Obtention de pires temps d'exécution (WCET, worst-case execution times)

Isabelle PUAUT

Professeur, Université de Rennes I - IRISA
Rennes, FRANCE

Septembre 2005

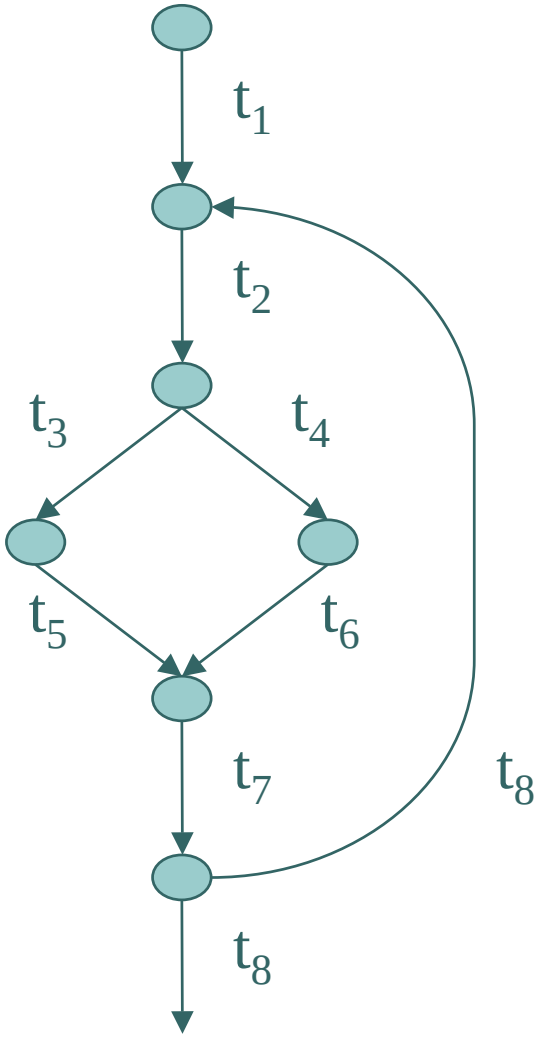
Plan

- Le WCET : définitions
- Classes de méthodes
  - Méthodes dynamiques
  - Méthode statiques
  - Comparaison
- Méthodes statiques d'obtention des WCET
  - Analyse de flot
  - Calcul des WCET
  - Analyse de bas niveau
- Points ouverts et recherches actuelles

WCET
Définition

- Borne supérieure pour les temps d'exécution de portions de code
  - Temps pris par le processeur pour l'exécution
  - Code considéré de manière isolée
  - Attention : WCET ≠ temps de réponse
- Défis pour l'obtention des WCET
  - Sûreté (WCET > tout temps d'exécution possible) : confiance tests de faisabilité
  - Précision
    - Surestimation ⇒ échec potentiel des tests de faisabilité, surdimensionnement des ressources matérielles nécessaires

Eléments influençant le WCET

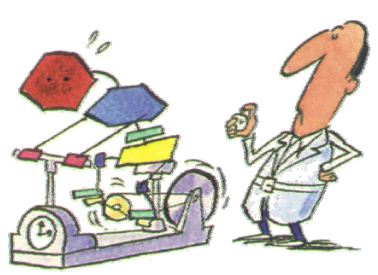


- Mises en séquence possibles des actions (chemins d'exécution)
  - Dépendent des données d'entrée
- Durée de chaque occurrence d'une action
  - Dépendant de l'architecture cible

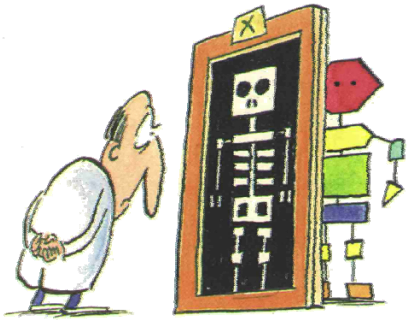
Méthodes d’estimation des WCET

Méthodes dynamiques

- Principe
  - Jeu de données d'entrée
  - Exécution (matériel ou simulateur)
  - Mesure
- Génération des jeux de test explicites
  - Définis par l'utilisateur : besoin d’expertise
  - Exhaustifs
    - Uniquement pour les entrées à domaines finis
    - Problème de combinatoire
  - Générés automatiquement (génétique,etc.)
  - Sûreté ?



