Processeurs embarqués spécifiques et production de code

François Charot
Irisa/Inria
Rennes

Plan de la présentation

- Quelques aspects concernant la production de logiciel pour les systèmes enfouis
- Compilateurs pour processeurs embarqués
- Compilateurs reciblables
- Approche ARMOR/CALIFE
- Conclusion

Plan de la présentation

- **Quelques aspects concernant la production de logiciel pour les systèmes enfouis**

  Types de processeurs

  codesign matériel/logiciel

  outils de spécification de systèmes
- **Compilateurs pour processeurs embarqués**
- **Compilateurs reciblables**
- **Approche ARMOR/CALIFE**
- **Conclusion**

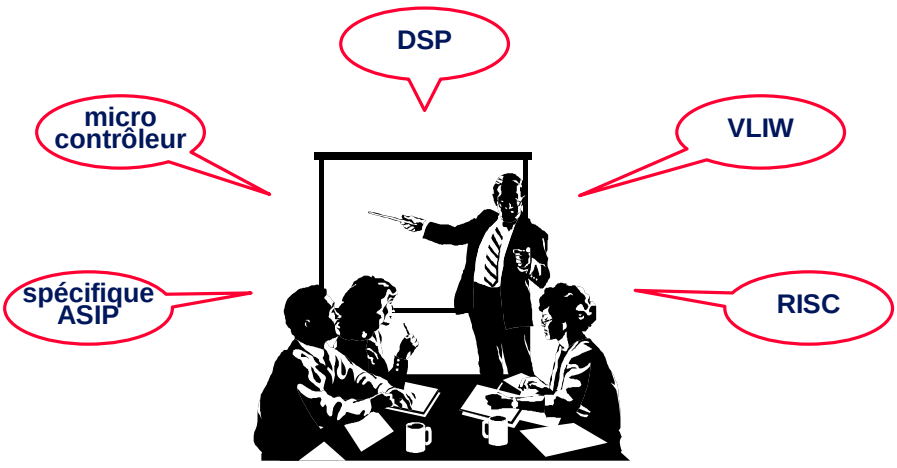
Logiciels et processeurs embarqués

- **Révolution technique des applications grand public du type multimédia, télécoms**
  - **Technologies d'intégration**
  - **Outils de conception**
- ☛ **système sur silicium, système mono-puce**
- **La loi du *time to market* impose réaction rapide et anticipation stratégique**
- ☛ **emploi du logiciel et intégration de cœurs de processeurs (temps de conception raisonnable)**
  - **flexibilité en terme de conception**
  - **flexibilité en terme d'évolution**

Caractéristiques d'un processeur embarqué

- Cœur de processeur
  - partie centrale d'un processeur (plus périphériques)
- Contrainte de performance
  - notion de temps réel
- Problème de la densité de code : taille du code applicatif (mémoire)
  - la mémoire tend à occuper la majeure partie d'une puce
  - impact de la taille du cœur dans le choix du processeur

Styles architecturaux des processeurs embarqués



Caractéristiques d'un logiciel embarqué

- Taille réduite de programme
  - quelques centaines, voire milliers de lignes de code C
- Gestion simplifiée de la mémoire
  - absence d'allocation dynamique
- Gestion simplifiée des entrées/sorties (flot de donnée)
- Exécution cyclique rythmée par le flot de données
- Contraintes temporelles fortes

☛ Ces caractéristiques influent sur la compilation

Caractéristiques d'un logiciel embarqué (suite)

- Historiquement, emploi de l'assembleur
  - notamment pour les parties critiques
- Aujourd 'hui, des langages de haut niveau comme C, C++ s 'imposent
  - augmentation de complexité
  - portabilité
  - progrès de la technologie de compilation
  - évolution des architectures

Caractéristiques d'un logiciel embarqué (suite)

- Impact du style d'écriture du code sur les performances
  - niveau comportemental (norme C ANSI)
    usage de constructions de haut niveau (tableaux, structures)
  - second niveau
    utilisation de pointeurs, de fonctions prédéfinies
  - troisième niveau
    assignation manuelle de variables à des registres de l'architecture cible (directives)
  - quatrième niveau
    mélange de C bas niveau avec directives et d'assembleur

Lisibilité portabilité ↑

efficacité ↓

Compilation de programmes embarqués

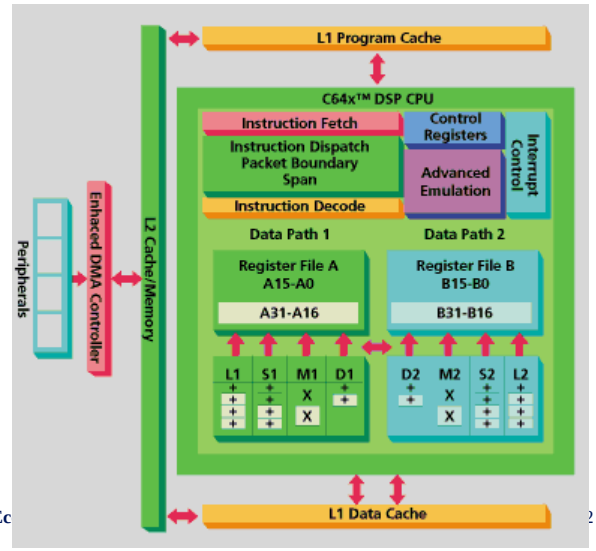
- Rôle essentiel du compilateur
  - générer rapidement des programmes efficaces
  - importance de l'optimisation de code
    - taille de code
    - vitesse d'exécution
    - puissance ou énergie consommée
- Automatiser la conception du compilateur
- ☛ Compilation reciblable

Interdépendance entre architecture cible, compilateur, application

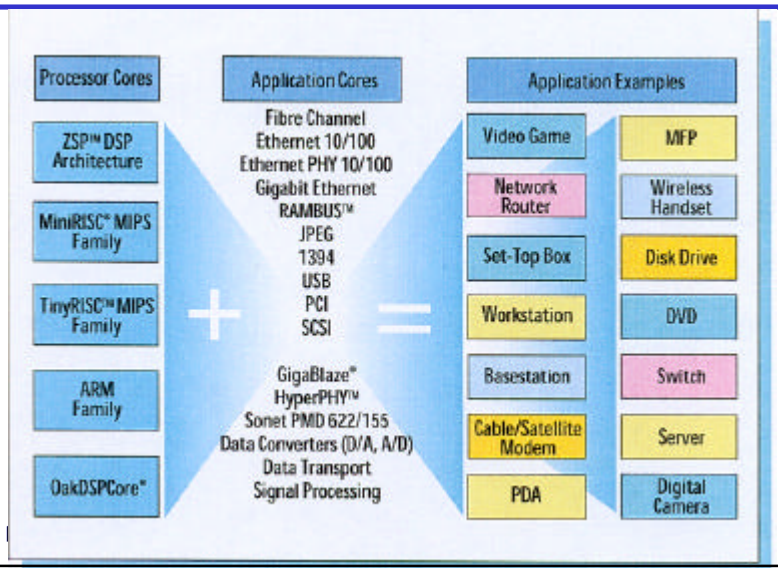
- Choix d'un cœur existant et son compilateur
  - écriture de l'application
- Cœur de processeur fixé
  - développement du compilateur, conjointement à l'écriture de l'application
- Conception conjointe processeur-compilateur en regard des besoins de l'application
  - ASIP : cible idéale pour la compilation ayant des caractéristiques pour une exécution optimisée

Processeurs embarqués : cœur TMS320C64x de Texas Instruments

1,1 GHz
8800 MIPS
4400 MAC (16 bits)



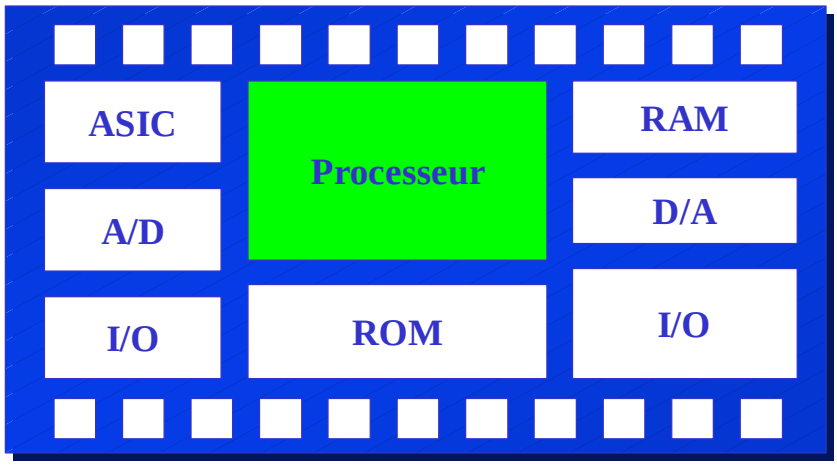
Processeurs embarqués : programme CoreWare de LSI Logic's



