

Co-Conception de Systèmes Spécialisés sur Composant

Michel Auguin



Les Algorithmes / Bâtiment Euclide B
2000 route des Lucioles, BP 121, 06903 Sophia-Antipolis Cedex

1

Plan de l'exposé

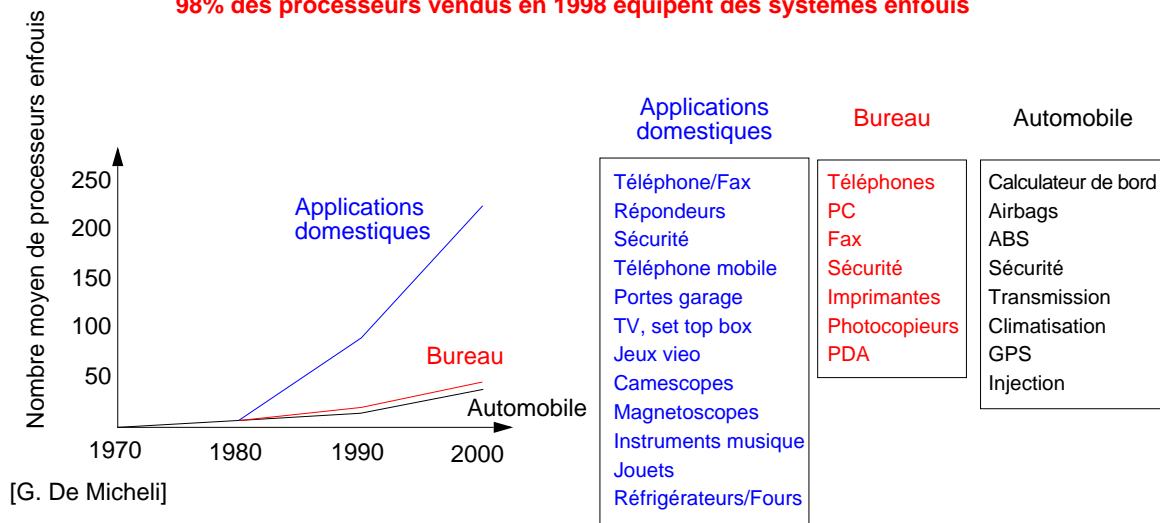


- 1 - Pourquoi la co-conception (codesign)**
- 2 - Modélisation des applications**
- 3 - Les architectures enfouies**
- 4 - Estimation logicielle et matérielle**
- 5 - Partitionnement**
 - Partitionnement manuel
 - Partitionnement automatique
- 6 - Synthèse des communications**
- 7 - Conclusion**

2

POURQUOI LA CO-CONCEPTION DE SYSTEMES ENFOUIS (CODESIGN)

98% des processeurs vendus en 1998 équipent des systèmes enfouis



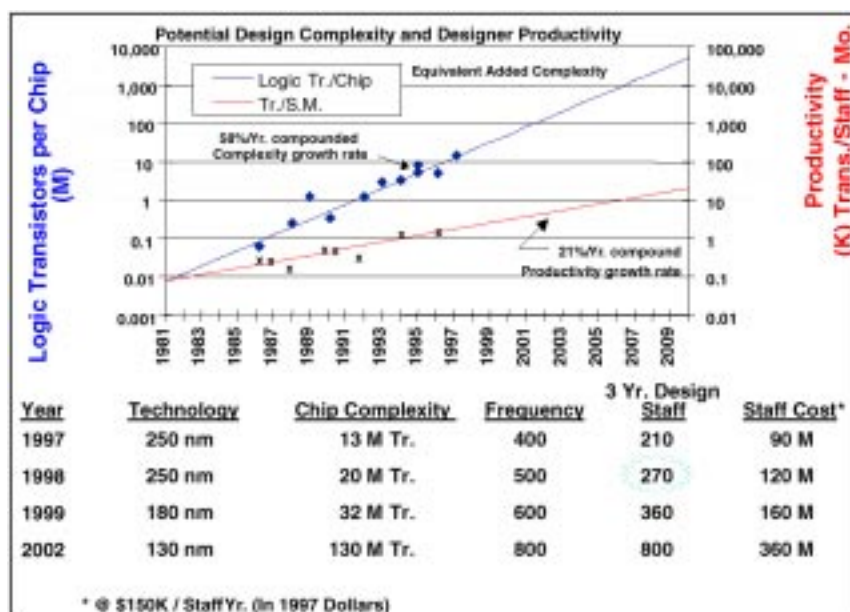
Estimations pour l'année 2003

1 milliard de téléphones portables

200 millions de PC

3

Divergence entre Possibilités d'Intégration et Coûts de Conception



A ce rythme on ne pourra plus exploiter les possibilités de la technologie

coûts trop élevés

International Technology Roadmap for Semiconductors (1999)

4

Spécificités des architectures enfouies

Les systèmes enfouis imposent des contraintes qui conditionnent leur conception :

- Faibles marges de coûts (1\$ sur un portable)
- Contraintes de temps sévères
- Consommation d'énergie (portables, PDA ...)
- Sécurité (automobile, aviation)
- Poids, taille (UMTS vs GSM)

La technologie permet des évolutions rapides des produits

- Architecture plus intégrée :
performances augmentées,
coûts de fabrication réduits
- Le coût global en vaut-il la peine ?

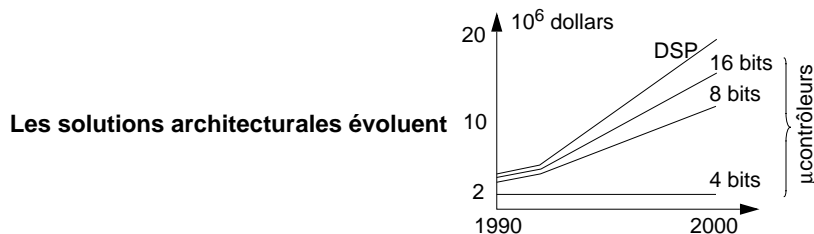
Vaste choix possibles offrant divers compromis performances/coûts

- RISC, DSP, ASIP, ASSP, IP, ASIC, FPGA, Analogique, MEMS
DSP 16, 20, 24, 32, 64 bits, Mono/MultiMac, superscalaires, VLIW
- Superscalaires & instructions SIMD (stations de base UMTS ?)

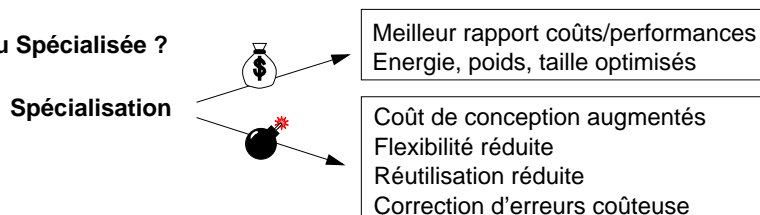
5

Les architectures des systèmes enfouis évoluent

		A310	A320	A340
Exemple : Avions Airbus	}			
	Composants numériques	77	102	115
	Taille du code embarqué	4 Mo	10 Mo	20 Mo
	Puissance de calcul	60 MIPS	160 MIPS	250 MIPS



Dilemme du concepteur :
Architecture Généraliste ou Spécialisée ?

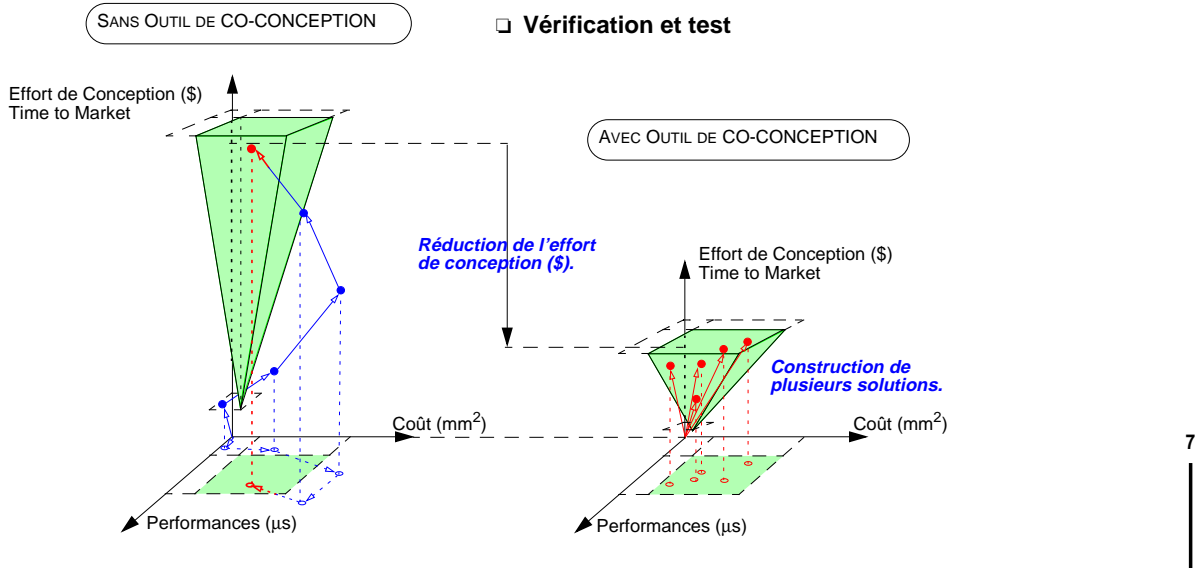


6

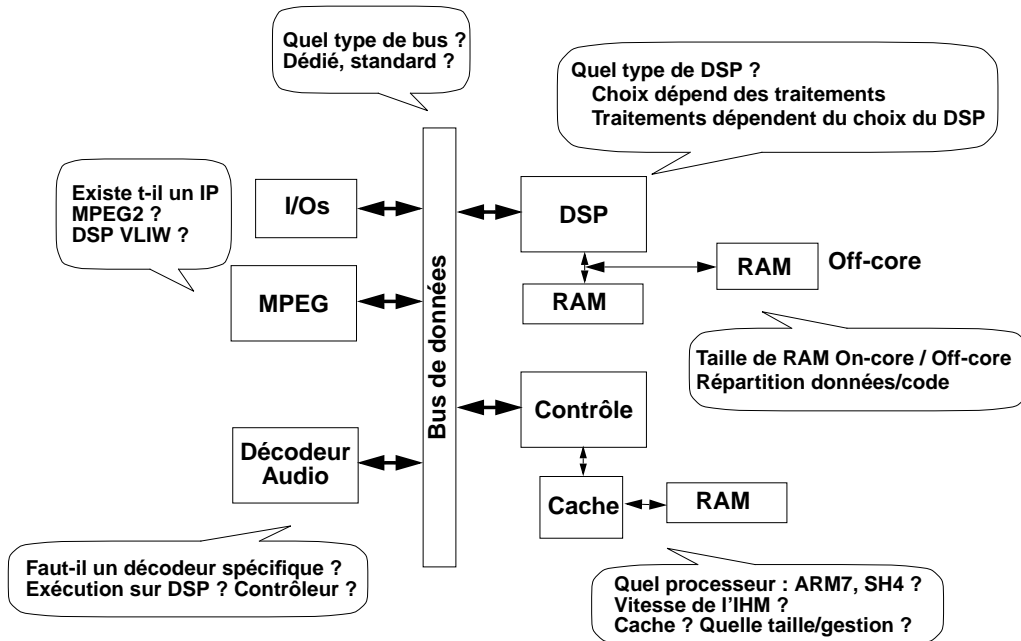
Application, Technologie et Architecture évoluent,... et les Méthodes de Conception ?

Evolution des méthodes et outils de conception :

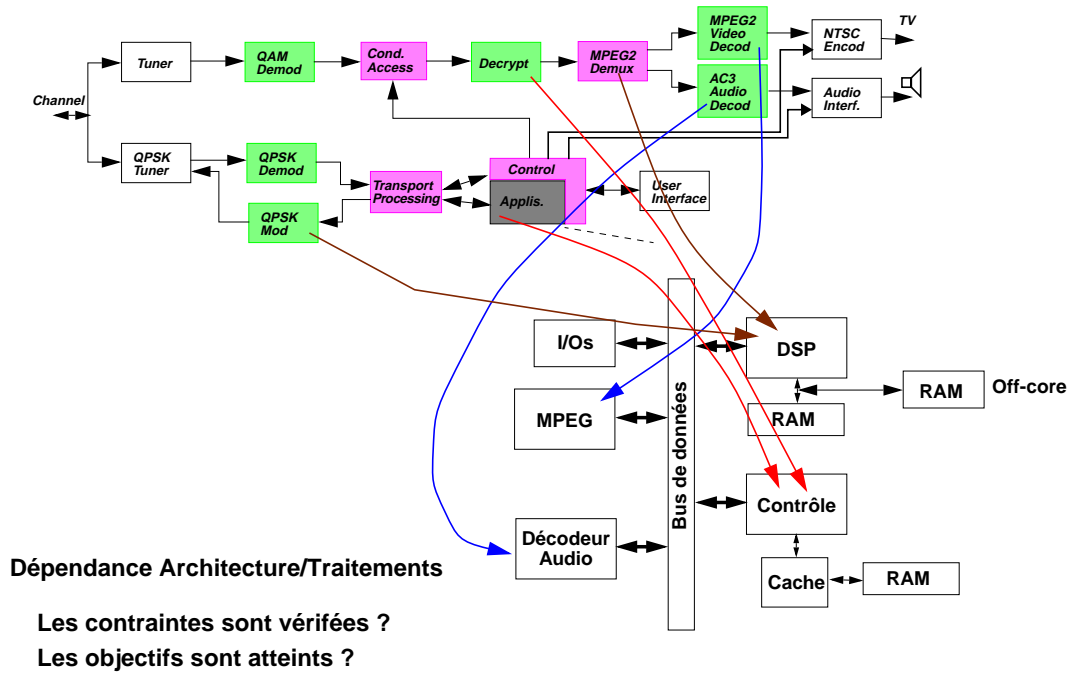
- Systèmes hétérogènes (RISC, DSP, ASIC, ASIP, IP)
- Réutilisation
- Exploration de solutions architecturales
- Vérification et test



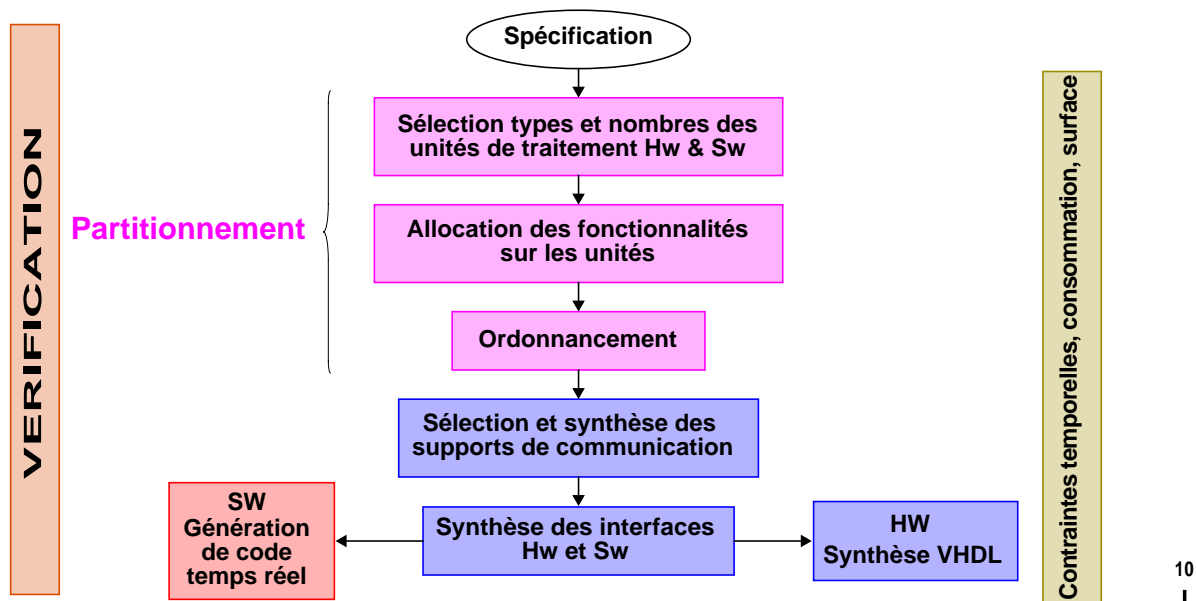
Méthode d'aide à la conception : Quelle architecture, Quelles caractéristiques ?



Méthode d'aide à la conception : Quelle répartition des traitements



Flot classique de co-conception



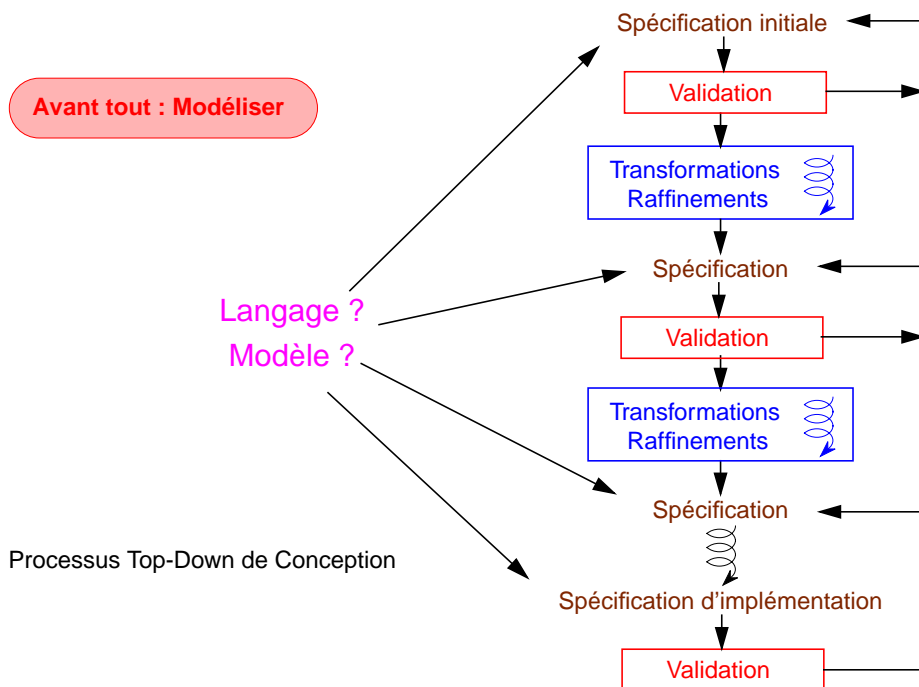
Plan de l'exposé



- 1 - Pourquoi la co-conception (codesign)
- 2 - Modélisation des applications
- 3 - Les architectures enfouies
- 4 - Estimation logicielle et matérielle
- 5 - Partitionnement
 - Partitionnement manuel
 - Partitionnement automatique
- 6 - Synthèse des communications
- 7 - Conclusion

11

MODÉLISATION DES APPLICATIONS



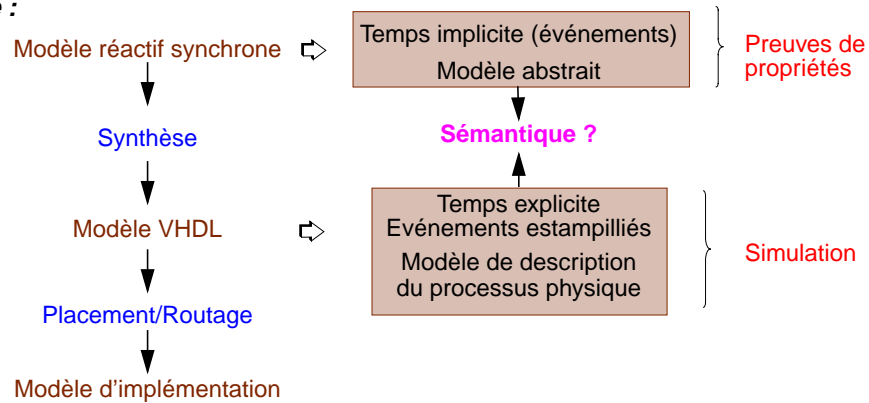
12

Les applications sont hétérogènes : Approche pragmatique

- ❑ Utiliser un langage adapté en fonction du niveau de spécification
(e.g. VHDL pour décrire un système matériel niveau RTL)

- ❑ Mais changer de modèle dans le processus de conception peut poser des difficultés.