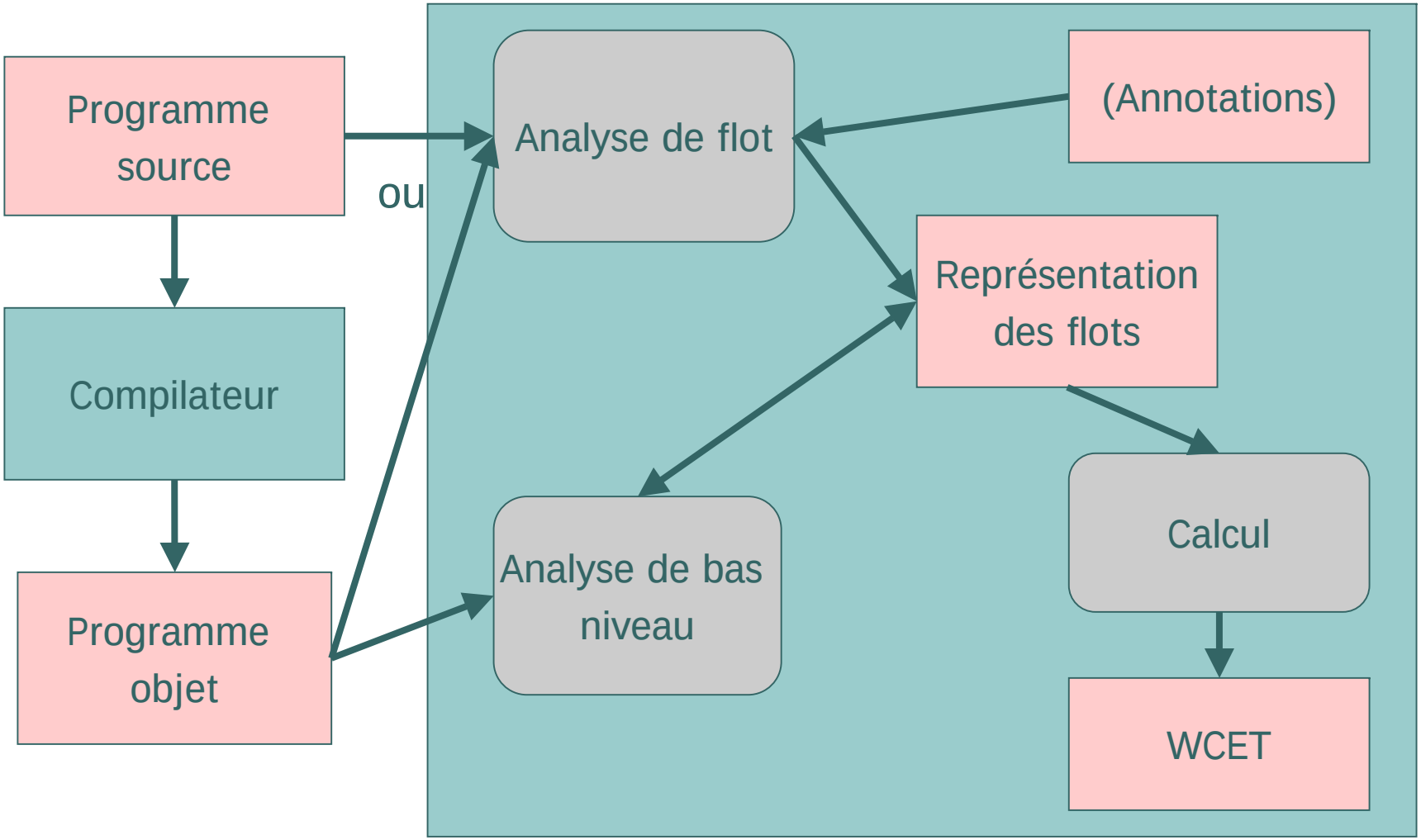
Méthodes statiques



- Principe
  - Analyser de la structure du programme (pas d’exécution)
  - Calculer la WCET à partir de la structure
- Composants de l'analyse
  - Analyse de flot
    - Détermine les chemins d’exécutions possibles
  - Analyse de bas niveau
    - Détermine le temps d’exécution d’une séquence d’instructions sur une architecture donnée
  - Calcul
    - Calcul du WCET à partir des composants précédents
    - Tous les chemins sont considérés : sûreté