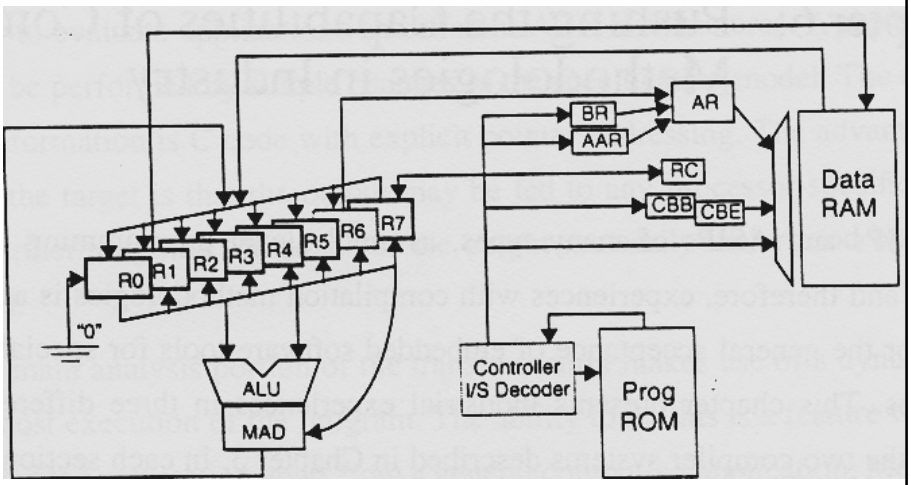
Processeurs embarqués : cœurs de DSP/VLIW

Compagnie	*Famille*	*Donnée*	*Instruction*	*Horloge*
ARM	*Piccolo*	*16 bits*	*16/32 bits*	*70 MHz*
DSP Group	*PineDSPCore*	*16 bits*	*16 bits*	*40 MHz*
	OakDSPCore	*16 bits*	*16 bits*	*80 MHz*
	PalmDSPCore	*16/20/24bits*	*16/32 bits*	*150 MHz*
IBM	*C54XDSP*	*16 bits*	*16/32 bits*	*66 MHz*
Mentor Graphics	*M320C50*	*16 bits*	*16 bits*	
Texas	*C27xx*	*16 bits*	*16 bits*	*50 MHz*
	C54x	*16 bits*	*16 bits*	*100 MHz*
StarCore	*140 (vliw)*	*16 bits*	*16 bits*	*300 MHz*
Siemens	*Tricore*	*32 bits*	*16/32 bits*	*80 MHz*
	Carmel (vliw)	*16 bits*	*24/48 bits*	*120 MHz*

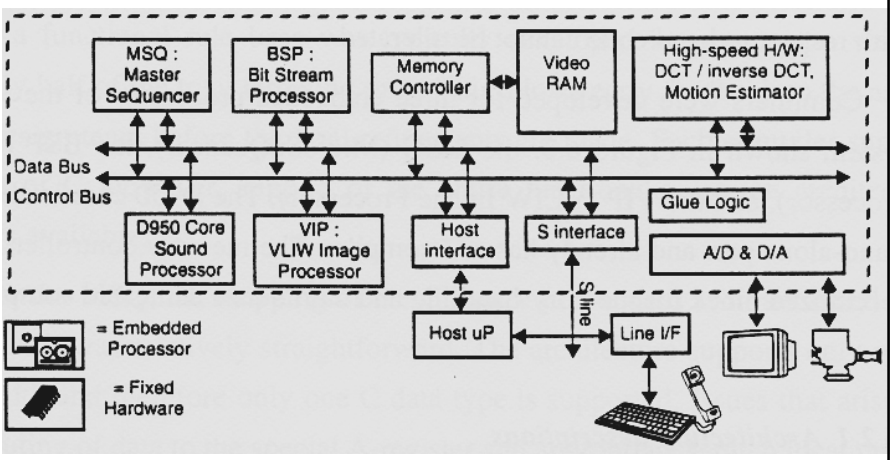
Cœur de processeur : utilisation dans des systèmes sur silicium



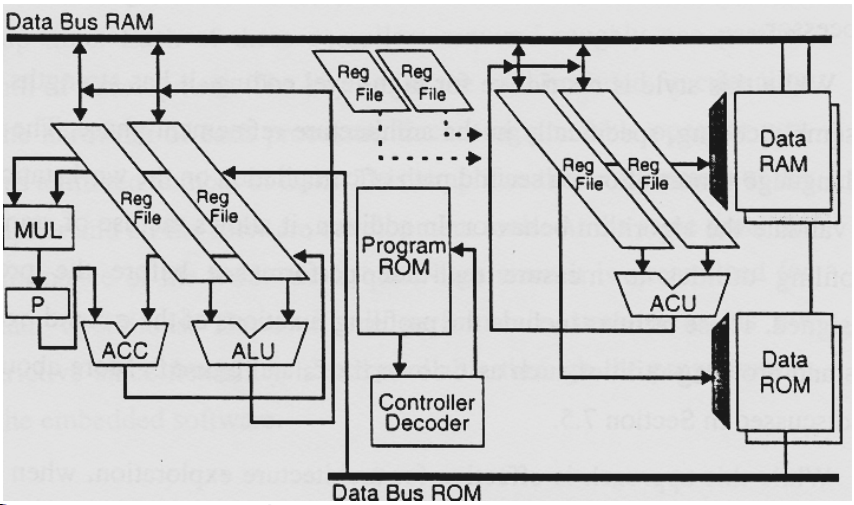
ASIP (Nortel) commutateur de téléphone local privé



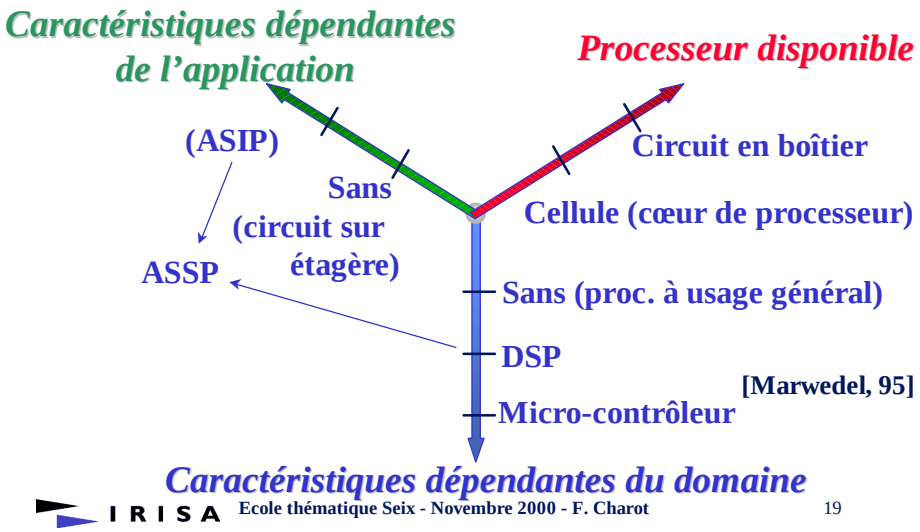
ASIP (STMicroelectronics)
visiophonie sur ligne téléphonique



ASIP (TCEC)
traitement audio haute fidélité



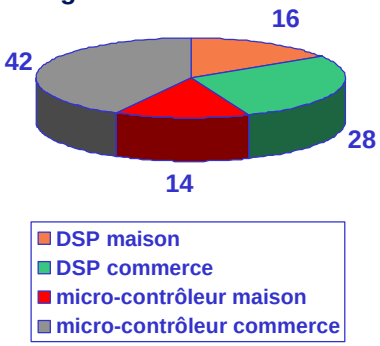
Classification des processeurs



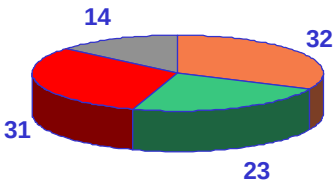
[Marwedel, 95]

Tendances en matière d'ASIP

Usage



Volume



[étude à Northern Telecom, Paulin, 1995]

Plan de la présentation

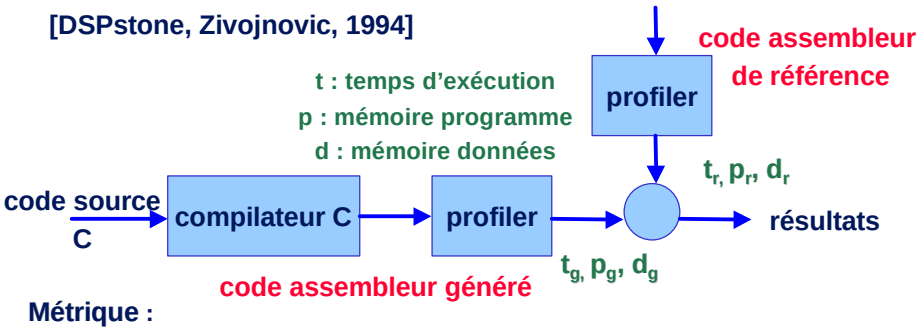
- Quelques aspects concernant la production de logiciel pour les systèmes enfouis
- **Compilateurs pour processeurs embarqués**

  Performances des compilateurs pour processeurs embarqués

  Nouvelles techniques d'optimisation
- Compilateurs reciblables
- Approche ARMOR/CALIFE
- Conclusion

Performance des compilateurs pour processeurs embarqués

[DSPstone, Zivojnovic, 1994]