Exemple :

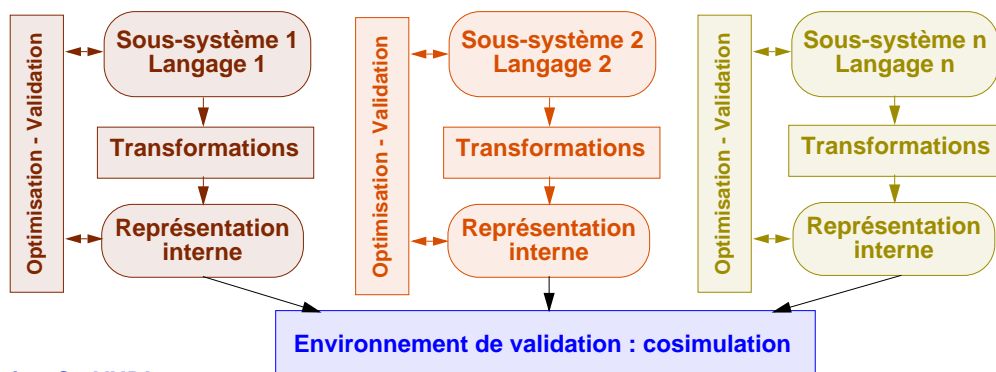


13

Les applications sont hétérogènes et parallèles. Comment les prendre en compte ?

Approche "langage"

La spécification est décomposée en sous-systèmes (manuellement)



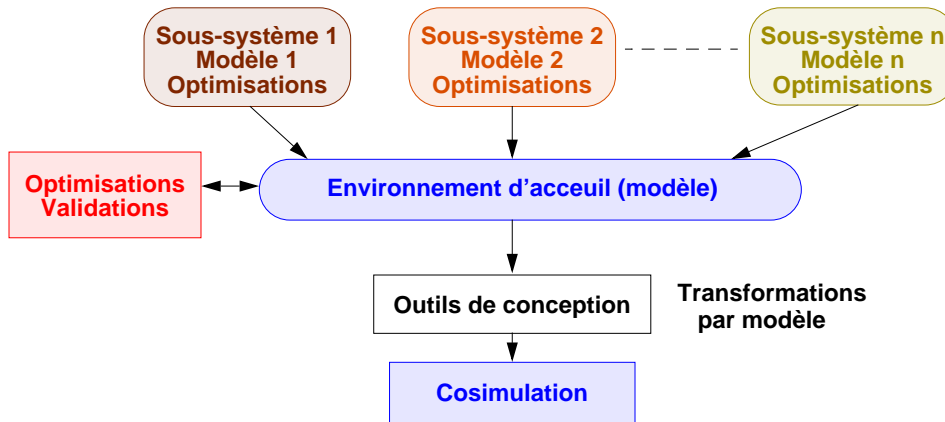
Cosimulation C - VHDL

Cosimulation Assembleur (ISS) - VHDL

- ❑ Aucune optimisation globale : optimisations par sous-système (dépendant langage)
- ❑ Nécessite de décrire précisément les communications et les interfaces
- ❑ Les migrations éventuelles entre sous-systèmes ne sont pas aisées

14

Approche "modèle"



- ❑ Optimisations locales et globale
- ❑ Exploration de l'espace de conception plus aisée
- ❑ Les communications et interfaces sont décrites de manière plus abstraite

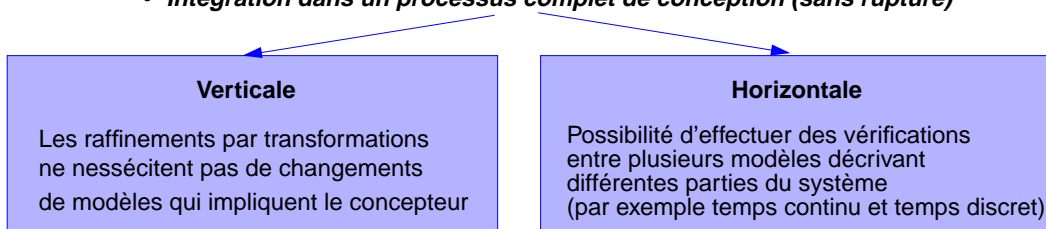
15

Modélisation d'une application hétérogène

- ❑ Quel(s) modèle(s) pour décrire la spécification initiale ?

Ce choix est guidé par plusieurs considérations, par exemple :

- *Type de traitements dans l'application*
- *Possibilité d'exécuter ou d'animer le modèle*
- *Disponibilité et efficacité des outils d'analyse et de synthèse*
Preuve formelle de propriétés - Simulation - Synthèse/Compilation
- *Possibilité de décrire des spécifications non-fonctionnelles*
- *Intégration dans un processus complet de conception (sans rupture)*



16

Choix du modèle de spécification en fonction du type de l'application

Classiquement les concepteurs considèrent qu'un système est dominé :

<p>Par le contrôle</p> <p>Réactions à des événements discrets :</p> <ul style="list-style-type: none">Pas d'hypothèse sur l'occurrence des événementsTous les événements doivent être pris en compte <p>⇒ Contrôle/Commande de processus</p>	<p>Systèmes à événements discrets</p> <ul style="list-style-type: none">Systèmes réactifs synchronesMachine d'Etats FinisRéseaux de PetriProcessus concurrents
--	--

<p>Par les données</p> <p>Les sorties sont fonctionnellement dépendantes des entrées :</p> <p>Les occurrences des valeurs sur les entrées sont périodiques</p> <p>⇒ Traitement du signal et des images</p>	<p>Kahn Process Networks</p> <p>Dataflow Process Networks</p>
---	---

17

Langages associés aux modèles

Systèmes réactifs synchrones	⇒	Esterel, Lustre, Signal, ECL	} Codesign Finite State Machine
Systèmes à événements discrets	⇒	VHDL, Verilog	
Machine d'Etats Finis	⇒	*Charts (StateCharts, ...)	} SDL (Standard de l'ITU)
Processus concurrents	⇒	CSP, Occam, Lotos	
Dataflow Process Networks Kahn Process Networks	⇒	Dynamic Data Flow Synchronous Data Flow	
Approche objet	⇒	Java, C++, OO-VHDL	

Exemple d'outils associés:

VCC de Cadence : asynchronisme des CFSM, C, C++, ECL

CoCentric de Synopsis : SystemC - Classes C++ pour décrire réactivité, processus concurrents

CoWare : Classes C++ pour décrire des processus concurrents (RPC)

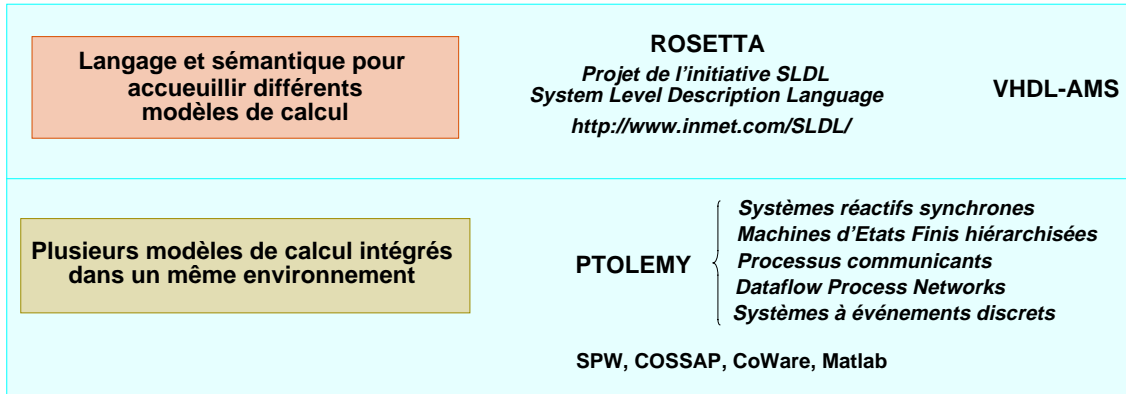
18

Environnements pour applications hétérogènes



Un seul modèle de calcul est insuffisant pour décrire toute l'application

Deux approches proposées :



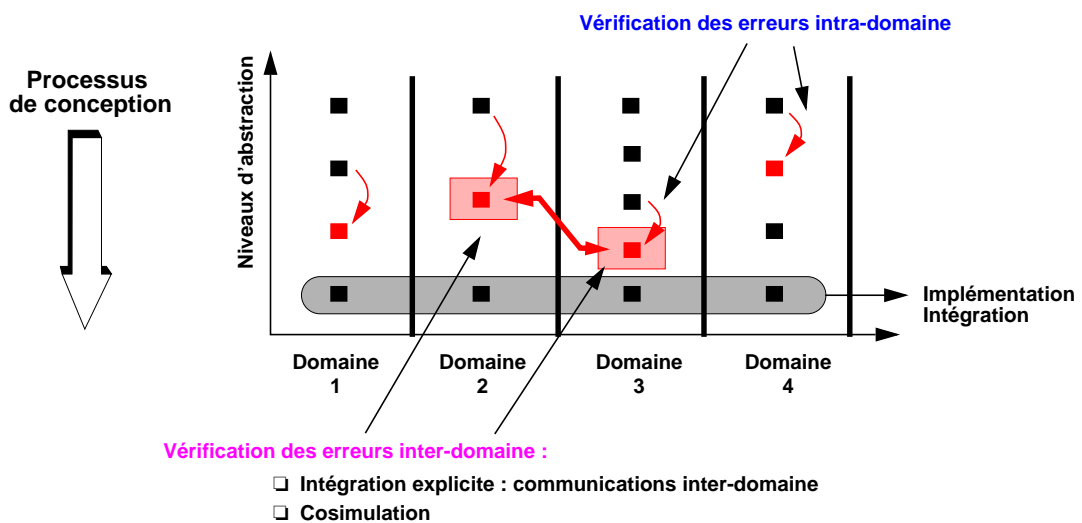
Hormis les techniques à base de cosimulation, il n'existe pas actuellement de méthodes de conception unifiées qui opèrent sur une description hétérogène.

19

Conception verticale par domaine (modèle de calcul)

Approche traditionnelle :

Décomposition en sous-systèmes et conception verticale



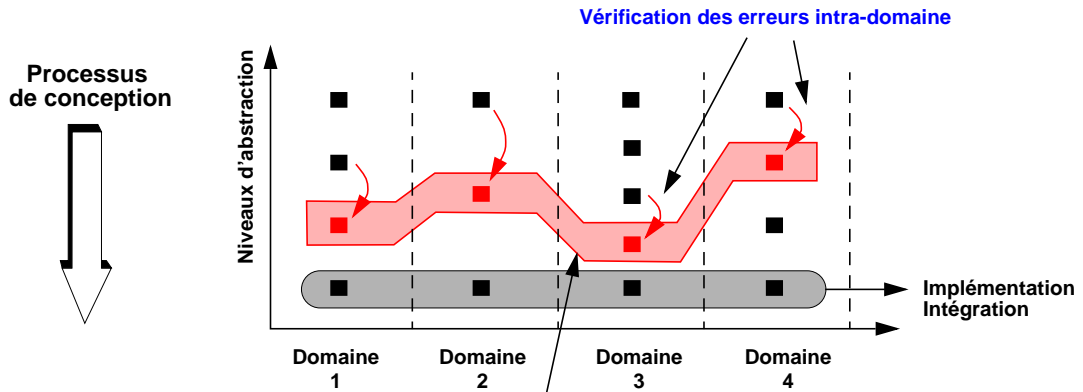
[www.vptinc.com]

20

Evolution souhaitée des méthodes de conception

Approche intégrée :

L'environnement permet des raffinements "sans coutures"



Vérification des erreurs inter-domaine :

- Communications inter-domaine : types prédéfinis
- Vérifications sémantiques

[www.vptinc.com]

21

Représentation d'informations non-fonctionnelles.

La description d'un système : **comportement** + **informations non fonctionnelles**

par exemple des contraintes : temps, surface, consommation, time-to-market...

Exemple dans la norme de communication Bluetooth

- Temps d'exécution global à considérer = 312,5 μ s,
c.a.d. une demi-fenêtre : 2 paquets peuvent être produits
par fenêtre de 625 μ s.
- Contrainte sur la partie bande de base :
 $321,5 - (T_{vco} + T_{power_up} + T_{uncertainty_window} + T_{access_code})$
- Consommation et surface minimum, coût réduit



Plusieurs modèles ou langages sont étendus pour décrire des contraintes

Exemple 1 : Langage **ROSETTA** (SLDL)

Types physiques et opérations associées : spécification des contraintes

Un système = composition de facettes : contraintes du système

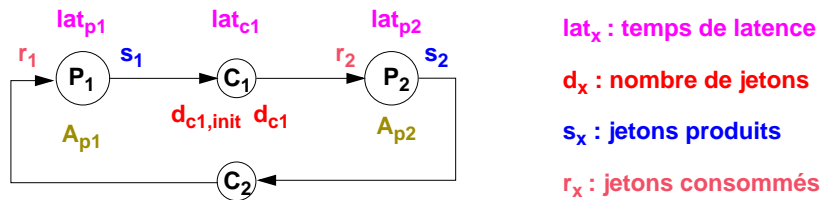
Exemple 2 : Modèle **SPI, Single Model with Intervals** (Université de Braunschweig)

22

En SPI un système est un ensemble de processus communicants

Communications par :

FIFO : lecture destructive, écriture non-destructive } C_1, C_2
Registre : lecture non-destructive, écriture destructive }



A_x : règle d'exécution

Les jetons peuvent être étiquetés (tag)

Les processus : différents modes ($m_{i,j}$)

P_1 : Si $m_{1,1} \wedge C_2.tag = y$

alors $m_{1,2}$; $s_1 = 2$; $C_1.tag = z$; $lat_{p1} = 3ms$

P_2 : Si $m_{2,1} \wedge C_1.tag = z$

alors $m_{2,2}$; $s_2 = 1$; $C_2.tag = y$; $r_2 = 1$; $lat_{p2} = 1,5ms$

Contraintes temporelles : $LC(C_1, C_2) = [lat_{min}, lat_{max}]$

23

Analyse statique des séquences possibles d'exécution (recherche de cas pire)

Plan de l'exposé

1 - Pourquoi la co-conception (codesign)

2 - Modélisation des applications



3 - Les architectures enfouies

4 - Estimation logicielle et matérielle

5 - Partitionnement

Partitionnement manuel

Partitionnement automatique

6 - Synthèse des communications

7 - Conclusion

24

ARCHITECTURES ENFOUIES

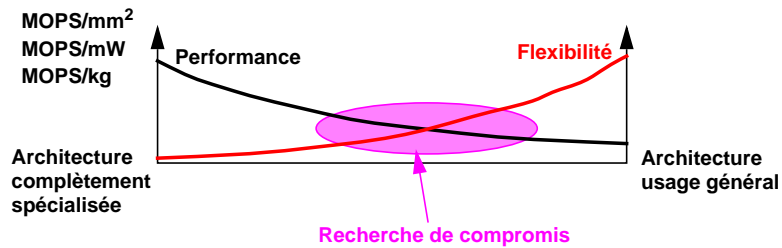
Les méthodes de codesign ciblent généralement des systèmes embarqués/enfouis (SOC)

Ces systèmes impliquent des contraintes :

produits largement diffusés : coûts réduits
contraintes temporelles strictes
consommation d'énergie
taille, poids
sûreté de fonctionnement (e.g. aéronautique)

Mais il faut aussi privilégier :

la réutilisation
la flexibilité : modifications tardives,
corrections d'erreurs



25

Très grande diversité des composants

- Les systèmes embarqués deviennent de plus en plus complexes.