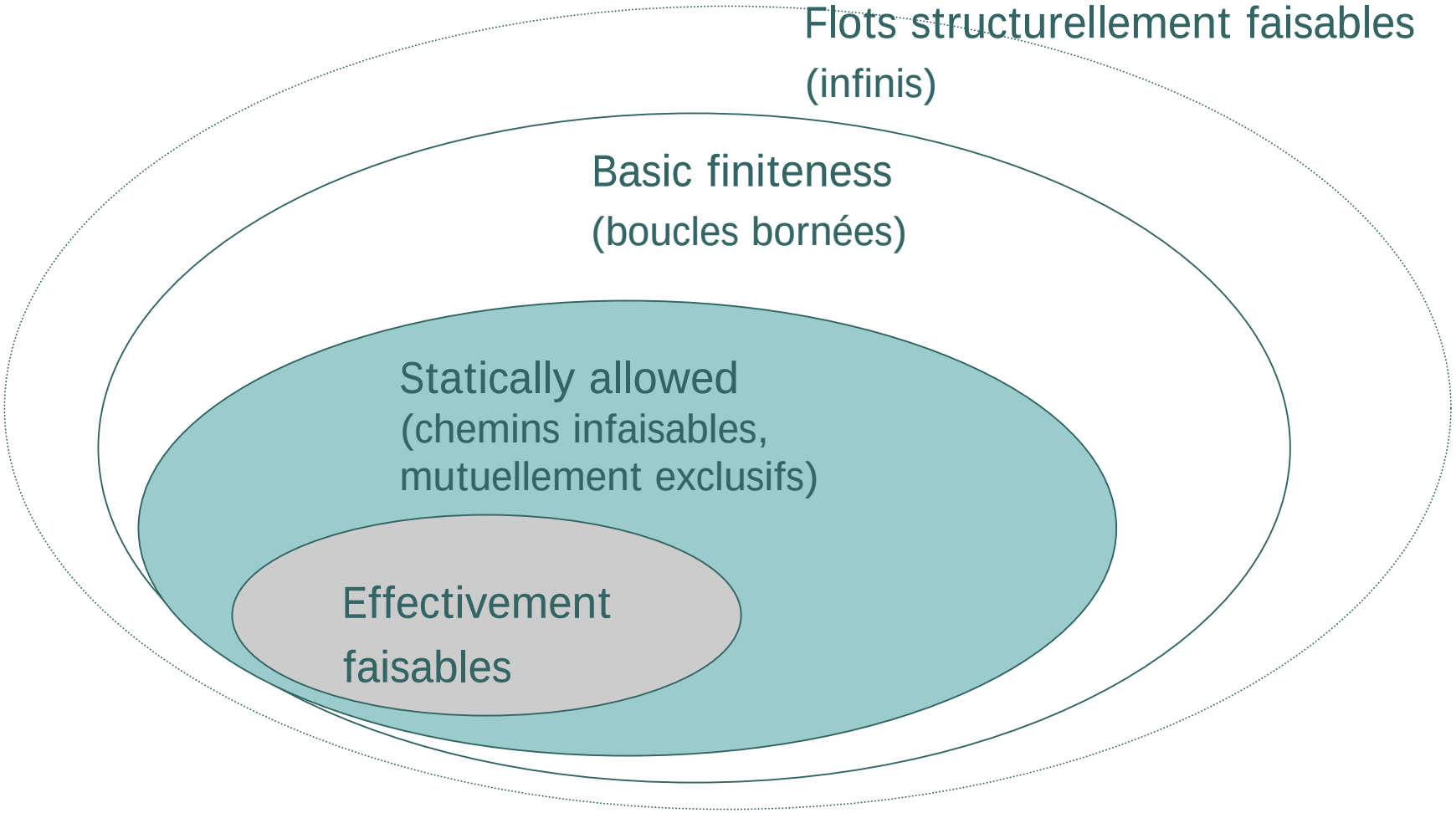
Méthodes statiques

Vue synthétique des composants



Méthodes statiques

Analyse de flot (1/4)



Méthodes statiques

Analyse de flot (2/4)

- Chemins infaisables

```
int baz (int x) {
    if (x<5)          // A
        x = x+1;      // B
    else x=x*2;       // C
    if (x>0)          // D
        x = sqrt(x); // E
    return x;         // F
}
```

- Le chemin ABDEF est infaisable
- Connaissance des chemins infaisables améliore la précision des estimations de WCET

Méthodes statiques

Analyse de flot (3/4)

- Nombres maximum d'itérations des boucles

```
for i := 1 to N do
    for j := 1 to i do
    begin
        if c1 then A.long
              else B.short
        if c2 then C.short
              else D.long
    end
```

← Borne de boucle: N

← Borne de boucle: N

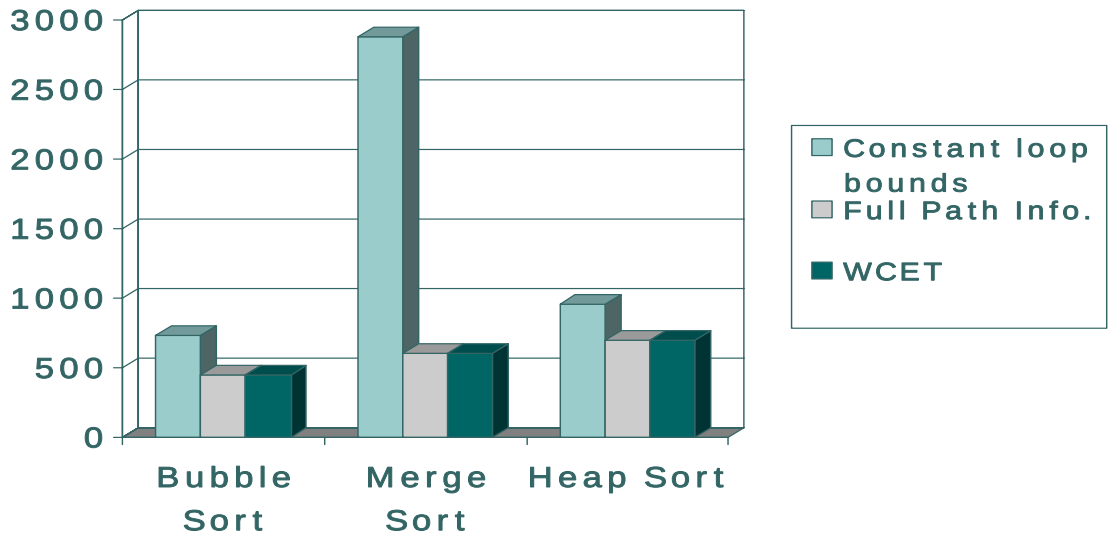
$\frac{(N+1)N}{2}$ executions

- Une estimation précise des bornes améliore l'estimation du WCET

Méthodes statiques

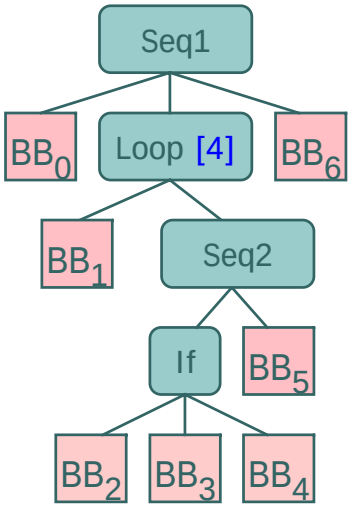
Analyse de flot (4/4)

- Modes de détermination des flots
  - Automatique : infaisable en général (halting problem)
  - Manuelle : annotations
    - Constantes, annotations symboliques
    - Annotations pour chemins infaisables / exclusifs
- Résultats de l'analyse de flot (P. Puschner)



Méthodes statiques : calcul

Analyse à base d'arbres (tree-based)

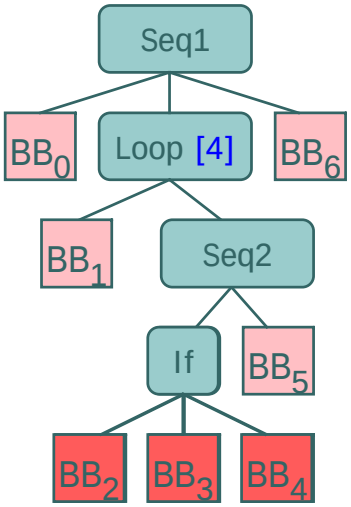


- Structures de données utilisées
  - Arbre syntaxique du programme
  - Blocs de base
- Principe de la méthode
  - Détermination des temps d'exécution des blocs de base (analyse de bas-niveau)
  - Calcul récursif sur les structures syntaxiques (timing schema)