Métrique :

$$\text{surcoût temps d'exécution : } \Delta t_g = (t_g - t_r) / t_r[\%]$$

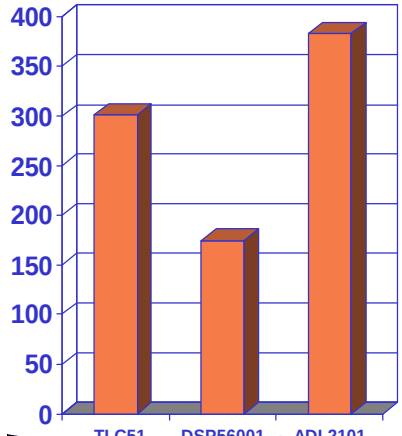
$$\text{surcoût mém. programme : } \Delta p_g = (p_g - p_r) / p_r[\%]$$

$$\text{surcoût mém. données : } \Delta d_g = (d_g - d_r) / d_r[\%]$$

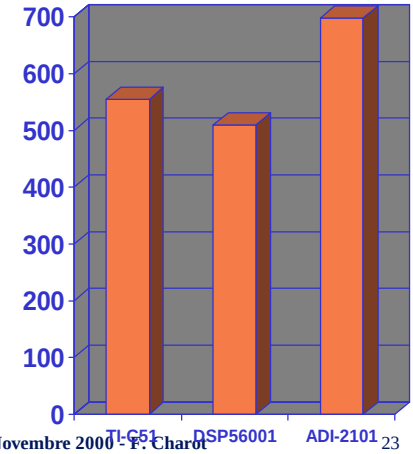
Performance des compilateurs pour processeurs embarqués

Exemple : ADPCM (codage parole)

Surcoût mémoire données (%)

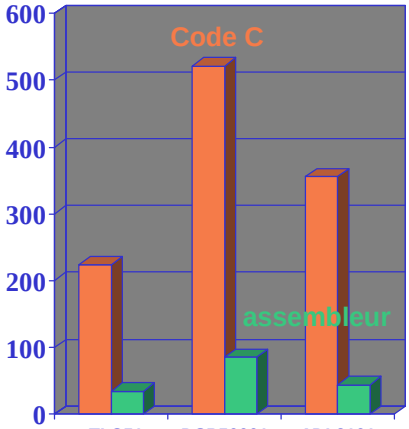


Surcoût vitesse (%)



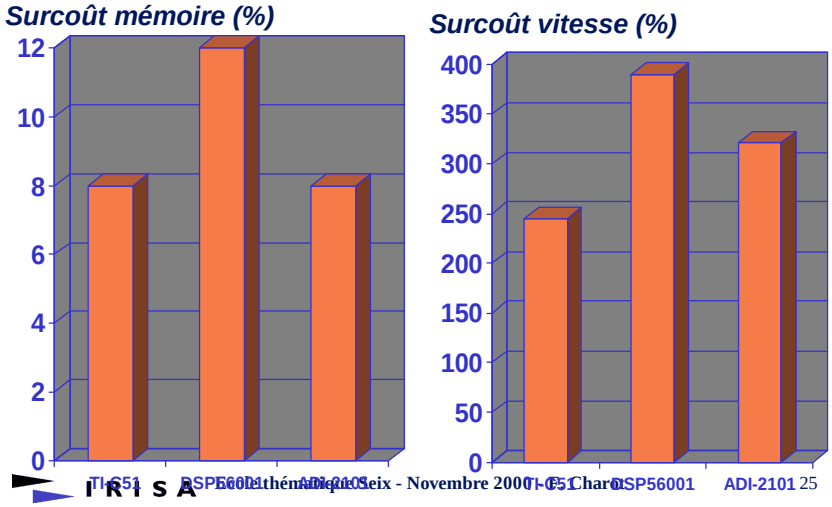
Performance des compilateurs pour processeurs embarqués

Performance absolue en *ms* pour l 'application ADPCM

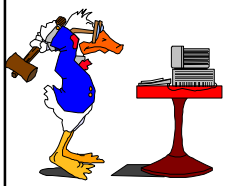


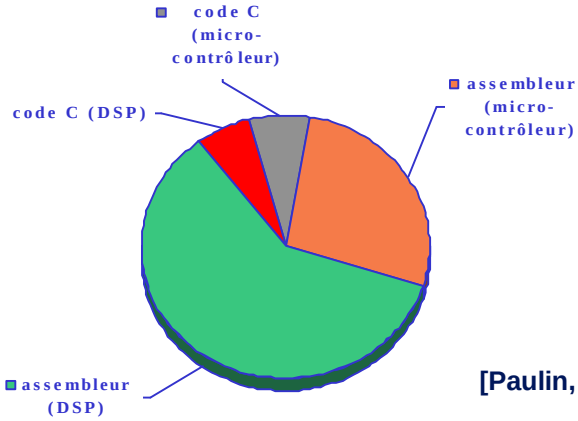
Performance des compilateurs pour processeurs embarqués

Exemple : multiplication de matrices (10x10)



Utilisation des langages d'assemblage dans les systèmes embarqués



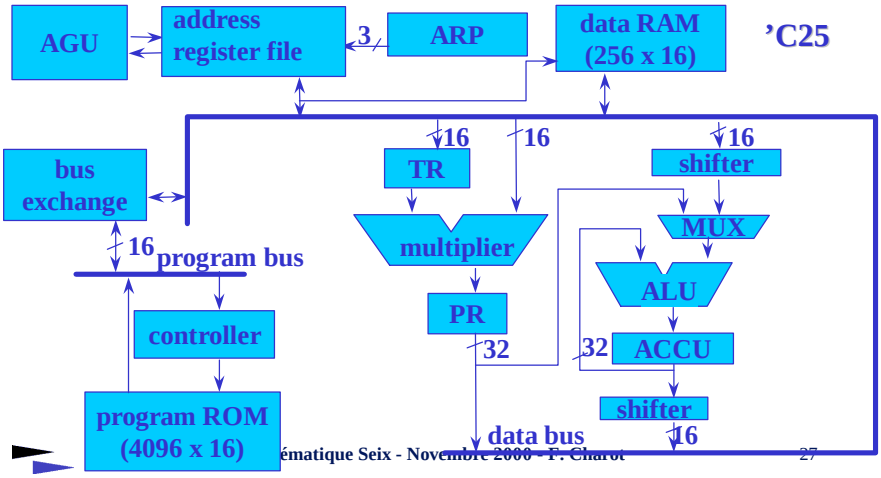


[Paulin, 1995]

proportions relatives de lignes d'assembleur et de code C

Raisons pour l'inefficacité des compilateurs

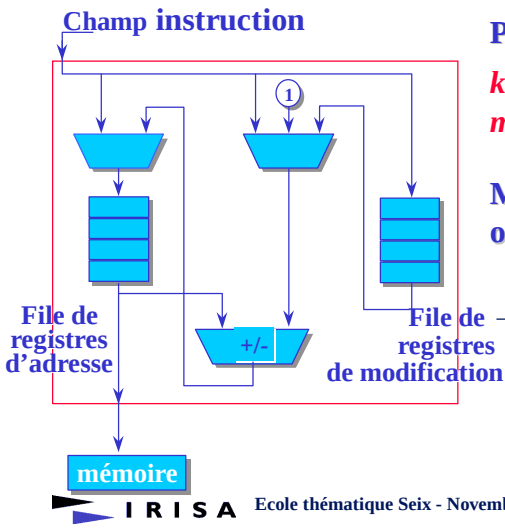
L'efficacité du matériel est privilégiée lors de la conception



Travaux récents concernant les techniques d'optimisation

- assignation de registres pour des bancs de registres hétérogènes [Araujo95,Wess95]
- optimisation des registres de mode [Liao96]
- intégration du dépliage de boucle et de l'allocation de registres [Nicolau]
- allocation en présence de plusieurs bancs mémoire [Sudarsanam95]
- compaction de code [Leupers95, Leupers99]
- assignation d'adresse pour une utilisation optimisée des unités de génération d'adresse [Liao95,Leupers96, Sudarsanam97,Basu99]
- sélection de code à partir de graphes [Leupers99]
- ...

Exploitation des calculs d'adresse



Paramètres:

k = # registres d'adresse
m = # registres de modification

Métrique de coût pour les operations UGA:

operation	coût
immediate AR load	1
immediate AR modify	1
auto-increment/decrement	0
.....	

