Outils d'optimisation de code pour applications embarquées

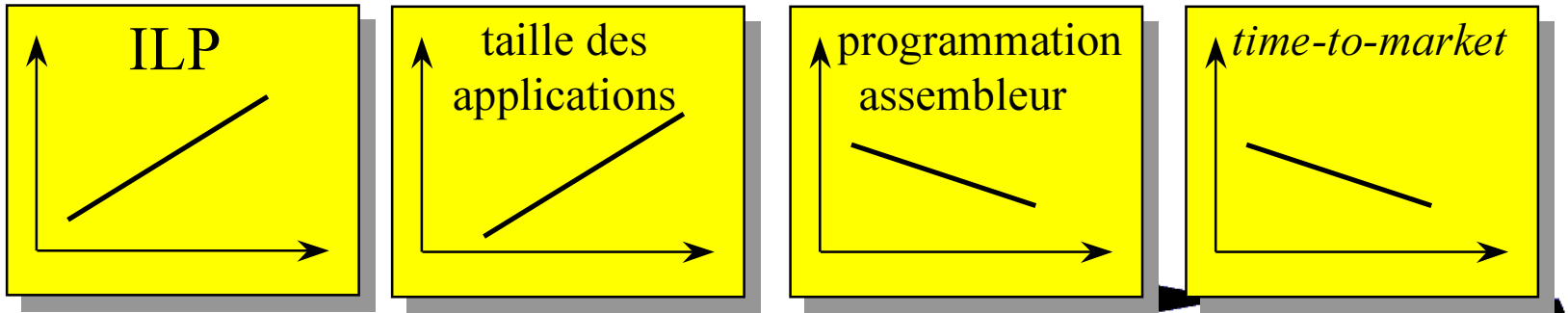
François Bodin (bodin@irisa.fr)
IRISA/IFSIC
Campus Universitaire de Beaulieu
35042 Rennes

IRISA

Evolution

code compact != performance
à jeu d'instructions fixé

- De la répartition logiciel/matériel
- Du codage des applications
- De la durée de développement
- Des architectures, techniques VLIW, instructions «multimédia»
- Importance croissante des compilateurs



IRISA

Spécificités

- Contexte
  - programmation sous/sur ensemble de C
  - architectures non orthogonales
  - génération de code difficile
  - extraction de parallélisme difficile
  - mais temps de compilation importants acceptables

Spécificités

- Contraintes
  - performances critiques
  - taille du code critique
  - consommation électrique limitée
  - ...
- Outils
  - nécessité de les recibler rapidement
  - interaction avec la CAO
  - ...

IRISA

Plan de l'exposé

- Architectures VLIW («gros DSP»)
  - principes
  - principales caractéristiques
  - exemples
- Optimisations de code orientées architectures
  - code source
  - code machine
  - optimiser globalement
- Quelques outils
  - FlexCC, Cosy, GCC, Salto, ...
- Quelques perspectives
  - schémas itératifs de compilation

Architectures VLIW

Principes

- Very Long Instruction Word
- Exécution :
  - une unité de contrôle émet une instruction longue par cycle
  - chaque instruction longue initialise simultanément plusieurs opérations indépendantes
  - le format des instructions comporte un grand nombre de bits
  - chaque instruction nécessite un petit nombre, statiquement évaluable, de cycles pour s'exécuter
  - les instructions peuvent être compressées en mémoire
- Principales caractéristiques
  - jeux d'instructions étendus (opérations de type RISC)
  - ordonnancement statique des instructions
  - chaque opération peut être pipelinée
  - branchements retardés
  - gardes
  - mémoire cache

Jeux d'instructions étendus

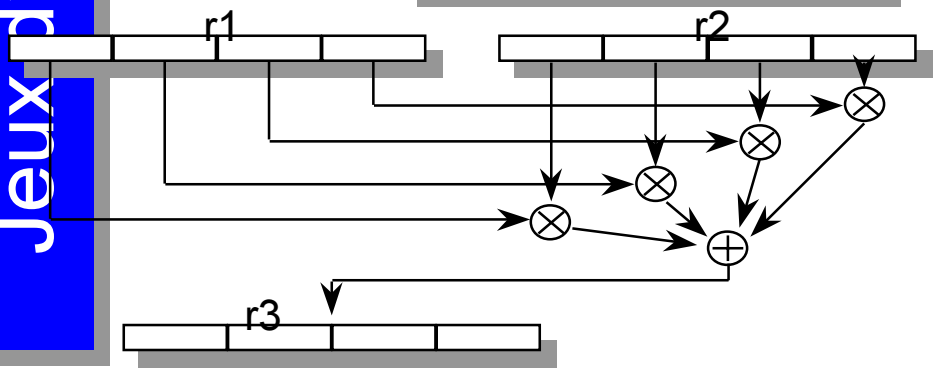
- Opérations «multimédia»
- Arithmétique saturée
- Opérations gardées

c = a+b
- si a+b > max_rep alors c = max_rep
- si a+b < min_rep alors c = min_rep
- sinon a+b

```
int sadd(int a, int b){
int result;
result = a+b;
if (((a^b)& 0x80000000) == 0){
 if ((result^a) & 0x80000000)
  result = (a<0)? 0x80000000:0x7fffffff
}
return result;
}
```

```
result = _sadd(a,b)
```

```
z=(x&15*y&15) +
   ((x>>8) &15*(y>>8)&15) +
   ((x>>16) &15*(y>>16)&15) +
   ((x>>24) &15*(y>>24)&15)
```



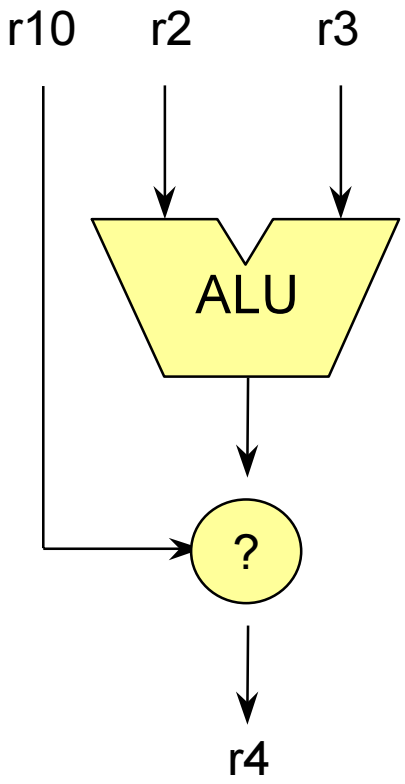
IRISA

Instructions gardées

```
if (x==3) y=y1+y2;
```

```
        cmp r11,3
        bne suite
        add r2 r3→r4
suite:
```

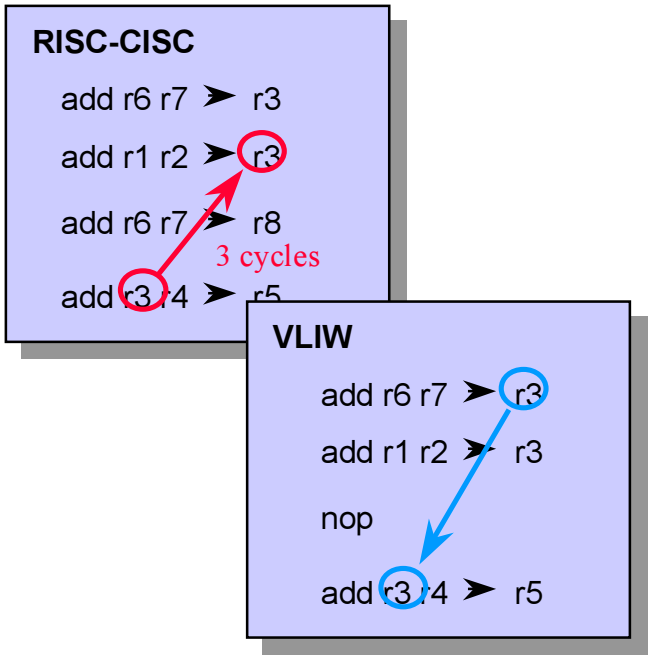
```
        iseq r11 3→r10
IF r10 add r2 r3→r4
```



IRISA

Ordonnancement des instructions

- Statiques, insertion de nop
- Placement dans les slots
- Branchements retardés




```
add r1,r2,r3
ld [r3],r4
add r5,1,r6
add r4,4,r4
jmp suite
...
```

Ordonnancement

instruction VLIW

	slot 1	slot 2	slot 3	slot 4	slot 5
1	add r1,r2,r3	add r5,1,r6			
2		~~ld [r3],r4~~		ld [r3],r4	
3					
4					
5	add r4,4,r4		jmp suite		

```
                add r1,r2,r3
                b suite
                nop
                ...
suite:
```

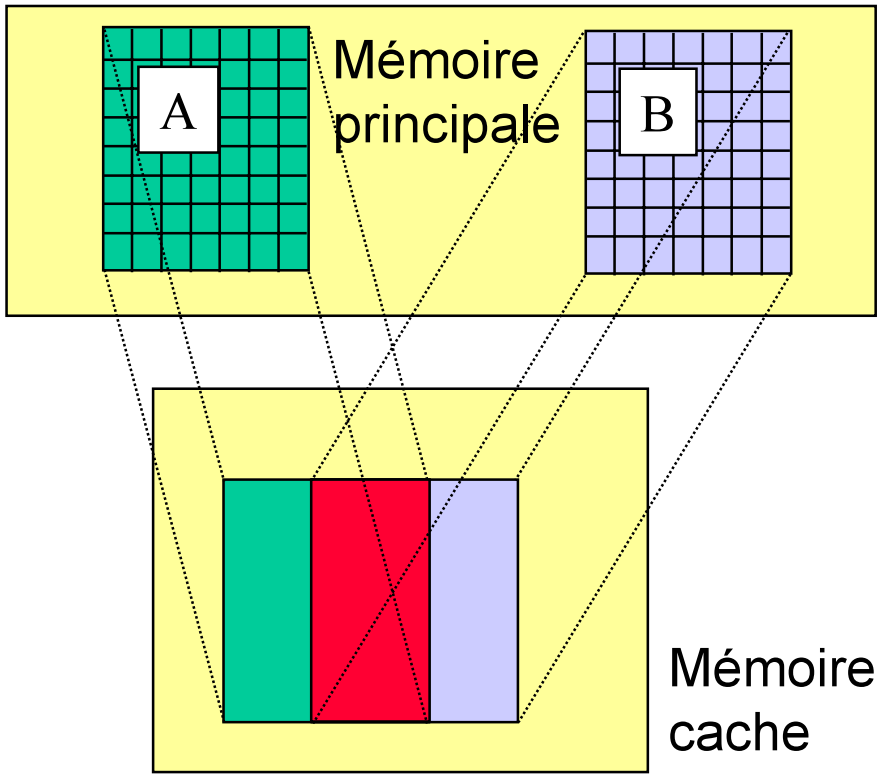
delay slot Trimedia
15 instructions!