Méthodes statiques : calcul

Analyse à base d'arbres

WCET(SEQ)	S1;...;Sn	WCET(S1) + ... + WCET(Sn)
WCET(IF)	if(test) then else	WCET(test) + max(WCET(then) , WCET(else))
WCET(LOOP)	for(;tst;inc) {body}	maxiter * (WCET(tst)+WCET(body+inc)) + WCET(tst)



Système d'équations

WCET(Seq1) = WCET_BB0 + **WCET(Loop)** +WCET_BB_6
WCET(Loop) = 4 * (WCET_BB1 + **WCET(Seq2)**) + WCET_BB1
WCET(Seq2) = **WCET(If)** + WCET_BB5
WCET(If) = WCET_BB2 + **max**(WCET_BB3 , WCET_BB4)

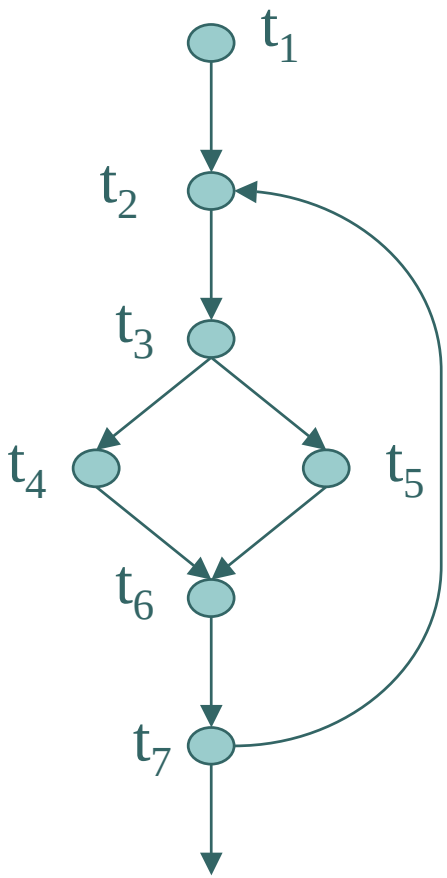
Méthodes statiques : calcul

Analyse à base d'arbres

- Attrait des méthodes tree-based
  - Lien avec le code source aisé
  - Complexité des calculs maîtrisée
  - Mais, ...
- Limitations
  - Ne supporte pas toutes les optimisations de compilation
  - Prise en compte d'informations de flot élaborées n'est pas aisée (si non conforme à l'arbre syntaxique)

Méthodes statiques : calcul

Analyse IPET (Implicit Path Enumeration Technique)



- Programmation linéaire en nombres entiers
  - But : max: $f_1t_1+f_2t_2+\ldots+f_nt_n$
  - Contraintes structurelles

    $\forall\ v$: $f_i = \Sigma\ a_i = \Sigma\ a_i$

    $f_1 = f_7 = 1$ $\quad a_i \in In(v)\ a_i \in Out(v)$
  - Contraintes sur les chemins

    $f_i \leq k$ (maxiter boucle)

    $f_i + f_j \leq 1$ (chemins mutuellement exclusifs)

Programmation par contraintes

- Expression contraintes plus complexes possible

Méthodes statiques : calcul

Analyse IPET

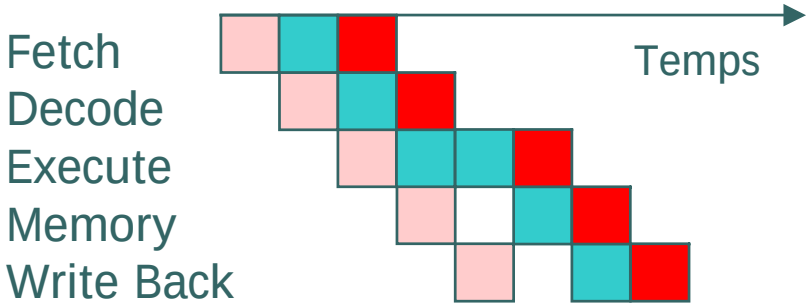
- Attrait des méthodes IPET
  - Travaille sur une structure de bas niveau, donc supporte les flots non structurés (goto, jumps, ...)
  - Supporte toutes les optimisations de compilation
  - Mais, ...
- Limitations
  - Complexité des calculs non maîtrisée
  - Liens avec le code source non évidents (maîtrise des optimisations de compilation)
  - Pas de retour explicite du pire chemin d'exécution

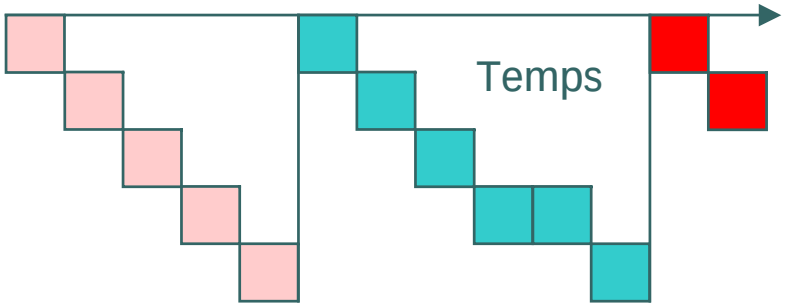
Introduction

- Architecture simple
  - Temps d'exécution d'une instruction dépend uniquement de son type et de ses opérandes
  - Pas de recouvrement entre instructions, pas de hiérarchie de mémoire
- Architecture complexe
  - Effets locaux
    - Recouvrement entre instructions (pipelines)
  - Effets globaux
    - Caches de données, d'instructions, prédicteurs de branchement
    - Demande une connaissance de l'ensemble du code

