Sorbonne Université
Paradigmes de Programmation Concurrente 5I553

Cours 13 - Programmation DataFlow

Carlos Agon

15 décembre 2024

Systèmes réactifs

- Systèmes informatiques qui réagissent à l'environnement.
- La vitesse des événements est imposée par l'environnement.
- Ils sont concurrents par nature.
- Contraints temporellement (i/o).
- Généralement déterministes.
- La fiabilité est importante.
- Généralement faits de software et hardware.

Approches software

- Automates déterministes
  - \+ Techniques de vérification connues
  - \- Objets plats (pas de hiérarchie, concurrence)
- Réseaux de Petri
  - \+ Plein des travaux, concurrents
  - \- Ne passe pas à l'échelle
- Modèles basés sur des tâches
  - \+ Techniques de scheduling
  - \- Trop bas niveau, dur à prouver
- L'approche synchrone

Systèmes complexes

- Interface interactive avec l'environnement
- Noyau réactif
- Tâches transformationnelles sur les données et sous le contrôle du noyau réactif

Mise en pratique

- Les langages synchrones ne sont pas des langages complets.
- Délégation du traitement des structures de données complexes à un langage hôte

Deux modes d'exécution synchrone classiques

- Système échantillonné :

```
Initialize Memory
for each clock tick do
   - Read Inputs
    -Compute Outputs
    -Update Memory
end
```

- Système événementiel :

```
Initialize Memory
for each input event do
         -Compute Outputs
         -Update Memory
end
```

S'assurer que le temps de calcul du corps de la boucle (réeaction) est inférieur à la période de l'horloge temps-réel ou du délai minimal entre deux événements successifs

Un exemple puredata

Dataflow

- Pour faire du calcul sur des flux de données .
- Les flux de données traversent une structure d'acteurs (le programme) qui vont les transformer.
- En théorie, ce modèle de computation ignore la notion de temps.
- Une exécution *dataflow* consiste en une sequence de *firings*, *triggers*, *bangs*, tirs.
- Chaque *fire* arrive comme une réaction à la disponibilité d'une donnée.
- Un tir est une unité de calcul (petite, en général) qui consomme une entrée et produit une sortie.

Dataflow

Les acteurs commencent leur exécution (sont déclenchés) quand leurs entrées sont prêtes.
C'est quoi être prêt ?
Deux familles dataflow :

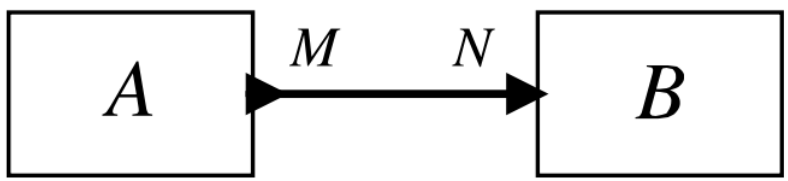
- *static dataflow (SDF synchronous dataflow)*
  -l'ordre d'exécution des acteurs ne dépend pas des données
- *dynamic dataflow (DDF)*
  - on peut avoir des conditionnels par rapport à la valeur d'une entrée

Static Dataflow

Definition (itération)

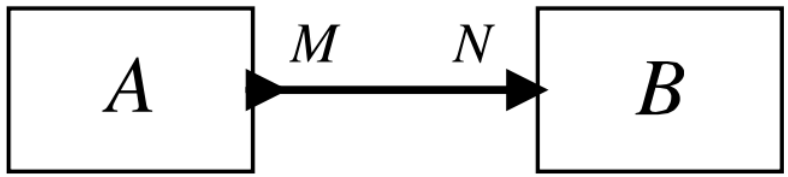
Une itération consiste en UN fire de tous les acteurs possibles.
Si chaque acteur produit et consomme un token, le programme est dit homogène

Les acteurs peuvent consommer et produire de multiples tokens à chaque déclenchement.



Exemple FFT

Les équations de balance



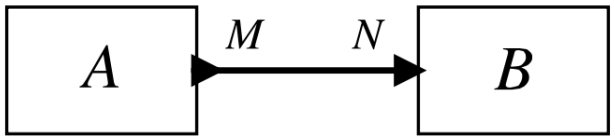
Quand A tire, on produit M *tokens* et quand B tire, on consomme N *tokens*.

Si A tire q_A fois et B tire q_B fois alors les tirs de A sont consommés par B si l'équation suivante est satisfaite :
$q_A M = q_B N$

On trouve les plus petites q_A et q_B qui satisfont l'équation et on construit un scheduler qui tire A q_A fois et B q_B fois. Le system est balancé : A produit le même nombre de tokens que B consomme.