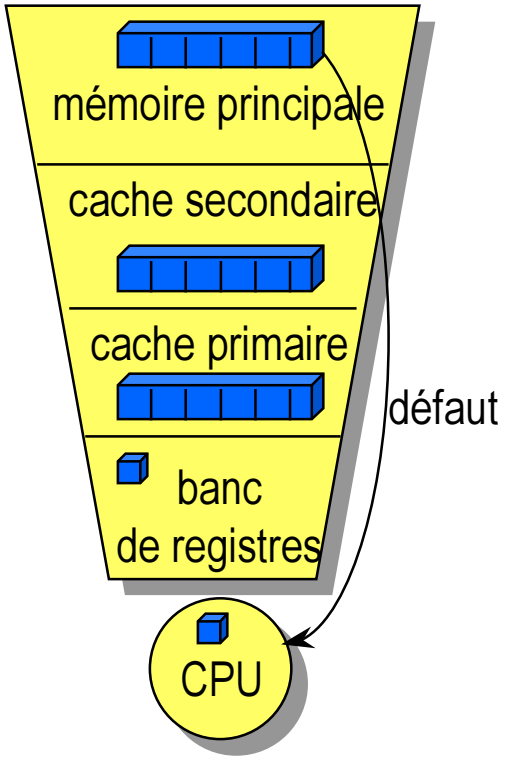
```
                b suite
                add r1,r2,r3
                ...
suite:
```


IRISA

Hiérarchie mémoire

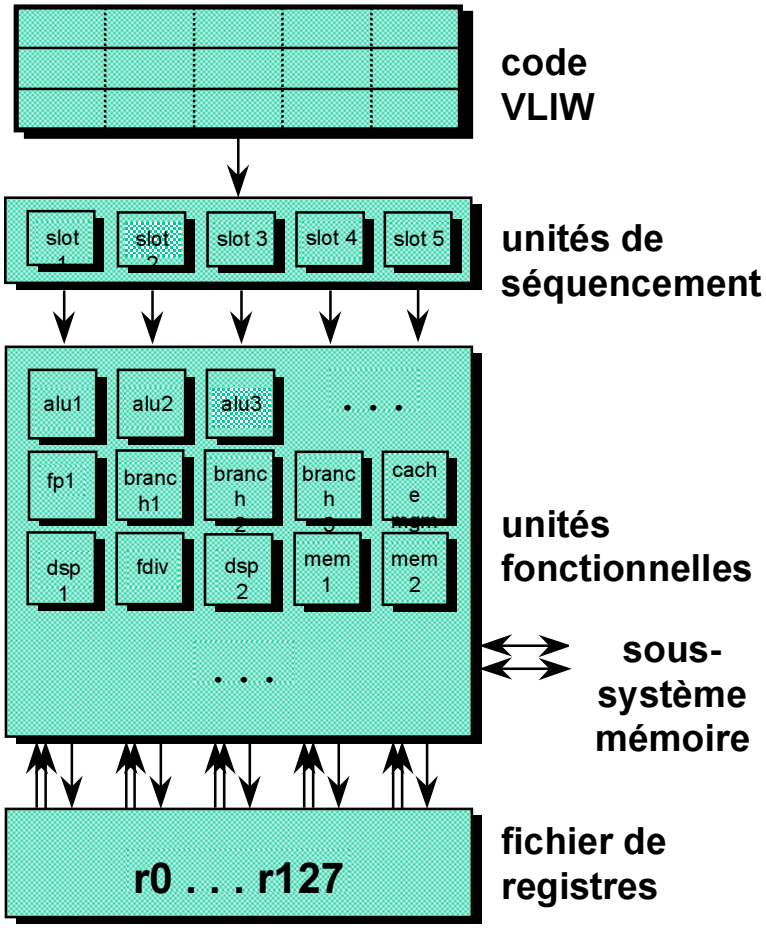
- Actuellement une tendance
- Cache instructions
- Cache de données
- Organisation en bancs





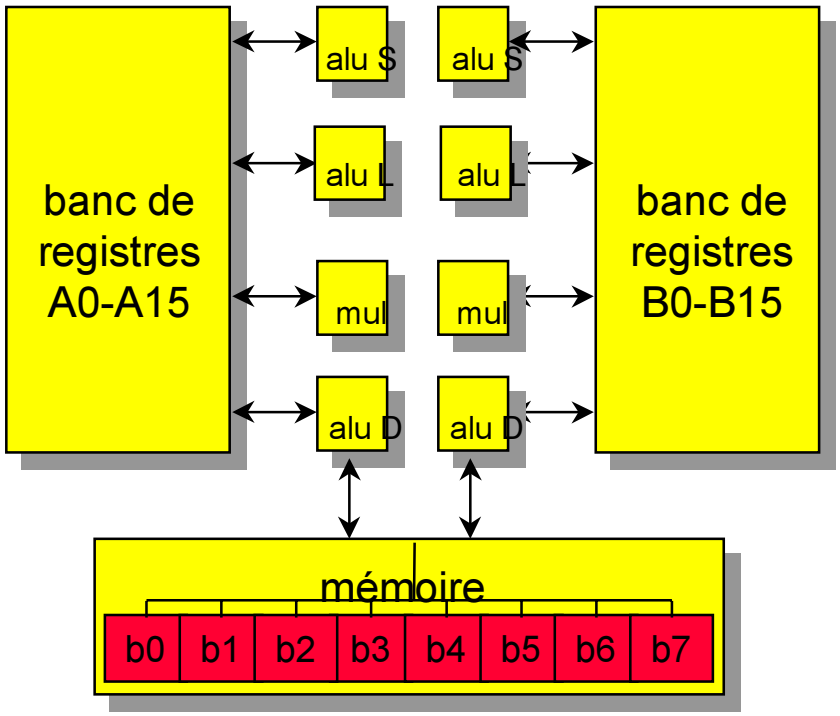
IRISA

Philips Trimedia



5 opérations RISC par cycle
32 k-octets cache instructions
16 k-octets cache données

Texas TMS320C6x



8 opérations RISC par cycle (256bits)
32 k-octets cache instructions
16 k-octets cache données

IRISA

Optimisations de code

Facteur de performance

- Structure et analyse des codes
  - alias sur la mémoire et dépendances de données
  - taille du code, mauvais placement des données
  - utilisation d'extension
- Optimisations orientées architectures
  - extraction du parallélisme
  - bonne utilisation de la hiérarchie mémoire
  - limitation du nombre de branchements
- Et bien sûr
  - génération de code efficace
  - optimisations « *peephole* », etc.

Optimisations du code source

- Ecriture du code spécifique à l ’embarqué
  - problèmes d’alias : pointeurs restreints
  - des instructions spécifiques : intrinsèques
- Optimisations orientées architectures
  - transformation des structures de données
  - *inlining*, *cloning* des fonctions
  - adressage des structures
  - transformations de boucle
  - ...
- Accroissent fréquemment la taille du code

Pointeurs restreints

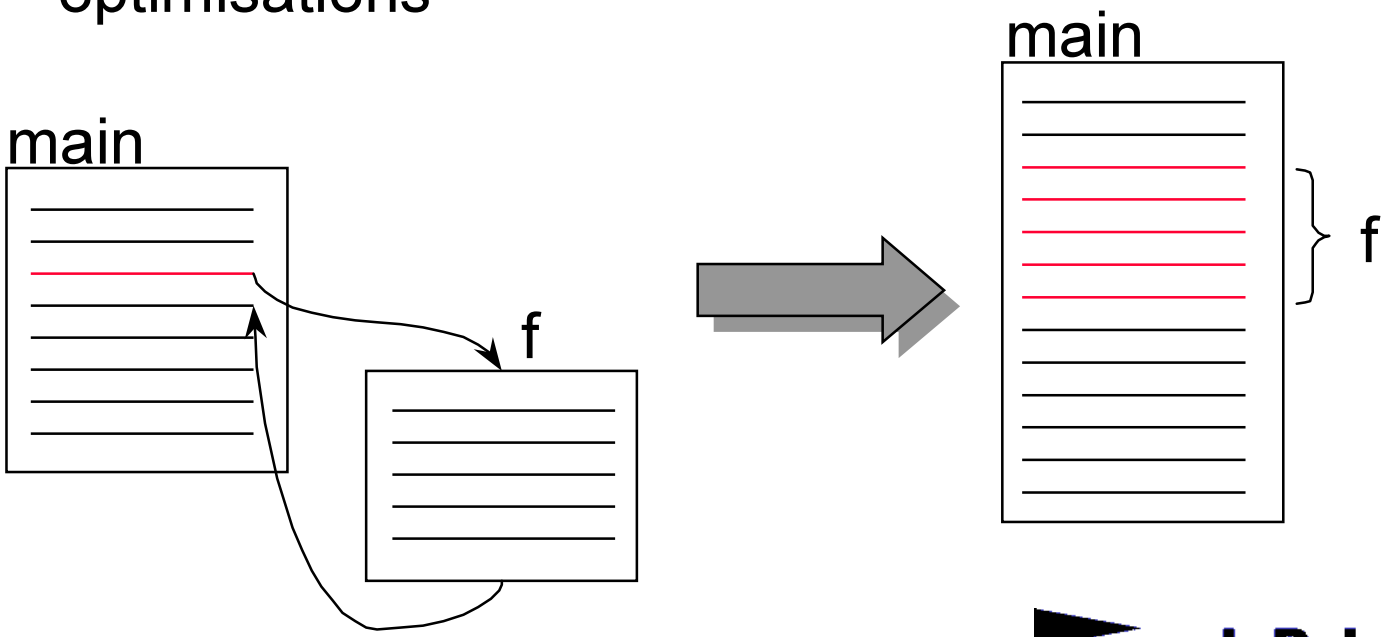
- Alias = dépendances
- Pointeurs restreints
  - gain : 50% de cycles en moins avec pointeurs restreints
- Danger : source d'erreur !

```
void f(int n, int *restrict a,
               int * restrict b,
               int * restrict c) {
  int   i;
  for(i=0; i<n; i=i+2)
    a[i]= b[i] + c[i];
    a[i+1] = b[i+1]+c[i+1];
}
```