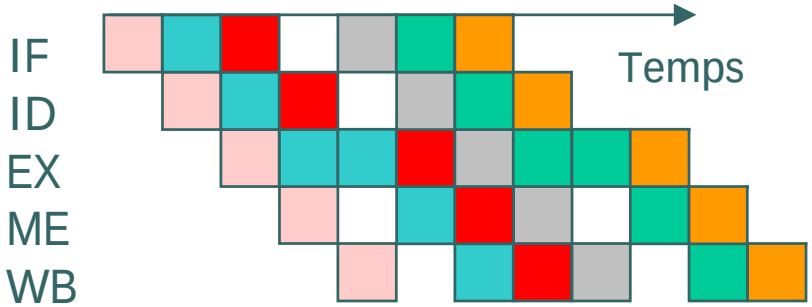
Prise en compte des pipelines

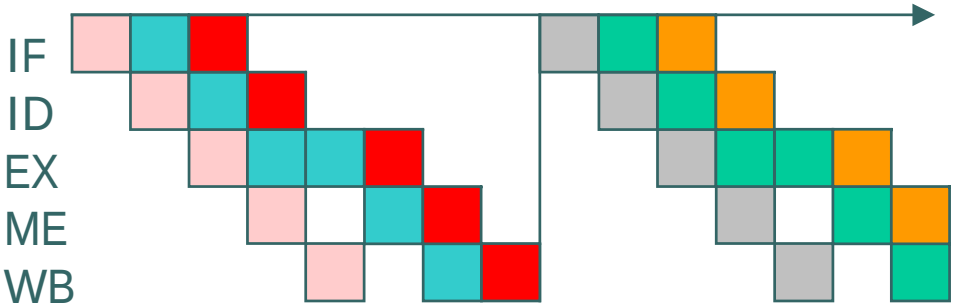
- Principe : parallélisme entre instructions
  - Intra bloc de base





  - Inter bloc de base





Prise en compte des pipelines

- Solution intra-BB
  - Tables de réservation décrivant quand chaque instruction accède les étages du pipeline
  - Obtenu par l'analyseur lui-même ou outil externe (simulateur, processeur cible)
- Solution inter-BB : modification de la méthode de calcul
  - Tree-based: opérateur d'addition spécifique
  - IPET: contraintes supplémentaires dans le problème d'ILP

Caches d'instructions

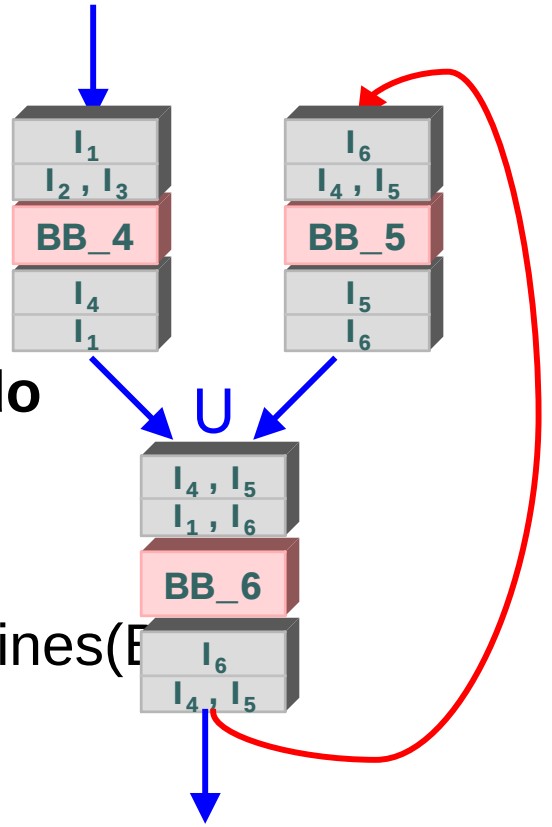
- Cache
  - Tire profit de la localité spatiale et temporelle des accès aux instructions
  - Spéculatif : comportement dépend du comportement passé des programmes
  - Bon comportement en moyenne, mais problème de déterminisme en contexte temps-réel strict
- Problème : estimation sûre du comportement des caches
  - Solution simple (tout miss) excessivement pessimiste
  - Objectif : prédire si une instruction causera un hit ou pourra causer un miss (de manière conservatrice)

Caches d'instructions

- Simulation statique de cache
- Calcul d'états abstraits de caches (ACS)
  - État du cache représentant tous les chemins d'exécution possibles
  - Itération de point fixe pour le calcul
- Division des instructions en catégories selon les ACS
  - *always hit*
  - *always miss*
  - *first miss*, *first hit* (cas d'instructions dans des boucles)

Caches d'instructions

input_state(top) = all invalid lines
while any change **do**
    for each basic block instance B **do**
        input_state(B) = null
        for each immediate predecessor P of B **do**
            input_state(B) += output_state(P)
        end for
        output_state(B) = (input_state(B)+ prog_lines(B
            conf_lines(B)
    end for
end while