Le nombre de cœurs de processeurs dans un SOC de 1996 à 1998 chez IBM :

1996 : 2 en moyenne, 3 max
1998 : 9 en moyenne, 30 max

- Grande variété de composants disponibles :

Cœurs de processeurs

ASIP : Application Specific Instruction-set Processor
ASSP : Application Specific Standard Product
Microcontrôleurs
RISC
DSP : Digital Signal Processors

Fonctions logicielles

Fonctions matérielles (ASIC)

Bus standardisés

Composants reconfigurables

SPGA : System Programmable Gate Array - FPGA + IP)

IP: Intellectual Property

26

ASIP et ASSP

ASIP : Application Specific Instruction set Processor

- ❑ Processeur spécialisé à l'exécution d'une (ou quelques) application (par exemple Modem)
- ❑ Jeu d'instruction et ensemble des ressources adaptés à l'application
- ❑ Meilleurs rapports MIPS/mW et MIPS/mm² que RISC et DSP
- ❑ Mais compilateur plus délicat, *time-to-market* plus long qu'avec des processeurs standards

ASSP : Application Specific Standard Product

- ❑ Composant complexe qui réalise une fonction spécifique (compression vidéo, modem)
- ❑ ASSP et interface standardisée : IP

27

Les Processeurs embarqués RISC

Grande variété de processeurs RISC disponibles sous forme d'IP (ARM, Hitachi, MIPS, LSI Logic)

Exemple : RISC 32 bits de Advanced Risc Machines (ARM) : **Famille ARM7** : 3 étages de pipeline

	Surface	Puissance	Horloge	MIPS	Cache
ARM7TDMI	0,54 mm ²	0,25mW/MHz	110 MHz	59 MIPS	-
ARM7TDMI-S synthétisable	0,8 mm ²			-	-
ARM7720T	2,9 mm ²	0,65 mW/MHz	75 MHz	-	8k unifié

ARM7TDMI : *Thumb Instruction Set*

Compression sur 8 ou 16 bits d'une partie des instructions 32 bits } Espace mémoire
Décompression à l'exécution } réduit de 30%

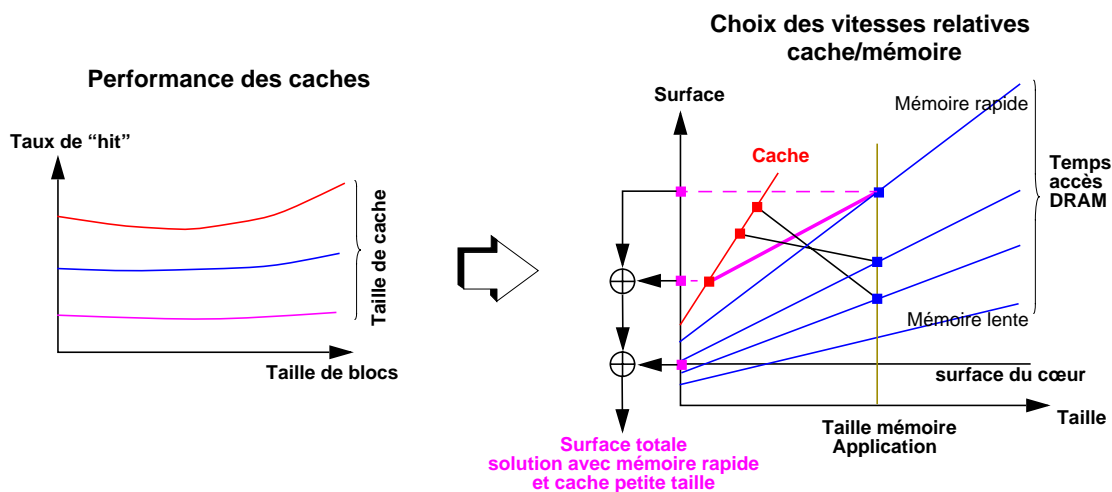
Famille ARM9 : 5 étages de pipeline

2,7 mm², 1,8 mW/MHz, 160MHz, 176 MIPS en 0,25 µm

28

Processeurs disponibles sur le Web : **OpenCore**, **Leon**

Processeurs embarqués RISC et mémoire cache : recherche de compromis



Quelle taille et quelle gestion de cache pour optimiser :

Performances
mm²

Performances
mW

Surface cœur ARM720T : 0,54 mm²
Surface cache 8K octets : 2,3 mm²

29

Les processeurs de traitement du signal DSP

Il existe de nombreux constructeurs : nombreux DSP et leurs variantes chez chacun d'eux

DSP virgule fixe 16 bits	DSP568xx, C54x, ADSP21xx, OakDSP, DSP16xx, Carmel, PalmDSP	Les plus communs. (Télécom. embarqué)
DSP virgule fixe 20 bits ou 24 bits	DSP563xx, PalmDSP	Applications audio
DSP virgule fixe et flottante	TigerSharc	Calcul intensif, précision
DSP flottant	ADSP2106x, C30, C40	Précision
DSP superscalaire	TigerSharc (4), StarCore (6)	Calcul intensif
DSP VLIW	C62xx, C67xx (8), TriMedia (5), REAL (instructions ASI), Carmel (instructions CLIW)	Calcul intensif. Traitement // sur octets
DSP flexible	REAL (unités AXU)	Unités fonct. dédiées
DSP hybride	Tricore, ST100	Applications diverses

Un DSP peut être particulièrement adapté à un type d'application (exemple : DSP56009, TMS320C54x)

Les performances des DSP peuvent varier significativement :

exemple : localisation des données en mémoire, localisation du code

30

Les performances des DSP sont très variables

Mesure de performance des DSP : BDTImark™

Temps exécution de 12 applications test (pas de compilation)

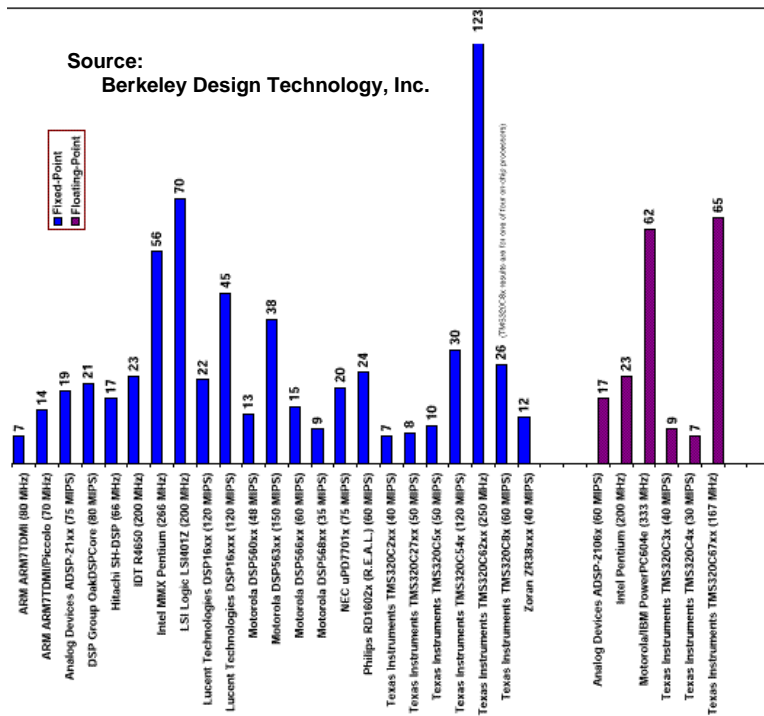


Mesure de performance



Performances très variables
Choix DSP important

Source: Berkeley Design Technology, Inc.



31

Un système : ensemble d'unités interconnectées

L'interconnexion devient un problème crucial dans la conception de circuits complexes
"Interconnect performance bottleneck"



Les constructeurs de SOC s'orientent depuis quelques années vers l'utilisation de bus génériques : protocoles de transferts de données bien définis.

Exemple la proposition AMBA de ARM : *Advanced Microcontroller Bus Architecture*

Bus Système : AHB ou ASB

bus rapide, multi-maître, transferts pipelines ou en mode *burst*, priorité sur les transferts

Bus Périphérique : APB

bus adapté à la connexion de périphériques "lents", non-pipeline, pas de priorité, optimisé en consommation

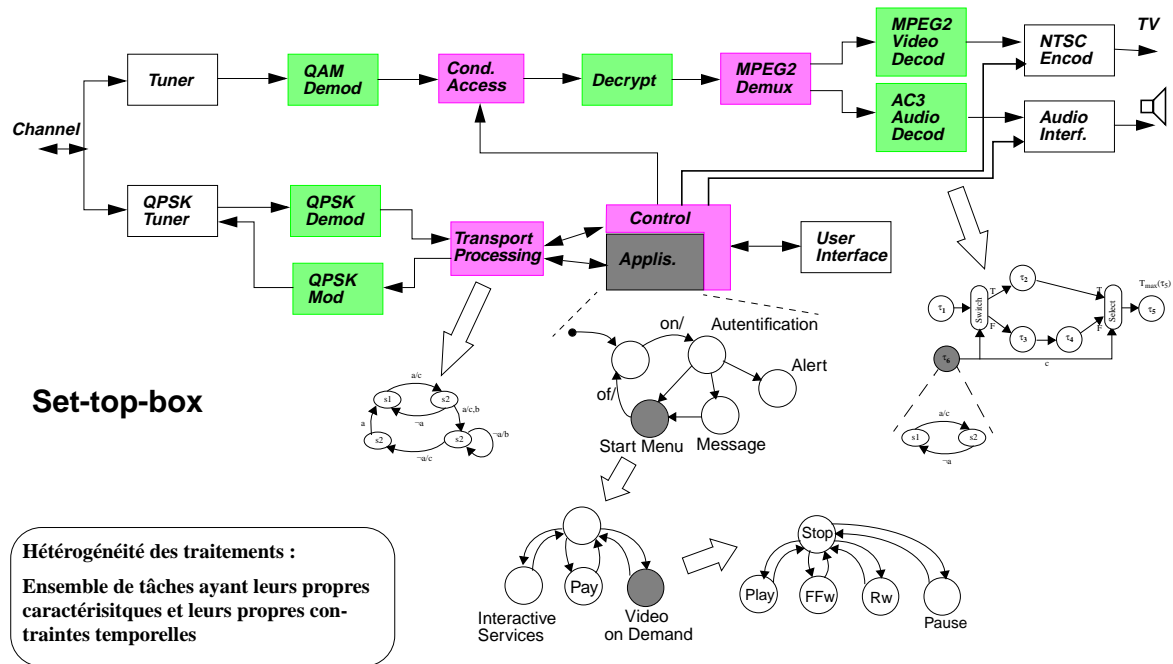
Interfaces entre bus : Bridges



La conception d'interface de communication est réduite
Les temps de communication dépendent des bus et des modes de transfert utilisés.

32

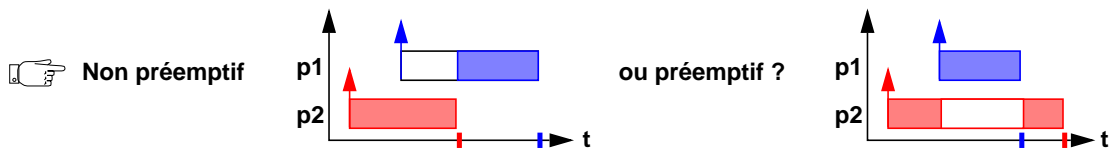
Système d'exploitation temps réel embarqué



33

Gestion des tâches par exécutif temps réel (RTOS)

Le comportement du système dépend des caractéristiques du RTOS



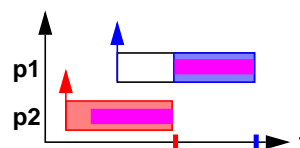
Ordonnancement statique ou dynamique ?

- ordonnancement dynamique** : actuellement peu ou pas d'analyse du comportement global
- ordonnancement statique** : quel politique ? Rate monotonic, Earliest Deadline First, ...

Pb de l'inversion de priorité

Priorité (p1) > Priorité (p2)

p1 & p2 : ressource commune R



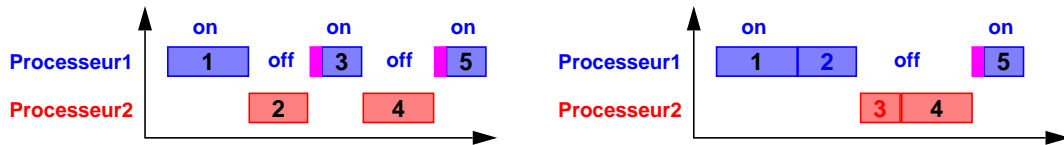
34

 **Changement de contexte entre deux tâches : consomme du temps**

Choix de la granularité des tâches, choix des événements

 **Ordonnancement avec réduction de la consommation d'énergie**

Passer de actif <-> mise en veille d'un processeur nécessite du temps



 **Ordonnancement monoprocesseur ou distribué ?**

Analyse comportement global avec ordonnancement distribué : plus délicat

Peu de vérification de propriétés : exemple toutes les tâches vérifient elles leur deadline ?

35

Conclusion sur les architectures

- ❑ **Concevoir les architectures embarquées à partir des composants les mieux adaptés vis à vis des traitements à exécuter**
 - ❑ **Objectifs** : globalement performances, consommation, time-to-market,... sont optimisés
 - ❑ **Hétérogénéité et évolution des composants** : microcontrôleur, DSP, ASSP, ASIP
 - Stations de base UMTS : DSP VLIW ou multi-MAC ou superscalaire + SIMD ?
 - Portable UMTS : RISC + DSP basse consommation + reconfigurable + accélérateurs HW
 - Migration progressive des accélérateurs dans le chemin de données du DSP
- ❑ **Des constructeurs proposent des architectures mixtes**
 - Philips/VLSI Technology VVS3771 : ARM7TDMI + OakDSPCore
 - Motorola DSP56651 : RISC M-CORE + DSP56600
- ❑ **Optimiser l'organisation et l'utilisation de la mémoire des composants**
 - La mémoire sur le composant** : rapide mais coûteuse en surface
 - Accès en mémoire externe au composant** : moins cher mais plus lent et plus coûteux en énergie
 - Défaut de cache** : consomme environ 6 fois plus d'énergie qu'une opération ALU

36

❑ **Manque d'efficacité des compilateurs pour les processeurs DSP, ASIP**

Evolutions ? architecture plus régulière, progrès en compilation

Spécialisation diminuée : performances/mm² diminuée aussi

❑ **Le comportement global dépend de l'exécutif temps réel (RMS, EDF, ...)**

❑ **Adéquation Algorithme/Architecture**

Choix des couples <fonctions, composants>

Algorithme plus rapide sur composant plus lent ?

❑ **Devant la diversité des solutions, choisir les composants d'un système est délicat**

- ❑ Est-ce que le système va satisfaire les contraintes temporelles de l'application ?
- ❑ Peut-on trouver une solution plus optimisée en surface, en consommation ?
- ❑ Le *time-to-market* est-il allongé ?
- ❑ L'architecture est-elle aisément testable ?
- ❑ Est-elle flexible pour supporter des corrections d'erreurs, des évolutions ?

37

Plan de l'exposé

1 - Pourquoi la co-conception (codesign)

2 - Modélisation des applications

3 - Les architectures embarquées



4 - Estimation logicielle et matérielle

5 - Partitionnement

Partitionnement manuel

Partitionnement automatique

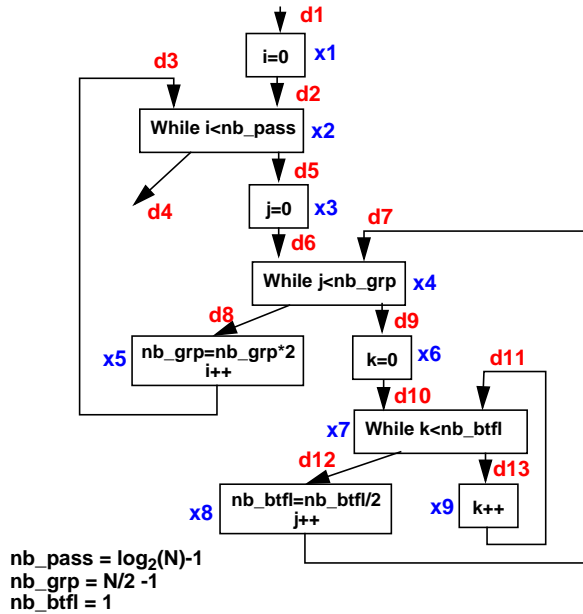
6 - Synthèse des communications

7 - Conclusion

38

Estimation du temps d'exécution maximum : Approche par programmation linéaire

[Malik 95]



Flux entrant = Flux sortant

$$\begin{aligned} x1 &= d1 = d2 \\ x2 &= d2 + d3 = d4 + d5 \\ x3 &= d5 = d6 \end{aligned}$$

Contraintes structurelles

$$\begin{aligned} x8 &= d12 = d7 \\ x9 &= d13 = d11 \\ x2 &= nb_pass * x1 \\ x4 &= nb_grp * x3 \\ x7 &= nb_btfl * x6 \end{aligned}$$

$$x1.c1 + x2.c2 + \dots + x9.c9 \text{ Maximum}$$

(ci = nb cycles pour exécuter le bloc i)

👉 Surestimation de 500% pour N = 1024

Ajout de 2 contraintes :

$$\begin{aligned} nb_grp * nb_btfl &= N/2 \\ x6 &= N * x1 \end{aligned}$$

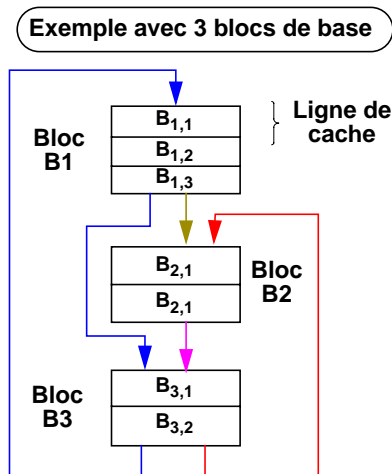
Contraintes fonctionnelles

👉 On obtient les valeurs des xi correctes

41

Estimation avec cache d'instruction

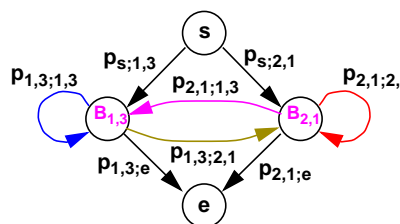
[Malik 95]



Lignes de cache	Blocs de base	
0	B _{1,1}	B _{3,1}
1	B _{1,2}	B _{3,2}
2	B _{1,3}	B _{2,1}
3		B _{2,1}

Conflicts

Contraintes supplémentaires

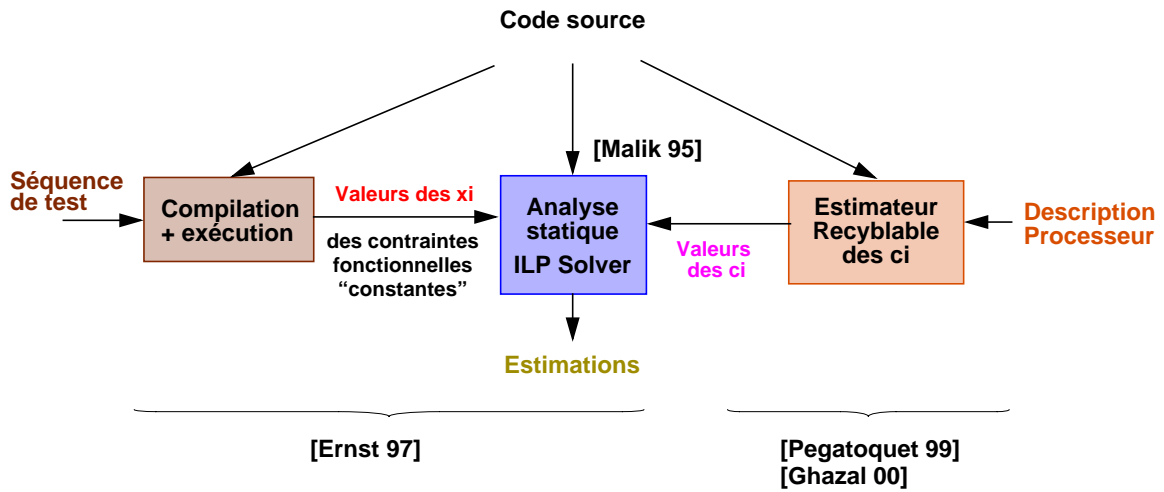


$$\begin{aligned} p_{i,j;i,j} &= x_{i,j}^{\text{hit}} \\ x_i &= \sum_{u,v} p_{u,v;i,j} = \sum_{u,v} p_{i,j;u,v} \\ p_{i,j;u,v} &\leq \min(x_i, x_u) \\ \sum_{u,v} p_{s;u,v} &= 1 \end{aligned}$$

👉 La méthode suppose les valeurs des ci connues.

