Atelier thématique RIS

Nouvelles architectures & technologies de processeurs et sûreté de fonctionnement

Architecture des processeurs et vérification de contraintes de temps-réel strict

Isabelle PUAUT

INSA/IRISA Rennes, FRANCE

Novembre 2002

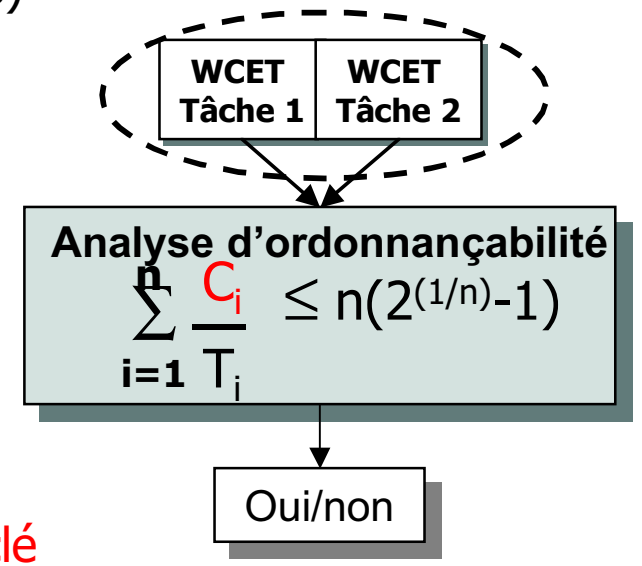
IRISA

Contexte

- Systèmes critiques
  - Fonctionnement correct en présence de fautes (d'origine physique, humaine)
  - Contraintes de temps-réel strict
    - Temps-réel : temps de réponse des logiciels est un critère de correction (échéance)
    - Temps-réel strict : conséquence graves en cas de faute temporelle (dépassement d'échéance) : les temps de réponse des applications doivent être garantis
  - Nécessité de validation du comportement temps-réel du système
- Systèmes à besoin en performances
  - Architectures matérielles élaborées (exécution pipelinée, caches, …)

Validation temporelle des systèmes temps-réel strict

- Comportement temporel du système dépend
  - De sa charge de travail (instants d'arrivée des tâches)
  - Du temps d'exécution de chacune des tâches
  - ➔ Test exhaustif du comportement du système impossible (combinatoire)
- Utilisation de modèles du système et méthodes analytiques
  - Charge de travail (exemple : tâches périodiques)
  - Pire temps d'exécution des tâches
  - Méthodes d'ordonnancement
  - Conditions de faisabilité du système
    - Exemple : analyse Rate Monotonic (RM)
      - Tâches périodiques
      - Ordonnancement à priorité fixe
  - Pire temps d'exécution (WCET - Worst-Case Execution Time) : entrée clé



Organisation de l'exposé

- Méthodes d'obtention des WCET : un bref tour d'horizon
- Architectures matérielles complexes : problèmes et solutions
  - Pipelines
  - Caches
  - Prédicteurs de branchement
- L'analyseur Heptane (IRISA)
  - Description
  - Quelques résultats
- Travaux en cours à l'IRISA
  - Alternatives à l'analyse de caches : gel statique de caches
- Architectures matérielles complexes : points ouverts

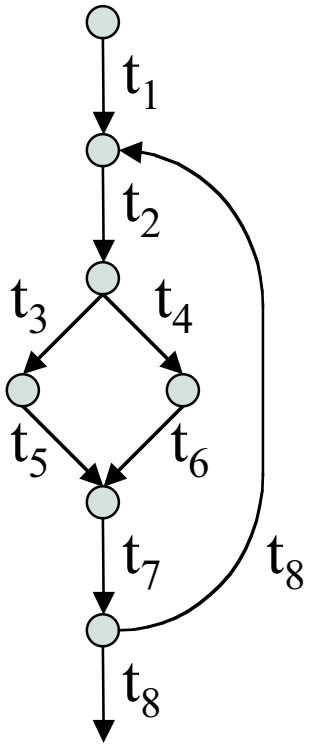
Obtention des WCET

- Calcule d'une borne supérieure pour les temps d'exécution de portions de code
  - temps pris par le processeur pour exécuter cette portion de code
  - code considéré de manière isolée

- Défis pour l'obtention des WCET
  - Les estimations de WCET doivent être sûres
    - Confiance dans les méthodes d'analyse
  - Les estimations de WCET doivent être précises
    - Surestimation ⇒ échec potentiel des conditions de faisabilité, surdimensionnement des ressources matérielles nécessaires

Eléments influençant le WCET

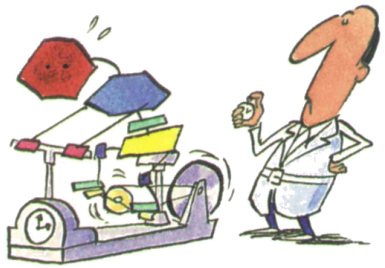
Programme



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

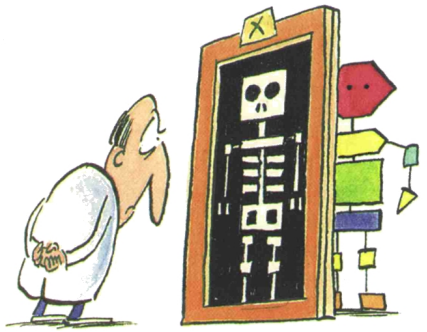
Méthodes d'estimation des WCET (1/2)

