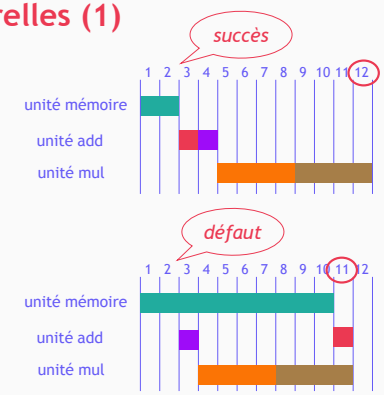
- **Mesure du temps d'exécution du bloc de base en prenant en compte le temps d'accès au cache (succès) ou à la hiérarchie mémoire (défaut)**
  - et quand on ne sait pas ? attention aux anomalies temporelles !
    le défaut n'est pas forcément le pire cas !
- **Dépliage du graphe de contrôle de flot lorsque le résultat n'est pas le même pour la première itération que pour les suivantes :**
  - un champ d'exécution pour la première itération
  - un champ pour les autres

Modélisation du cache de données

- **Différence avec le cache d'instructions**
  - les adresses de données sont souvent dynamiques
    - exemple : LDR r3,[r4] / LDR r8,[r9] — même adresse ?
- **Solutions**
  - Analyse statique préliminaire pour identifier les adresses (alias)
  - Simulation symbolique

Anomalies temporelles (1)

```
ldr r4,[r3]
add r5,r4,r4
add r11,r10,r10
mul r11,r8,r11
mul r19,r1,r2
```



Anomalies temporelles (2)

Soient 2 instructions I_1 et I_2 exécutées en séquence

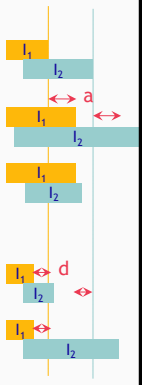
- 1er cas :
  - on suppose que la durée de I_1 augmente de *a* cycles
  - si le temps d'exécution de la séquence I_1-I_2 :
    - augmente de plus de *a* cycles

    OU
    - diminue

    anomalie temporelle
- 2nd cas :
  - on suppose que la durée de I_1 diminue de *d* cycles
  - si le temps d'exécution de la séquence I_1-I_2 :
    - diminue de plus de *d* cycles

    OU
    - augmente

    anomalie temporelle

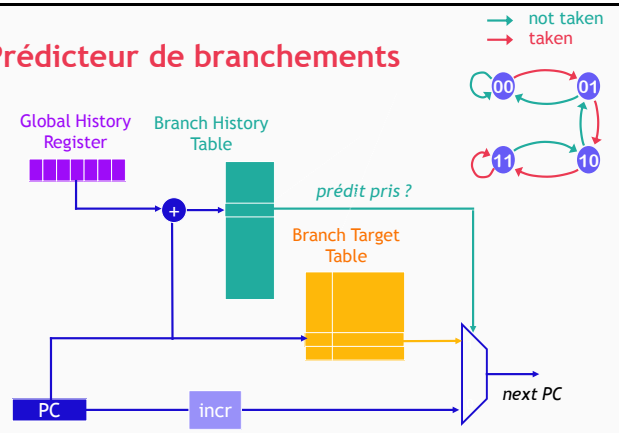


Gel de cache d'instructions

Une solution à la non prévisibilité des accès (latences)

- **Sélection du contenu :**
  - profiling
    - trace des instructions exécutées pour un ensemble de jeux d'entrées
    - sélection des instructions les plus fréquentes
  - algorithme génétique
    - individu = vecteur de *nl* bits (1 si la ligne est dans le cache)
    - sélection des individus *valides* correspondant aux WCETs les plus courts
    - croisement
    - mutation (réduction ou augmentation aléatoire du nombre de lignes verrouillées dans le cache, changement aléatoire d'une ligne)