Conséquence d'un arrangement optimisé de la mémoire

Variables dans un bloc de base:

V = {a, b, c, d}

Séquence d'accès:

S = (b, d, a, c, d, c)

0	a
1	b
2	c
3	d

```
Load AR,1 ;b
AR += 2   ;d
AR -= 3   ;a
AR += 2   ;c
AR ++     ;d
AR --     ;c
```

cost: 4

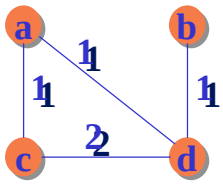
0	b
1	d
2	c
3	a

```
Load AR,0 ;b
AR ++     ;d
AR +=2    ;a
AR --     ;c
AR --     ;d
AR ++     ;c
```

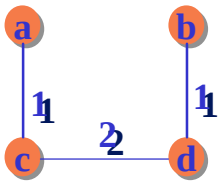
cost: 2

Assignation d'adresse pour un unique registre d'adresse

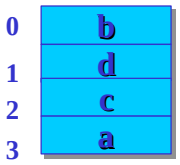
Séquence d'accès : b d a c d c



Graphe d'accès



Chemin de poids maximum



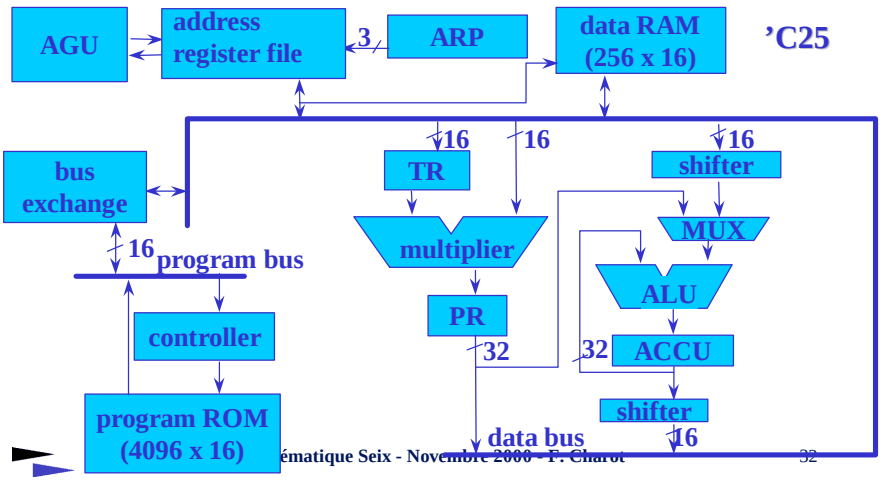
mémoire

Recherche du chemin (heuristique)

[Bartley, 1992, Liao, 1995]

Exploitation du parallélisme de niveau instruction

Transfert vers plusieurs registres possible dans la même instruction



Exploitation du parallélisme de niveau instruction

u(n) = u(n - 1) + K0 × e(n) + K1 × e(n - 1);
e(n - 1)= e(n)

De 9 à 7 cycles après compaction

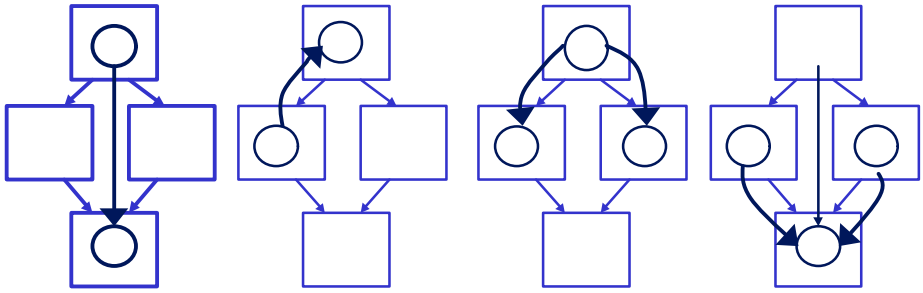
```
ACCU  := u(n - 1)
TR    := e(n - 1)
PR    := TR × K1
TR    := e(n)
e(n - 1)  := e(n)
ACCU  := ACCU + PR
PR    := TR × K0
ACCU  := ACCU + PR
u(n)  := ACCU
```

```
ACCU:= u(n - 1)
TR    := e(n - 1)
PR    := TR × K1
e(n - 1):= e(n) || ACCU:= ACCU + PR
                || TR:= e(n)
PR    := TR × K0
ACCU:= ACCU + PR
u(n)  := ACCU
```


Exploitation du parallélisme de niveau instruction

- Ordonnancement local pour les blocs de base
  - ordonnancement par liste
  - programmation linéaire en nombres entiers
- Ordonnancement global pour exploiter le parallélisme de l'architecture
  - ordonnancement de traces (VLIW) [Fisher,85]
    - trace : séquence de blocs de base (sur la base de probabilité d 'exécution de branchement conditionnel)
  - ordonnancement de super-bloc [Hwu,93]
    - super-bloc : région de programme à un point d 'entrée

Ordonnancement global : mouvement de code en présence de conditionnelle

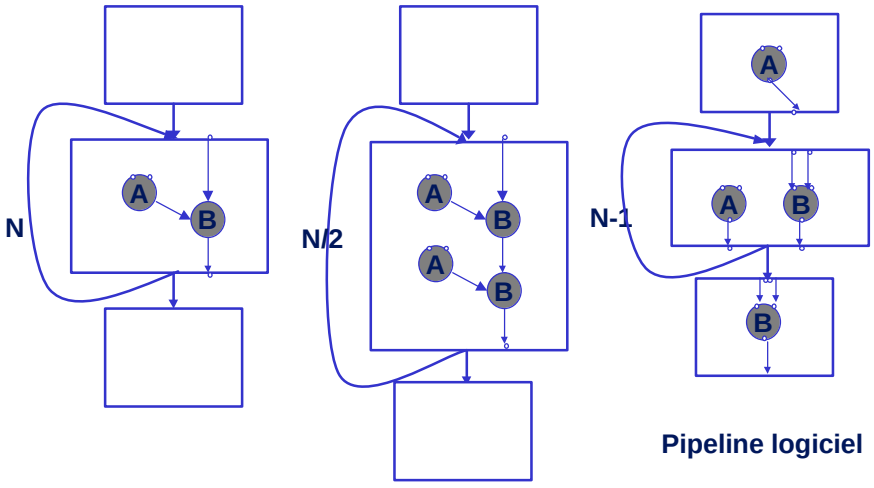


profitable | spéculatif | Copie (bas, haut) | Remontée (bas, haut)

instruction avant mouvement

instruction après mouvement

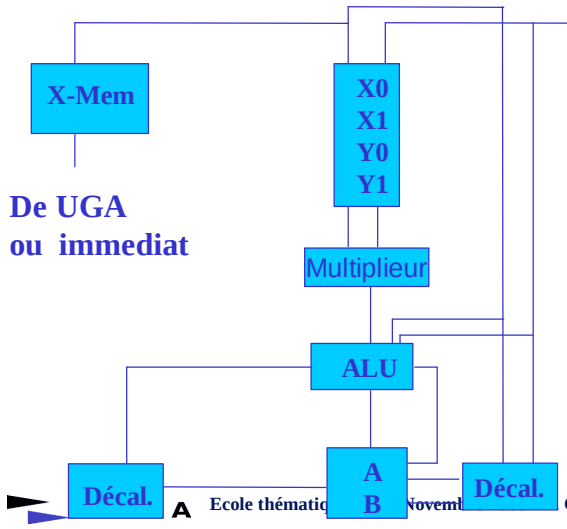
Optimisation globale : les boucles



Dépliage

Pipeline logiciel

Bancs mémoire multiples



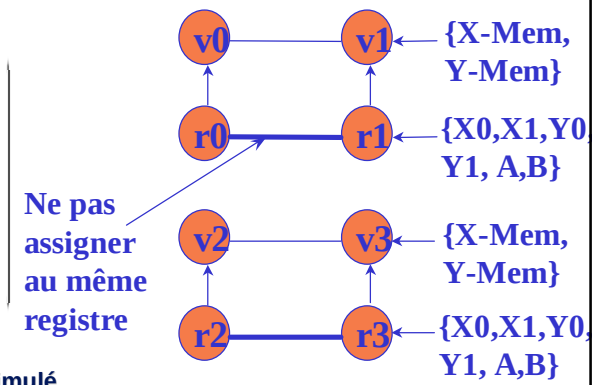
Mouvement de donnée en parallèle possible si différents bancs mémoire sont utilisés

Bancs mémoire multiples

Code Précompacté (variables symboliques et registres)

move v0,r0 v1,r1
move v2,r2 v3,r3

Etiquetage d'un graphe de contraintes

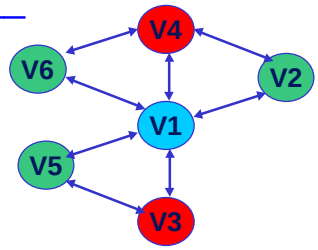


utilisation du recuit simulé

[Sundarnanam, 95]

Allocation de registres par coloriage d'un graphe d'interférence

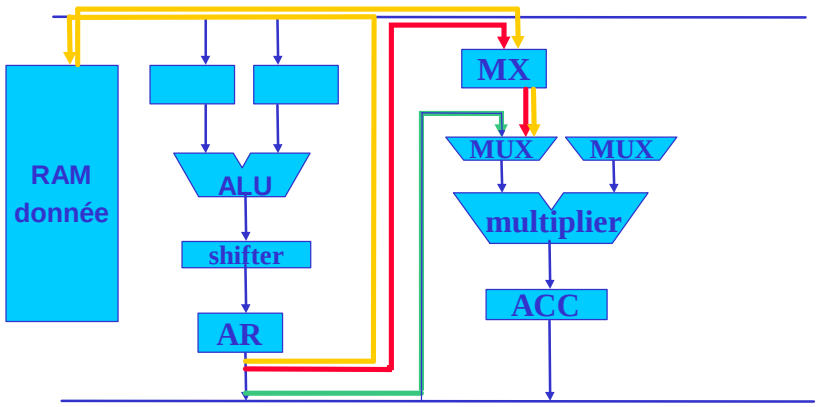
valeur	durée de vie
V1	
V2	
V3	
V4	
V5	
V6	