IRISA

Inlining

- Remplacement d'un appel de fonction par le corps de la fonction
  - élimination du coût de l'appel
  - augmentation du parallélisme potentiel
  - expansion du code
  - amélioration des analyses et autres optimisations



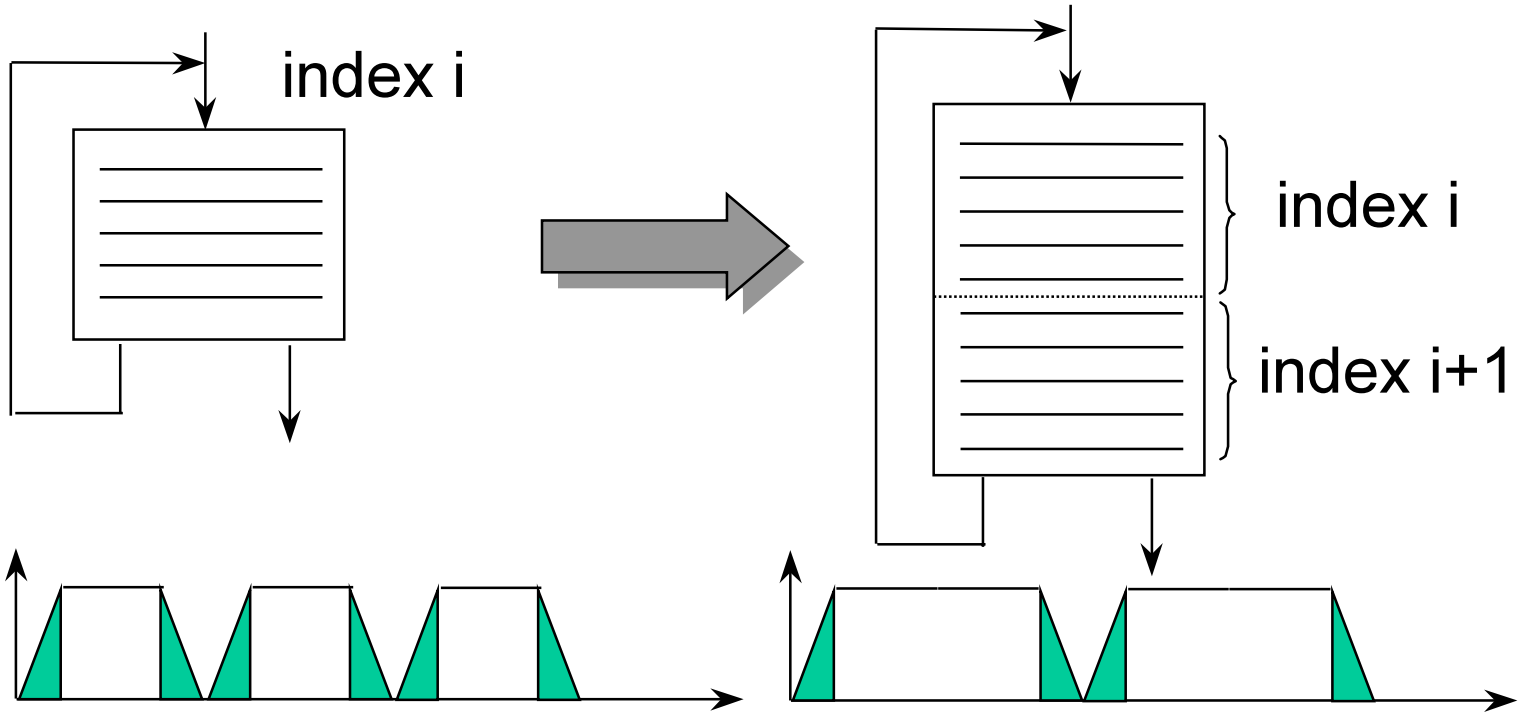
IRISA

Transformations de boucles

- Exploitation du parallélisme
  - dépliage de boucle
  - pipeline logiciel
  - effondrement («loop collapsing»)
  - ...
- Amélioration de la localité
  - blocage de boucles
  - échange de boucles
  - fusion/distribution
  - transformations unimodulaires
  - «array padding»
  - ...
- Divers
  - «loop peeling»,
  - reconnaissance d'idiomes
  - «unswitching»
  - ...

Dépliage de boucle

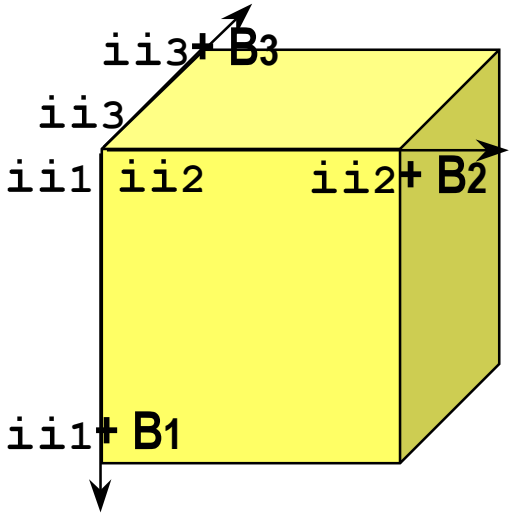
- Duplication d'un corps de boucle
  - accroît la taille des blocs
  - diminue le surcoût des boucles
  - diminue le nombre de branchements

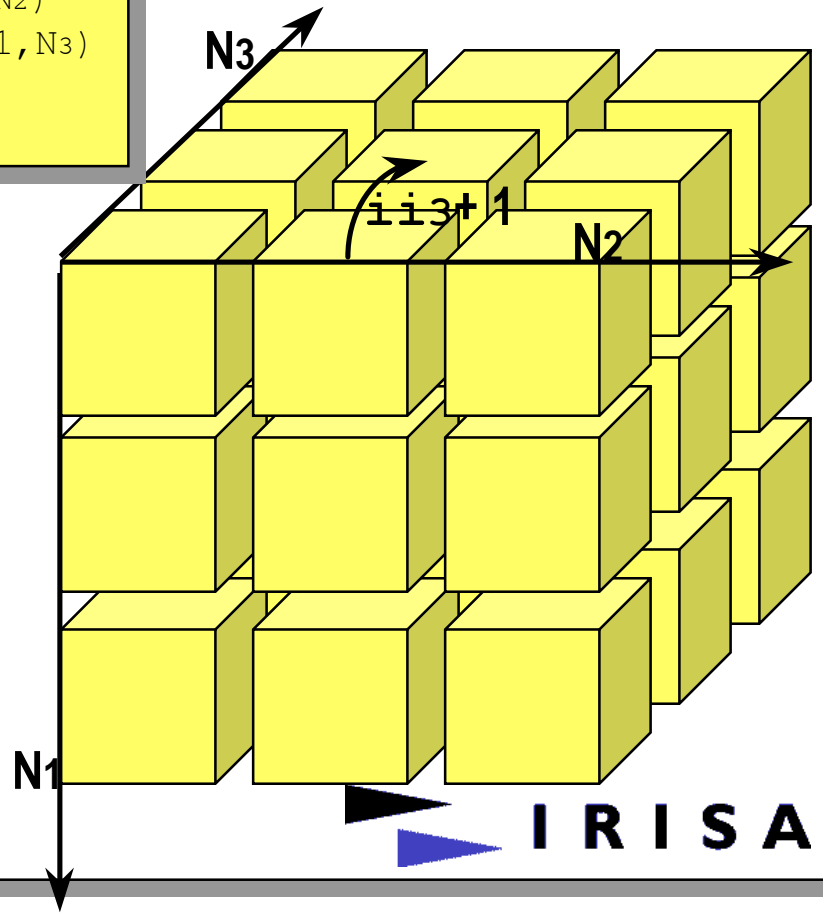


IRISA

- Blocage de boucles
  - l'espace d'itération est découpé en blocs
  - données d'un bloc < taille du cache

```
DO 10 ii1 = 1, N1, B1
 DO 10 ii2 = 1, N2, B2
  DO 10 ii3 = 1, N3, B3
   DO 10 i1 = ii1, min(ii1 + B1 -1, N1)
    DO 10 i2 = ii2, min(ii2 + B2 -1,N2)
     DO 10 i3 = ii3 , min(ii3 + B3 -1,N3)
       ....
10    CONTINUE
```