Amélioration : prise en compte niveaux de boucles

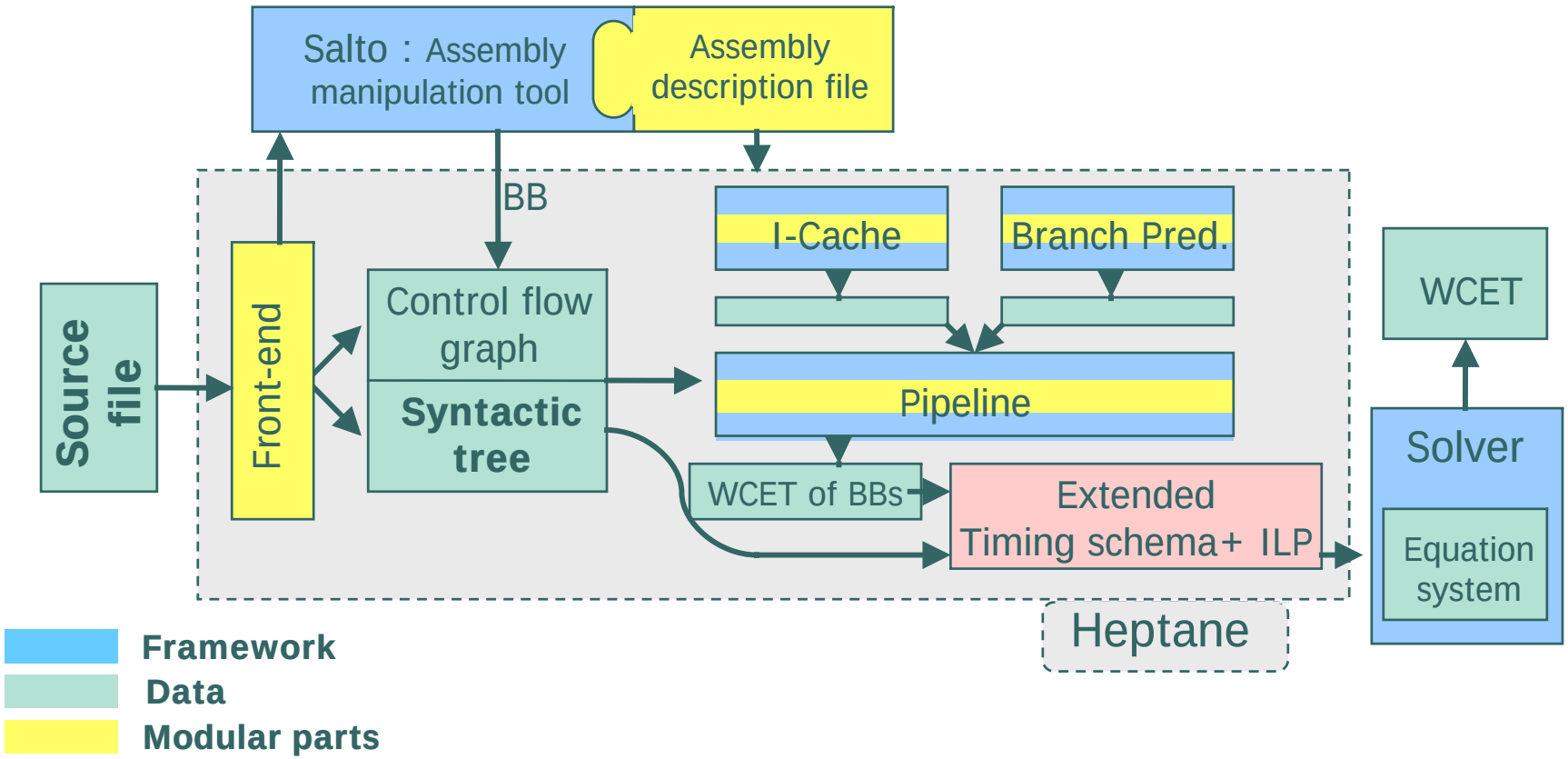
Caches de données

- Problème supplémentaire : obtention de l'adresse des données (dynamique)
- Solution 1
  - Analyse statique pour l'obtention des adresses
  - Pas de mise en cache des données d'adresses inconnues
- Solution 2 : exécution symbolique [Chalmers]
  - Extension d'un simulateur de processeurs à des données inconnues
  - Exemple : cas d'un branchement
    - Valeur testée connue : on connaît le flot d'exécution
    - Valeur testée inconnue : on explore les deux branches
  - Fusion de chemins pour éviter de dérouler les boucles
  - Intérêt : prise en compte de toutes les caractéristiques de l'architecture. Limite : temps de simulation