N registres
trouver un coloriage utilisant N couleurs

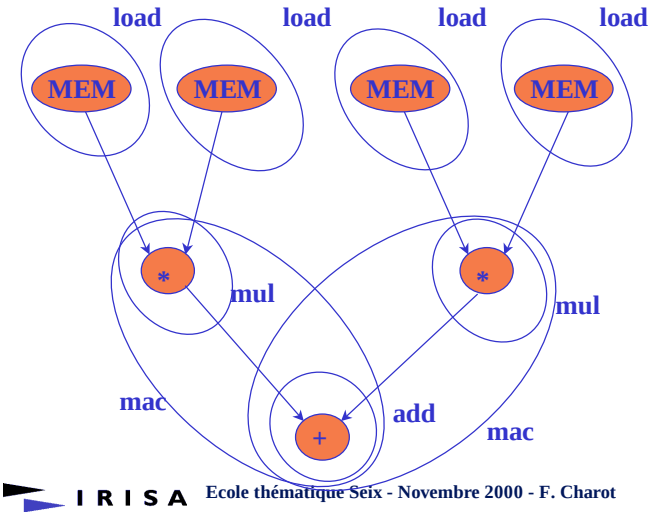
Technique du routage de données

Calcul d'une solution à partir des routes possibles



donnée de AR utilisée par le multiplieur

Etape clé dans tout compilateur : la sélection de code



Sélection de code par analyse d'arbre (1)

Spécification d'un grammaire pour générer un analyseur iburg* :

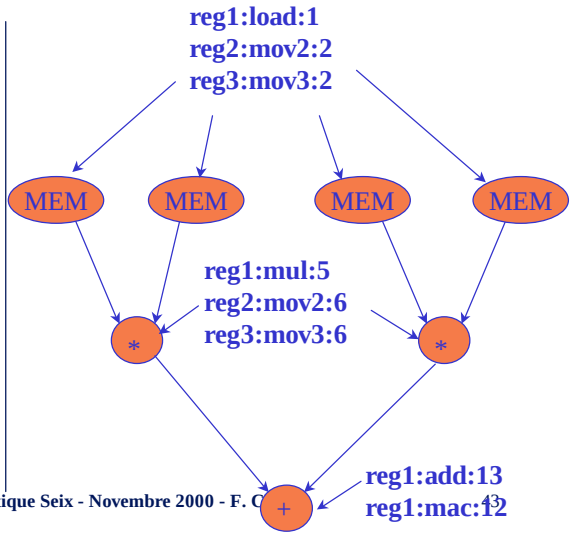
terminals: {MEM, *, +}
non-terminals: {reg1,reg2,reg3}
start symbol: reg1
rules:
"add" (cost=2): reg1 -> + (reg1, reg2)
"mul" (cost=2): reg1 -> * (reg1,reg2)
"mac" (cost=3): reg1 -> + (*(reg1,reg2), reg3)
"load" (cost=1): reg1 -> MEM
"mov2"(cost=1): reg2 -> reg1
"mov3"(cost=1): reg3 -> reg1

* C.W. Fraser et al., 1992

Sélection de code par analyse d'arbre (2)

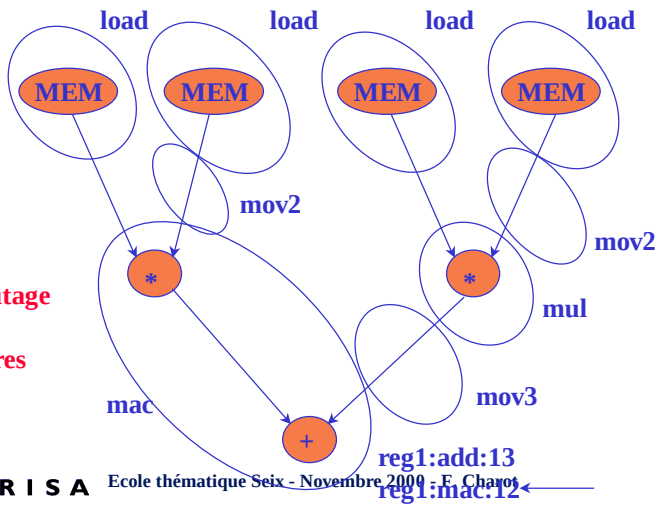
"load"(cost=1):
reg1 -> MEM
"mov2"(cost=1):
reg2 -> reg1
"mov3"(cost=1):
reg3 -> reg1

"add" (cost=2):
reg1 -> +(reg1, reg2)
"mul" (cost=2):
reg1 -> *(reg1,reg2)
"mac" (cost=3):
reg1->+(*(reg1,reg2), reg3)



Sélection de code par analyse d'arbre (3)

Inclut le routage de données entre registres



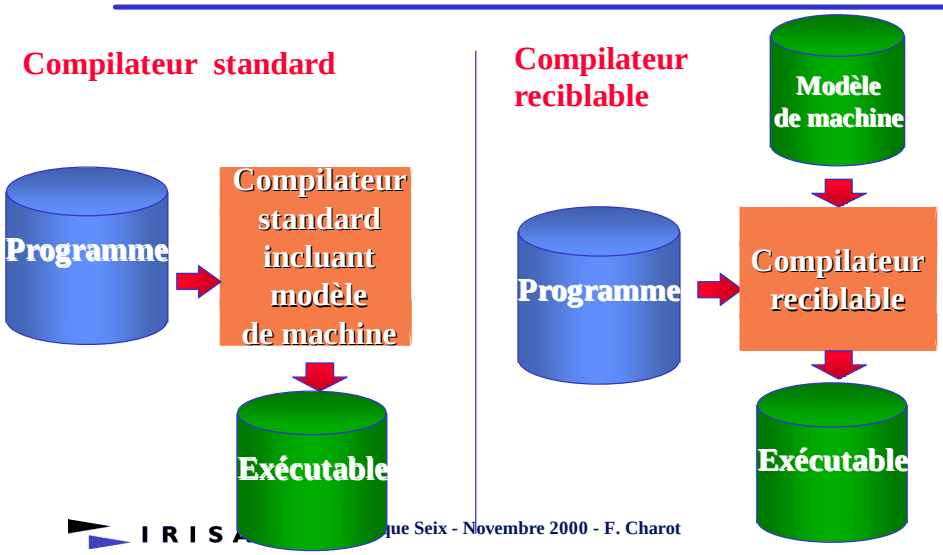
Plan de la présentation

- Quelques aspects concernant la production de logiciel pour les systèmes enfouis
- Compilateurs pour processeurs embarqués
- **Compilateurs reciblables**

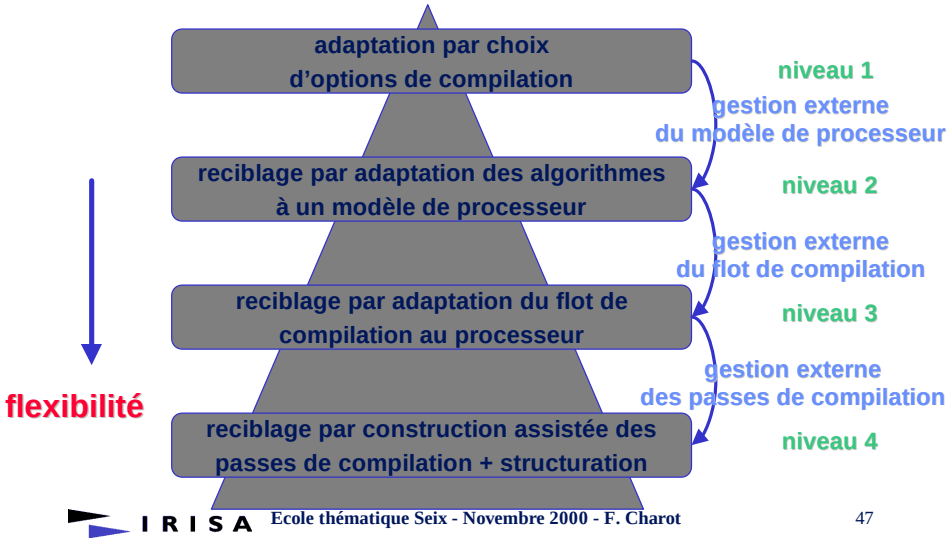
  Motivations pour des compilateurs reciblables

  Quelques exemples
- Approche ARMOR/CALIFE
- Conclusion

Compilateur reciblable versus compilateur standard

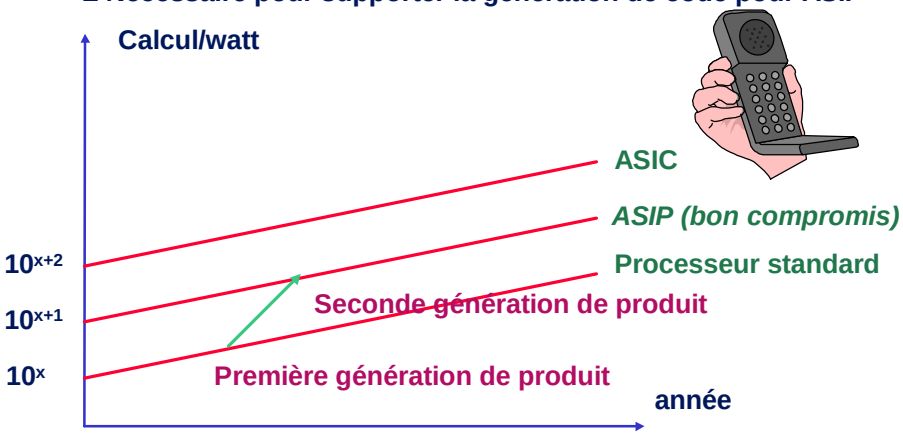


Flexibilité d'un compilateur



Pourquoi le reciblage ?