- Génération de jeux d'entrée et mesure



  - Exécution du programme avec des données d'entrée
  - Mesure du temps d'exécution
    - ➔ Comment exhiber le pire comportement du programme ?
    - ➔ Test exhaustif : impossible en pratique

- Analyse



  - Analyser le code source ou objet du programme (pas d'exécution)

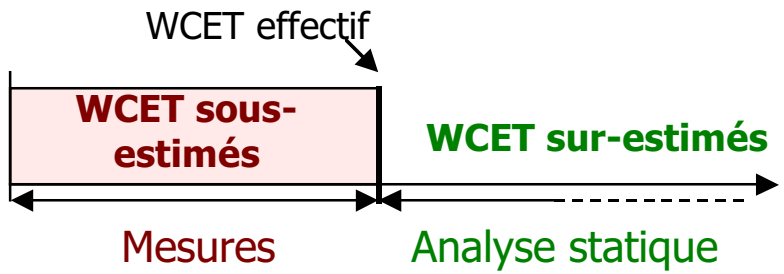
Méthodes d'estimation des WCET (2/2)

- Test et mesure
  - Temps mesuré ≤ WCET
  - WCETsous-estimé: l'analyse d'ordonnançabilité peut accepter des ensembles de tâches non faisables

➡ Non sûr

- Analyse statique
  - Temps estimé ≥ WCET
  - WCET surestimé: l'analyse d'ordonnançabilité acceptera seulement des jeux de tâches faisables

➡ Sûr



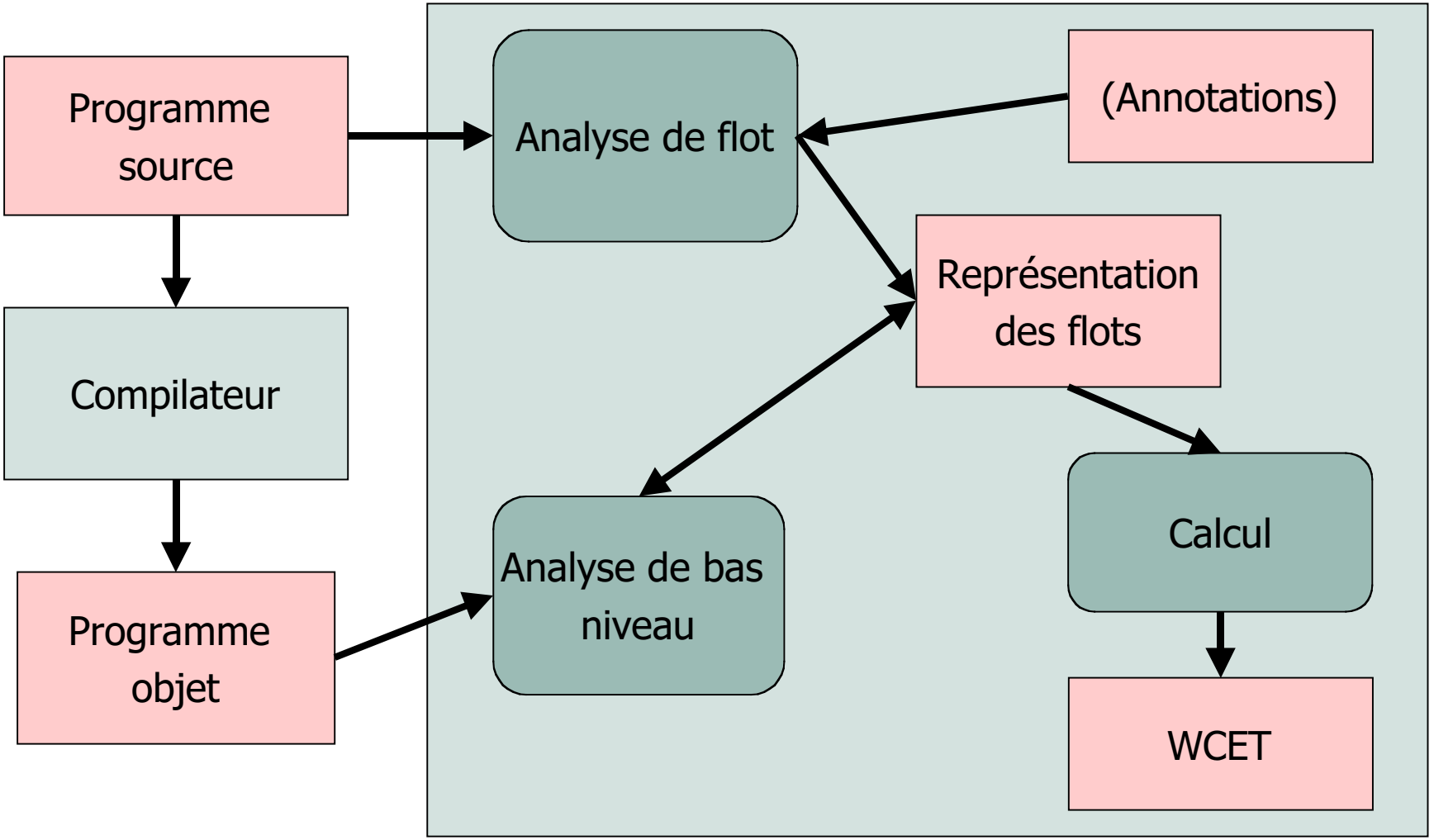
Systèmes temps réel strict ont besoin de sûreté ⇧ analyse statique