Autres éléments maîtrisés

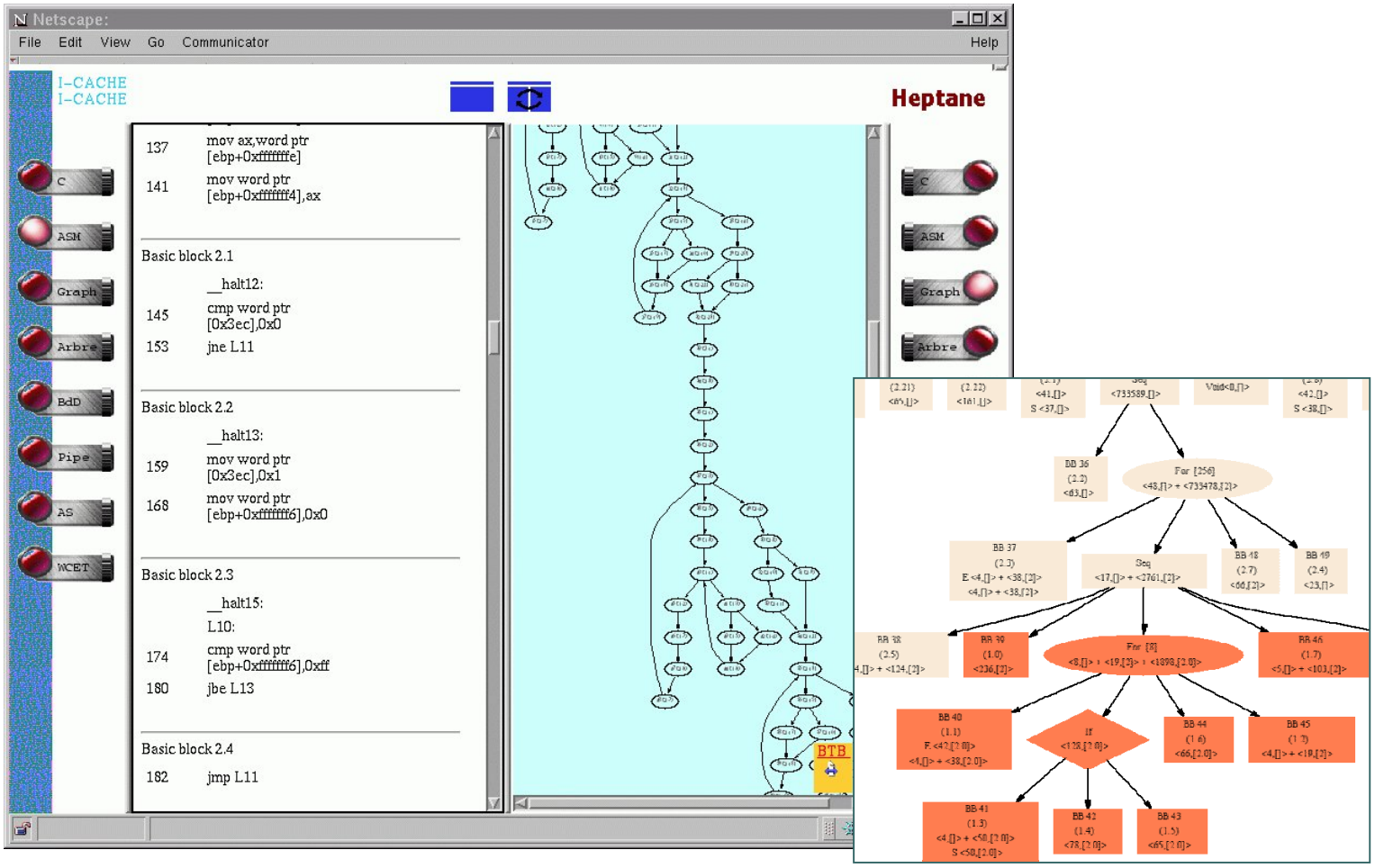
- Prédicteurs de branchement (dynamiques)
  - Locaux (basé sur historique dernières prédictions du branchement) [IRISA]
    - Méthode issue de la simulation statique de cache
    - Passage du contenu du BTB (Branch Target Buffer) à une classification
  - Globaux (basé sur historique du branchement + autres branchements exécutés récemment) [Singapour]
    - Mapping vers un problème d'ILP

Méthodes statiques
Heptane



Méthodes statiques

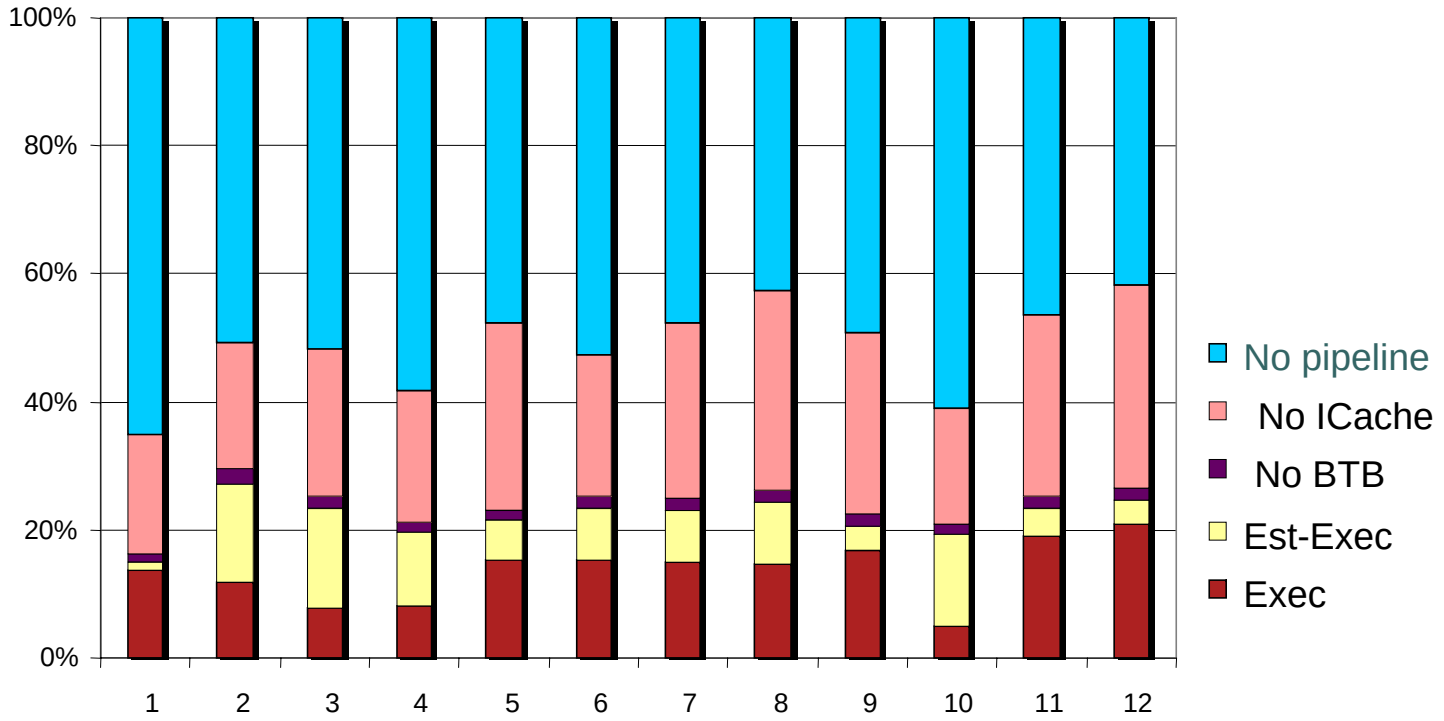
Heptane



Méthodes statiques

Heptane : résultats quantitatifs

- Impact de la modélisation d'architecture (analyse noyau RTEMS, Irisa)



Méthodes d’estimation des WCET

Quelle méthode pour quel usage ?