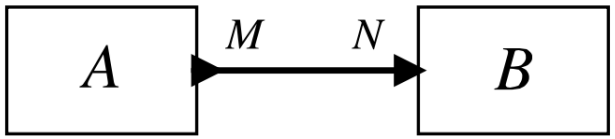
Puisqu'on peut répéter cette opération autant de fois que l'on veut (avec une quantité de mémoire fixe), on dit qu'on réalise une **unbounded execution** avec **bounded buffers**.

Exemple



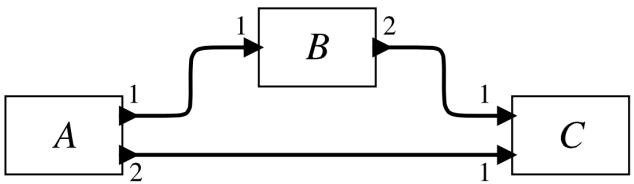
- Pour $M = 2$ et $N = 3$, une solution de l'équation de balance est donnée par : $q_A = 3$ et $q_B = 2$.
- On trouve les plus petites q_A et q_B qui satisfont l'équation et on construit un scheduler qui tire A q_A fois et B q_B fois.
  Alors, on peut tirer la sequence $AAABB$

Exemple



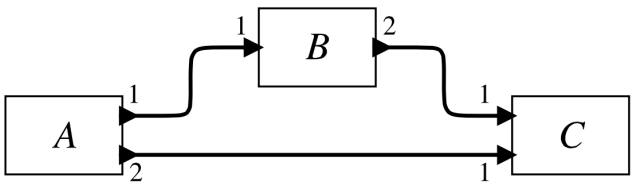
- Pour $M = 2$ et $N = 3$, une solution de l'équation de balance est donnée par : $q_A = 3$ et $q_B = 2$.
- On trouve les plus petites q_A et q_B qui satisfont l'équation et on construit un scheduler qui tire A q_A fois et B q_B fois.
  Alors, on peut tirer la sequence $AAABB$
- Cependant, la séquence $AABAB$ a besoin de moins de mémoire pour garder les *tokens* intermédiaires.
- Des solutions plus grandes (e.g. $q_A = 3$ et $q_B = 2$) compliquent l'optimisation de la mémoire.
  La solution $q_A = 0$ et $q_B = 0$ n'est pas utile, ainsi que les solutions négatives.

Exemple



Les équations de balance sont :

Exemple



Les équations de balance sont :

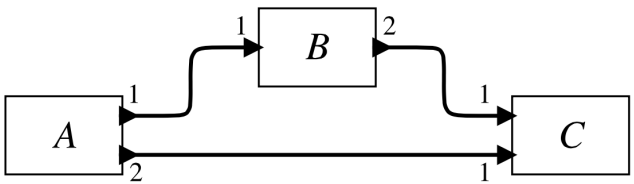
$q_A = q_B$

$2q_B = q_C$

$2q_A = q_C$

La solution la plus petite est :

Exemple



Les équations de balance sont :

$q_A = q_B$

$2q_B = q_C$

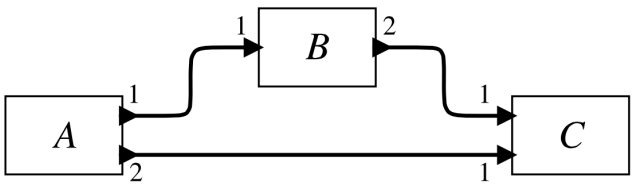
$2q_A = q_C$

La solution la plus petite est :

$q_A = 1$, $q_B = 1$ et $q_C = 2$

La séquence optimale en mémoire est :

Exemple



Les équations de balance sont :

$q_A = q_B$

$2q_B = q_C$

$2q_A = q_C$

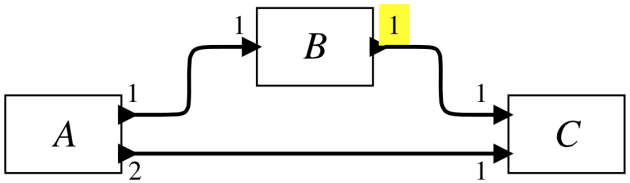
La solution la plus petite est :

$q_A = 1$, $q_B = 1$ et $q_C = 2$

La séquence optimale en mémoire est :

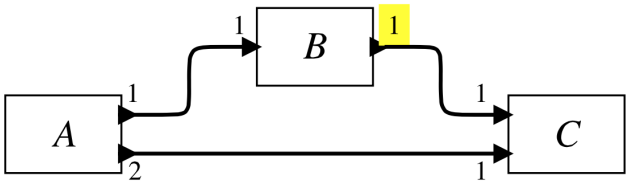
A, B, C, C

Exemple



Les équations de balance sont :