Niveaux d'analyse

- Analyse de haut-niveau
  - Détermine statiquement le chemin d'exécution le plus long dans un programme
  - Calcule le WCET le long de ce chemin d'exécution
  - ➔ Le langage doit être adapté à l'analyse statique
    - Langage dédié
    - Langage existant avec restrictions

- Analyse de bas-niveau (niveau matériel)
  - Détermine le temps d'exécution d'une séquence d'instructions sur une architecture donnée
  - ➔ Nécessite de connaitre précisément le matériel utilisé
  - ➔ Dépendant du matériel

Sous-problèmes de l'analyse statique de WCET



Analyse de flot (1/2)

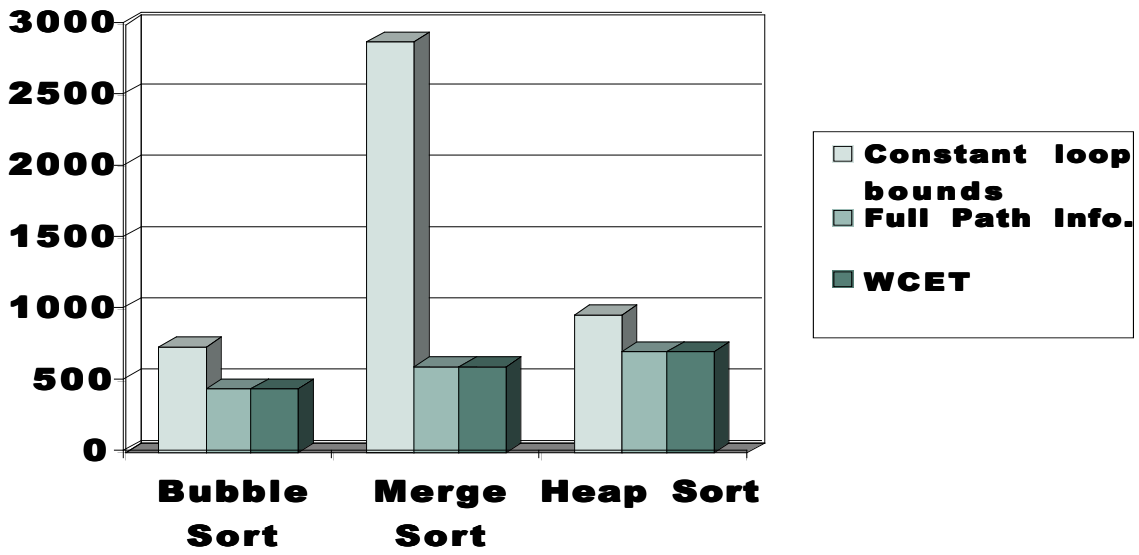
- Doivent être déterminés
  - Nombres maximum d'itérations des boucles
  - Autres éléments pouvant améliorer la qualité des estimations des WCET
    - Chemins infaisables ou mutuellement exclusifs
    - Bornes de boucles dans le cas de boucles non rectangulaires

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

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

Analyse de flot (2/2)

- Modes de détermination des flots
  - Automatique : infaisable dans le cas général (halting problem)
  - Manuelle : annotations
    - Simples constantes sur les boucles
    - Annotations symboliques
- Résultats de l'analyse de flot (P. Puschner)



Méthodes de calcul

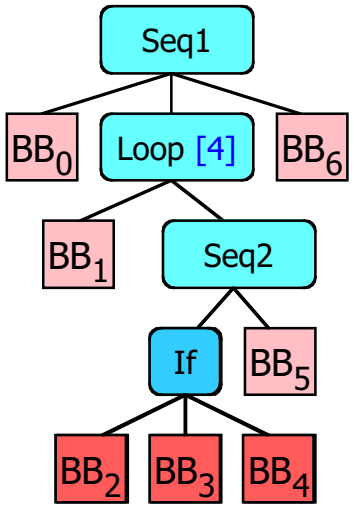
Analyse à base d'arbres (tree-based)

- Identification des blocs de base
- Détermination des temps d'exécution des blocs de base
  (analyse de bas-niveau)
- Calcul des temps d'exécution des chacune des structures syntaxiques du langage en utilisant les temps des blocs de base et un timing schema pour chaque construction

Méthodes de calcul

Analyse à base d'arbres (tree-based)

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) + WCET(test)



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)

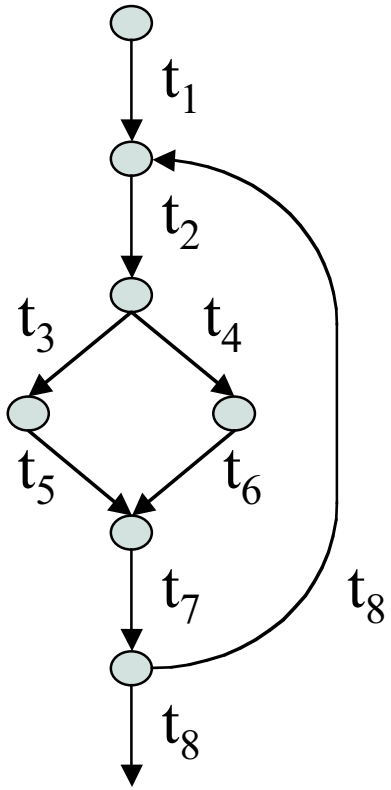
Méthodes de calcul

Analyse à base d'arbres (tree-based)