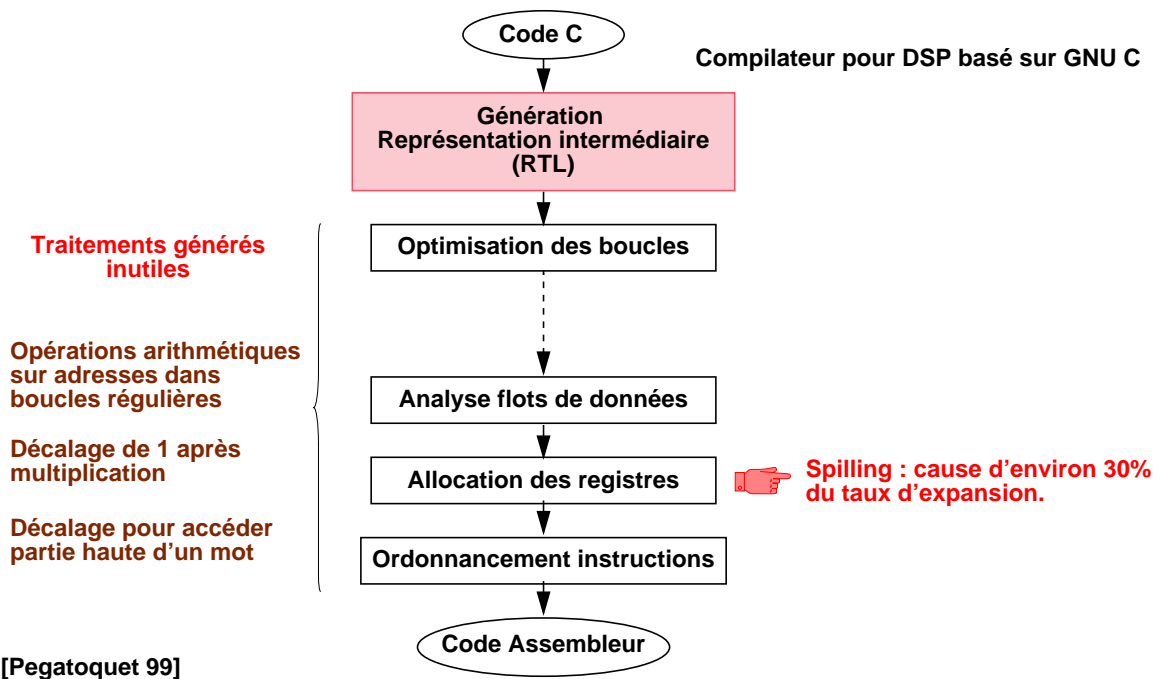
42

Estimation logicielle : Estimateur recyclable



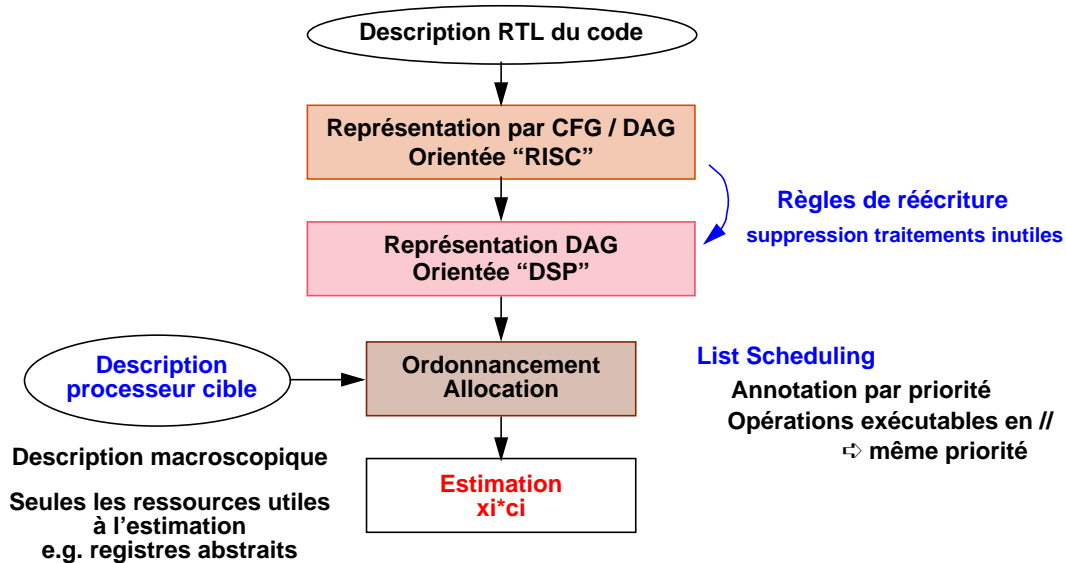
43

Estimateur recyclable: VESTIM (I3S - Philips Semiconductors Sophia)



44

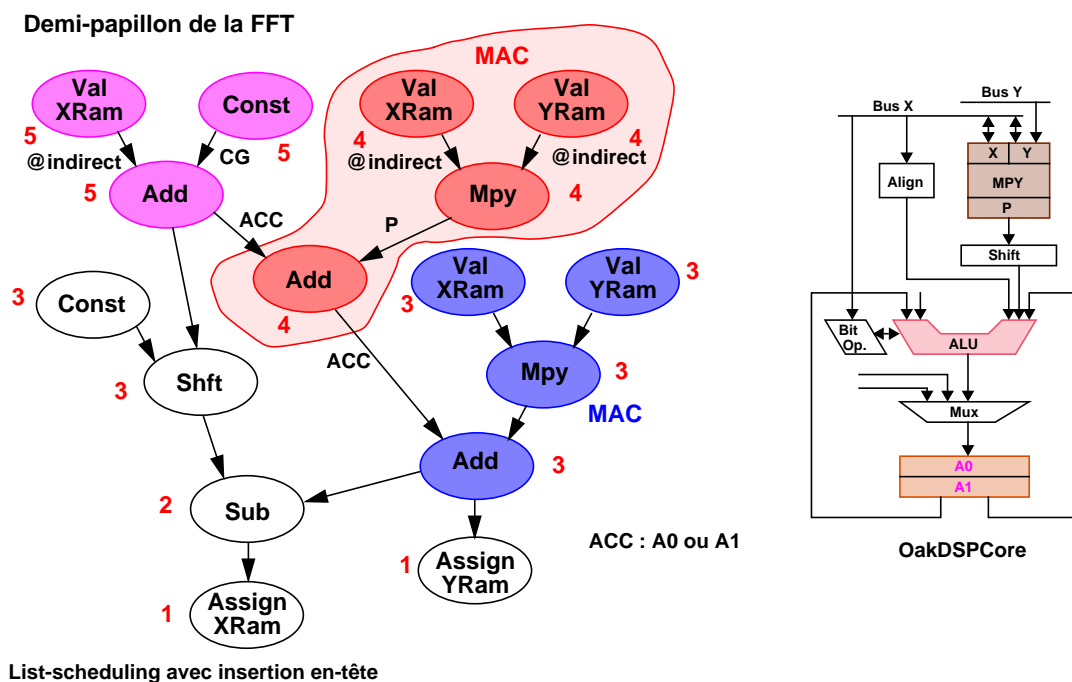
Estimateur recyclable : VESTIM



45

Ordonnancement au niveau opération (adapté aussi aux DSP VLIW, multi-MAC)

Estimateur recyclable : VESTIM



46

Résultats d'estimation

	Nb lignes de code C (x 1000)	Code ASM optimisé (MIPS)	Code ASM généré (MIPS)	Code estimé (MIPS)	Durée
Encodeur G728	1,1	23	62,3	26,7	2 s
G729	5,6	12,2	21	15,7	1,5 mn
G723.1	2,7	23,5	32	22	2 mn
Décodeur audio AC3	4	39,5	75	32	1 mn
Modem V22bis	19	4	10	5,2	1 s



Les estimations d'un code assembleur optimisé sont précises



L'architecture reste assez simple (pas de cache, parallélisme ILP intra-bloc de base)

47

Conclusion sur l'estimation

La complexité des architectures rend difficile le problème de l'estimation

Estimation logicielle :

Aléas de pipeline, lancement dynamique des instructions, superscalaire

Hierarchie mémoire : caches, répartition des données

Système d'exploitation, communications (DMA)

Des approches existent pour des cas simples

Approche pratique : mesures sur l'architecture elle-même : développer le code

Estimation matérielle :

Estimation temps à opérateurs connus ou estimation opérateurs à temps fixé

Influence du placement/routage, influence de la technologie

Approche IP caractérisé avec précision : disponibilité ? (e.g. rake receiver UMTS)

Et pourtant avec des estimations précises, la conception devient plus simple et rapide

Au LESTER : Estimation hiérarchisée

Estimations systèmes : caractérisation macroscopique

Estimation architecturale : caractérisation détaillée

48

Plan de l'exposé

1 - Pourquoi la co-conception (codesign)

2 - Modélisation des applications

3 - Les architectures enfouies

4 - Estimation logicielle et matérielle



5 - Partitionnement

Partitionnement manuel

Partitionnement automatique

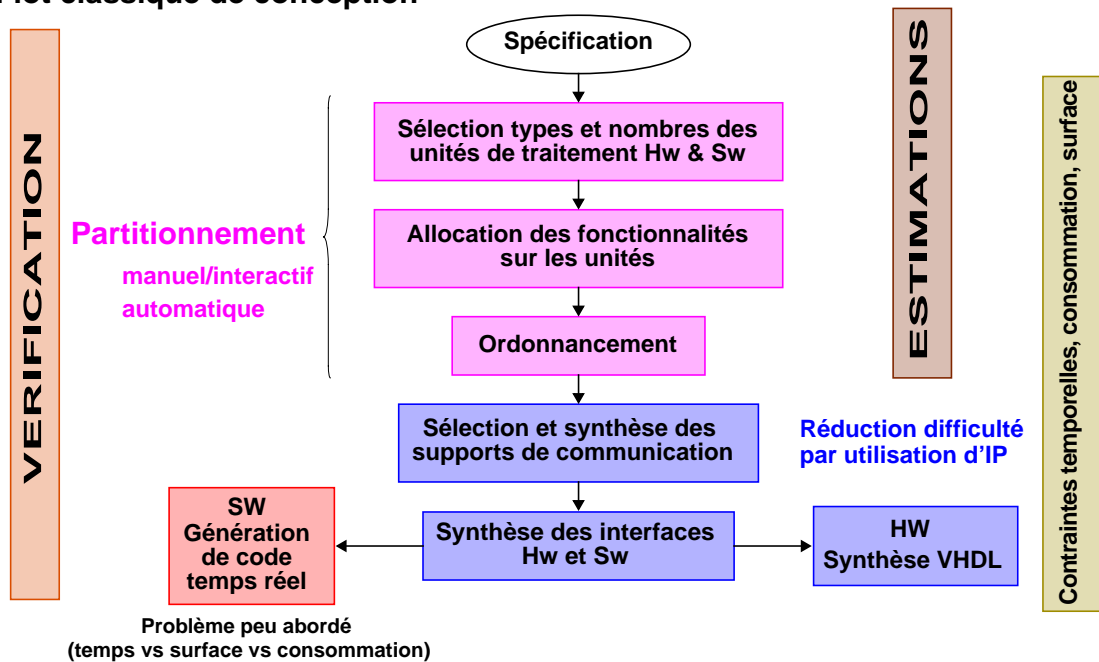
6 - Synthèse des communications

7 - Conclusion

49

PARTITIONNEMENT

Flot classique de conception

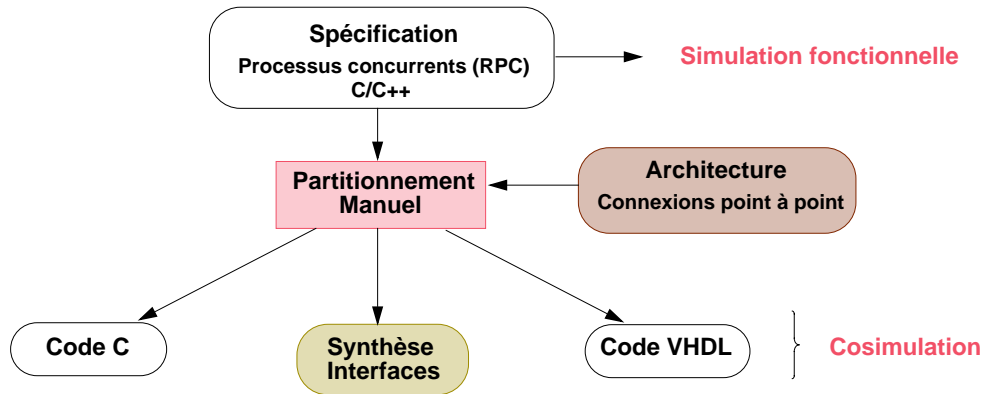


50

Partitionnement manuel : CoWare

Conception de SOC par partitionnement manuel : utilise généralement la cosimulation

Exemple : l'environnement CoWare

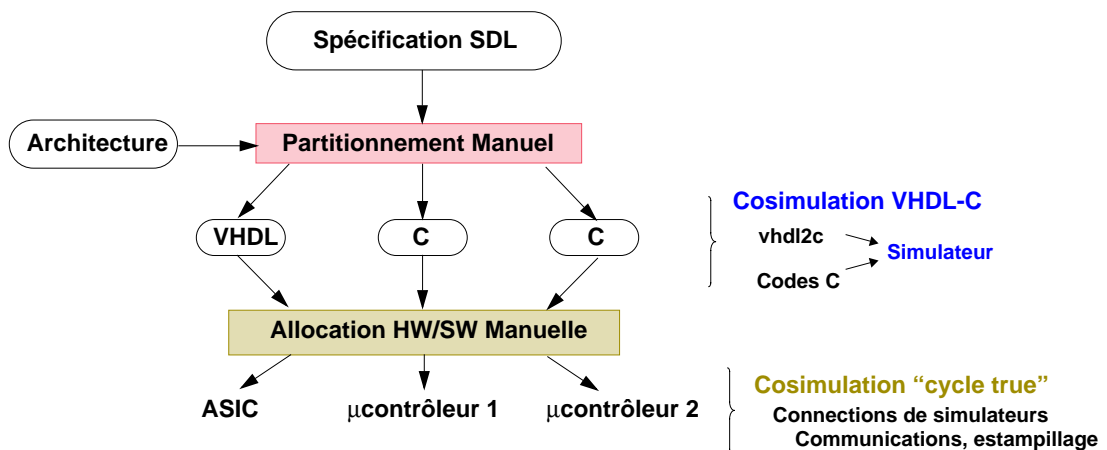


51

Partitionnement manuel : Tima/Arexsys

Environnement de conception à partir de **SDL** : Specification & Description Language

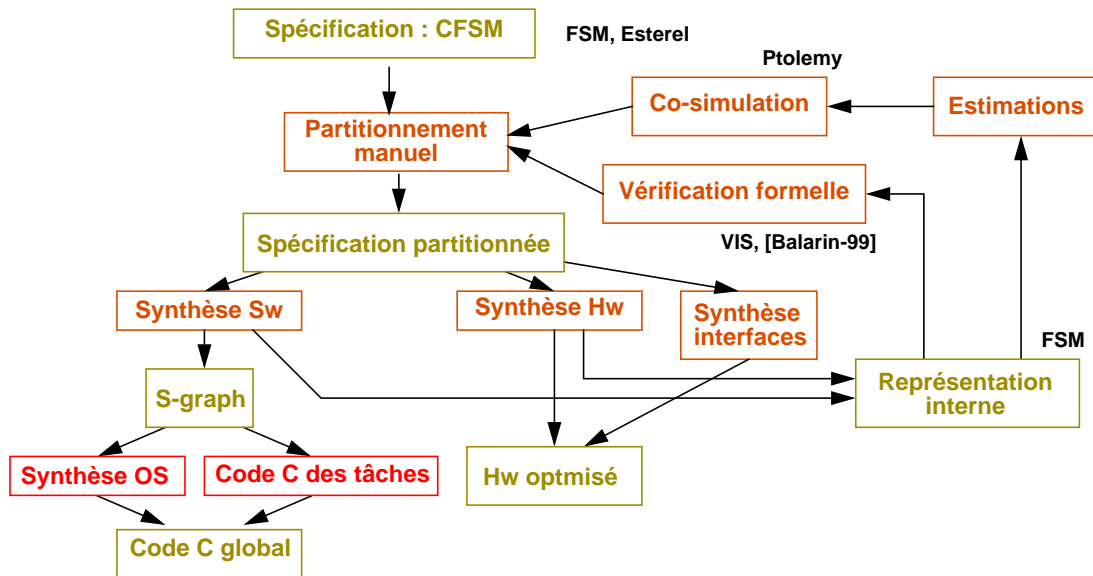
- ❑ Standard ITU-T
- ❑ Langage graphique : Processus communicants, par passage de messages
- ❑ Description de protocoles



52

Partitionnement manuel : exemple l'environnement POLIS (Berkeley)

