Réseaux de Processus Réguliers

Paul Feautrier

ENS de Lyon
Paul.Feautrier@ens-lyon.fr

15 juin 2009

Plan

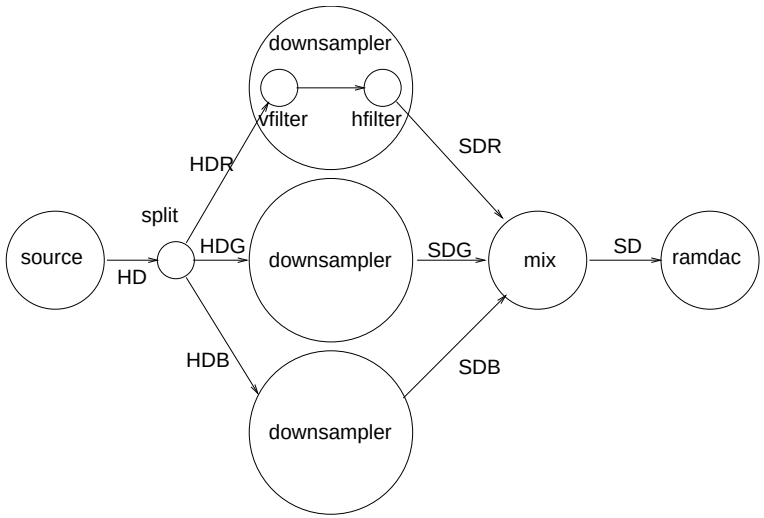
- Objectifs
- Les réseaux de processus
- *Communicating Regular Processes*
  - Vérification, ordonanncement, génération de code.
  - SYNTOL
- Au delà des CRPs.
  - Le contrôle
  - Insertion dans MARTES

Objectifs

- Aide à la réalisation rapide de systèmes embarqués.
- Parallélisation automatique (performances, économie d'énergie).
- *Codesign* : partage des fonctionalités entre le logiciel et le matériel.
- Synthèse de haut niveau des parties matérielles.

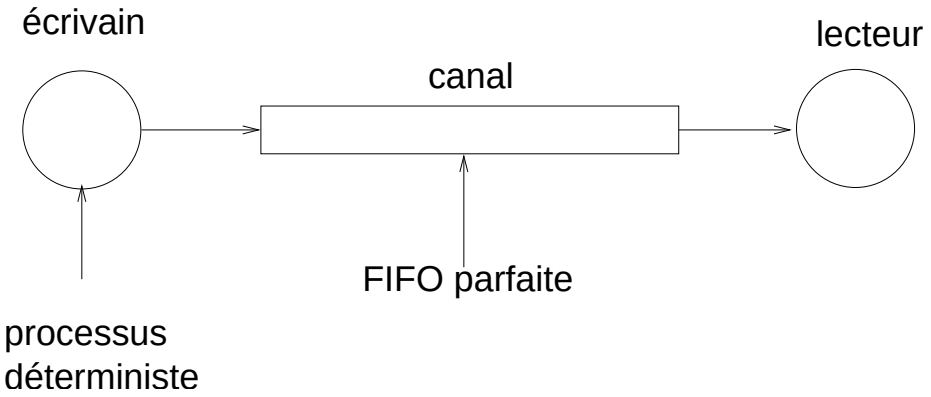
Réseaux de processus communicants. Pourquoi ?

- Représentation naturelle des systèmes réactifs.
- Représentation graphique intuitive.



- Nombreuses variantes : réseaux de processus de Kahn (KPN), réseaux de processus réguliers (CRP), réseaux à rendez-vous, réseaux anarchiques.
- Le problème central : le déterminisme.

KPN, I/II

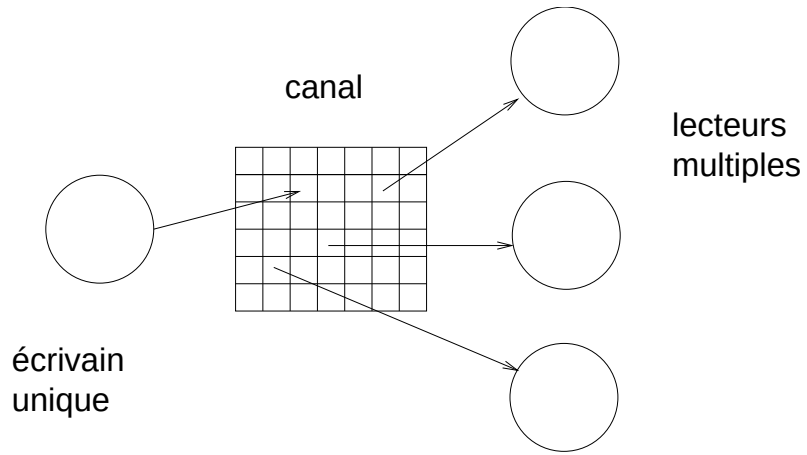


- Un seul lecteur et un seul écrivain par canal.
- Lecture destructive.
- Garantie de déterminisme.
- Deadlock possible.

KPN, II/II

- **Avantage** : garantie de déterminisme.
- **Inconvénients** :
  - L'appariement des *send* et des *receive* est difficile.
  - Il faut écrire des processus de copie si un canal doit être distribué à plusieurs lecteurs.
  - Mémorisation locale si une donnée reçue doit être utilisée plusieurs fois.