- Travaille sur une représentation du programme de haut niveau (arbre syntaxique)
  - Lien avec le code source aisé
  - Complexité des calculs maîtrisée
- Adaptée aux programmes bien structurés
  - Ne supporte pas toutes les optimisations de compilation
  - Ne supporte pas les constructions non conformes aux portées de l'arbre (jumps and gotos)
  - Prise en compte d'informations de flot élaborées n'est pas aisée (si non conforme à l'arbre syntaxique)
  - Mais, ...
- Rapide

Méthodes de calcul

Analyse IPET (WCET comme un problème d'optimisation)



Programmation linéaire en nombres entiers

- But

  max: $f_1t_1+f_2t_2+\ldots+f_nt_n$

- Contraintes structurelles

$$\forall\ v: \sum_{e_i \in In(v)} f_i = \sum_{e_i \in Out(v)} f_i$$

- Contraintes sur les chemins

  exemples : $f_i \leq k$ (maxiter boucle)

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

  $\Sigma\ c_i f_i \leq k$ (ratio entre nombres d'exécutions)

Méthodes de 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)

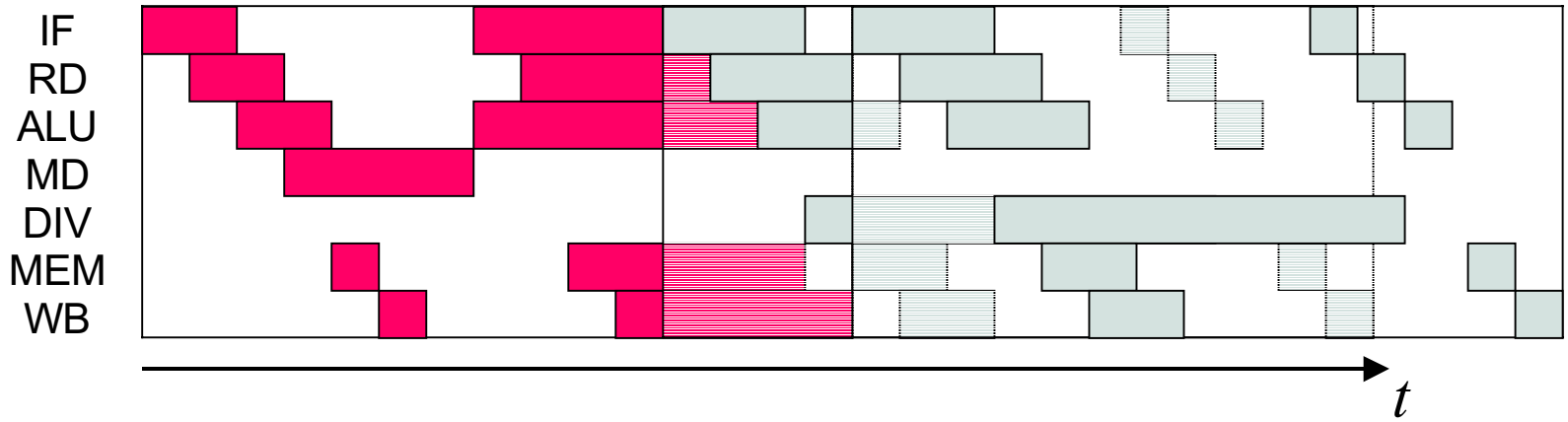
Analyse de bas niveau (introduction)

- 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

Analyse de bas niveau locale

Prise en compte des pipelines

- Principe : parallélisme entre instructions sucessives (effet local)
- Problème : simple ajout des temps d'exécution trop pessimiste
- Solution usuelle : tables de réservation décrivant quand chaque instruction accède les étages du pipeline



- Modification de la méthode de calcul
  - Tree-based: opérateur d'addition spécifique
  - IPET: contraintes supplémentaires dans le problème d'ILP

Analyse de bas niveau globale

Caches d'instructions : problème

- 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 inadéquat en contexte temps-réel strict
- Problème : estimation sûre du comportement des caches
  - Solution simple (tout miss) est excessivement pessimiste
  - Objectif : prédire si une instruction causera un hit (de manière sûre) ou pourra causer un miss (de manière conservatrice)

Analyse de bas niveau globale

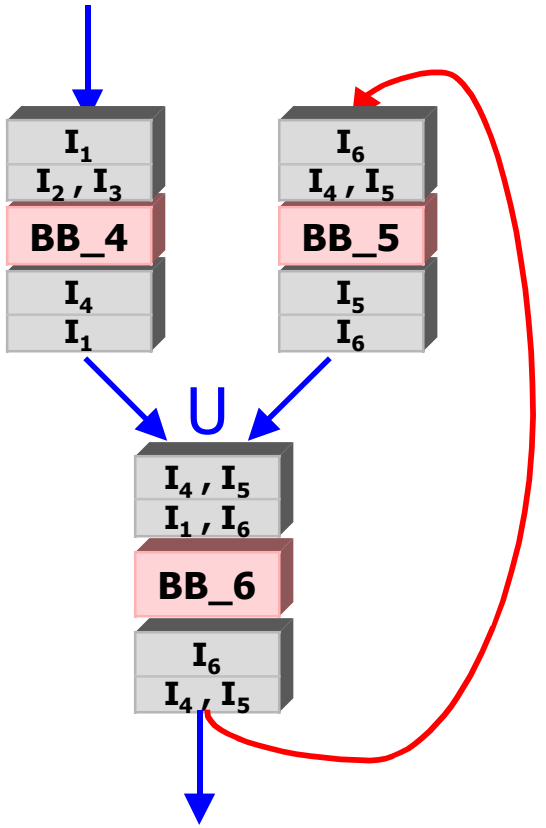
Caches d'instructions : simulation statique de cache