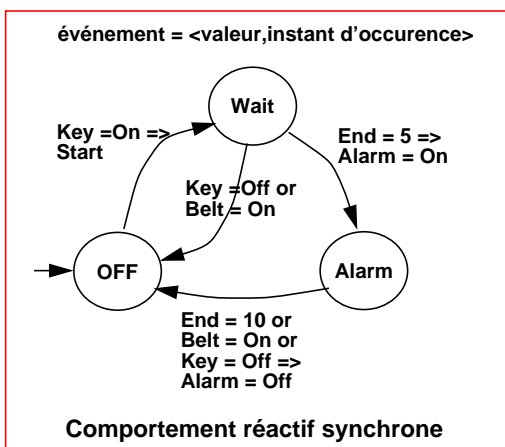
- POLIS : environnement de codesign pour des applications orientées contrôle/commande



53

Modèle CFMS de POLIS

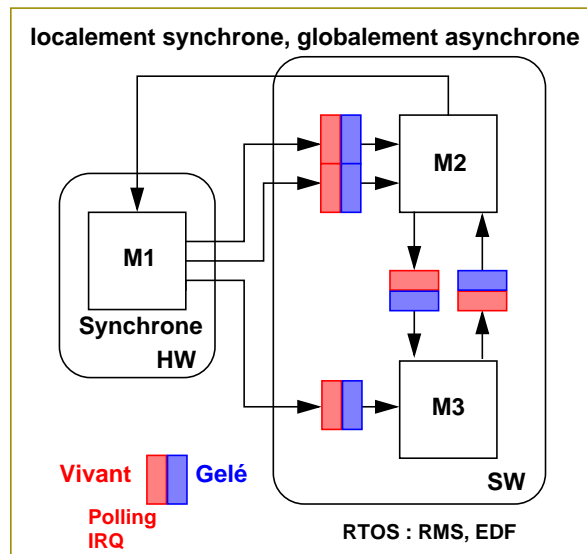
1) Un module : Machine d'état étendue



3) Analyse de l'ordonnançabilité

Estimations temps exécution
Politique d'ordonnancement
Ensemble de contraintes temporelles
Aucun événement perdu ? [F. Balarin]

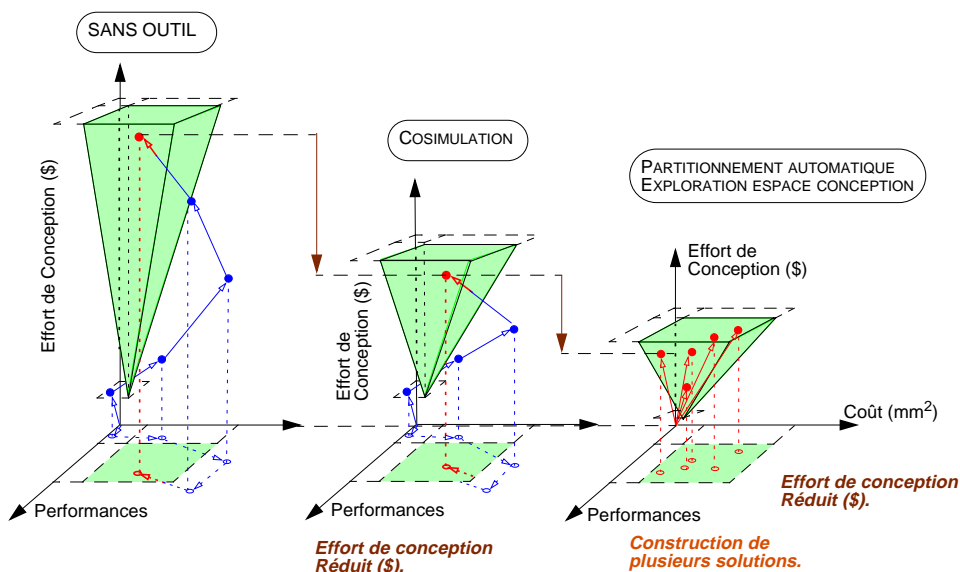
2) réseau de modules (avec partitionnement)



Réseau de Modules : outil VCC de Cadence
Description d'un Module : C, C++ ou ECL

54

Conception manuelle vs cosimulation vs partitionnement



55

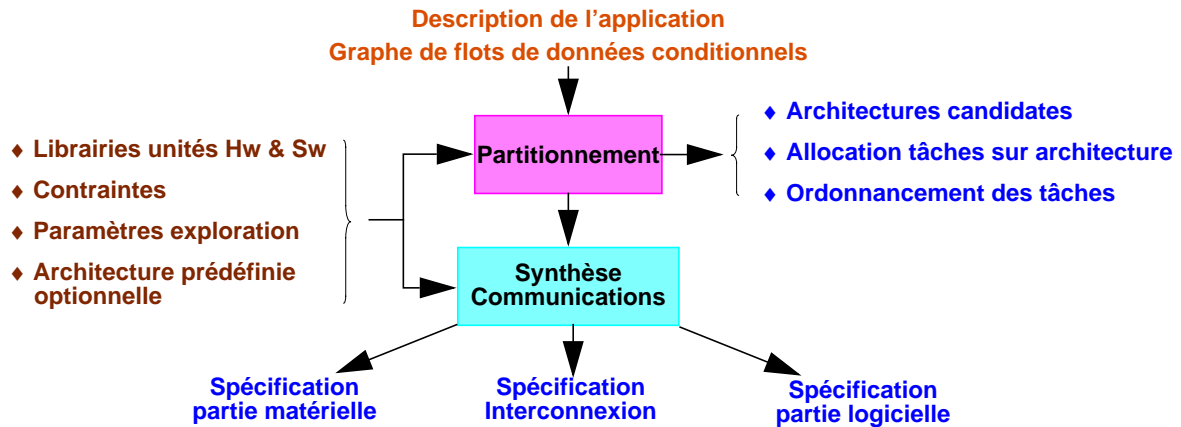
Partitionnement automatique : nombreuses techniques proposées

Comparaisons de quelques techniques

	Modèle/Granularité	Architecture	Objectif	Technique	Contrôle	Nombre solutions
COSYMA [Ernts 93]	C étendu/Instruction	Figée: 1 proc. 1 Asic	Temps	Recuit Simulé, SW -> HW	nonpréemptif compile time	1
Vulcan [De Micheli 94]	C/Instruction	Figée: 1 proc. 1 Asic	Temps	HW -> SW Glouton	nonpréemptif	1
GCLP [Kalavade 94]	DFG/Tâche	Figée: 1 proc. 1 Asic	Temps/Surface	List scheduling modifié	nonpréemptif compile time	1
[Vahid 97]	C/Fonction	Figée: 1 proc. 1 Asic	Temps/Surface/ IO	Min Cut	nonpréemptif compile time	Exploration
[Teich 97]	C/Fonction	SOC	Temps	Algorithme génétique	?	Exploration
COSYN [Dave 97]	Graphe de tâches	Multiprocesseur	Temps/Consommation	Clustering	préemptif/non-préemptif	1
[Karkowski 98]	C/boucles	multi-ASIC	Temps/nb proc.	Enumération	séquentiel	Exploration
[Kuchcinski 99]	DFG/Tâche	SOC	Temps/Surface	Prog. par contraintes	compile time	Exploration
[Li 00]	C/boucles	1 proc. 1 FPGA	Temps	Clustering	compile time	1
SynDEX [Sorel]	DFG/Tâche	multiprocesseur	Temps	List Scheduling modifié	nonpréemptif compile time	1
[Oh 99]	Graphe de Tâches	multiprocesseur	Temps/Surface	Ordonnancement	préemptif	1
[Wolf 95]	Graphe de Tâches	multiprocesseur	Temps/Surface	Ordonnancement	préemptif	1

56

Méthode développée dans CODEF (I3S - Philips Semiconductors Sophia)



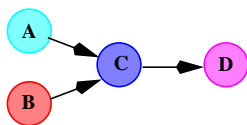
CODEF opère en deux étapes

- ◆ **Partitionnement/Ordonnancement** : exploration de l'espace de conception
- ◆ **Synthèse des Communications** : ressources, protocoles pour réaliser les transferts

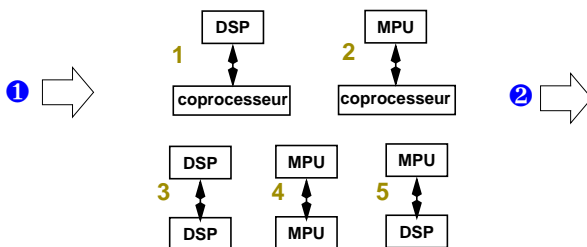
57

Problèmes : sélection architecture/partitionnement/ordonnancement

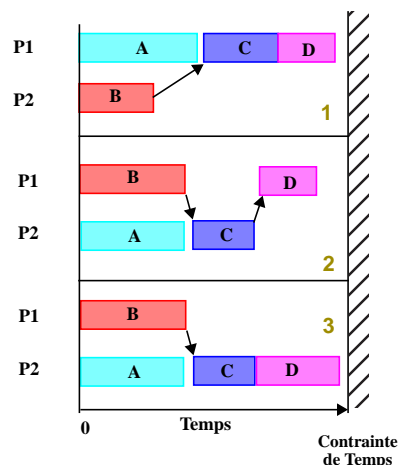
Graphe flots de données



Quelle Architecture ?



Allocation et ordonnancement ?



Éléments architecturaux dans un SOC :

- ◆ RISC, DSP, coprocesseurs, accélérateurs Hw
- ◆ Mémoires locales/externes : data & programme
- ◆ Caches
- ◆ Ports I/O, caractéristiques des bus ...

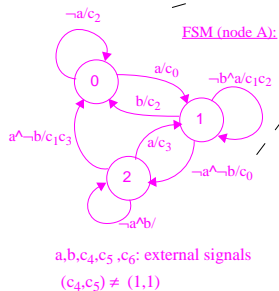
Ordonnancement statique non préemptif

58

Modèle de l'application dans CODEF

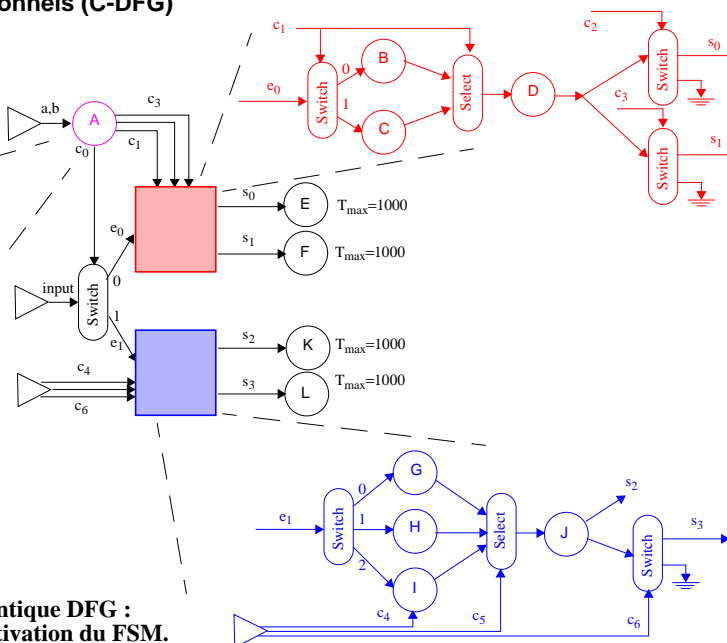
Graphes de flots de données conditionnels (C-DFG)

◆ *Exemple :*



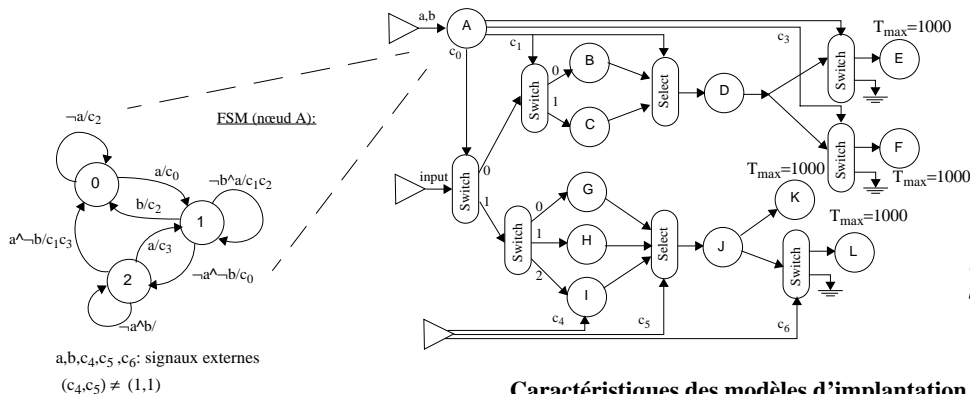
Application de traitement du signal
(SPW, Ptolemy)

➔ Les nœuds des FSM ont la sémantique DFG :
une seule transition à chaque activation du FSM.



59

Librairies des modèles d'implantation



T_{max}
Contraintes
temporelles

◆ *Librairies:*

Surfaces des cœurs

Sw1	Sw2	HwA	HwD	HwE	HwF	HwK	HwL
6.66	6.35	2.31	0.59	1.82	1.89	1.91	1.83

Caractéristiques des modèles d'implantation.

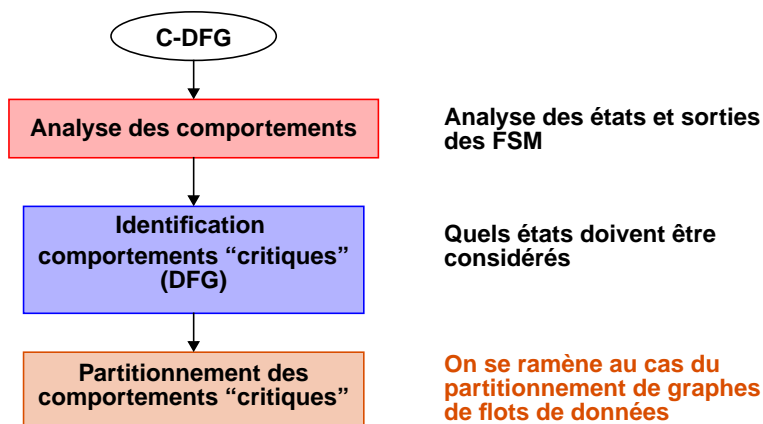
Nodes	Sw1	Sw2	HwA	HwD	HwE	HwF	HwK	HwL
A	329/1.16	409/1.09	38					
B	410/1.55	250/1.10						
C	324/0.21	188/0.42						
D	328/1.17	458/1.40		44				
E	385/0.97	346/1.33			53			
F	402/0.77	318/1.04				74		
G	379/0.67	283/0.84						
H	357/1.15	432/0.93						
I	435/0.94	407/1.17						
J	292/0.11	264/1.02						
K	300/1.13	371/0.87					70	
L	339/0.90	257/1.06						69

Temps exécution/surface mémoire. Temps exécution des accélérateurs

60

Principe de partitionnement d'un C-DFG

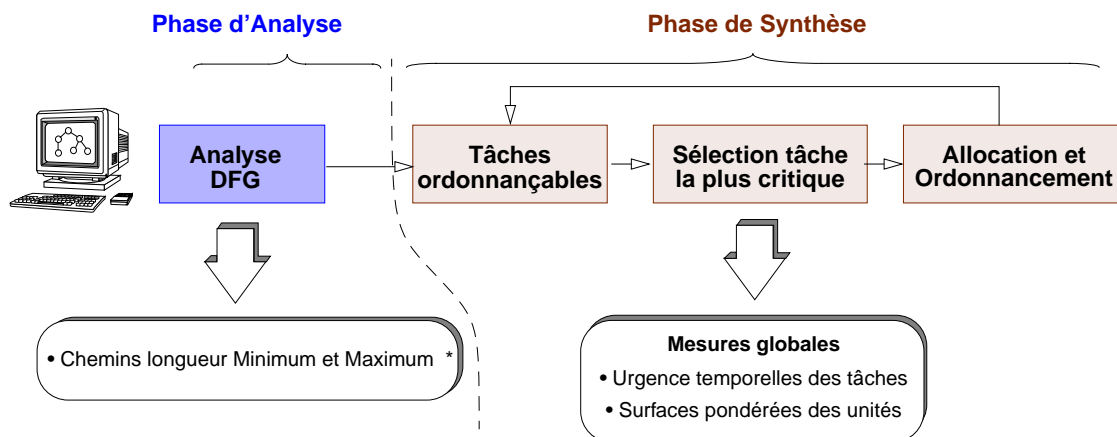
Objectif : Simplifier le problème



61

Partitionnement de graphes de flots de données statiques (DFG)

♦ On considère une heuristique basée sur le *List Scheduling*



Path Analysis Partitioning Algorithm (PA2)
[Bianco-99]

62

* Extension de [P. Bjørn-Jørgensen, J. Madsen - 1997]

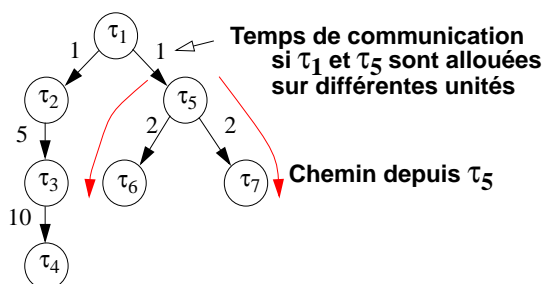
Phase d'analyse de PA2

◆ Evaluation récursive des longueurs des chemins

$\forall \tau_i$ une tâche et $\forall u_k$ une unité (HW ou SW)

$$pl_{\min}(\tau_i, u_k) = t_{\text{exe}}(\tau_i, u_k) + \text{Max} \left(\text{Min}_{\tau_j \in \text{succ}(\tau_i), u_1} \left(t_{c, u_k, u_1}(\tau_i, \tau_j) + pl_{\min}(\tau_j, u_1) \right) \right)$$