- Nécessaire pour supporter la génération de code pour ASIP



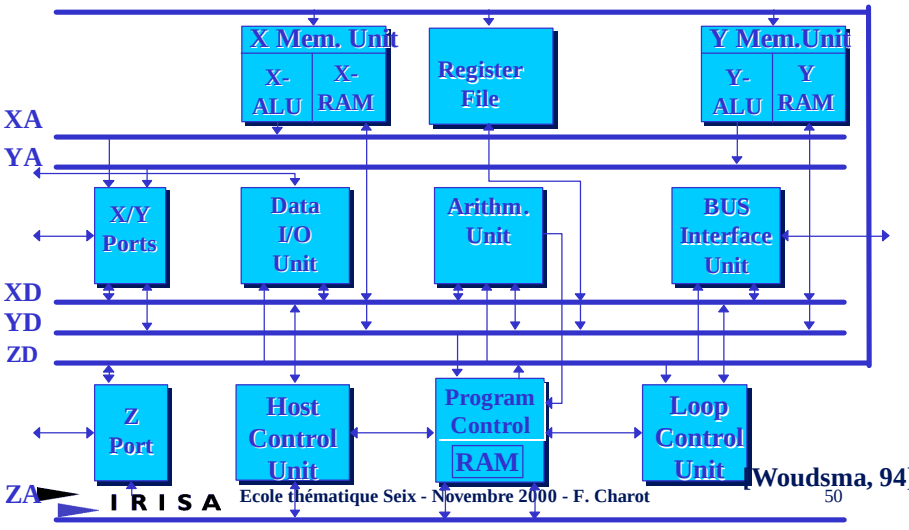
Pourquoi les ASIP sont-ils utilisés ?

	Processeurs	Applications
Philips	Trimedia VLIW EPICS	Multimédia GSM
STMicroelectronics	DSP D950 + coprocesseur	Vidéophone GSM, modem
Italtel[1]	DSP ASIP	Station de base GSM
FT R&D[2]	ASIP DSP LIW	Terminal main libre

(1) Réduction du volume (2 ASIP au lieu de 8 DSP) et de la fréquence d'horloge (facteur 5 à 10)

(2) Réduction de 50% de la consommation par rapport à des solutions commerciales (plus de parallélisme, fréquence horloge réduite)

EPICS : architecture reconfigurable



[Woudsma, 94]

EPICS : caractéristiques dépendantes de l'application

- Mémoire : types et taille
- Largeur des données et adresses en bit
- file de registres optionnelle
- unité de contrôle de boucle optionnelle
- interfaces paramétrables
- utilisation de générateurs de SRAM et ROM
- .....

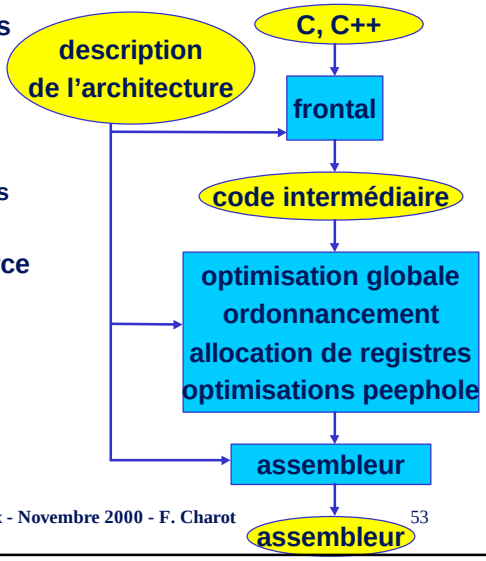
Approche intermédiaire entre l'ASIP et le cœur standard

Pourquoi le reciblage ?

- Produire rapidement des compilateurs pour une large palette de processeurs, de façon économique
- Comparer différentes architectures
- Fournir une meilleure compréhension de l'interdépendance entre architecture et compilateur

GCC : Gnu C Compiler

- optimisations indépendantes de l'architecture cible
  - élimination des sous-expressions communes
  - élimination du code mort
  - propagation de constantes
  - ...
- large diffusion du code source
- inadapté aux DSP, ASIP



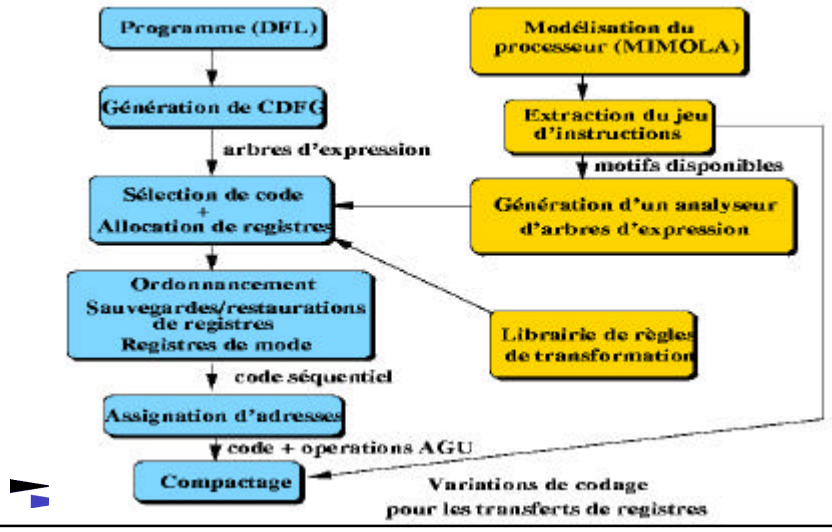
Record

- université de Dortmund
- cible une classe de DSP
- modélisation structurelle de la cible (Mimola)

```
MODULE ual (IN A,B : (15:0));
                IN Ctrl :(1:0);
                OUT Res : (15:0))
 CONBEGIN
   Res <- CASE Ctrl OF
     0 : A+B;
     1 : A-B;
     2 : A AND B;
     3 : A ;
   END;
 CONEND;
```

```
MODULE register (IN input : (15:0);
                 OUT output : (15:0);
                 IN enable : Bit; CLK clock : Bit)
 VAR reg : (15:0)
  CONBEGIN
    CASE enable OF
      0 : NONLOAD;
      1 : AT clock UP DO reg := input;
    END;
    output <- reg;
  CONEND;

 CONNECTIONS
  ual.Res -> register.input
  ...
```

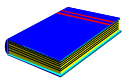
Record : flot de compilation



Chess

- Target compiler technologies (http://www.retarget.com)
- cible DSP
- modélisation comportementale de la cible

Idée: partir du manuel de référence

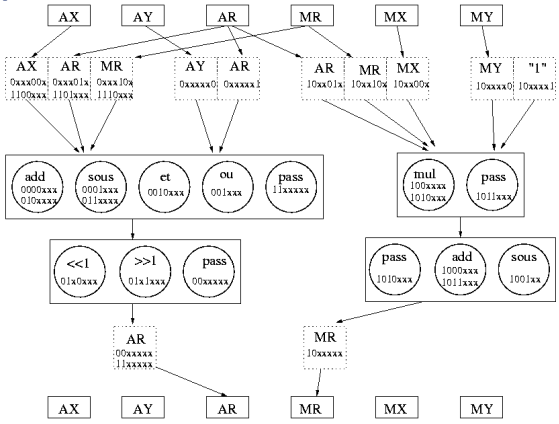


format opcode	operands
0 0	.....
0 1	.....
format opcode	operands
1 0	.....
format opcode	operands
1 1:	.....

```
opn exa (alu_sh_inst
        | macc_inst
        | shift_inst);
opn alu_sh_inst ( ....
);
opn macc_inst ( ....);
opn shift_inst  ( ... );
```

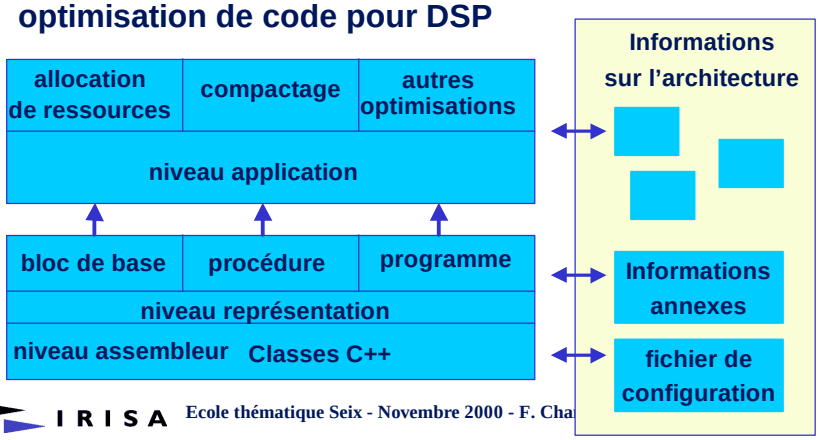
Chess

- ISG : modèle combiné du jeu d'instructions et du chemin de donnée, base pour la sélection de code