- Estimation sûre du comportement des caches
  - Prédit si une instruction causera un hit (de manière sûre) ou pourra causer un miss (de manière conservatrice)
- Division des instructions en catégories
  - *always miss*
  - *always hit*
  - *first miss*
  - *first hit*
- Classification des instructions calculée à partir d'états abstraits de caches (ACS)
  - État du cache représentant tous les chemins d'exécution possibles
  - Itération de point fixe pour le calcul

Analyse de bas niveau globale

Caches d'instructions : simulation statique de cache

```
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
```



Analyse de bas niveau globale

Caches d'instructions : autres méthodes

- Méthodes d'interprétation abstraite [Sarbrüken]
  - Similitudes avec la simulation statique de cache
- Méthodes utilisant l'ILP [Séoul]
  - Lignes de programme: sequences d'instructions associées à un bloc de cache
  - Deux temps d'exécution par ligne de programme : hit ou miss
  - Nombre d'exécution d'une ligne de programme = nb hits + nb misses
  - Intégré dans méthode de calcul par ILP

    max: $\Sigma\ (f_{hit,i}\ t_{hit,i} + f_{miss,i}\ t_{miss,i})$

    Contraintes pour le comportement les lignes de prog. / cache
  - Modélisation très fin grain ➔ explosion des temps de calcul

Analyse de bas niveau globale

Caches de données

- Caches de données [Chalmers]
  - Problème : obtention de l'adresse des données (dynamique)
  - Solution : 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
  - Intérêt : prise en compte de toutes les caractéristiques de l'architecture
  - Limite : temps de simulation

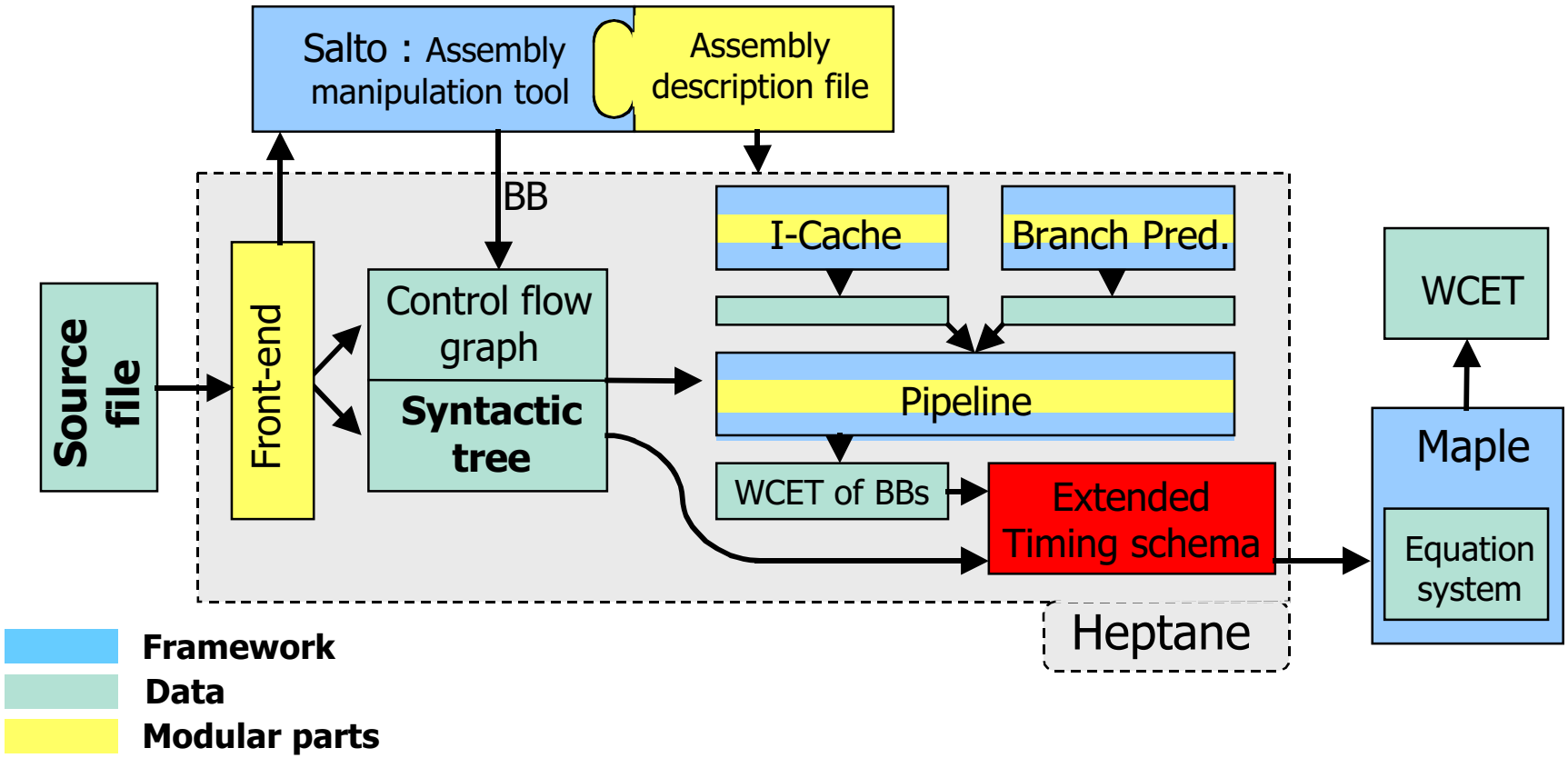
Analyse de bas niveau globale

Autres éléments architecturaux 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 à une classification
  - Globaux (basé sur historique du branchement + autres branchements exécutés récemment) [Singapour]
    - Mapping vers un problème d'ILP