Exemple simplifié



Temps d'exécution des tâches sur 3 unités

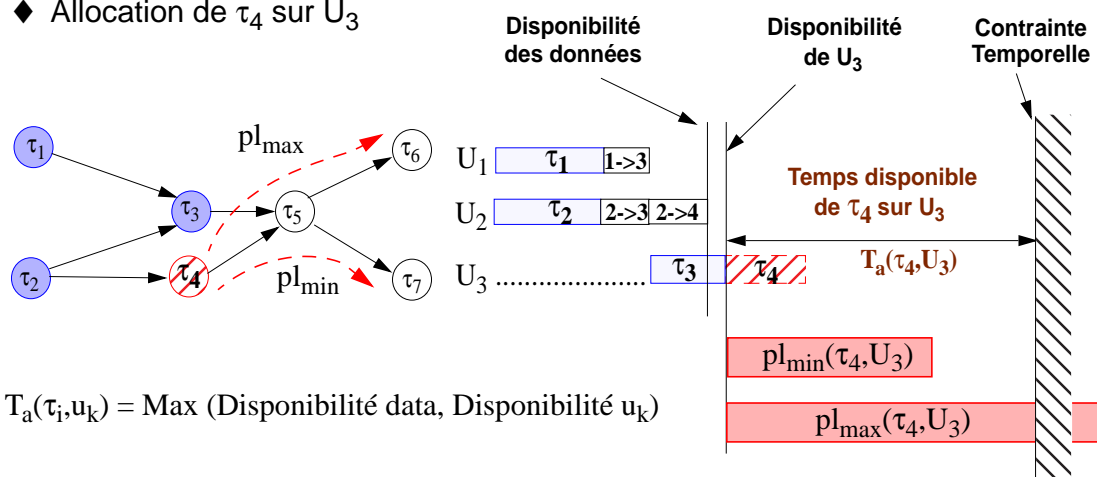
Tâche	U1	U2	U3
τ_1	2	1	2
τ_2	4	1	3
τ_3	6	2	3
τ_4	5	3	4
τ_5	3	3	2
τ_6	6	2	3
τ_7	3	1	1

$$pl_{\min}(\tau_5, U_1) = 3 + \text{Max}(\text{Min}((0+6, 2+2, 2+3), \text{Min}(0+3, 2+1, 2+1))) = 7$$

63

Phase de synthèse de PA2 : Urgence temporelle

◆ Allocation de τ_4 sur U_3



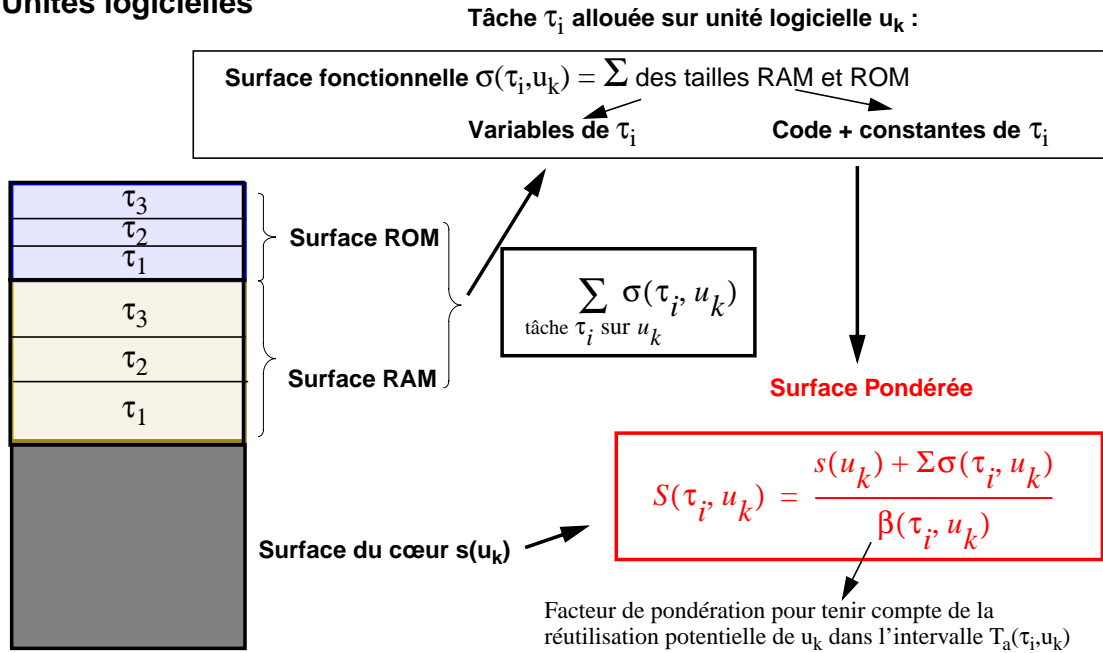
$$T_a(\tau_i, u_k) = \text{Max} (\text{Disponibilité data}, \text{Disponibilité } u_k)$$

- Comparaison de $T_a(\tau_i, u_k)$ avec pl_{\min} et $pl_{\max} \Rightarrow$ Mesure de l'urgence de τ_i sur u_k

64

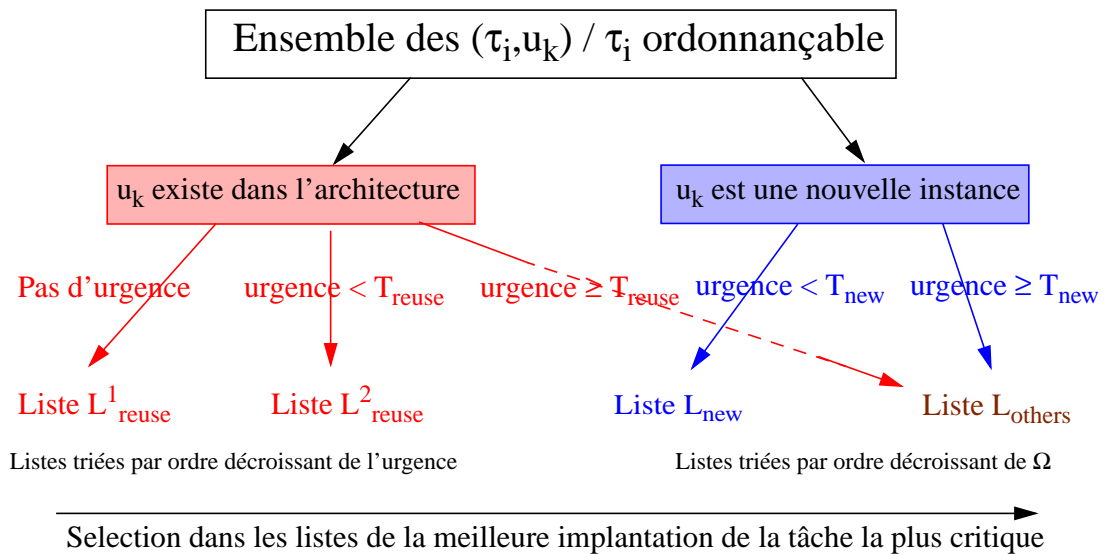
Phase de synthèse de PA2 : Surface pondérée

◆ Unités logicielles



65

Phase de synthèse de PA2 : Algorithme de partitionnement



$$\Omega = \alpha(\text{urgence relative}) + (1-\alpha)(\text{surface pondérée relative})$$

$T_{new}, T_{reuse}, \alpha$: trois paramètres définis par l'utilisateur

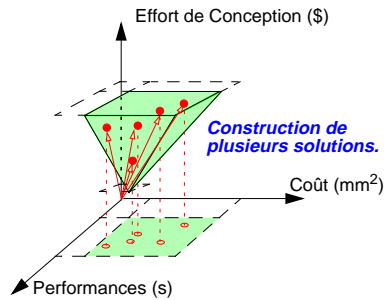
66

Phase de synthèse de PA2 : Exploration automatique de l'espace de conception

- ◆ L'algorithme de partitionnement/ordonnancement est récursif

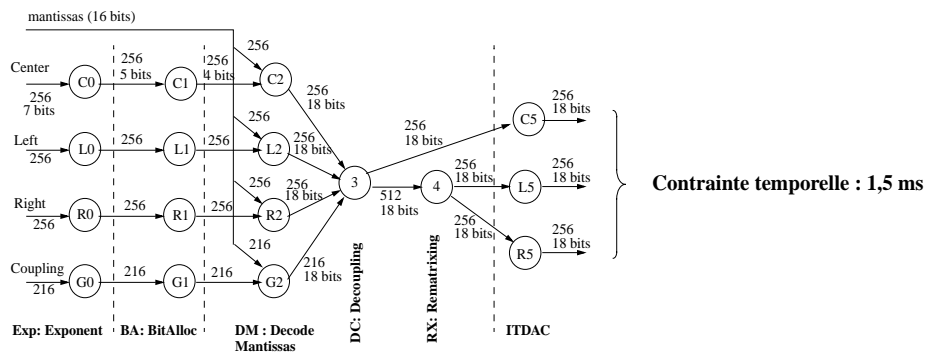
A l'exécution N les variations minimum sur T_{new} and T_{reuse} sont calculées telles qu'à l'exécution N+1 une allocation différente soit réalisée

- ◆ L'algorithme construit un ensemble de solutions qui correspondent à différentes répartitions des tâches dans le temps (ordonnancement) et/ou dans l'espace (allocation)



67

Partitionnement statique : Exemple d'un décodeur audio AC3 (3 canaux)



Temps d'exécution(μs), ROM (octets), RAM (octets) des tâches

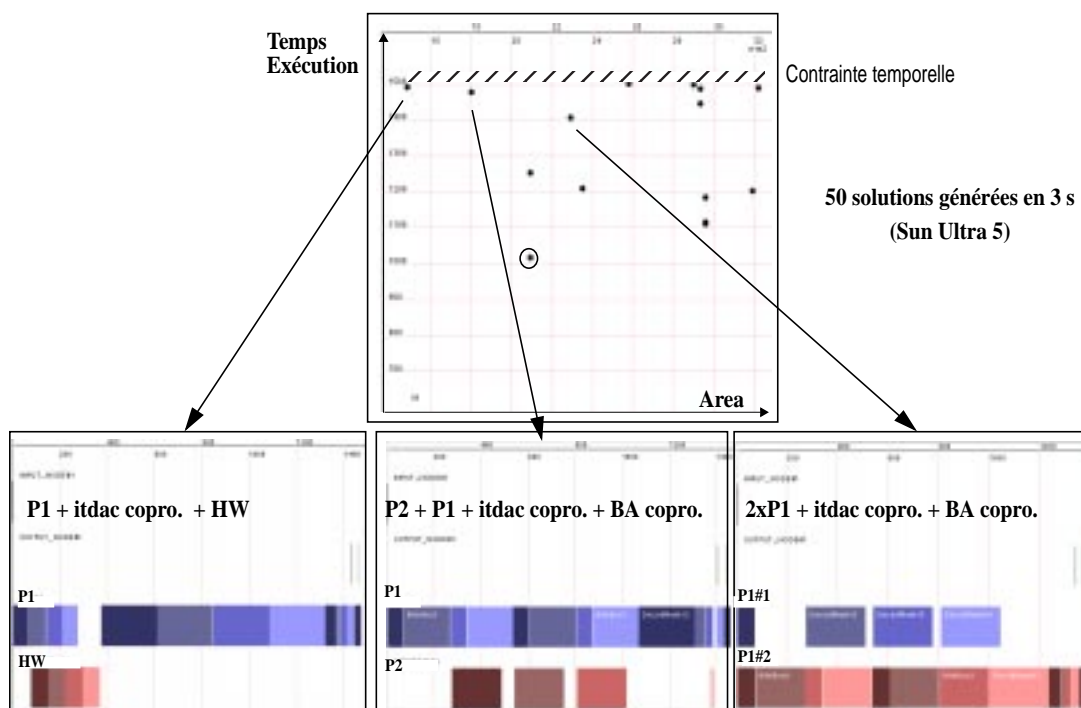
Unités: { 2 processeurs P1, P2
1 accélérateur HW

Unité	Surface Cœur
P1	11 mm ²
P2	5 mm ²
HW	3.5 mm ²

	Processeur P1	Processeur P2	accélérateur HW
Exp	Sw: 69 / 774 / 258	Sw: 81 / 920 / 258	-
BA	Sw: 532 / 1092 / 200	Sw: 589 / 1680 / 200	Hw: 73 / 0 / 0
	Sw+coprocessor: 197/800/200		
DM	Sw: 237 / 274 / 150	Sw: 213 / 392 / 150	-
DC	Sw: 47 / 120 / 48	Sw: 52 / 190 / 48	-
RX	Sw: 10 / 20 / 0	Sw: 13 / 42 / 0	-
ITDAC	Sw: 130 / 776 / 384	-	-
	Sw+coprocessor: 31 / 311/384		

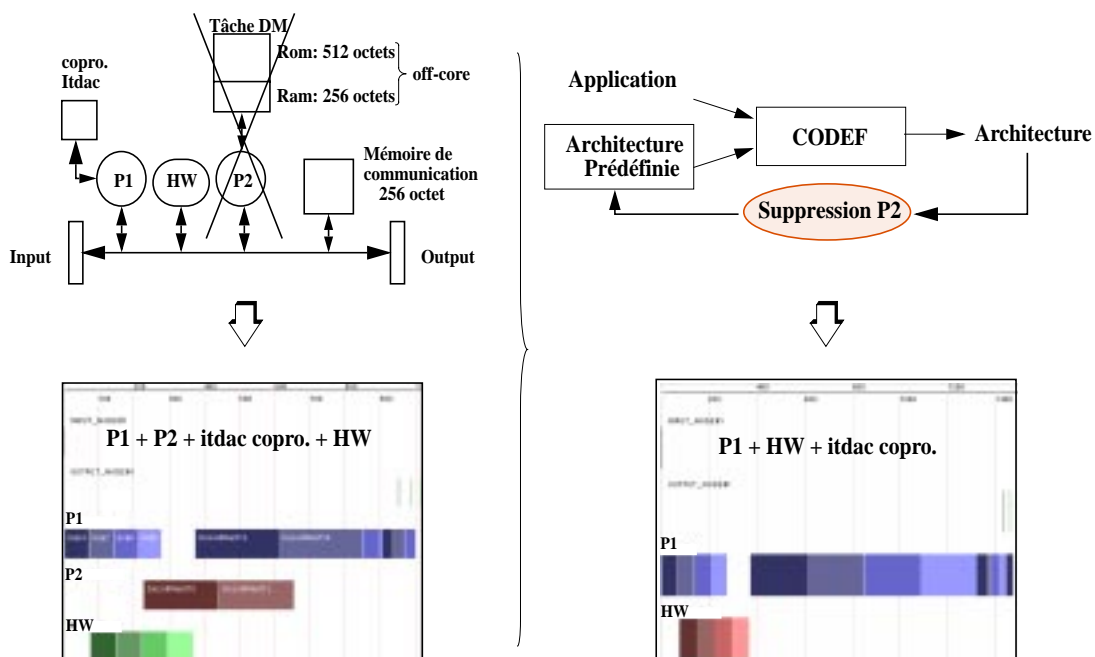
68

Résultats de l'exploration de l'espace de conception



69

Exemple d'optimisation interactive



70

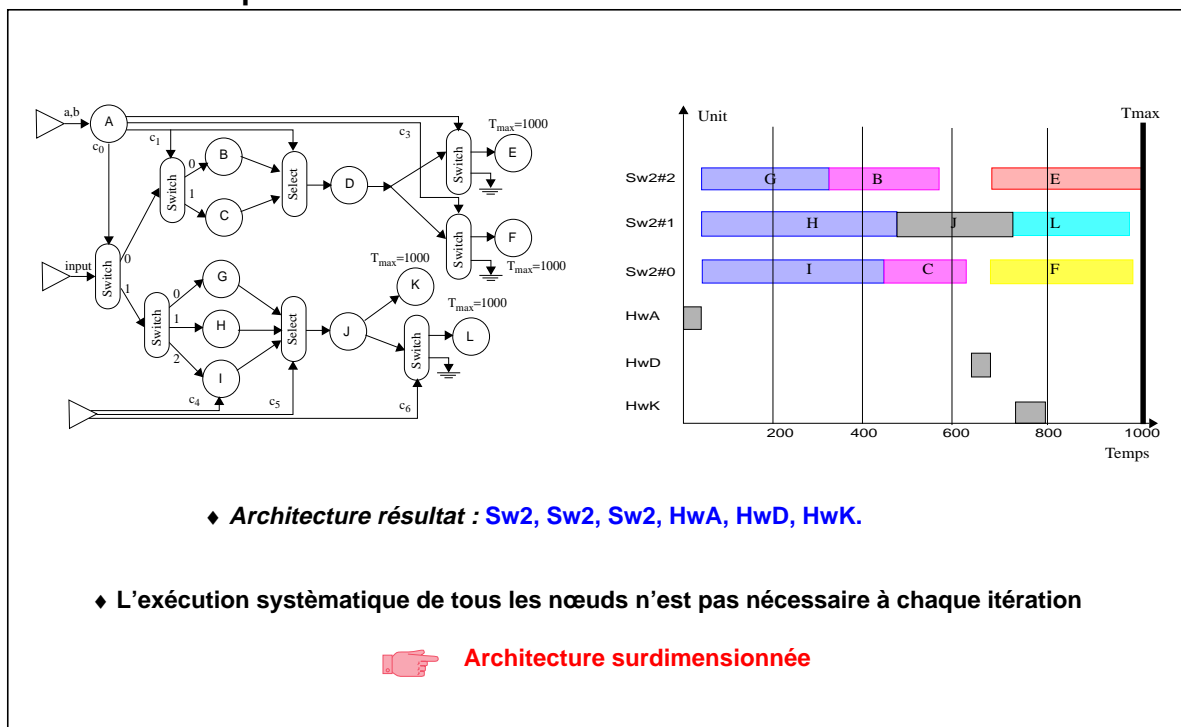
Conclusions sur le partitionnement statique

- ◆ L'algorithme de partitionnement prend en compte :
 - ❑ Unités logicielles, matérielles, Coprocesseurs, Accélérateurs matériels
 - ❑ Mémoire cache (L1) des processeurs (modèle simple)
 - ❑ Mémoires RAM/ROM locales ou externes des processeurs
 - ❑ Répartition des données et programmes dans les hiérarchies mémoires (glouton)
 - ❑ L'architecture prédéfinie optionnelle peut contenir :
 - Instances d'unités HW & SW
 - Allocations de tâches à ces instances
 - Hiérarchie mémoire des processeurs,
 - Interconnexion

◆ Mais partitionner un C-DFG avec PA2 peut être inefficace (surface, consommation)