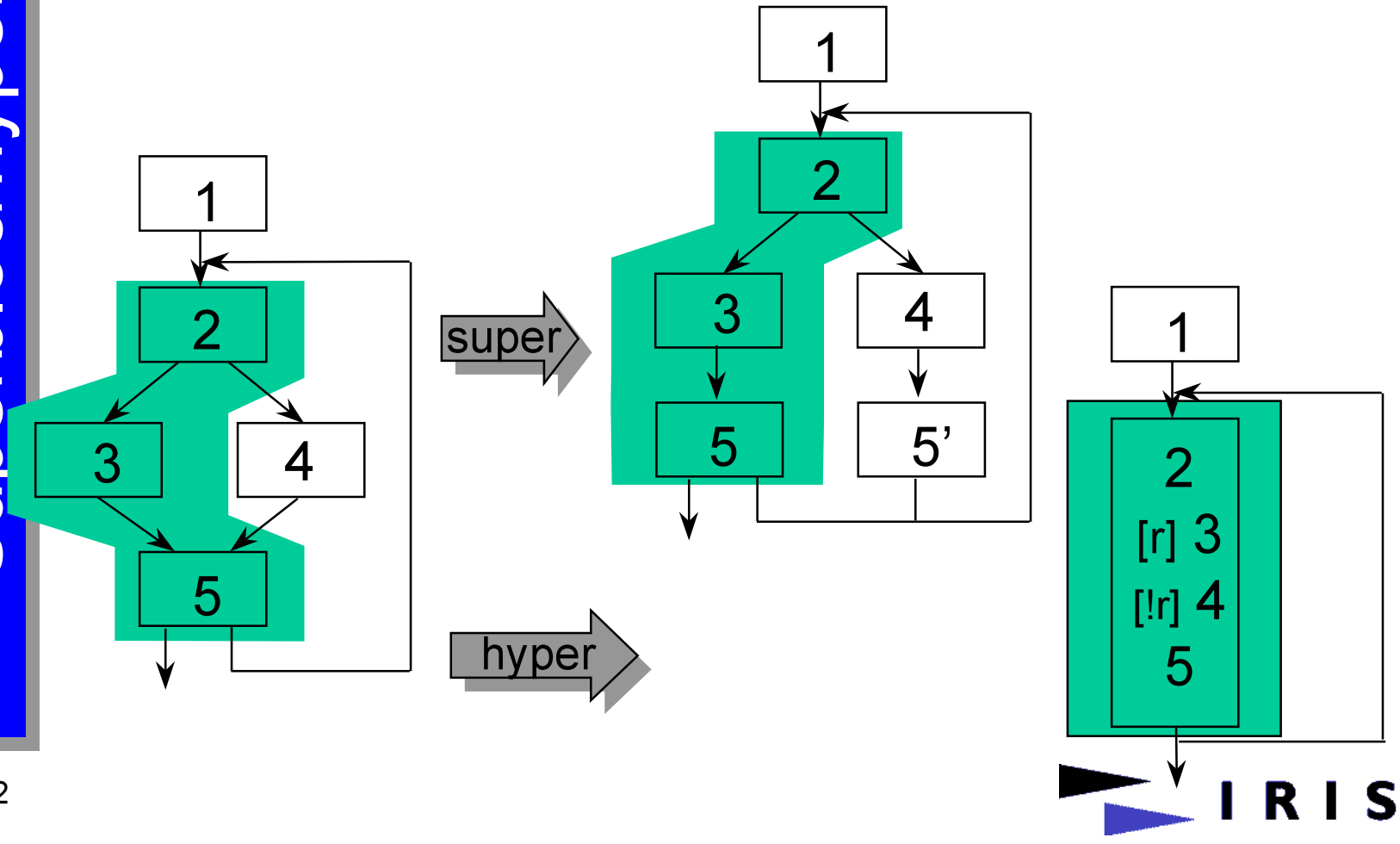




- Exploitation de l'ILP
  - bloc de base
    - compactage
  - boucle avec un corps simple
    - pipeline logiciel
  - contrôle de flots complexe
    - *trace scheduling*
    - *superblocs*
  - utilisation des gardes
    - *hyperblocs*
- Allocation de registres
- Accroissent fréquemment la taille du code

Superbloc/hyperbloc

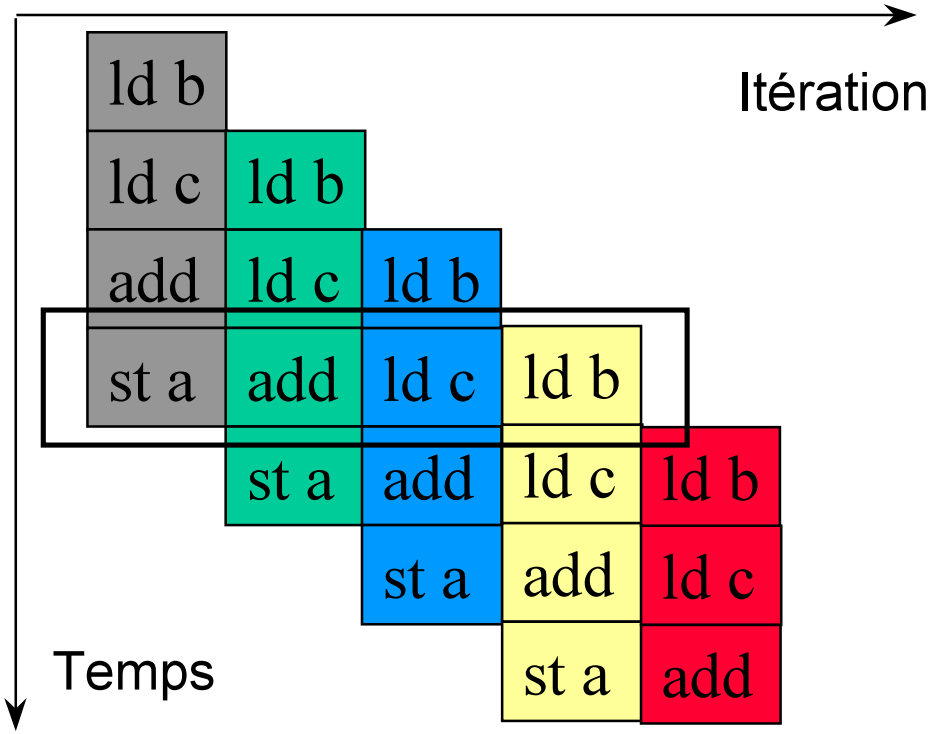
- Région de programme avec un seul point d ’entrée
  - migrations des instructions au delà des branchements
  - augmentation du parallélisme potentiel



IRISA

Pipeline logiciel

- Extraction de parallélisme pour les boucles
- Utilise le dépliage de boucle
- Taille de code généralement importante
- Complexe à mettre œuvre



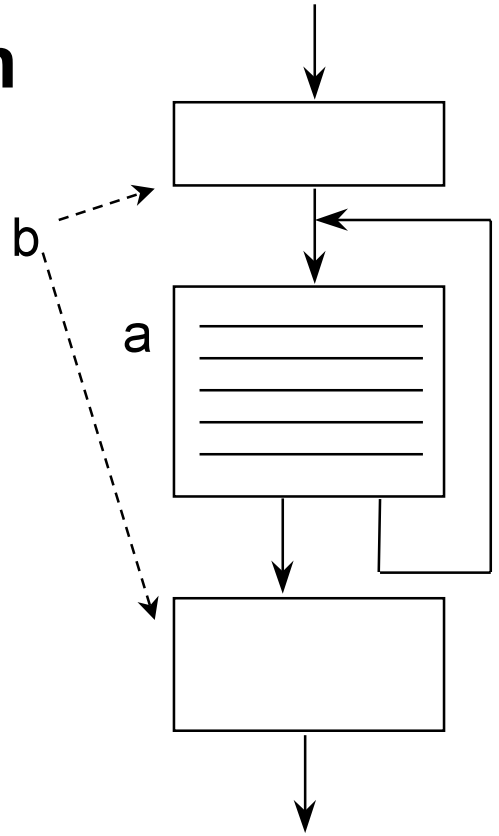

```
for(i=0; i<n; i++) {
    a[i] = b[i] + c[i]
}
```


IRISA

Optimiser globalement

- Objectifs globaux
  - taille du code
  - performance du cache instructions
  - ...
- Compilation classique locale
- Optimiser globalement
  - comment évaluer les alternatives
  - où autoriser un accroissement important du code
  - paramètres d'exécution
  - infrastructure

- **Global Constraints Driven Strategy (GCDS)**
- Evaluation de plusieurs optimisations par groupe d'instructions
- Stratégie globale de choix
- Modèle de performance
- Profiling