Prédicteur de branchements

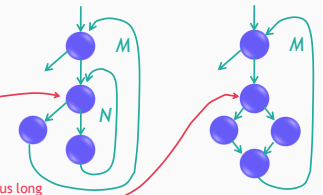


Analyse du prédicteur de branchement (1)

- **Objectif :**
  - borner le nombre de mauvaises prédictions (maximum et minimum)
- **Deux aspects :**
  - algorithme de prédiction (par ex., compteur 2-bits) :
    - facile à analyser pour certains branchements (boucles)
    - plus complexe pour d'autres (ex : conditionnelle dépendant des données en entrée)
  - interférences entre branchements dans la table de prédiction :
    - prédicteur local ⇨ complexité similaire à l'analyse de cache
    - prédicteur global ⇨ plus compliqué

Analyse du prédicteur de branchement (2)

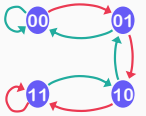
- **Première approche**
  - analyse algorithmique
    - boucle répétée



nb erreurs ≤ M+2

    - conditionnelle répétée

nb erreurs ≤ { 2 si un chemin est plus long ; M si les chemins sont de longueur équivalente



  - analyse de l'aliasing
    - mêmes techniques que pour les caches

Analyse du prédicteur de branchement

- **Deuxième approche**
  - modélisation ILP
    - ajout de contraintes pour modéliser l'évolution du compteur

$$x_i^{00} = x_{0\,i\to}^{00} + x_{0\,i\to}^{01} + x_{\Rightarrow i}^{00}$$
$$x_i^{01} = x_{0\,i\to}^{01} + x_{1\,i\to}^{00} + x_{\Rightarrow i}^{01}$$
$$x_i^{10} = x_{0\,i\to}^{11} + x_{1\,i\to}^{01} + x_{\Rightarrow i}^{10}$$
$$x_i^{11} = x_{1\,i\to}^{11} + x_{1\,i\to}^{10} + x_{\Rightarrow i}^{11}$$

$$\forall c,\quad x_i^c = x_{0\,i\to}^c + x_{1\,i\to}^c + x_{i\Rightarrow}^c \qquad x_i = \sum_c x_i^c$$

$$\forall d,\quad x_{i\to}^d = \sum_c x_{i\to}^{c\,d} + \sum_c x_{i\Rightarrow}^{c\,d}$$

$$\sum_c x_{\Rightarrow i}^c = 1 \qquad \sum_c x_{i\Rightarrow}^c = 1$$

    - ajout de contraintes pour modéliser l'aliasing : complexe !

Exécution spéculative

- **Peut affecter le contenu des caches**
  - chargement sur le mauvais chemin
  - effets positifs ou négatifs

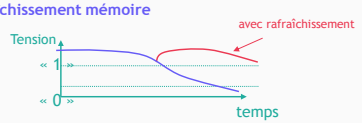
Pas vraiment de solution à ce jour …

Plan

- Système temps-réel et temps d'exécution pire cas
- Comment évaluer le WCET d'une tâche ?
  - Mesures
  - Analyse statique
- Modélisation de l'architecture cible
  - pipeline
  - caches
  - prédicteur de branchement
- Prise en compte de l'environnement
  - événements asynchrones
  - processeur multithread
  - processeur multicoeur
- Conclusion

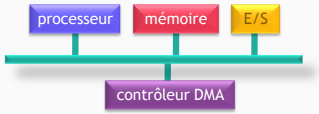
Evénements asynchrones (1)

- Ils influent sur la latence des accès mémoire
  - rafraîchissement mémoire



  - détection et correction d'erreurs (*SEU = Single Event Upset*)

  - transferts DMA → conflits bus



Evénements asynchrones (2)