Réseaux de Processus réguliers



- Un seul écrivain, multiples lecteurs.
- Assignation unique.
- Lecture non destructive.
- Appariement lecture / écriture facile.
- Détermisme garanti.

Que peut-on faire avec les CRPs ?

- Vérifier la conformité du réseau.
- Ordonnancer et détecter les deadlocks.
- Borner la taille des canaux.
- Générer du code synchrone (VLIW ou ASIC / FPGA).

Vérifier le réseau

- Vérifier l'unicité de l'écrivain.
- Vérifier que l'écriture est en assignation unique.
- Absence de *hang up* : lecture d'une valeur qui ne sera jamais définie.
- Optionnel : vérifier que toutes les valeurs écrites sont lues.

Ordonnancer

- Trouver la date d'exécution (symbolique) de chaque opération du programme.
- Contrainte : respecter les dépendances.
- Pas de solution = haute probabilité de deadlock.
- Ordonnancement modulaire : permet de traiter des problèmes de taille réaliste.

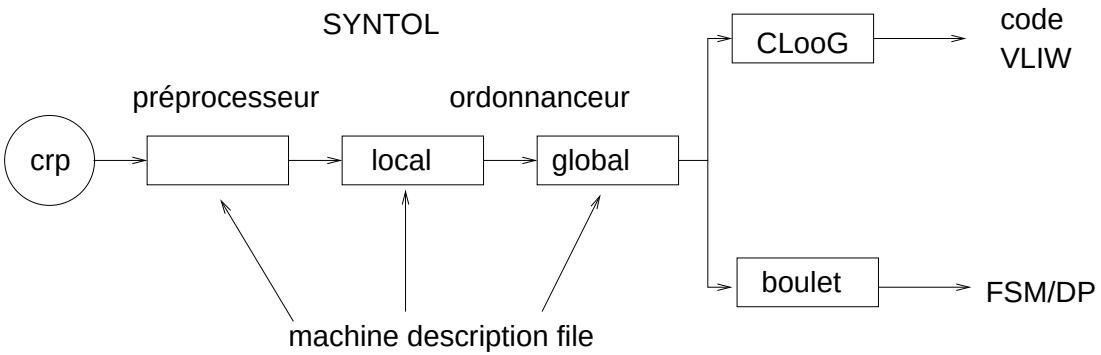
Borner la taille des canaux

- On connait la date de création de chaque valeur, et la date de sa dernière lecture.
- On en déduit la taille minimale du canal (symboliquement).
- On impose à cette taille d'être au plus égale à une valeur donnée.

Génération de code

- A partir du même ordonnancement :
- il est possible de générer un programme en boucle (CLooG, Cédric Bastoul, 2004)
- ou un FSMDP (en cours de réalisation),
- ou une combinaison des deux (codesign).

Etat d'avancement des travaux



- Préprocesseur : analyse syntaxique, recherche des boucles DO, vérifications, calcul des dépendances.
- Ordonnancement local, processus par processus.
- Ordonnancement global : ordonnancement des communications.
- *Machine Description File* renseignement divers sur le matériel : délais, ressources, taille des canaux.

En cours

- Complétion de l'analyseur syntaxique
- Eclatement d'instructions et réduction de force
- Génération de VHDL
- Gestion des ressources
- Traitement conservatif des dépendances

Extension : le contrôle

- Contrôle local à un processus
- Contrôle des lectures et des écritures des canaux
- Contrôle du réseau, reconfiguration

Contrôle local

- Il n'y a pas de problème pour les prédicats *régulier* : il suffit de découper les domaines suivant le "pointillé".
- Les expressions conditionelles $p?a : b$ sont traitées comme des expressions normales, mais on contrôle les effets de bord. En synthèse, on engendre des tests *cablés*.
- Les prédicats généraux sont traités de façon pessimiste : on calcule les dépendances comme si les deux branches étaient exécutées, sauf pour les dépendances indépendantes des boucles.
- L'ordonnancement se fait sans modification.

Contrôle des communications, I/II

- Que se passe-t-il si un test contrôle une lecture ou une écriture dans un canal ?
- Noter que pour les processus de Kahn, toute analyse devient impossible : les messages se décalent les uns par rapport aux autres.
- Dans le modèle CRP, il n'y a pas de difficulté pour les lectures, puisqu'elles ne sont pas destructives. L'ordonnancement et l'allocation de la mémoire se font de façon pessimiste.

Contrôle des communications, II/II

- Le probléme est plus difficile pour les écritures :

```
process a(outport int x[]){
   if (p) x[i] = ...;
}
```

```
process b(inport int x[]){
   r = x[i];
}
```

Si p est faux, le processus b se bloque. Remèdes :

- Laisser faire : c'est au programmeur de s'arranger pour que ça n'arrive pas.
- Vérifier l'*équilibre* des *if*s (toute instruction suivant le test est dominée par une écriture), puis :
  - Avertir le programmeur, ou
  - Refuser le programme, ou
  - Forcer l'équilibrage.

Reconfiguration du réseau, I/II

- Très difficile. Mais il n'est pas obligatoire de purger le réseau !
- Reconfiguration statique : tous les réseaux possibles sont connus d'avance.
- Reconfiguration dynamique : création de nouveaux processus dans un réseau existant.