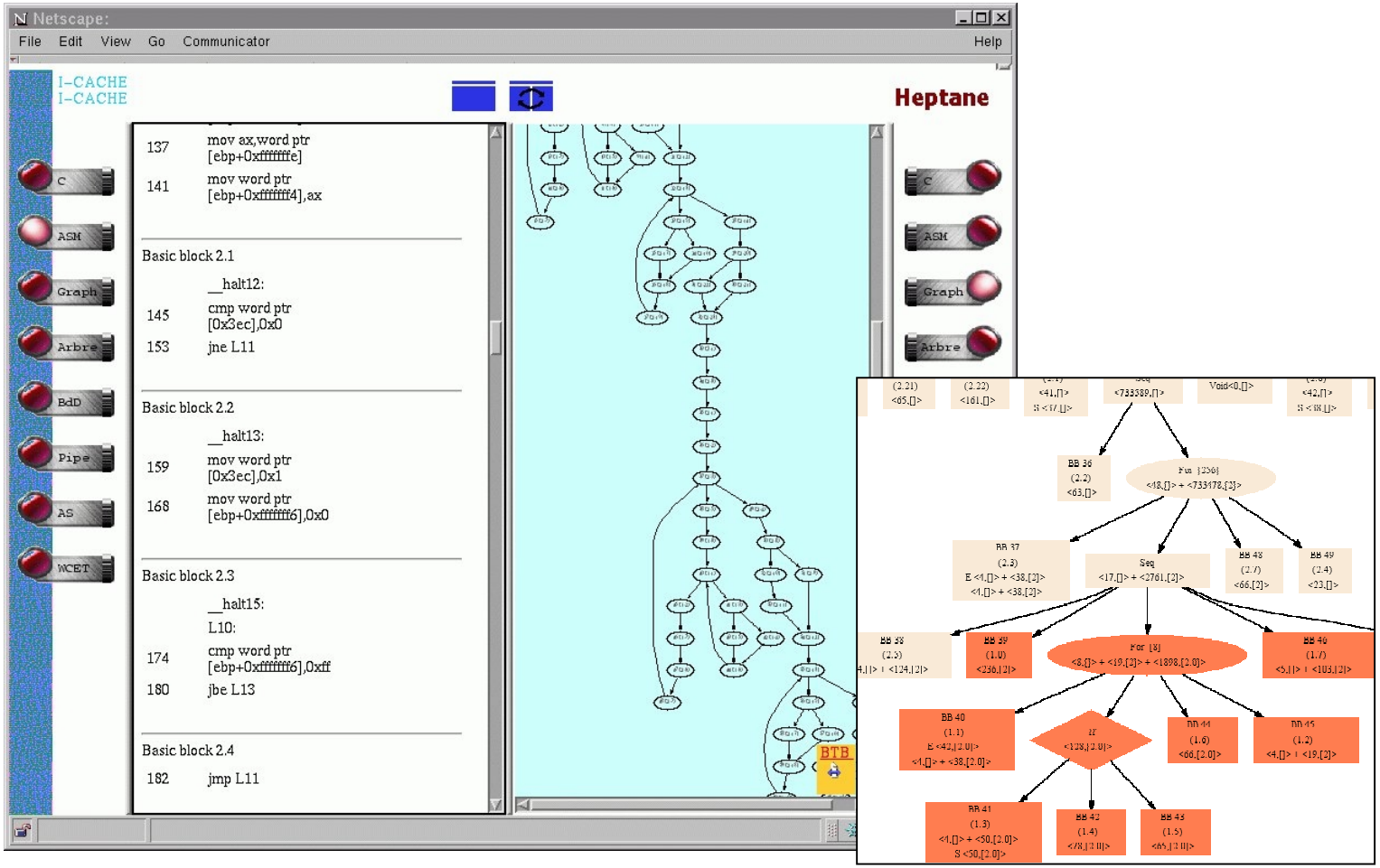
Analyse de WCET

Heptane



Analyse de WCET

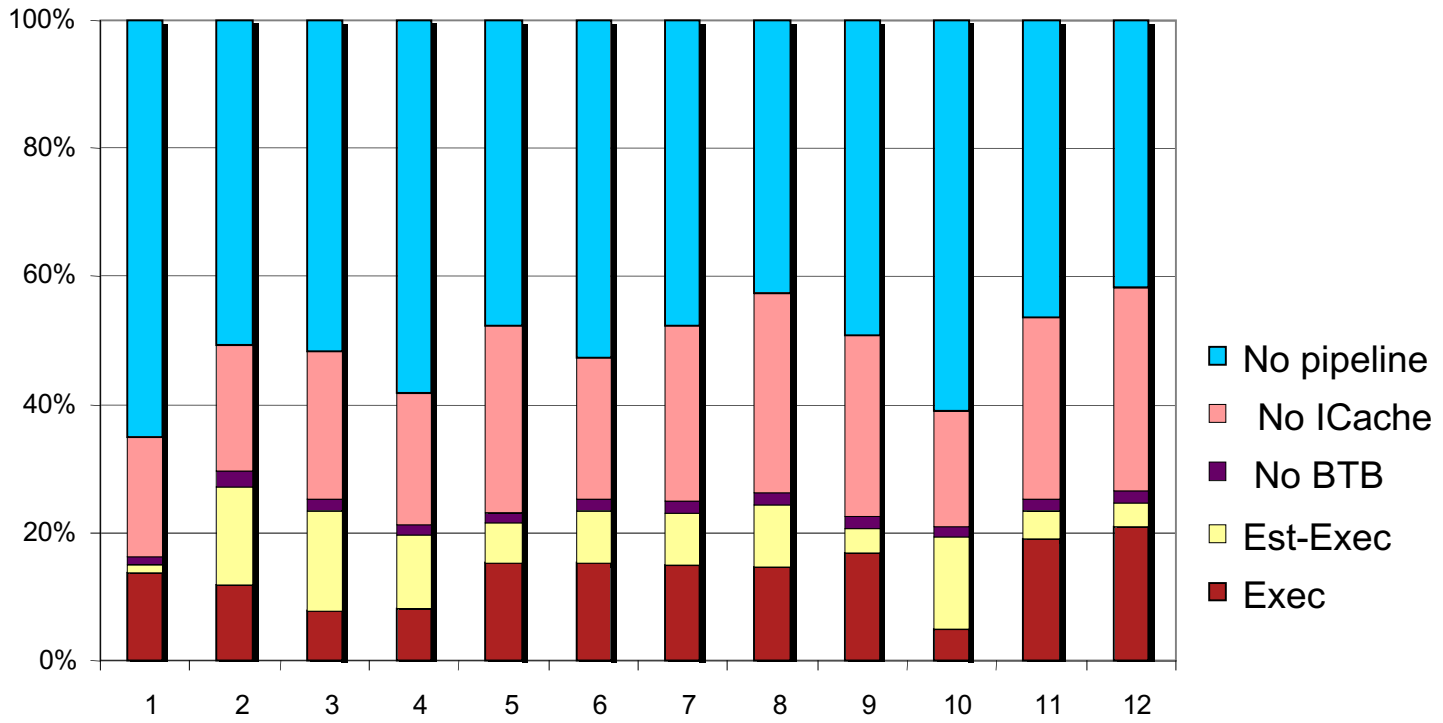
Heptane



Analyse de bas niveau

Quelques résultats quantitatifs

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



- Sans prise en compte architecture, facteur de surestimation = 12.5
- Prise en compte architecture réduction pessimisme d'un facteur 6.6

Analyse de bas niveau

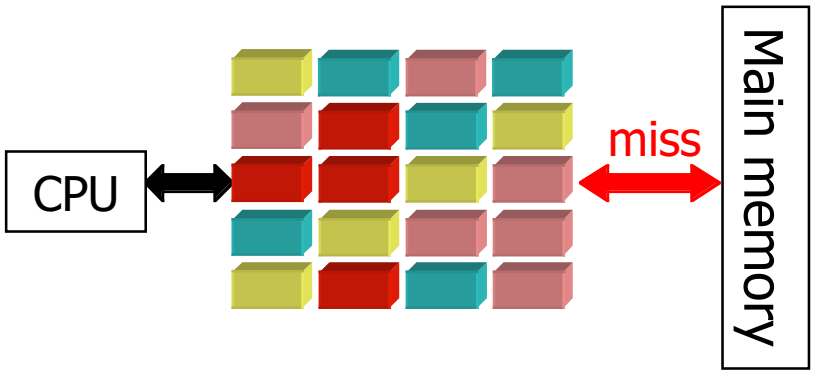
Travaux en cours à l'IRISA - gel de caches [RTSS02]

- Limites de l'analyse de caches
  - Degrés d'associativité élevés, caches associatifs
  - Politiques de remplacement de caches non déterministes
    - ➔Dégradation des performances des méthodes d'analyse
  - Latence lors des changements de contexte
- Gel du contenu du cache (cache locking)
  - Rend le temps d'accès à chaque information connu
  - Pas d'impact sur les temps de changement de contexte
  - Disponible dans de nombreuses architectures (PowerPC, Motorola MPC7451, Coldfire, MIPS, ARM)
  - Simple à mettre en œuvre, obtention des WCETs simplifiée, indépendant de la politique de remplacement de cache
  - Gel partiel (MPC 7451) : cohabitation applications temps-réel strict et souple