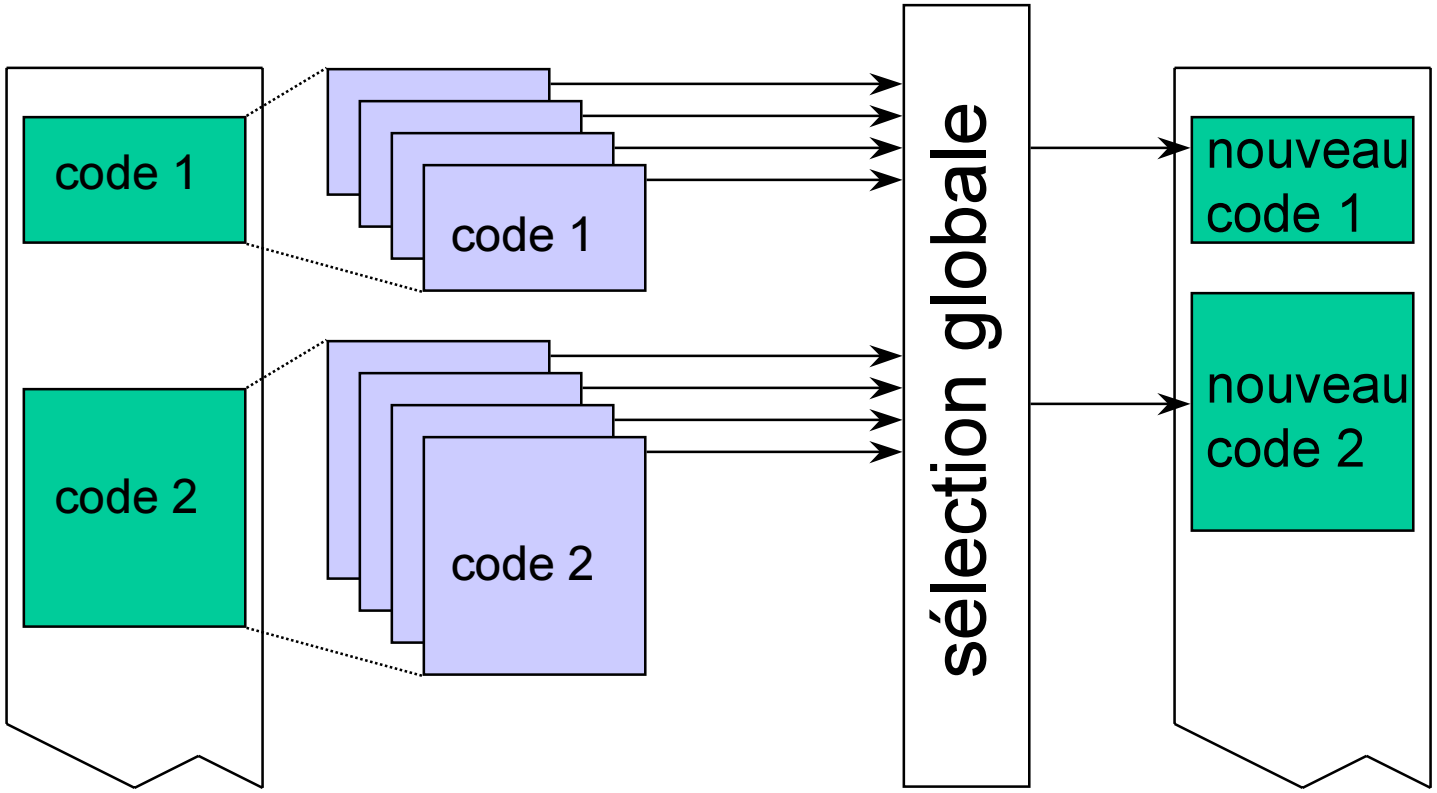
$$s = a + b$$

$$t = W \times (a \times n + b(n))$$

IRISA

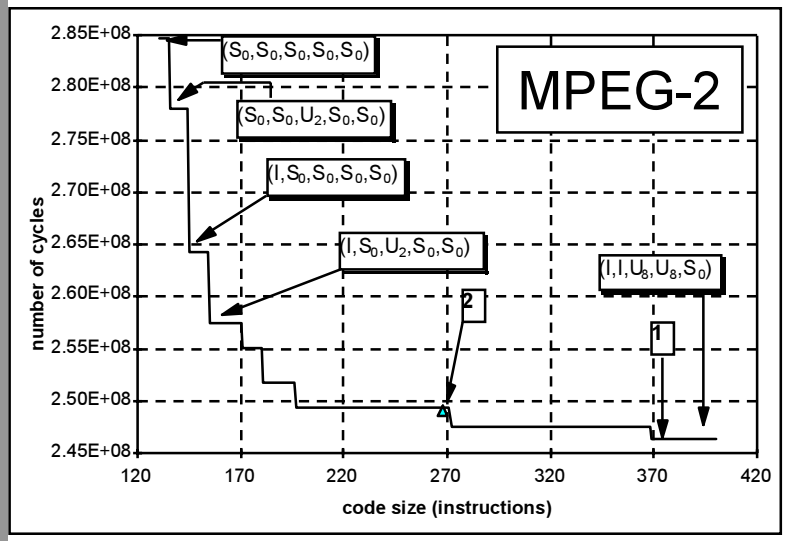
Principes de GCDS

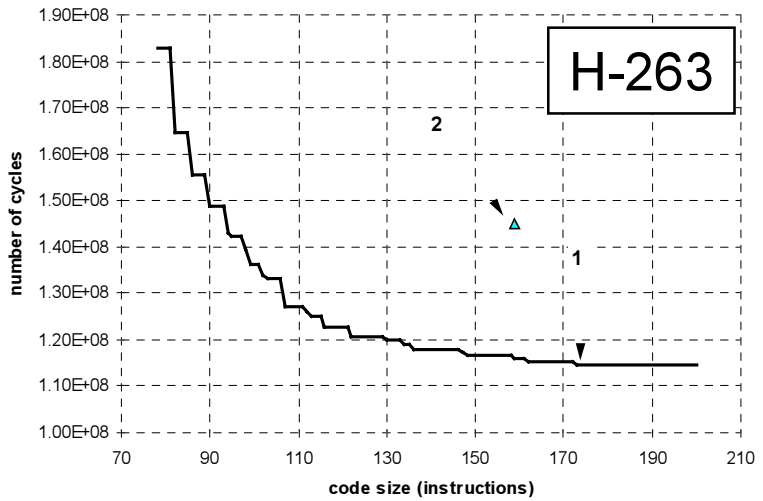
- Multiples versions optimisées
- Fonction de choix globale (simplex)



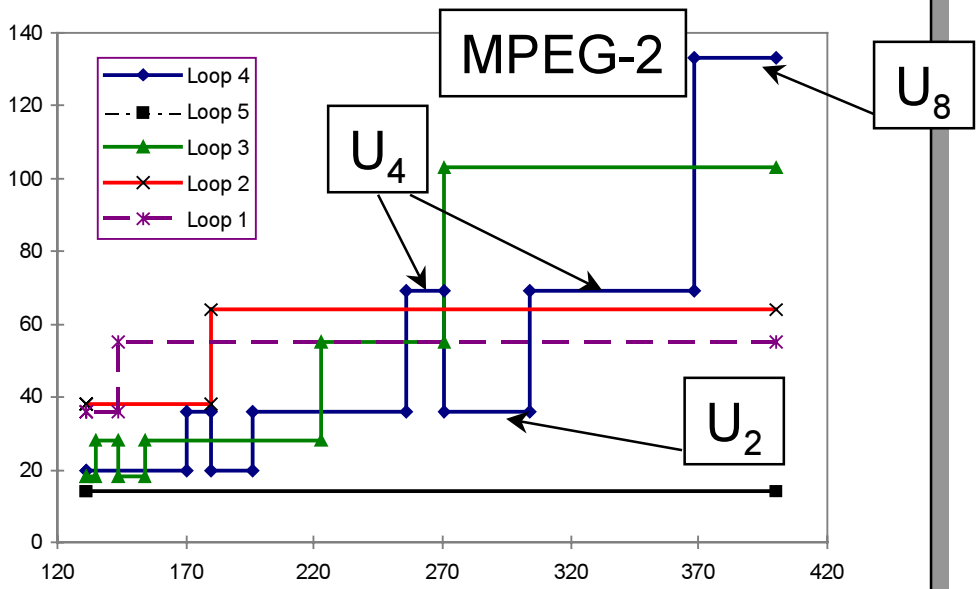
IRISA

Exemples de Résultats





- **Programmes: H-263 and MPEG-2**
  - petites boucles
  - petit nombre d ’itérations (typiquement n=8)
- **Transformations**
  - compactage
  - dépliage
  - pipeline logiciel
  - *inlining*