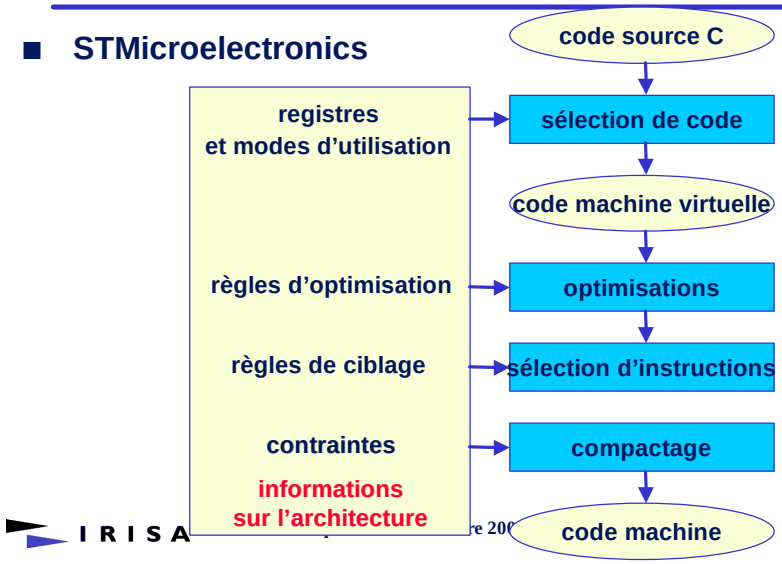
SPAM

- université de Princeton (http://www.ee.princeton.edu/spam)
- bibliothèque d'algorithmes de génération et optimisation de code pour DSP



FlexCC

- STMicroelectronics



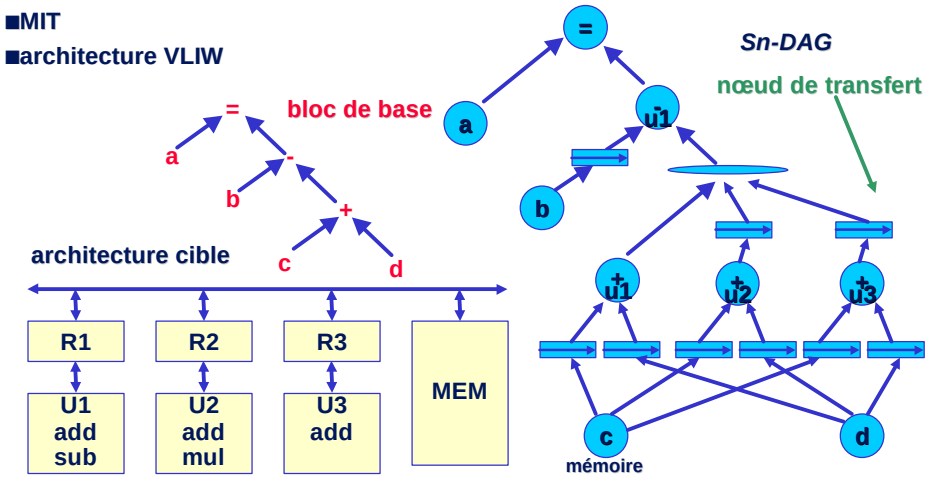
SUIF

- université de Stanford (http://www.suif.stanford.edu)
- à la base de la *National Compiler Infrastructure*
- regroupe Harvard, Rice, Stanford, UCSB, ...

- optimisation de code source, parallélisation
- niveau d'abstraction du format intermédiaire

AVIV/ISDL

- MIT
- architecture VLIW



AVIV/ISDL

- Une description ISDL comporte les sections suivantes
  - format du mot d'instruction
  - déclaration des ressources de mémorisation, avec leur taille et leur largeur en bits
  - instructions du processeur, informations sur la syntaxe assembleur, le masque binaire, la sémantique, la taille et la durée d'exécution
  - spécification explicite des contraintes imposées par l'architecture (limitations de parallélisme dues au codage du jeu d'instructions)
- Deux bases de données sont construites
  - mise en relation des opérations de base avec toutes les instructions du processeur susceptibles de les implémenter
  - transferts de données autorisés par le processeur

Cosy

- Développé par ACE (http://www.ace.nl)
- Résultat des projets esprit Prepare et Compare
- C, C++ DSP-C
- Points forts
  - organisation des passes de compilation
  - complet (générateur de générateur de code, optimisation de boucles)
- Points faibles
  - adaptation aux DSP et ASIP ?

Plan de la présentation

- Quelques aspects concernant la production de logiciel pour les systèmes enfouis
- Compilateurs pour processeurs embarqués
- Compilateurs reciblables
- **Approche Armor/Calife**

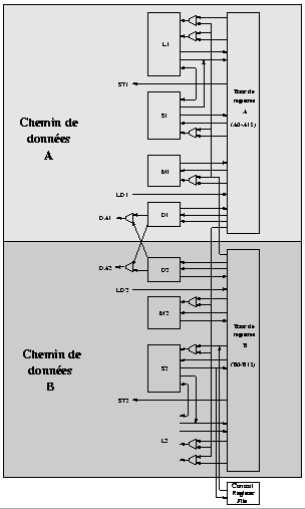
  Modélisation des processeurs en Armor

  Plate-forme de compilation Calife
- Conclusion

Le langage Armor

- Comportement du jeu d'instructions
  - sémantique opérationnelle
  - utilisation des ressources
  - parallélisme interne
- Pas d'informations sur la forme
- Concision -> factorisation des informations
  - réinvestissement des principes de nML

Structure du jeu d'instructions



Tms320c6x

```
instructionSet =
[
   inst || inst || inst || inst
|| inst || inst || inst || inst
]

gp inst = add | sub | ...
```

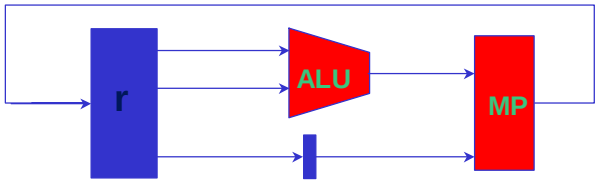
Structure du jeu d'instructions

D950

Instructions comprenant une opération arithmétique avec des load/store éventuels :

```
gp ALUo2_LDSTopt =
     [ st_YM_1 || ALUo2_3 || ld_XM_1 ] |
     [ st_YM_1 || ALUo2_3 ]            |
     [ ALUo2_3 || ld_XM_1 ]            |
     [ st_XM_1 || ALUo2_4 || ld_YM_1 ] |
     [ st_XM_1 || ALUo2_4 ]            |
     [ ALUo2_4 || ld_YM_1 ]
```


Flot de données

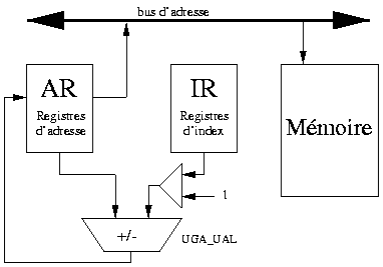



```
df asmul is { r=mul(adsub(r,r),r) }
op adsub = add | sous
op add(x,y)  is  ADD(x,y) <ress=ALU>
op sous(x,y) is  SUB(x,y) <ress=ALU>
op mul(x,y)  is  MUL(x,y) <ress=MP>

fu ALU <cycle=1>    fu MP <cycle=2>
regFile r(16,word) <access Rd=1 Wr=2>
type word is integer(32)
```