71

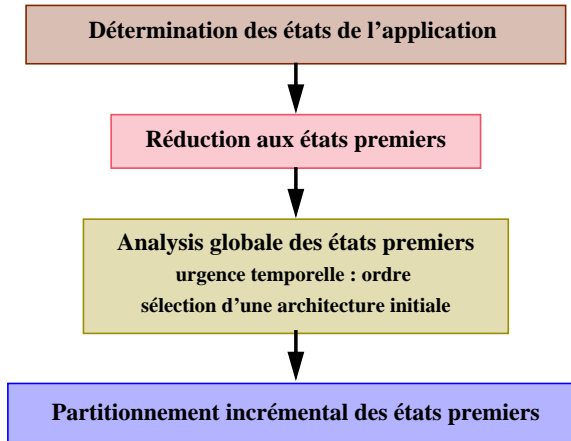
Exemple d'inefficacité : le C-DFG considéré comme un DFG



72

Partitionnement de graphes de flots de données conditionnels (C-DFG)

- ◆ Par rapport à un DFG, seuls les nœuds autorisés par les contrôles doivent être exécutés dans un C-DFG
- ◆ On considère une approche incrémentale de partitionnement :



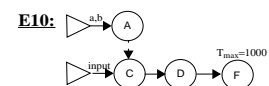
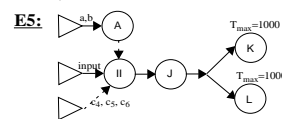
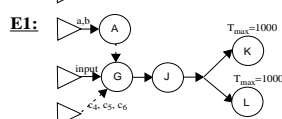
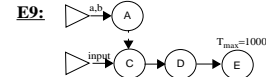
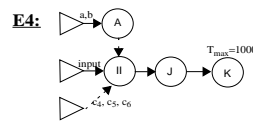
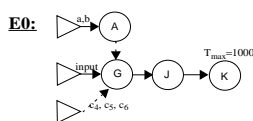
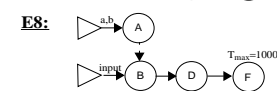
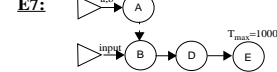
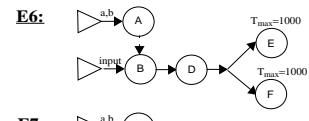
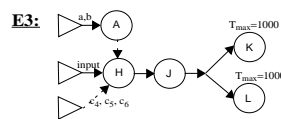
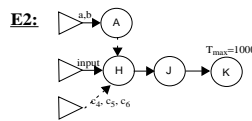
[Capella-00]

73

Analyse des états de l'application

- ◆ De l'analyse des états et des sorties des FSM on déduit :

États	c_0, \dots, c_6	sous-graphes
E0	1---000	AGJK
E1	1---001	AGJKL
E2	1---010	AHJK
E3	1---011	AHJKL
E4	1---100	AJJK
E5	1---101	AJJKL
E6	0000---	ABDEF
E7	0001---	ABDE
E8	0010---	ABDF
E9	0101---	ACDE
E10	0110---	ACDF



74

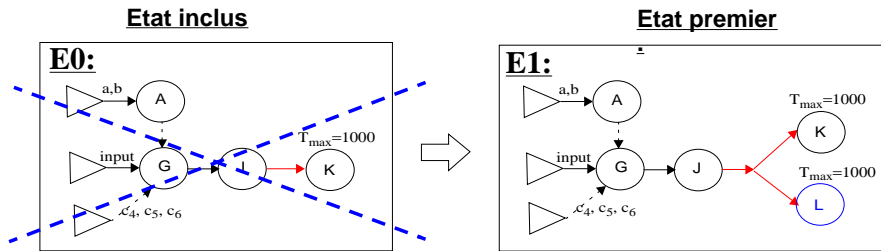
Réduction des états de l'application : les états premiers

Remarque :

- ◆ Si une architecture respecte les contraintes temporelles pour un DFG, elle les respecte aussi pour tout sous-graphe de ce DFG.

➔ L'approche statique est basée sur cette remarque.

- ◆ On ne considère que les **états premiers** pour déduire l'architecture finale : Un **état premier** est un état non inclus dans tout autre état de l'application.



Etats premiers de l'exemple

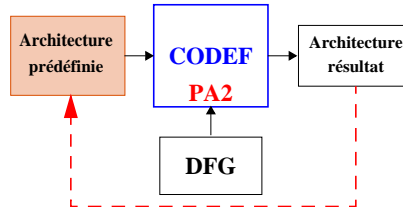
Etats	c_0, \dots, c_6	Sousgraphes	Etat englobant
E0	1---000	GJK	E1
E1	1---001	GJKL	
E2	1---010	HJK	E3
E3	1---011	HJKL	
E4	1---100	IJK	E5
E5	1---101	IJKL	
E6	0000---	BDEF	
E7	0001---	BDE	E6
E8	0010---	BDF	E6
E9	0101---	CDE	
E10	0110---	CDF	

Partitionnement incrémental

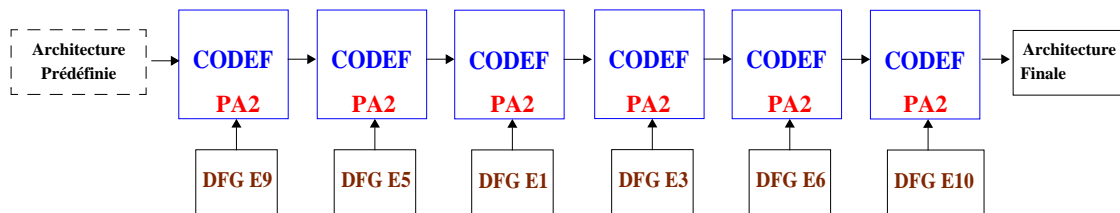
◆ Choix d'une approche incrémentale : contexte industriel

Nombre d'états et taille du graphe d'applications industrielles
Compromis qualité des solutions/ temps exécution du partitionnement

Utilisation de l'entrée *architecture prédéfinie* de CODEF

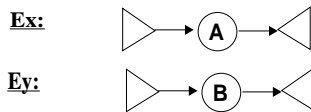


◆ Conception incrémentale :



Le partitionnement incrémental est sous-optimal

◆ Exemple :



Surface

	P1	P2	P3
Cœurs	100	100	100
A	<u>40</u>	100	60
B	110	<u>30</u>	40

} Surface Mémoire

250 230 200

◆ Partitionner **Ex avant Ey** ⇒ P1 est sélectionné ⇒ 250 unités de surface

◆ Partitionner **Ey avant Ex** ⇒ P2 est sélectionné ⇒ 230 unités de surface

Ey->Ex meilleur ordre que Ex->Ey

Les résultats dépendent de l'ordre d'analyse des états premiers

Mais la solution avec 200 unités de surface (i.e. P3) ne peut être trouvée

Amélioration du partitionnement incrémental

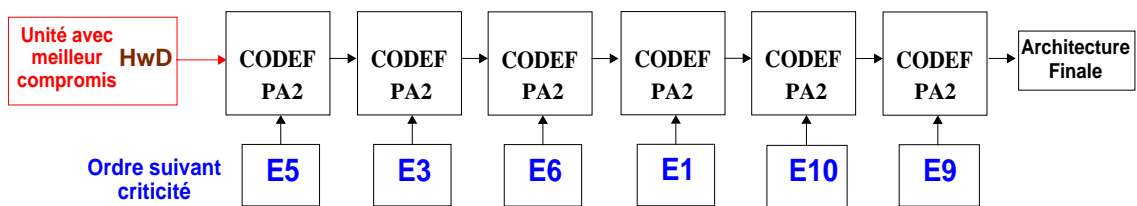
- ➔ Dans l'architecture prédéfinie initiale on place l'unité avec le meilleur compromis surface/réutilisation potentielle, en fonction des contraintes temporelles.

$$S(\tau_i, u_k) = \frac{s(u_k) + \sum \sigma(\tau_i, u_k)}{\beta(\tau_i, u_k)} \quad \forall \text{ état premier} \\ \forall u_k$$

- ➔ Les états premiers sont partitionnés dans l'ordre décroissant de leur criticité

$$\text{Min}_{\text{états}} \left(\text{Min}_{\tau_i} \left(\text{Max}_{u_k} \left(T_a(\tau_i, u_k) - pl_{\min}(\tau_i, u_k) \right) \right) \right)$$

Sur l'exemple à 6 états premiers



79

Résultats

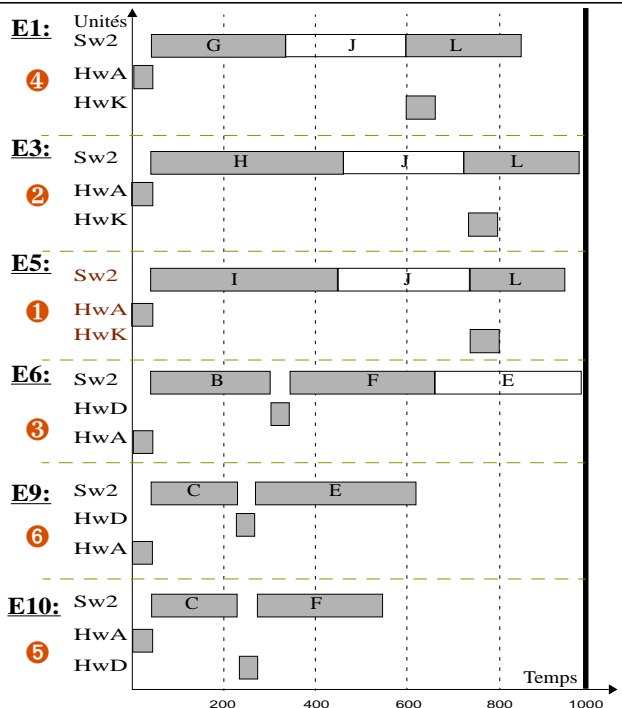
- On obtient une architecture composée de **Sw2, HwA, HwD, HwK**

- Partitionnement du DFG complet **Sw2, Sw2, Sw2, HwA, HwD, HwK**

👉 Deux processeurs en moins

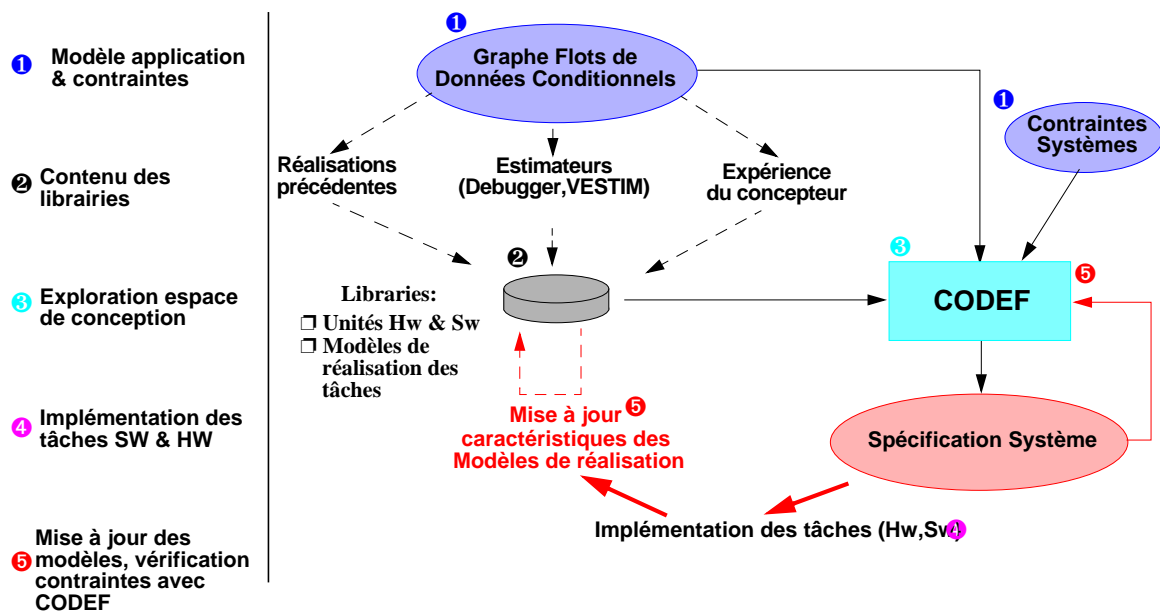
- L'architecture obtenue n'a pas la plus petite surface. Seulement 20% des 720 ordres possibles ont une surface plus petite (la meilleure : - 10%)

- Sur les 720 ordres on a :
< Surface solutions avec HwD dans A_0 < surface solutions partitionnée A_0 vide



80

Etapes de la conception système avec CODEF



CODEF : Vérification des contraintes pendant les étapes de raffinement système (**3**, **5**)

81

Plan de l'exposé

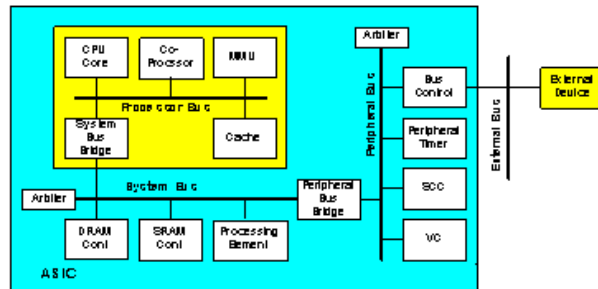
- 1 - Pourquoi la co-conception (codesign)
- 2 - Modélisation des applications
- 3 - Les architectures enfouies
- 4 - Estimation logicielle et matérielle
- 5 - Partitionnement
 - Partitionnement manuel
 - Partitionnement automatique
- 6 - Synthèse des communications
- 7 - Conclusion



82

SYNTHÈSE DES COMMUNICATIONS

La spécification d'une architecture spécialisée inclut l'interconnexion



□ Déterminer une interconnexion implique :

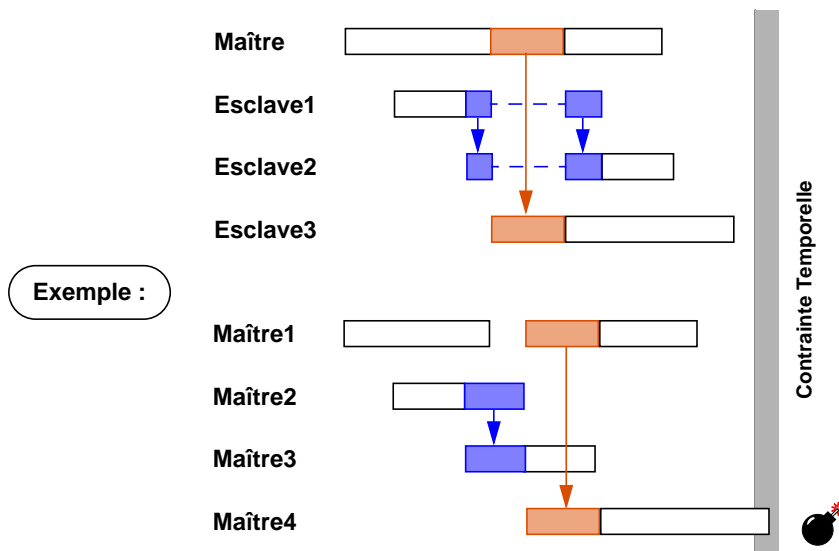
- Choisir les ressources de communication : types et nombres de bus, FIFO, Dual-Port, Arbitre de bus ...
- Allouer les communications de l'application sur les ressources sélectionnées
- Choisir des modes de transfert :
 - protocole de transfert : mode burst, pipeline, ...
 - transfert synchrone ou asynchrone
 - priorités allouées aux transferts

□ Classiquement la synthèse des communications est réalisée :

- Pendant le partitionnement (modèle simplifié utilisé)
- Après partitionnement

83

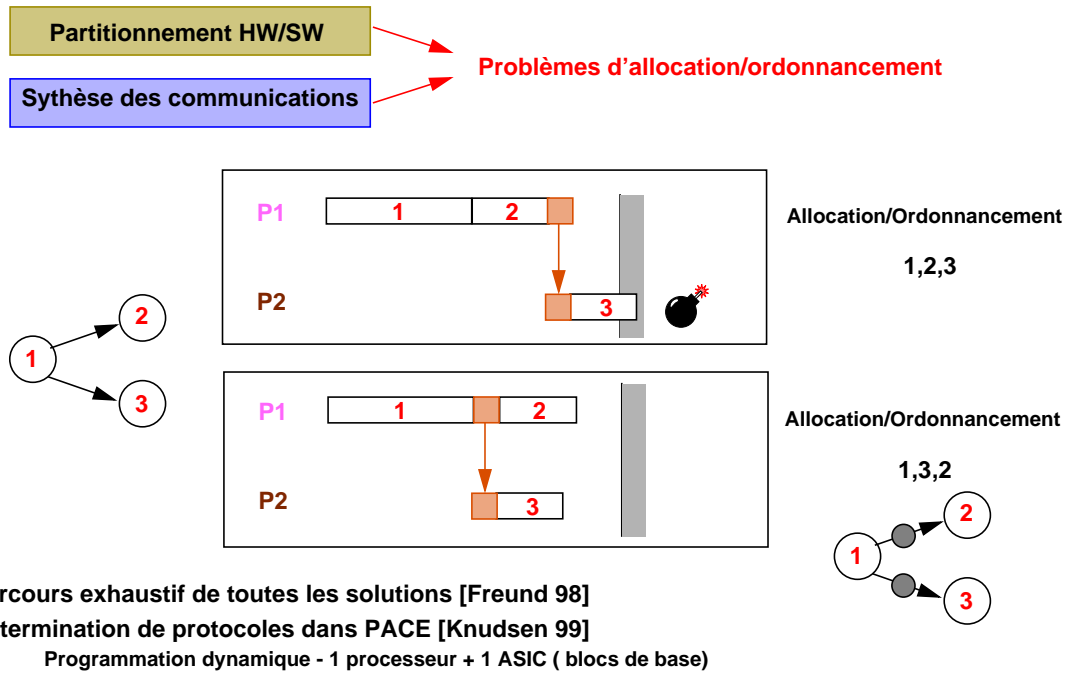
La réalisation des communications conditionne le comportement global



Respecter les contraintes temporelles et minimiser surface de silicium/Consommation
FIFO, Dual-Port, Drivers,...