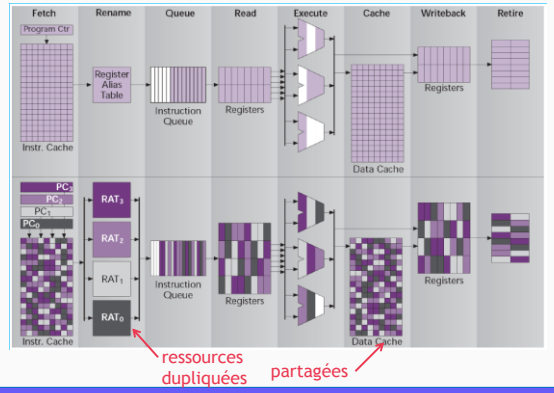
- Prise en compte dans le calcul du WCET
  - période ?
  - statistiques d'occurrence ?

remet en cause la précision de l'analyse du processeur …

Système multi-tâches

- Impact des autres tâches et de l'OS ?
  - sur le contenu des mémoires caches
    - analysable ?
      - sans doute si ordonnancement statique
      - difficilement dans le cas général (système pré-emptif)
    - solution : partitionnement de cache
      - une partition par tâche
      - les tailles doivent être soigneusement fixées
      - données partagées ?

Processeur multiflot simultané :SMT (1)

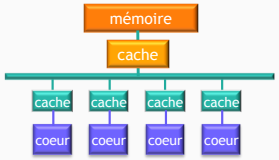


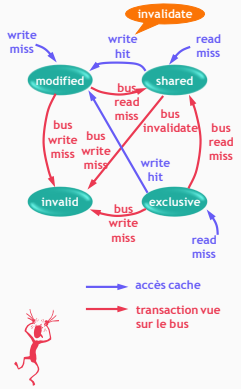
Processeur multiflot simultané (2)

- **Impact des autres threads**
  - ressources partagées
    - caches
    - prédicteur de branchement
    - files d'instructions
    - unités fonctionnelles
  - ordonnancement
    - quelle politique de sélection ?

Pas vraiment de solution à ce jour …

Processeurs multicoeurs





- **Impact des autres tâches**
  - accès au bus (risque de conflit)
  - contenu du cache de niveau 2
  - cohérence des caches de niveau 1

Pas vraiment de solution à ce jour …

Plan

- Système temps-réel et temps d'exécution pire cas
- Comment évaluer le WCET d'une tâche ?
  - Mesures
  - Analyse statique
- Modélisation de l'architecture cible
  - pipeline
  - caches
  - prédicteur de branchement
- Prise en compte de l'environnement
  - événements asynchrones
  - processeur multithread
  - processeur multicoeur
- Conclusion

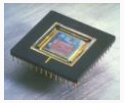
Ce qu'on sait (à peu près bien) analyser …

- **Des tâches « isolées »**
  - non interrompues, seules à s'exécuter sur le matériel
- **Des architectures moyennement complexes :**
  - pipelines superscalaires, exécution non ordonnée
  - caches pas trop compliqués
  - prédicteur de branchement si pas d'exécution spéculative

Ce qu'on ne sait pas (bien) analyser …

- **Les architectures complexes**
  - caches avec politique de remplacement non prévisible
  - prédiction de branchements et surtout exécution spéculative
  - architectures qui introduisent du parallélisme entre tâches
    - multiflot simultané
    - multicoeurs
- **Les interactions avec l'environnement**
  - événements asynchrones
    - rafraîchissement mémoire, correction d'erreur
    - conflits avec acteurs externes (DMA, …)
  - préemption, interruptions, …

Portrait du processeur idéal



- **Un pipeline pas trop compliqué**
  - éviter un ordonnancement dynamique si les latences ne sont pas maîtrisées
- **Des caches analysables**
  - politique de remplacement prévisible
    - application directe ou remplacement LRU
  - caches verrouillables
- **Pas de prédiction de branchement**
  - ou débrayable
- **Pas de parallélisme de tâches**