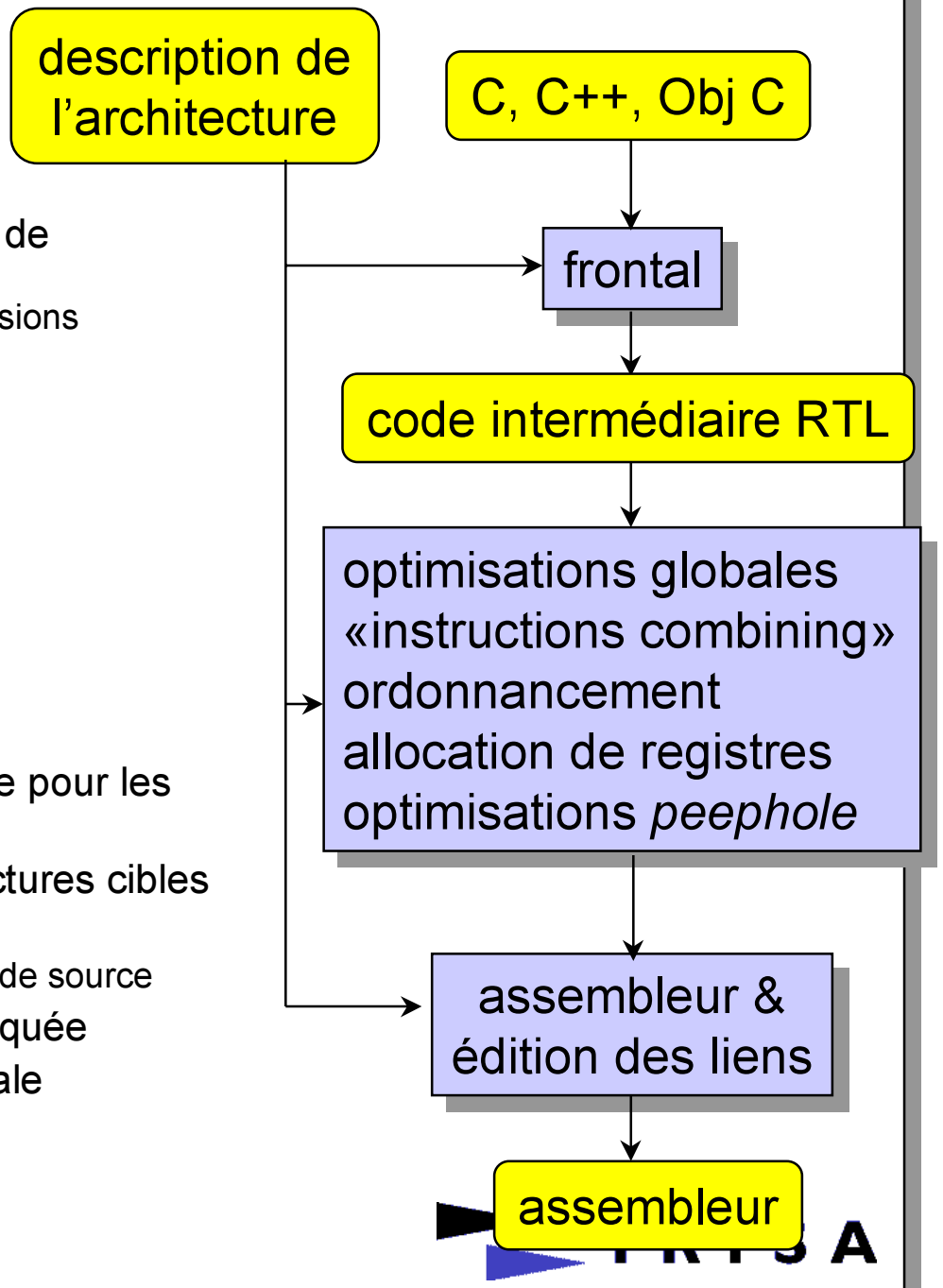


IRISA

Quelques outils

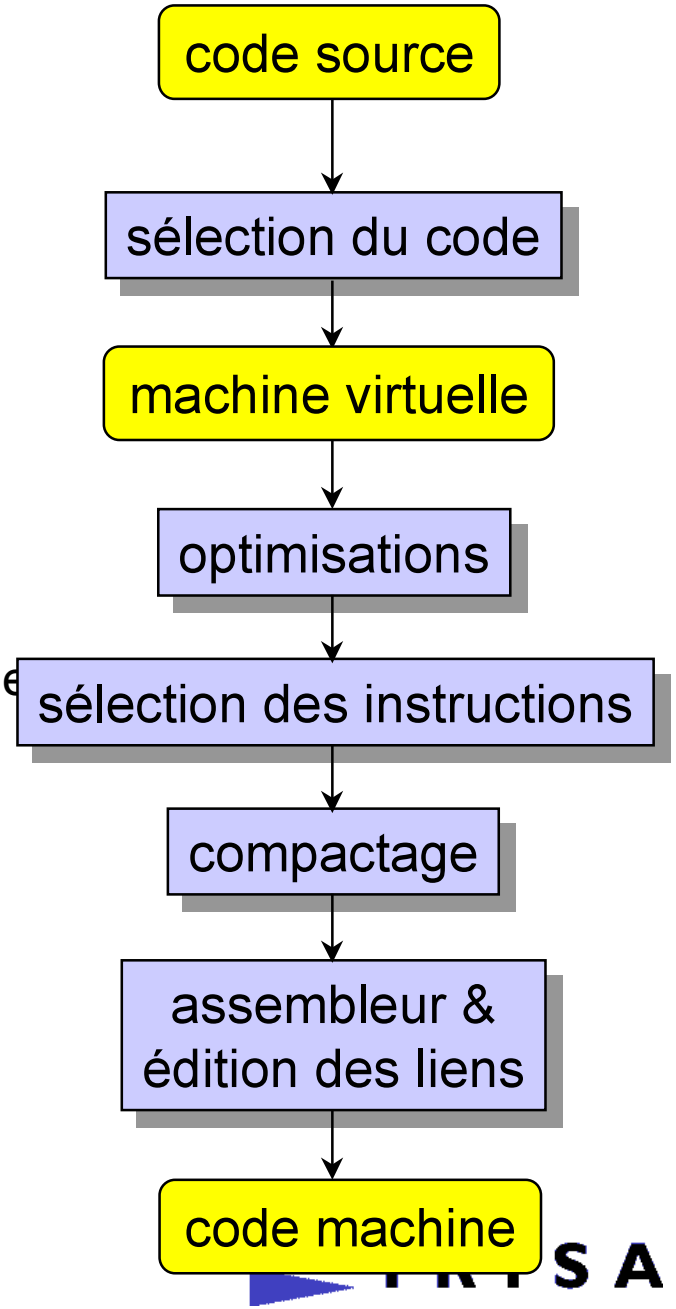
GCC

- GNU C Compiler
- Points forts
  - optimisations indépendantes de l'architecture cible
    - élimination des sous expressions communes
    - élimination de code mort
    - propagation de constantes
    - ...
  - gratuit
  - code source, diffusion
  - C et C++
- Points faibles
  - génération de code inadaptée pour les DSP?
  - optimisation pour les architectures cibles
    - gestion de l 'ILP minimale
    - pas de restructuration du code source
  - pas d'extension pour l'embarquée
  - pas d 'analyse interprocédurale
  - ...



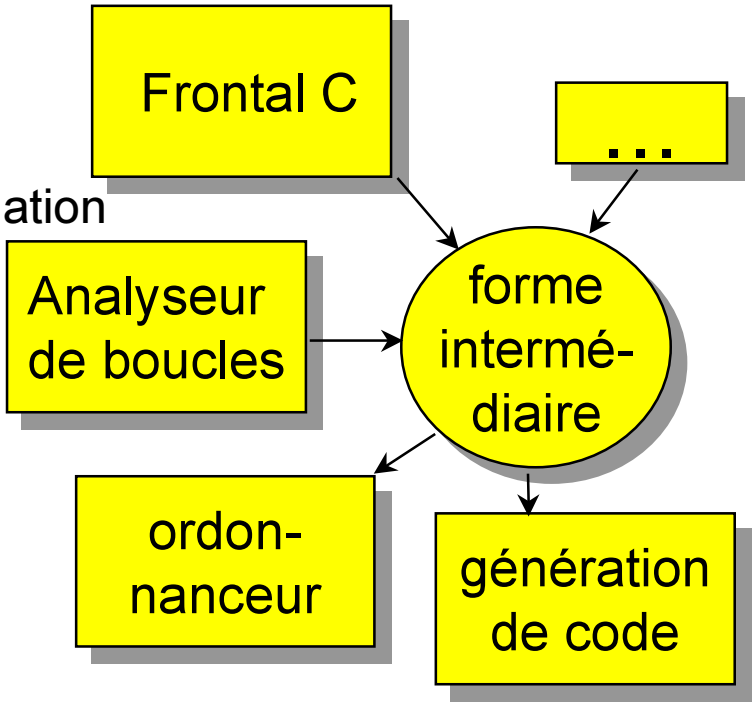
FlexCC

- FlexCC, ST Microelectronics
- Points forts
  - génération de code « rule-driven »
  - reciblable
  - extensions DSP
- Points faibles
  - optimisation pour les architectures cible
    - gestion de l ’ILP minimale
    - pas de restructuration du code source



Cosy

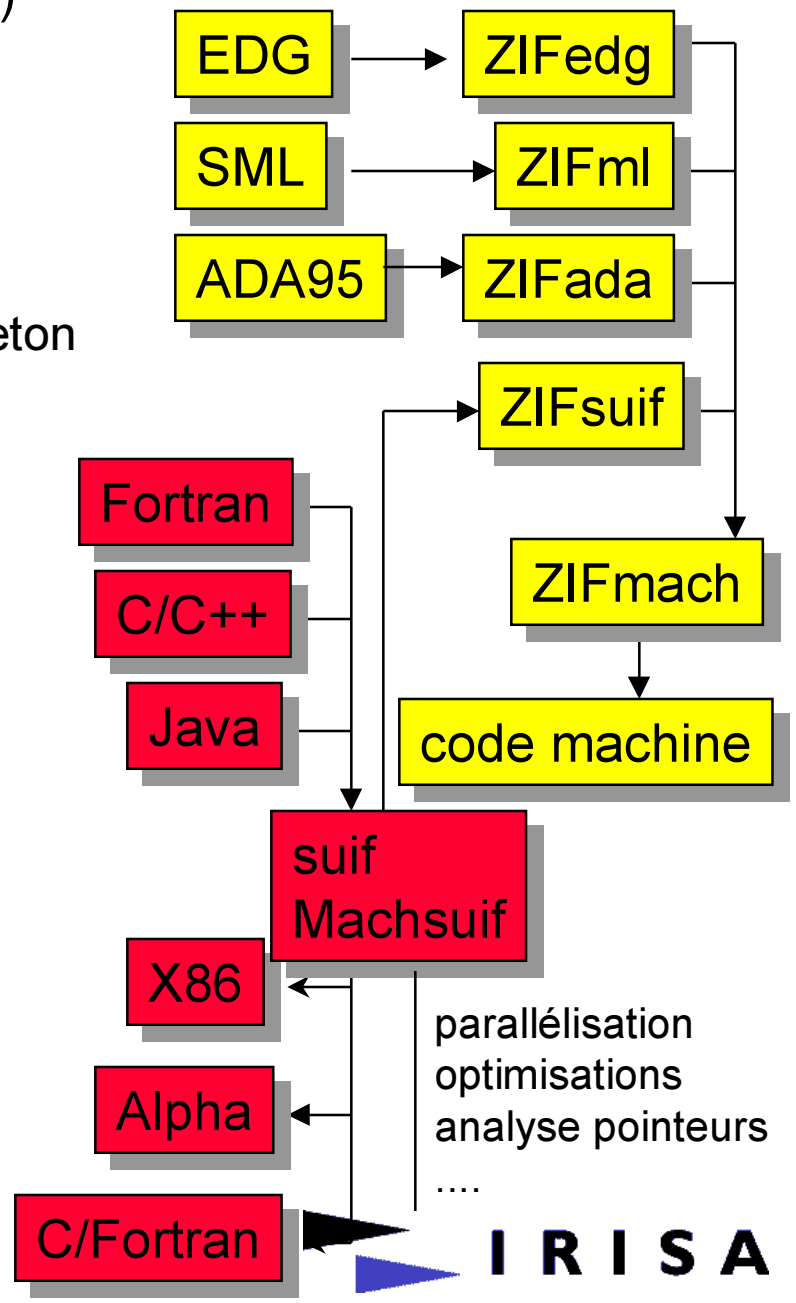
- Développé par ACE (http://www.ace.nl/)
- Résultat des projets esprit Prepare et Compare
- C, C++, DSP C, Java, etc.
- Points forts
  - reciblable,
  - organisation des phases de compilation flexibles
  - générateur de générateur de code
  - optimisations de boucle
  - DSP C (fixed-point data types, ...)
  - ...
- Points faibles
  - adaptation au DSP?
  - pas de pipeline logiciel
  - ...



IRISA

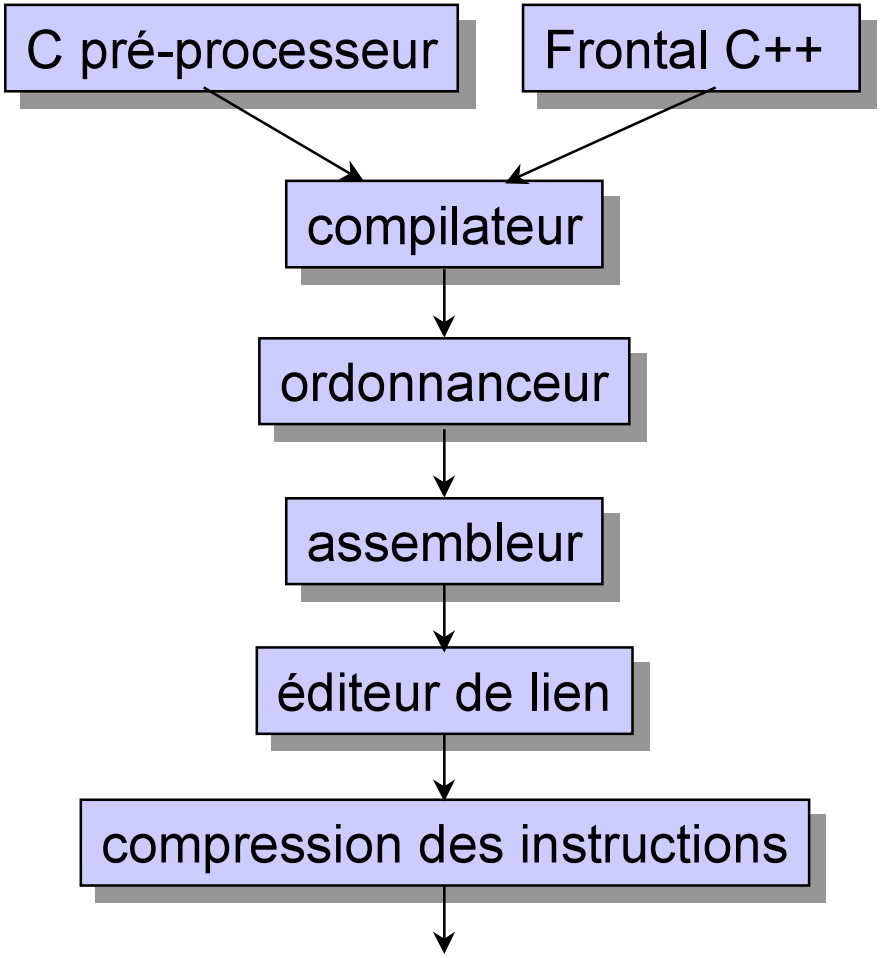
SUIF

- Suif (http://www-suif.stanford.edu/)
- A la base de la National Compiler Infrastructure US
  - infrastructure pour la recherche
  - regroupe Harvard, Rice, Stanford, UCSB, Portland Group Inc, Princeton University, Virginia University, ...
- Points forts
  - optimisations du code source
  - code source, diffusion, ...
  - parallélisation
  - effort important
  - ...
- Points faibles
  - robustesse?
  - génération de code DSP ?
  - évolution?
  - extension de type DSP C?
  - ...