Analyse de bas niveau

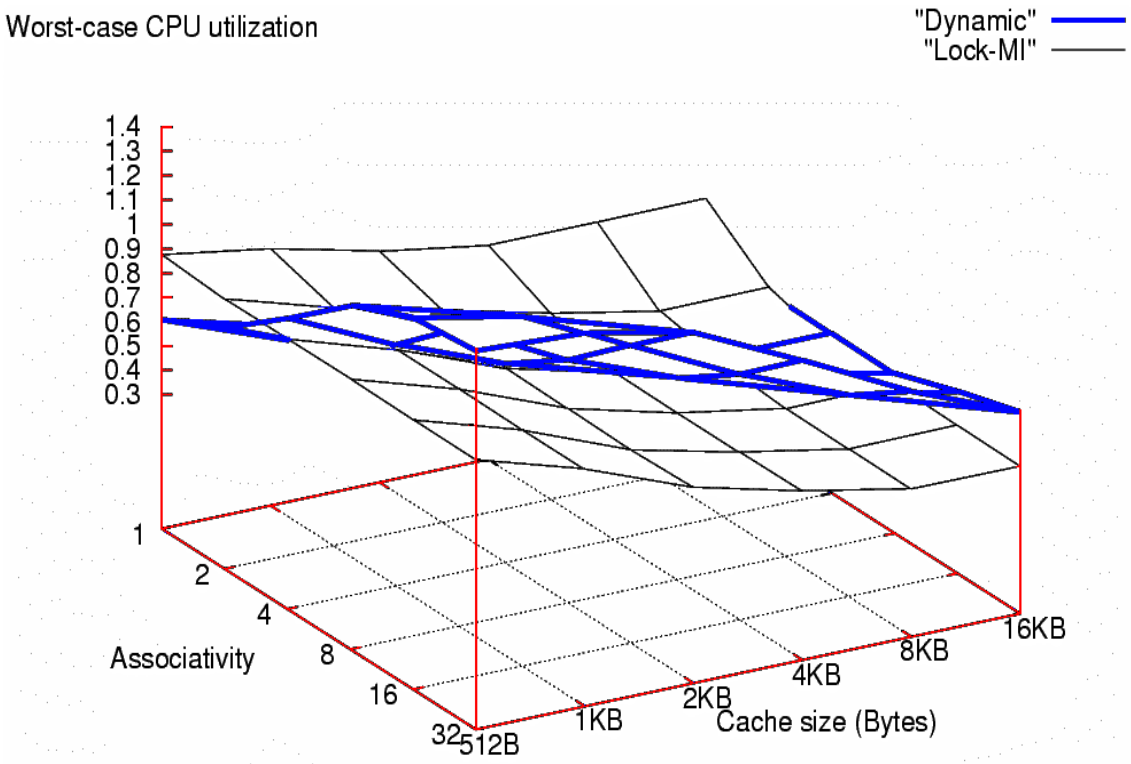
Gel statique de cache

- Gel du contenu du cache
  - Pour l'ensemble des tâches
  - Pour toute leur durée de vie (gel statique)
- Problème de sélection du contenu
  - Critère de sélection
  - Combinatoire
- Solution
  - Critère de sélection : ordonnançabilité (comportement pire-cas)
    - Sélection pour minimisation de la charge CPU ($\Sigma C_i/P_i$)
    - Sélection pour minimisation des interférences entre tâches
  - Réduction de la combinatoire :
    - optimisation sur le chemin d'exécution pire cas sans cache
    - faible complexité (pseudo-polynomial)



Analyse de bas niveau

Gel statique de cache [RTSS02] - performance pire-cas



- Meilleur que l'analyse pour
  - Caches de taille importante
  - Degrés d'associativité élevés

Analyse de bas niveau

Travaux en cours à l'IRISA

- Gel de cache
  - Gel partiel pour cohabitation applications temps-réel strict et souple (voire non temps-réel)
  - Caches de données
  - Gel statico-dynamique pour applications très grandes par rapport à la taille du cache
- Espaces d'adressage séparés
  - Impact sur contraintes de temps-réel, cohabitation d'applications de criticités différentes
- Placement en scratchpad

Analyse de bas niveau

Points ouverts (1/2)

- Exécution non ordonnée, exécution spéculative
  - Anomalie : un accès hit peut provoquer un WCET plus long qu'un accès miss [Chalmers]
  - Anomalie : un chemin exécuté spéculativement peut remettre en cause l'analyse de cache
- Architectures multiprocesseur ou SMT
  - On ne peut plus considérer les tâches de manière isolée
  - Remet en cause la séparation (analyse de WCET <-> analyse d'ordonnançabilité)
- Caches
  - politique de remplacement non déterministes (random), degrés d'associativité élevés
  - gel de cache
- Temps de raffraichissement DRAM, traduction d'adresse, …

Analyse de bas niveau

Points ouverts (2/2)

- Validation des modèles temporel du matériel
- Jusqu'ou aller dans la prise en compte du matériel ?
  - Cohabitation des différents éléments architecturaux
  - Complexité des analyseurs, retard / arrivée du matériel
  - Approches probabilistes
- Conception ou utilisation de matériel prévisible ?

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 (2 éditions en 2001 et 2002), en conjonction avec la conf. Euromicro on real-time systems
- Estimation de temps d'exécution au pire-cas par analyse statique et application aux systèmes d'exploitation temps-réel. A. Colin, thèse de l'université de Rennes I, octobre 2001
- Calcul de majorants de pires temps d'exécution: état de l'art. A. Colin, I. Puaut, C. Rochange, P. Sainrat, PI IRISA 1461, mai 2002
- Low-complexity algorithms for static cache locking in hard real-time systems, Proc. of the 2002 IEEE Real Time Systems Symposium (RTSS 2002) - Austin, Texas, décembre 2002

Quelques pointeurs

Outils d'obtention de WCET

- Universitaires
  - Cinderella [Princeton] - 68000
    - Calcul par ILP, cache, pipeline, annotations demandées interactivement
  - Heptane [IRISA] - Pentium- MIPS - autres architectures (planifié)
    - Calcul tree-based, cache (dynamique ou gelé), pipeline, prédiction de branchement, annotations symboliques
    - Diffusion en open-source (http://www.irisa.fr/solidor/work/hades.html)
  - En cours : Outil commun 3 universités Suédoises
    - Analyse de flot automatique + analyse bas-niveau
- Commerciaux
  - Bound-T [SSF] : pipelines, pas de caches
  - Absint [Sarbrüken] : code objet, utilisation interprétation abstraite (cache, pipeline)