84

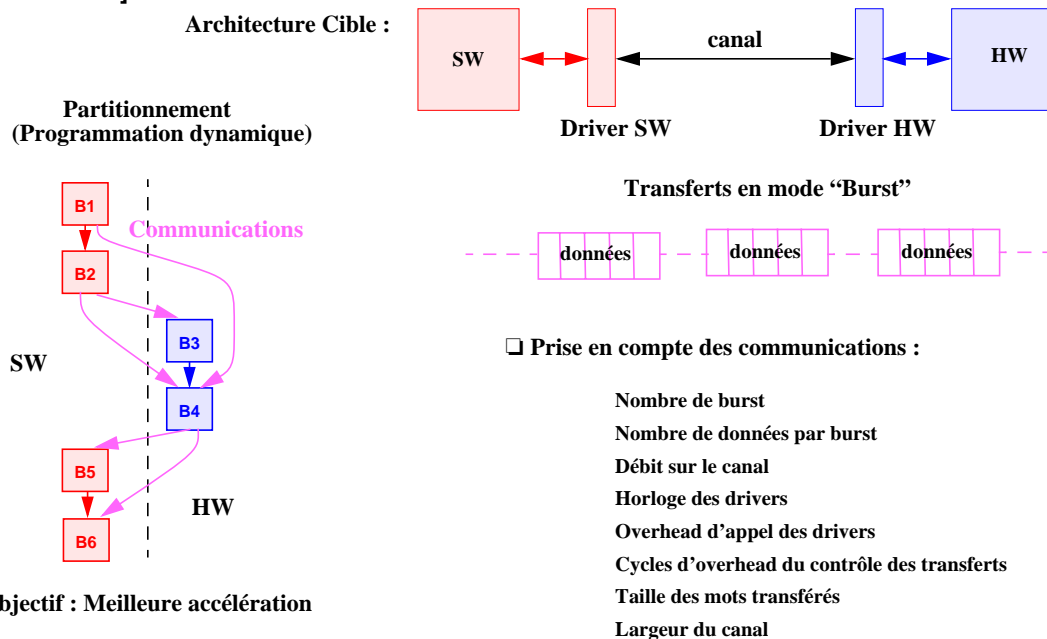
Synthèse des communications pendant le partitionnement



85

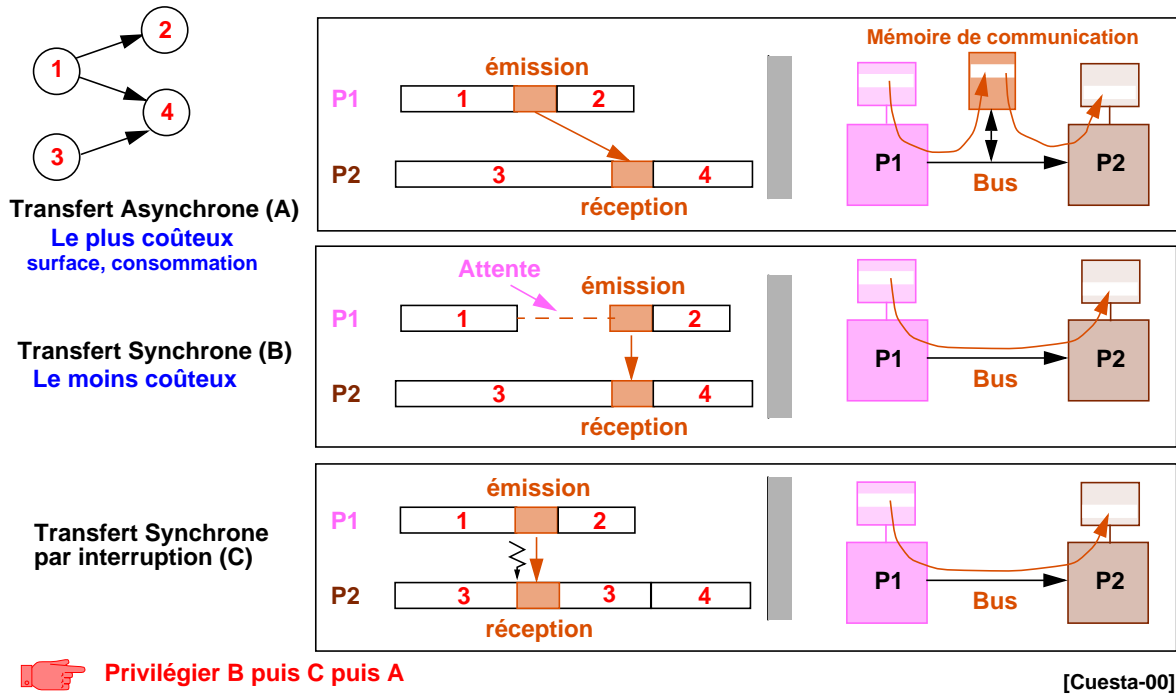
Synthèse des communications intégrée au partitionnement

[Knudsen 99]



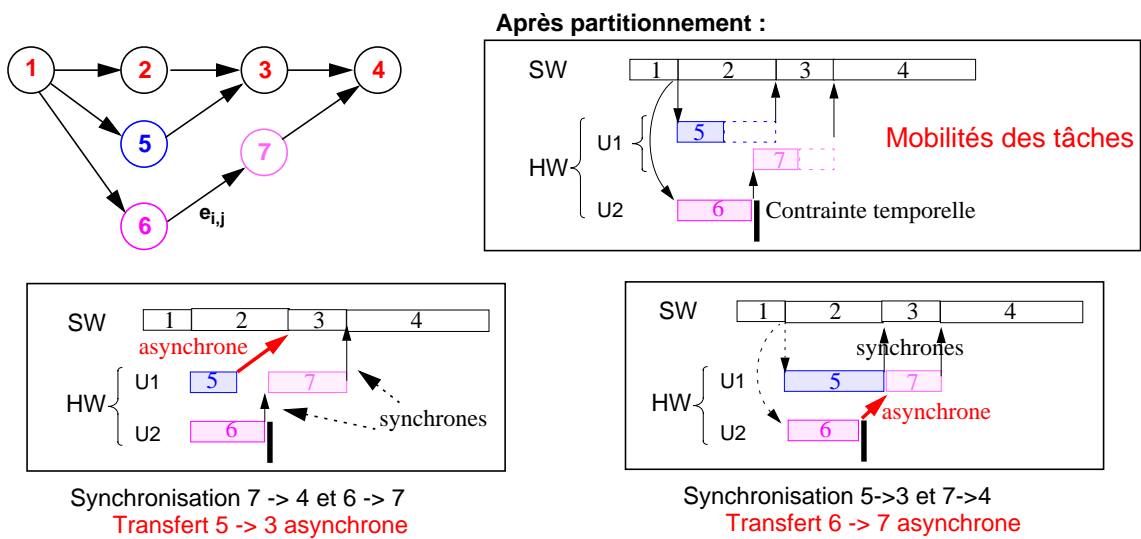
86

Synthèse des communications après partitionnement (CODEF)



87

Synchronisation par utilisation des mobilités



1) Synchronisation des arcs qui ne provoquent pas de transferts asynchrones

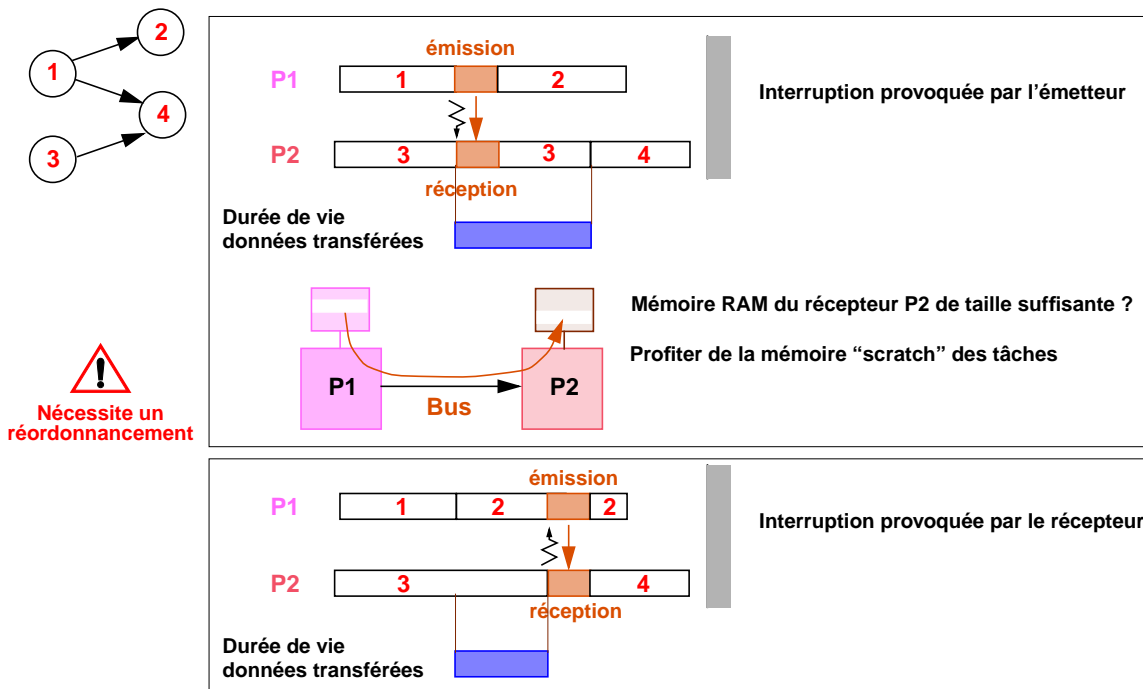
2) Synchronisation des arcs $\frac{\text{Volume données arcs asynchrones}}{\text{Volume données arcs synchrones}}$

$\frac{\text{Volume données transférées } e_{i,j}}{\text{Mobilité de } e_{i,j}}$

[Gogniat-97]

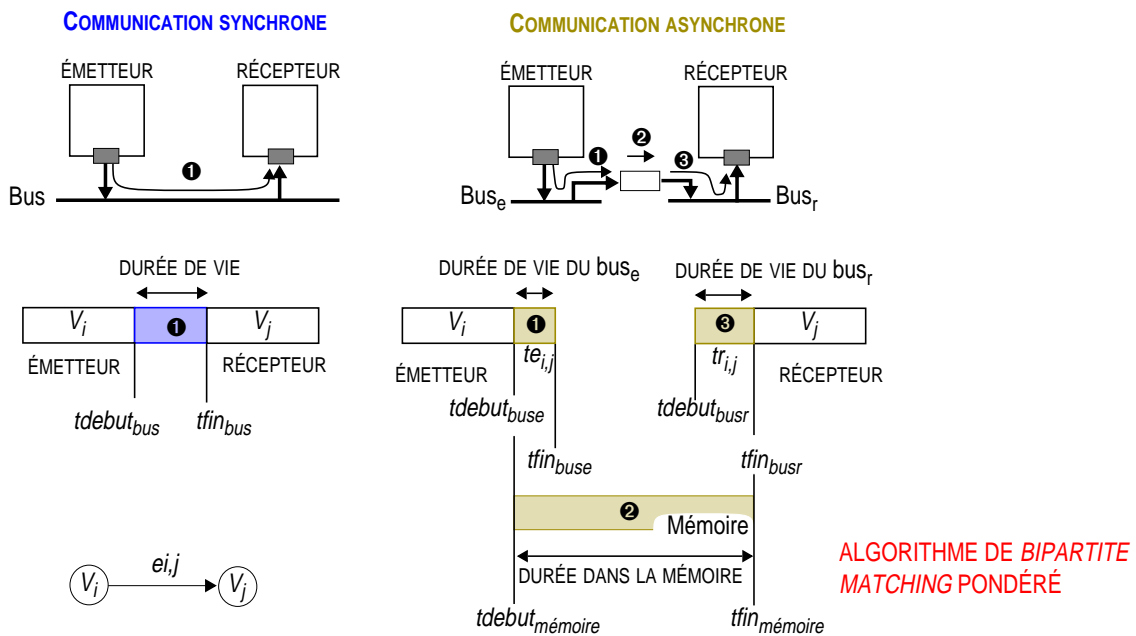
88

Synchronisation par interruptions



89

Synthèse des supports de communication



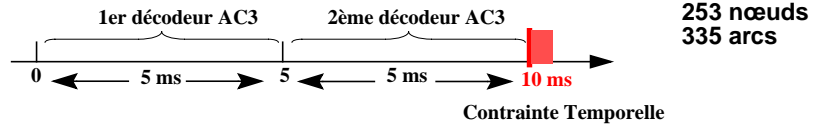
90

Exemple de résultats avec CODEF

Application considérée :

Sous-ensemble d'un set-top-box : **décodeur audio AC3 5.1** et un **modem**.

Contraintes de temps :



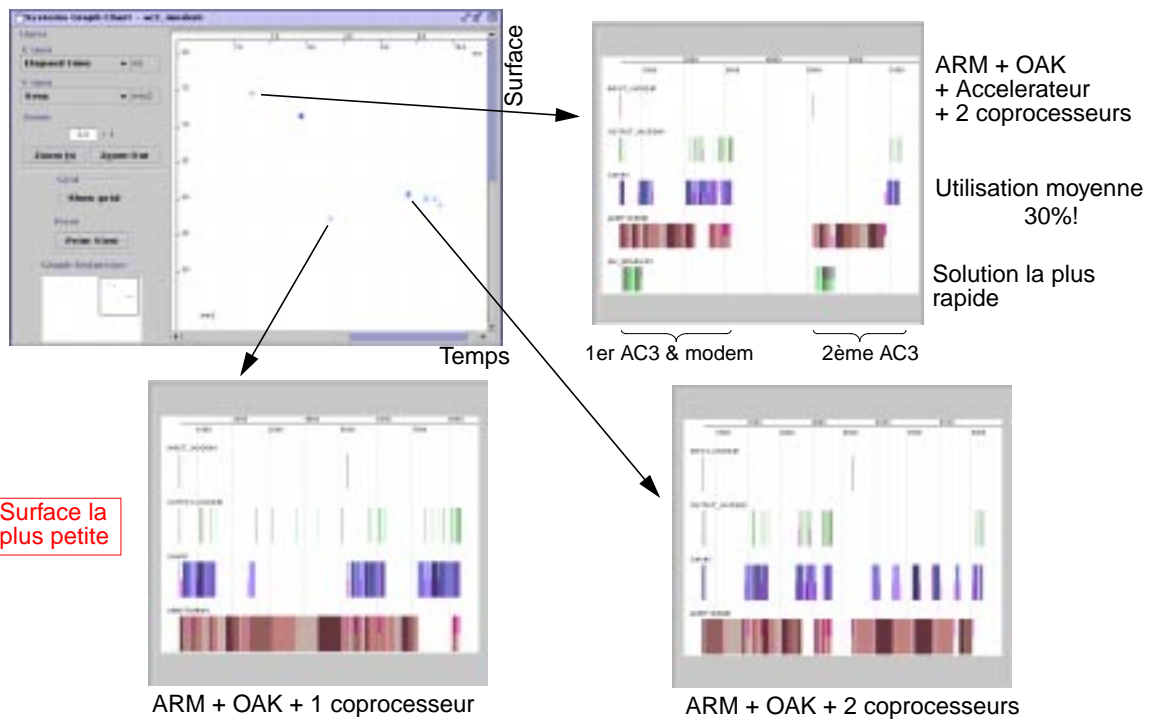
Unités de la librairie :

- ARM7TDMI (100 MHz),
- OAKDspCore (80 MHz),
- coprocesseurs & accélérateurs matériels

Problème : Architecture, Allocation, Ordonnancement ?

91

Exploration de l'espace de conception



92

Optimisation et synthèse des communications

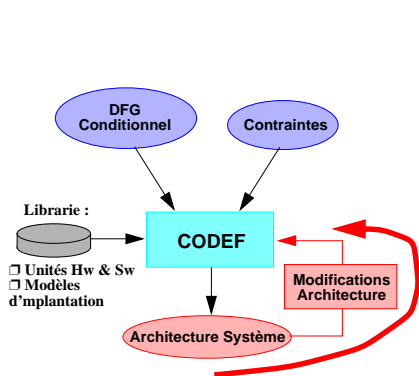
Temps d'exécution de la solution avec la plus petite surface : **8.3 ms.**

 **Optimizations possibles : contrainte = 10 ms**

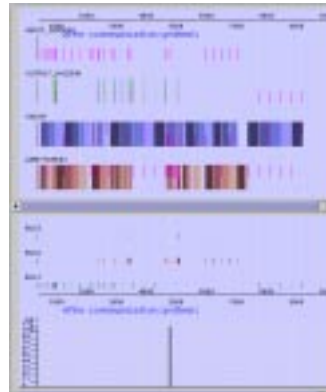
 **On modifie cette solution qui est réutilisée dans CODEF.**

Modifications :

- OAKDspCore : 40 MHz au lieu de 80 MHz
- Mémoires RAM et ROM Off-core : 50 ns de temps d'accès
- ARM7TDMI : 80 MHz au lieu de 100 MHz



On obtient



Meilleure exploitation parallélisme entre les 2 processors.

3 bus, 1 mémoire : à améliorer !

93

Conclusion sur la synthèse des communications

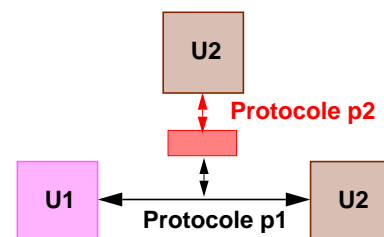
Peu d'études sur la synthèse des communications

Plusieurs travaux abordent la synthèse d'interface

Les modèles considérés dans le partitionnement sont en général simples [Knudsen 99]

Avancées basées sur des bus standardisés (AMBA, Bus Core)

Travaux sur les composants virtuels (IP)



94

CONCLUSION



Le processus de conception d'un système embarqué est complexe

- ❑ **Grande variété de réalisation des traitements**

Microcontrôleurs, RISC, DSP, ASIP, ASIC, algorithmes, RTOS
Hiérarchie et structure mémoire, ressources de communication (DMA, bus...)

- ❑ **Hétérogénéité des traitements**

Flots de données

signal : pression sur traitement/mémoire

image : pression sur mémoire

Contrôle : pression sur réactivité, temps de réponse

- ❑ **Contraintes fortes (différentes pour chaque produit)**

temps, surface, consommation, time-to-market, flexibilité, coûts, poids...

95



Privilégier la réutilisation



Problèmes importants d'actualité en co-conception

- ❑ Environnement de modélisation unique : éviter de fractionner la spécification/conception
- ❑ Partitionnement des fonctionnalités de l'application
- ❑ Vérification/preuve plutôt que (co-) simulation (ordonnançabilité, réactivité, comportement)
- ❑ Estimations de caractéristiques SW, HW : performances, surface, consommation...
- ❑ Définition d'interfaces standardisées des composants virtuels (IP)
- ❑ La synthèse des communications (Bus Core)
- ❑ Intégration avec d'autres technologies : optique, analogique ...

96