Compilateur TriMedia

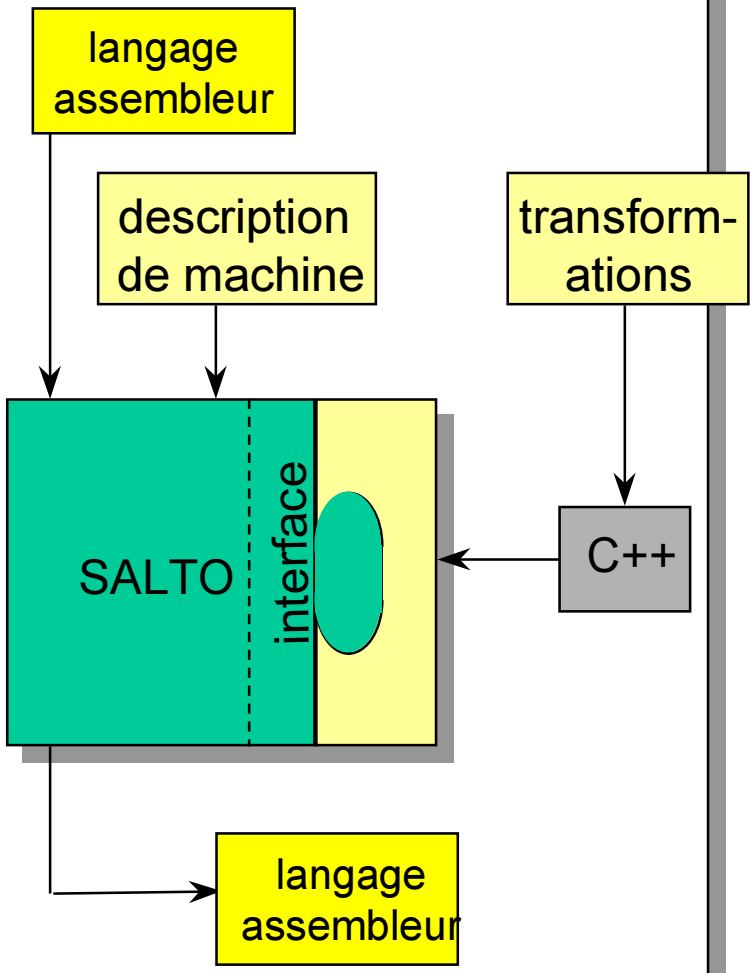
- Compilateur Trimedia C et C++
- Points forts
  - grafting / profiling
  - gestion des instructions gardées
  - dépliage de boucle
  - pointeurs restreints
  - simulateur
  - ...
- Points faibles
  - pas de pipeline logiciel
  - reciblage limité
  - pas de vectorisation
  - ...



IRISA

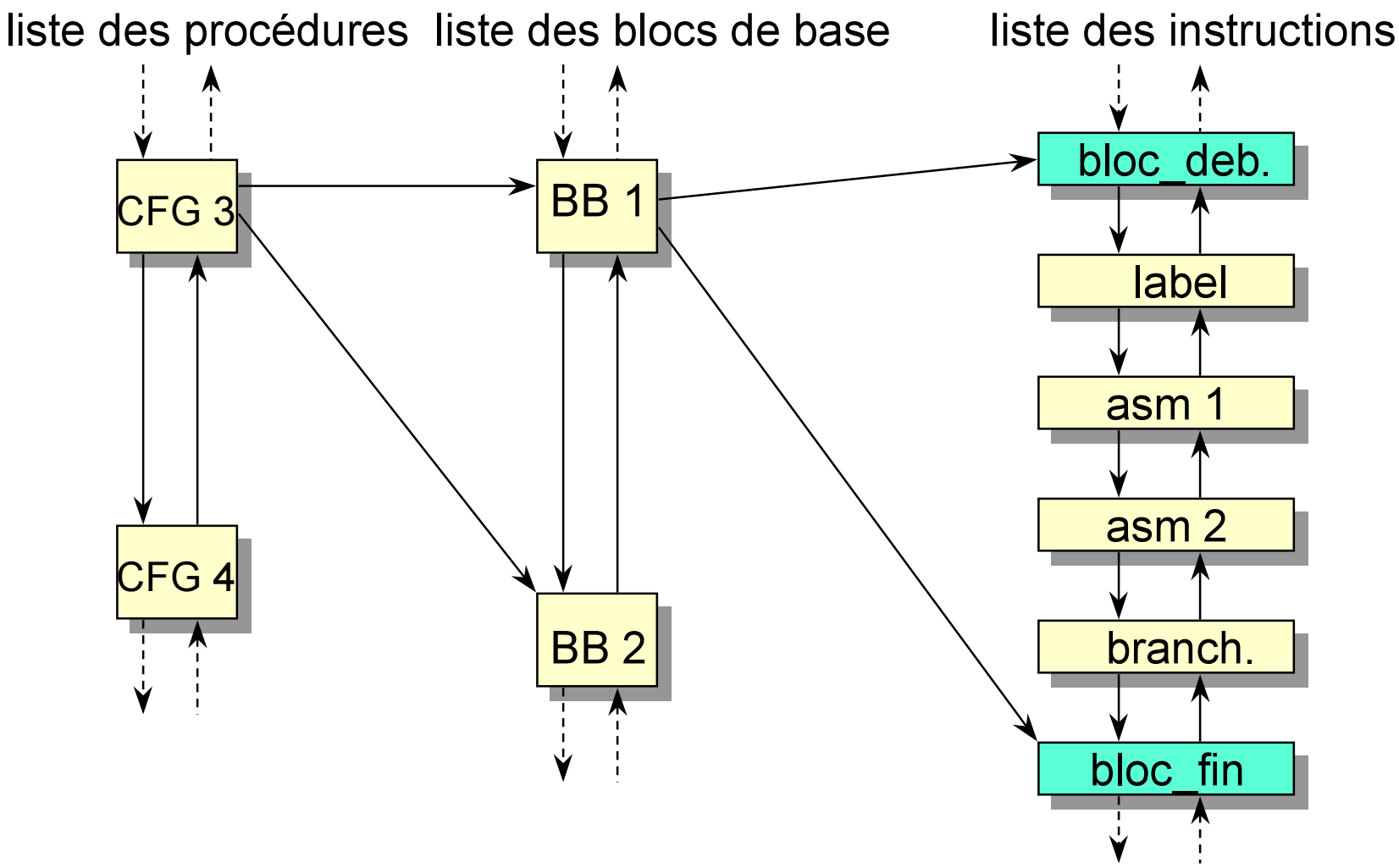
Salto

- Outils niveau assembleur
  - description fine de la machine cible
  - interface orienté objets pour l'écriture d'applications
- Cibles décrites par
  - le jeux d'instructions, format assembleur
  - utilisation des ressources du processeur pour chaque instruction, etc.
- Permet
  - l'instrumentation ou la transformation de code assembleur,
  - la mise en œuvre de techniques d'ordonnancement,
  - la mise en œuvre d'allocations de registres, etc.
- Existe pour Sparc, Alpha, Mips, Tri-Media, TMSC62X, etc.
- Partiellement soutenu par le projet LTR Esprit OCEANS



IRISA

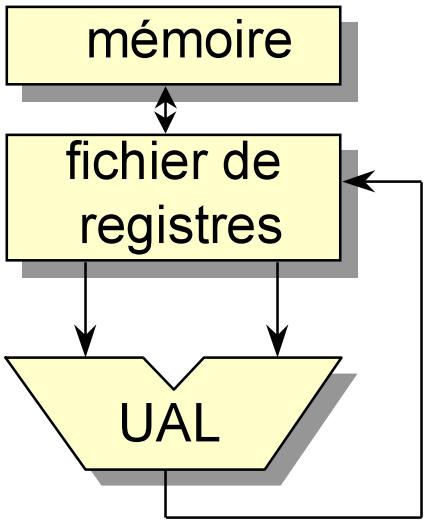
Abstraction du code assembleur



IRISA

Description de l'architecture

- Description des unités fonctionnelles
- Description de l'assembleur
- Tables de réservation




```
_main:
        !#PROLOGUE# 0
        save %sp,-104,%sp
        !#PROLOGUE# 1
        call ___main,0
        nop
        cmp %i0,1
        bg L2
        sethi %hi(__iob+40),%o0
        or %o0,%lo(__iob+40),%o0
        sethi %hi(LC0),%o1
        or %o1,%lo(LC0),%o1
        call _fprintf,0
```

cycles	0	1	2	3
lanc.	util.			
reg 1		lect.		
reg 2		lect.		
UAL			util.	
reg 3				écrit.