- Méthodes statiques
  - Sûreté ☺
  - Pessimisme ☹
  - Besoin de modèle du matériel ☹
  - Compromis précision-rapidité du calcul (tree-based / IPET) ☺
- Méthodes dynamiques
  - Sureté ? Probabiliste
  - Pessimisme ☺
  - Pas de modèle du matériel ☺
- Une méthode pour chaque usage

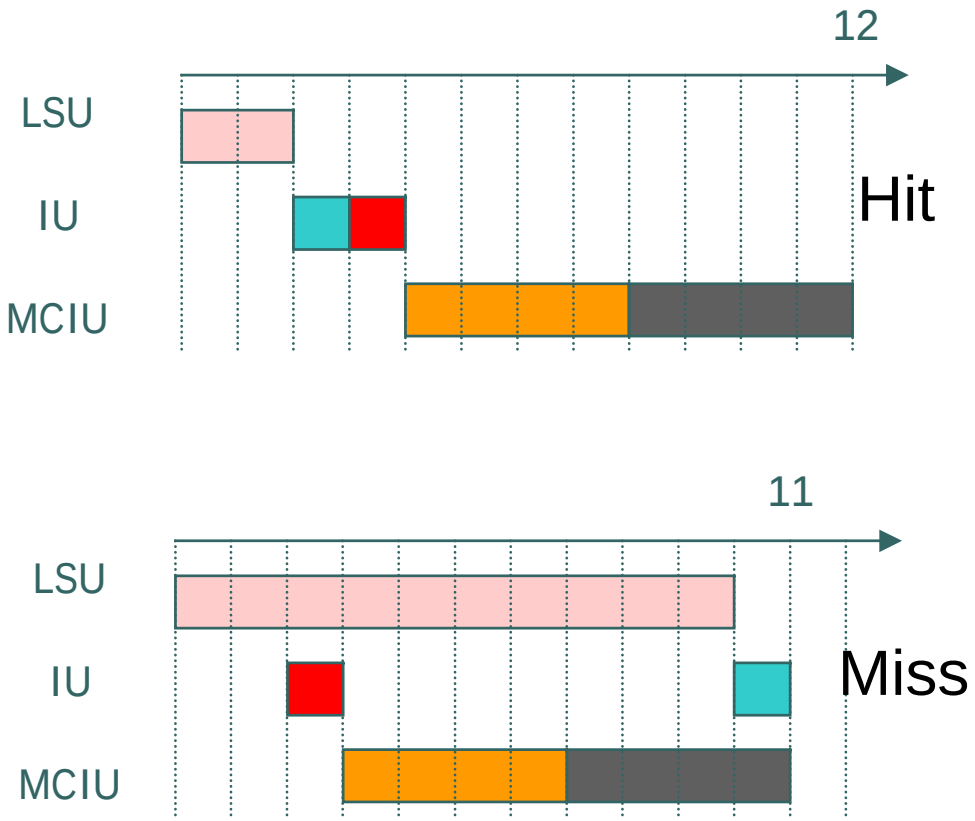
Points ouverts et recherches actuelles (1/4)

- Prise en compte du matériel
  - Constat : complexité croissante du matériel
    - Complexité des analyseurs
    - Timing anomalies
    - Cohabitation des différents éléments architecturaux, retard par rapport à l'arrivée du matériel

Points ouverts et recherches actuelles (2/4)

Timing anomalies

	Disp. Cycle	Instruction
A	1	LD r4,0(r3)
B	2	ADD r5, r4, r4
C	3	ADD r11, r10, r10
D	4	MUL r11, r11, r11
E	5	MUL r13, r12, r12



Points ouverts et recherches actuelles (3/4)

- Prise en compte du matériel : recherches
  - Conception de matériel prévisible [Vienna]
  - Modèles de matériel complexe : exécution spéculative [IRIT], prédicteurs de branchement [Séoul,York]
  - Utilisation prévisible de matériel complexe : gel de cache [IRISA,Mälardalen]
  - Prise en compte probabiliste du matériel [York]

Points ouverts et recherches actuelles (4/4)

- Analyse de flot et calcul
  - Détermination automatique des flots [Mälardalen]
  - Paradigmes de programmation pour le temps-réel facilitant le déterminisme (donc l'obtention des WCET) : single path paradigm [Vienne]
  - Méthodes dynamique [CEA, IRISA]
- Compilation pour le déterminisme [IRISA]

Quelques pointeurs
Outils d'obtention de WCET

- Universitaires
  - Cinderella [Princeton]
  - Heptane [IRISA]
  - Otawa [IRIT Toulouse]
  - SWEET [Sweden]

- Commerciaux
  - Bound-T [SSF]
  - AIT [AbsInt, Sarbrüken]
  - Rapitime [Rapita systems, York]

Quelques pointeurs

Références bibliographiques

- Journal of real-time systems, special issue on static worst-case execution-time analysis, 2 numéros parus en 1999 et 2000
- Worshop on static worst-case execution time analysis (4 éditions en 2001..2004), en conjonction avec la conf. Euromicro on real-time systems
- Calcul de majorants de pires temps d’exécution: état de l’art. A. Colin, I. Puaut, C. Rochange, P. Sainrat. TSI, issue spéciale temps-réel, 2003