Exemple



Les équations de balance sont :

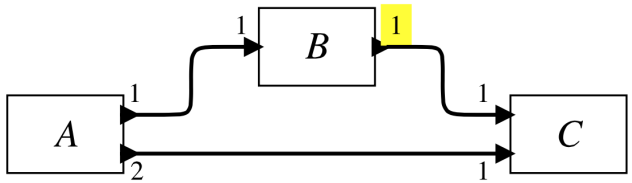
$q_A = q_B$

$q_B = q_C$

$2q_A = q_C$

La solution la plus petite est :

Exemple



Les équations de balance sont :

$q_A = q_B$

$q_B = q_C$

$2q_A = q_C$

La solution la plus petite est :

$q_A = 0$, $q_B = 0$ et $q_C = 0$

Le modèle est inconsistant. Il ne possède pas une **unbounded execution** avec **bounded buffers**.

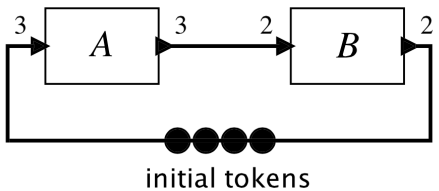
Exemple **unbounded execution** avec **bounded buffers**

Feedback loops

Exemple feedback loops

Feedback loops

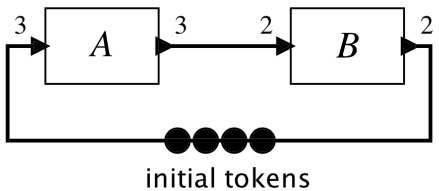
Exemple feedback loops



Les équations de balance sont :

Feedback loops

Exemple feedback loops

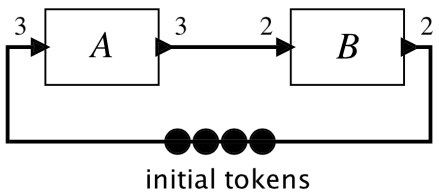


Les équations de balance sont :
$3q_A = 2q_B$
$2q_B = 3q_A$

La solution la plus petite est :

Feedback loops

Exemple feedback loops



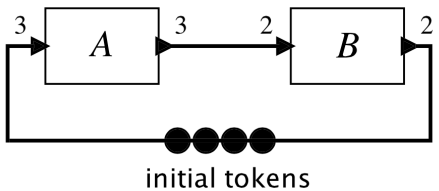
Les équations de balance sont :
$3q_A = 2q_B$
$2q_B = 3q_A$

La solution la plus petite est :
$q_A = 2$ et $q_B = 3$

Une séquence possible est donnée par :

Feedback loops

Exemple feedback loops



Les équations de balance sont :
$3q_A = 2q_B$
$2q_B = 3q_A$

La solution la plus petite est :
$q_A = 2$ et $q_B = 3$

Une séquence possible est donnée par :
A, B, A, B, B
La consistance est suffisante pour assurer **bounded buffer** mais pas **unbound execution**, car le modèle peut *deadlock* même s'il est consistant.
Heureusement, on peut trouver, s'il est possible, la séquence qui assure **unbound execution**. On peut aussi savoir si une telle séquence n'existe pas.

Extensions

- fire,
- iteration,
- complete iteration.

Beaucoup d'extensions, e.g. dynamically varying rates
i.e. on peut faire un static data flow avec variable rate (M et N peuvent changer) mais uniquement après une iteration complete.

Dynamic dataflow

- Le problème avec SDF est le pouvoir d'expression. En particulier, on n'a pas de tir conditionnel, c-à-d qu'un acteur tire pour une valeur particulière.
- Dans DDF, chaque acteur a une règle de tir (une condition qui doit être satisfaite avant que l'acteur soit tiré).
- SDF est un cas particulier de DDF avec pour unique condition les *tokens* nécessaires en entrée.
- Une autre différence est qu'avec DDF, le nombre de tokens de sortie pour un acteur peut varier.
- Même la règle d'un acteur est dynamique, elle peut changer après un tir.

Select et *Switch*

- `BoolenSelect` mélange deux flux dans un seul par rapport à un troisième flux de type boolean.
- `GeneralSelect` mélange n flux dans un seul par rapport à un troisième flux de type integer.
- `BoolenSwitch` sépare un flux en deux par rapport à un troisième flux de type boolean.
- `GeneralSwitch` sépare un flux en plusieurs autres par rapport à un troisième flux de type integer.

Exemple DDF et **no bounded buffer**

Exercice

Prendre en flux les entiers 0,1,2,... et à chaque fois diviser par 2 jusqu'à que la valeur soit <= 0.5