IRISA

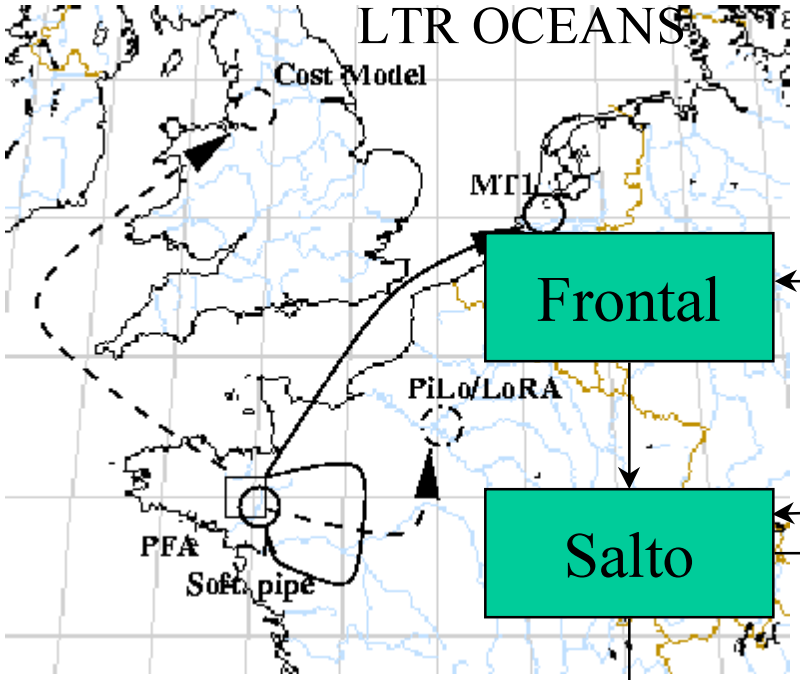
Autres

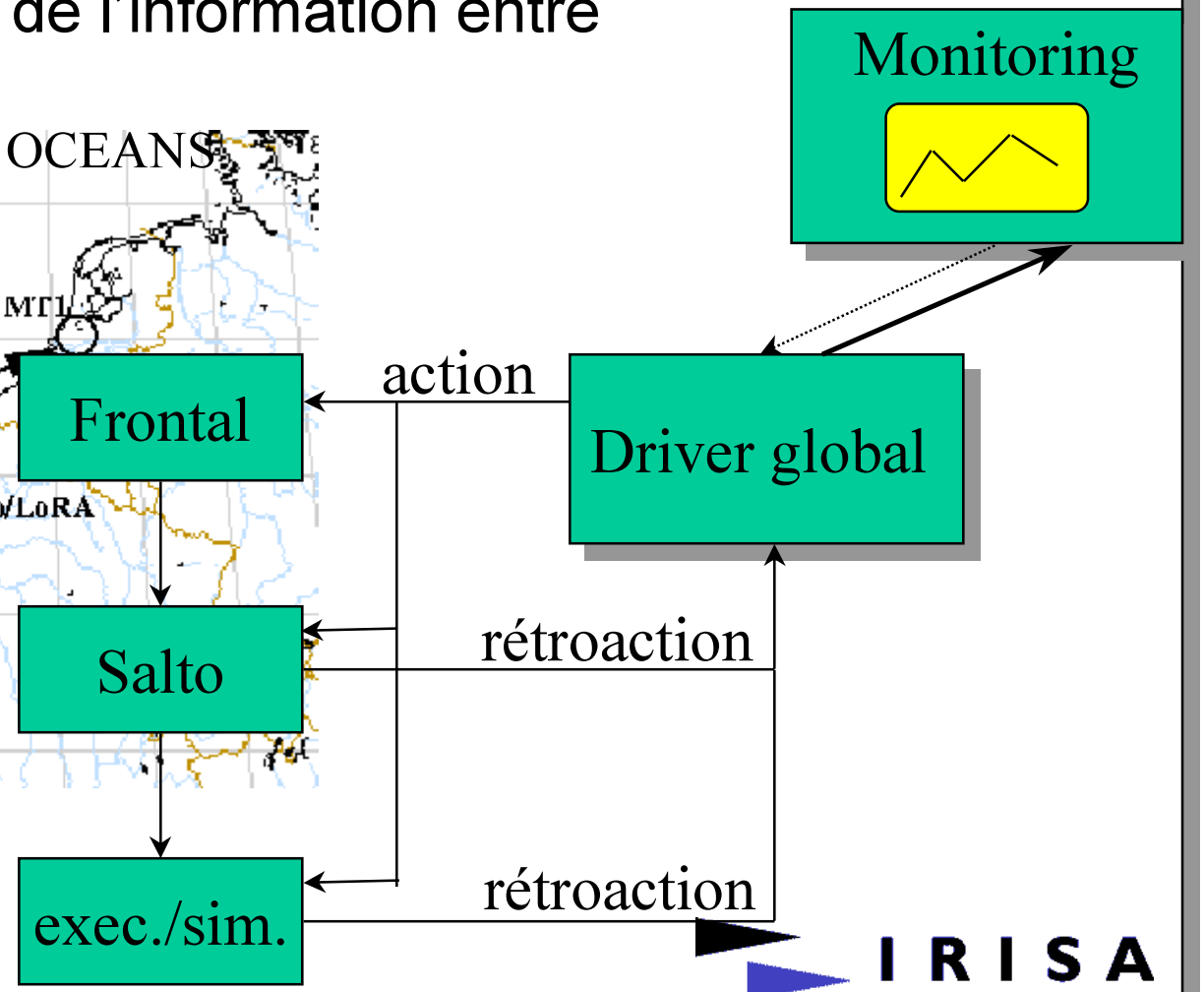
- ISDL (http://caa.lcs.mit.edu/caa/)
- SPAM (http://www.ee.princeton.edu/spam)
- ZEPHYR (http://www.cs.virginia.edu/zephyr/)
- IMPACT (http://www.crhc.uiuc.edu/Impact/)
- Trimaran (http://www.trimaran.org/)
- Saxo (CNET)
- Edinburgh Portable Compilers (http://www.epc.co.uk/)
- Acropolis (http://www.imec.be/vsdm/projects/mm_comp/)
- ...

Quelques perspectives

Schéma itératifs de compilation

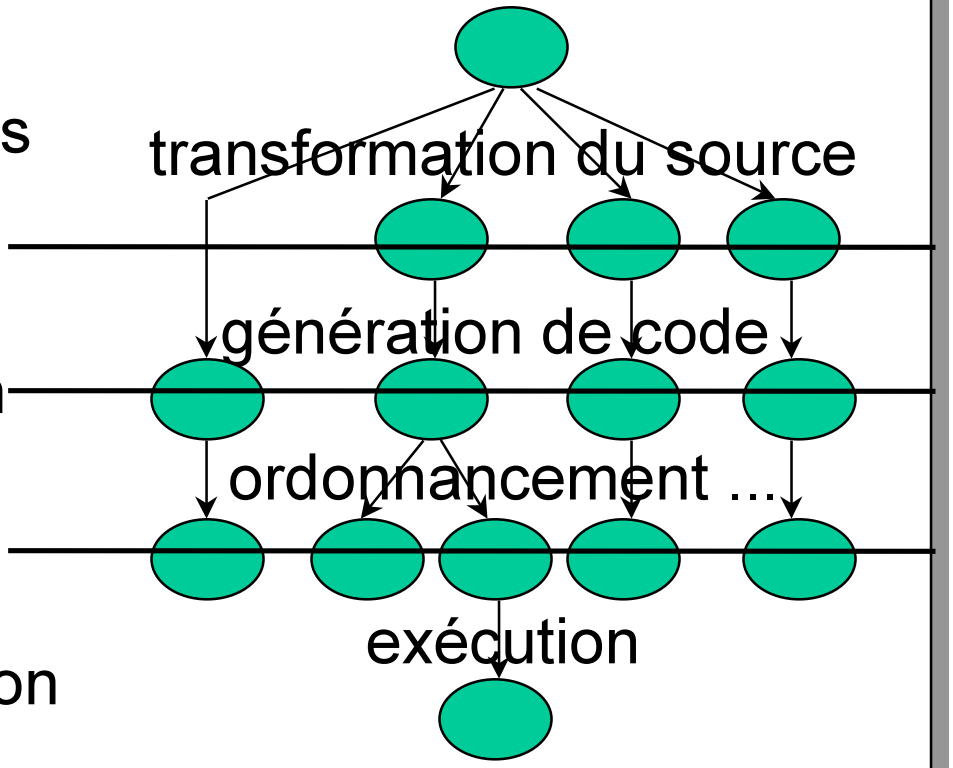
- Interaction haut-bas niveaux
- Exploration de l'espace des optimisations
- Infrastructure de compilation
- Conservation de l'information entre phases





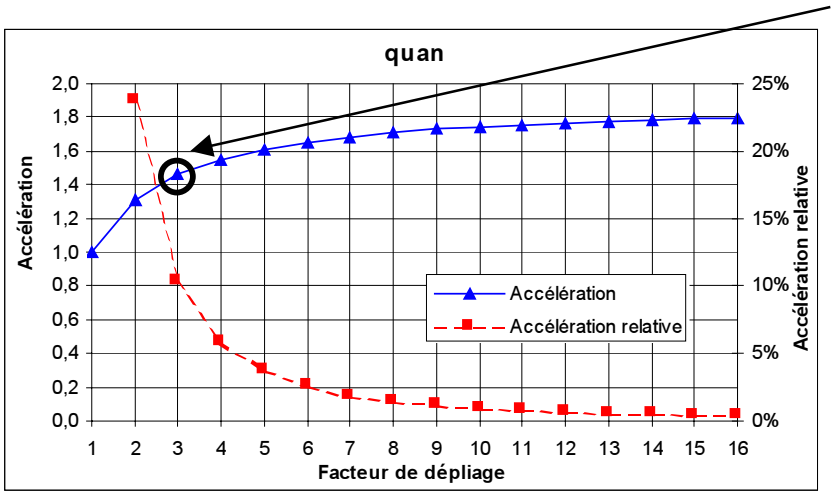
IRISA

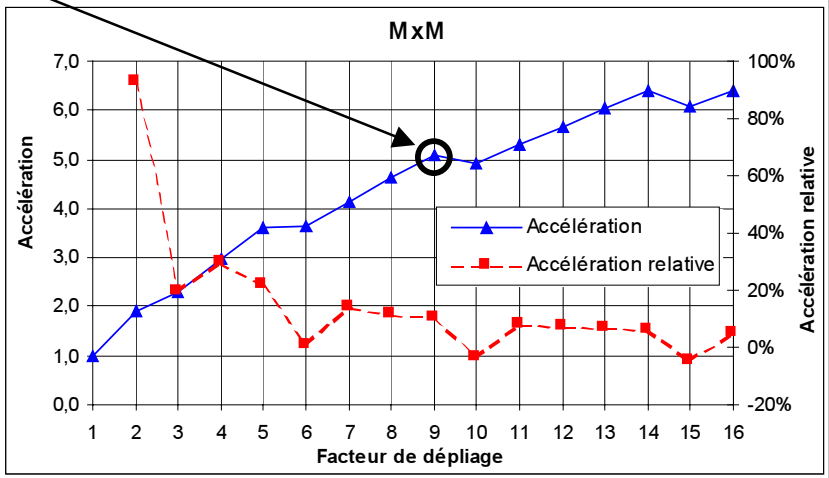
- Exploration
  - mémorise les solutions
  - tous les niveaux du compilateur contrôlés
  - création dynamique des alternatives
  - estimation/exécution
- Evaluation locale
  - chaque transformation s'évalue



Dépliage de boucle

- Exemple simple: contrôle du dépliage de boucles dans le code source
  - mesure statique de la vitesse de la boucle
  - cesser quand le gain n'atteint pas 10%
  - éviter les pertes de place (code)





IRISA

Conclusion

- Domaine en évolution rapide
- Technologie de compilation critique
- Capacité de reciblage essentielle
- Nécessité d'optimisations « *état de l'art* »
- Contraintes non standards: taille de code, consommation électrique, ...
- Multiprocesseurs
- Intégration dans une chaîne de conception conjointe