ADDMUL r,r,r,r

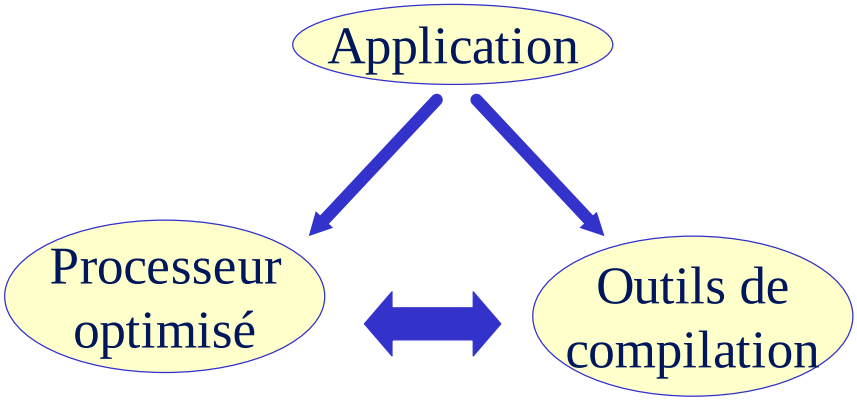
SUBMUL r,r,r,r

Unités de génération d'adresses

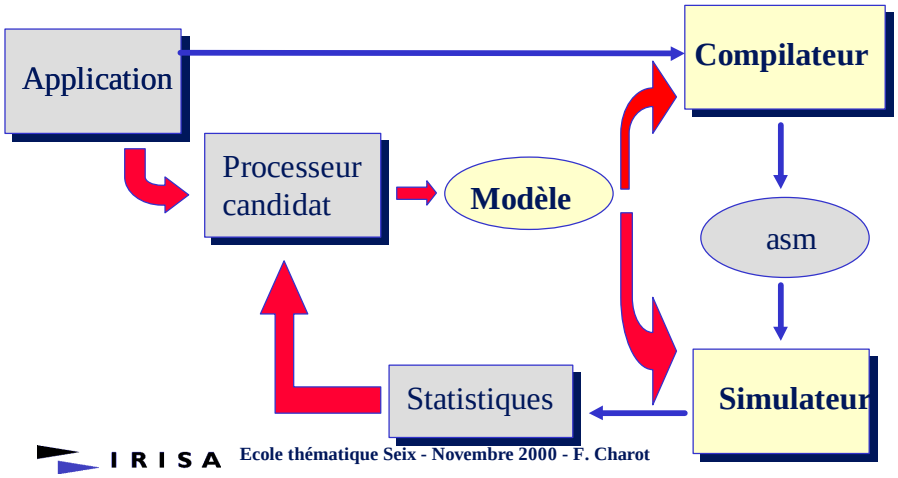



```
df load is {r=mem[ad]}

address ad = simple
           | postmod

address simplep is AR
address postmod is
 AR%i{opAd(AR%i,src)}

op opAd = inc | dec
op inc(x,y) is
  ADD(x,y) <ress=UGA_UAL>

mode src = IR | const(1)
```

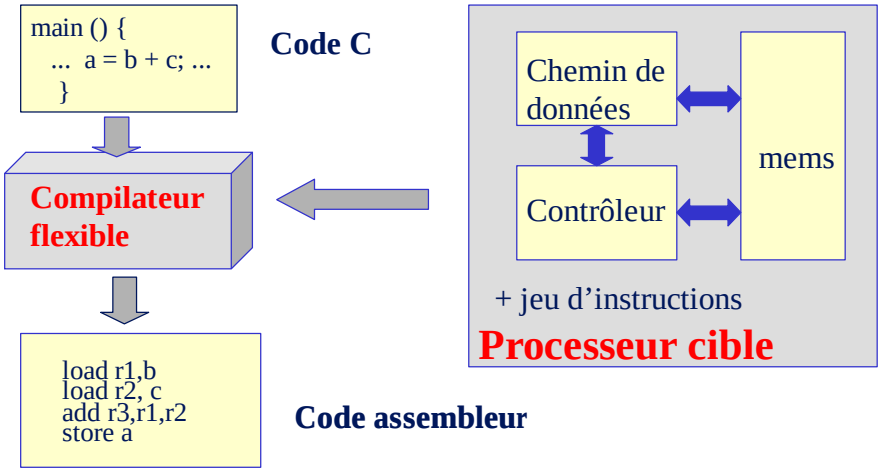

Adéquation application/ architecture/compilateur



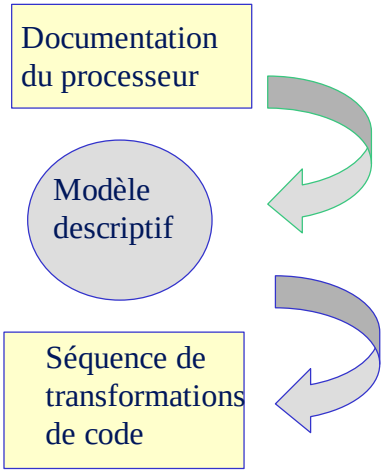
Adéquation application/ architecture/compilateur



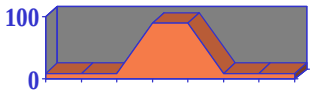
Compilation flexible



Flexibilité : approche courante

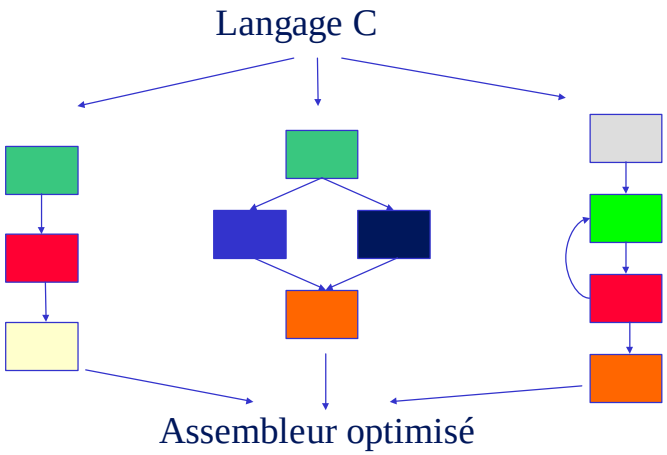


- **Reciblage rapide**
- **Pas de compétences en compilation**



- Gamme architecturale restreinte

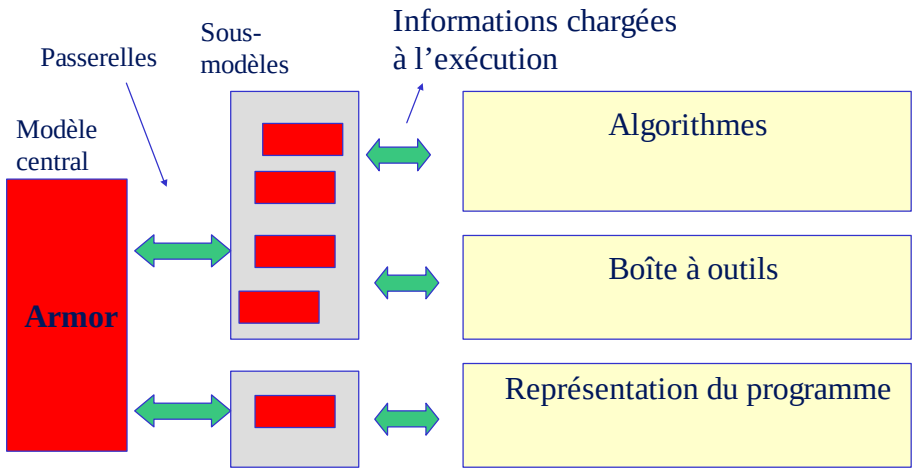
Notre approche de flexibilité : au niveau du flot de compilation



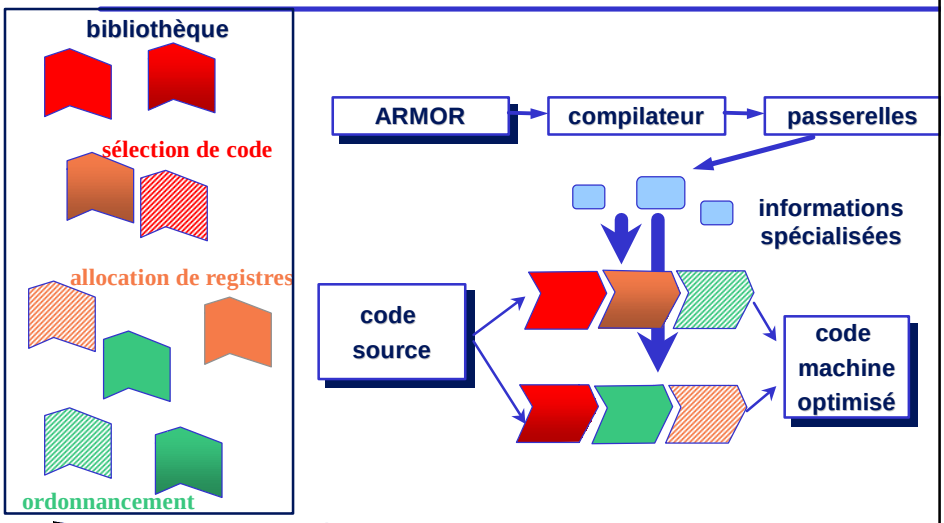
Fondements de CALIFE

- Créer des transformations génériques
- Offrir un moyen de les paramétrer
  - description du processeur
- Offrir un moyen de les assembler
  - plate-forme commune

Bibliothèque CALIFE



CALIFE en résumé



Plan de la présentation

- Quelques aspects concernant la production de logiciel pour les systèmes enfouis
- Compilateurs pour processeurs embarqués
- Compilateurs reciblables
- Approche Armor/Calife
- **Conclusion**

Conclusion

- Tendance marquée vers les cœurs en général et les cœurs de processeurs embarqués en particulier.
- Les compilateurs pour processeurs embarqués ont du mal à générer du code de qualité. Les travaux récents montrent que des améliorations sont possibles.
- Les compilateurs reciblables offrent un support pour les ASIP, mais également pour sélectionner le processeur adéquat et comprendre le processus de reciblage.
- Les compilateurs reciblables doivent être interfacés à d'autres outils (outils de CAO, de synthèse, de parallélisation)