Outils existants pour le calcul de WCET

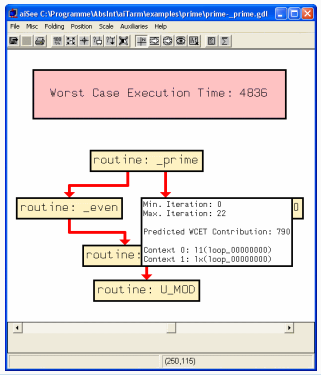
Outils académiques

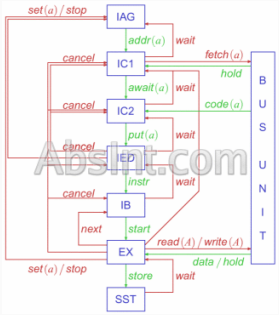
Heptane	Chronos	SWEET	OTAWA
• IRISA (Rennes) • méthode ETS (arbres syntaxiques)	• SNU (Singapour) • graphes d'exécution + IPET	• Mälardalen (Suède) • simulation + IPET	• IRIT (Toulouse) • environnement accueillant différentes méthodes

Outils commerciaux

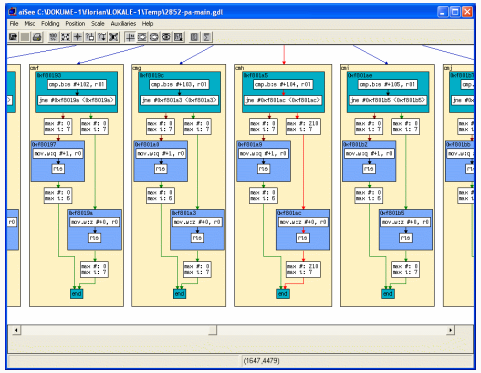
aiT	BOUND-T
• AbsInt (Allemagne) • interprétation asbtraite + IPET	• Tidorum (Finlande) • analyse de flot

Outil aiT de AbsInt (1)

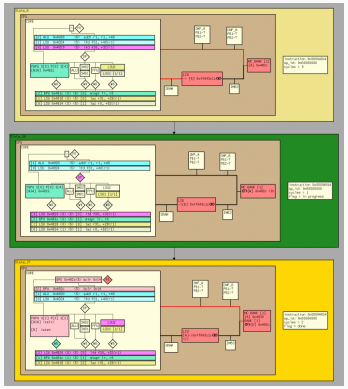




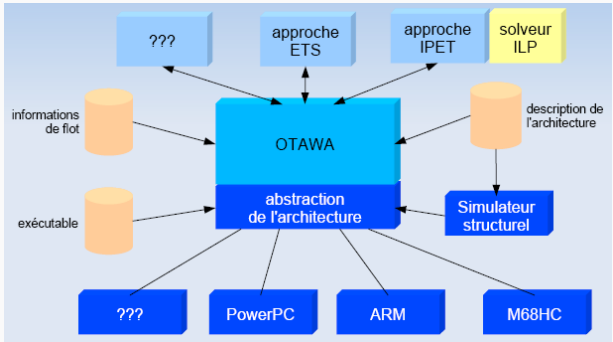
Outil aiT de AbsInt (2)



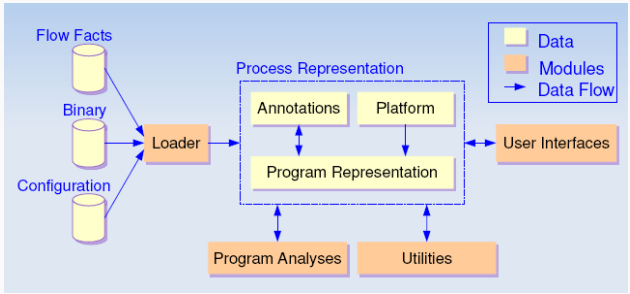
Outil aiT de AbsInt (3)



OTAWA (1)



OTAWA (2)


Flow Facts
Binary
Configuration
Loader
Process Representation
Annotations
Platform
Program Representation
Program Analyses
Utilities
User Interfaces
Data
Modules
Data Flow