ASE
Cours 3 : Flots de Données

Pablo Oliveira
<pablo.oliveira@exascale-computing.eu>

Langages de Flots de Données

- Les langages impératifs décrivent l'ordre des calculs.
- Les langages dataflow décrivent le mouvement des données.
- Premiers langages dataflow dans les années 70.

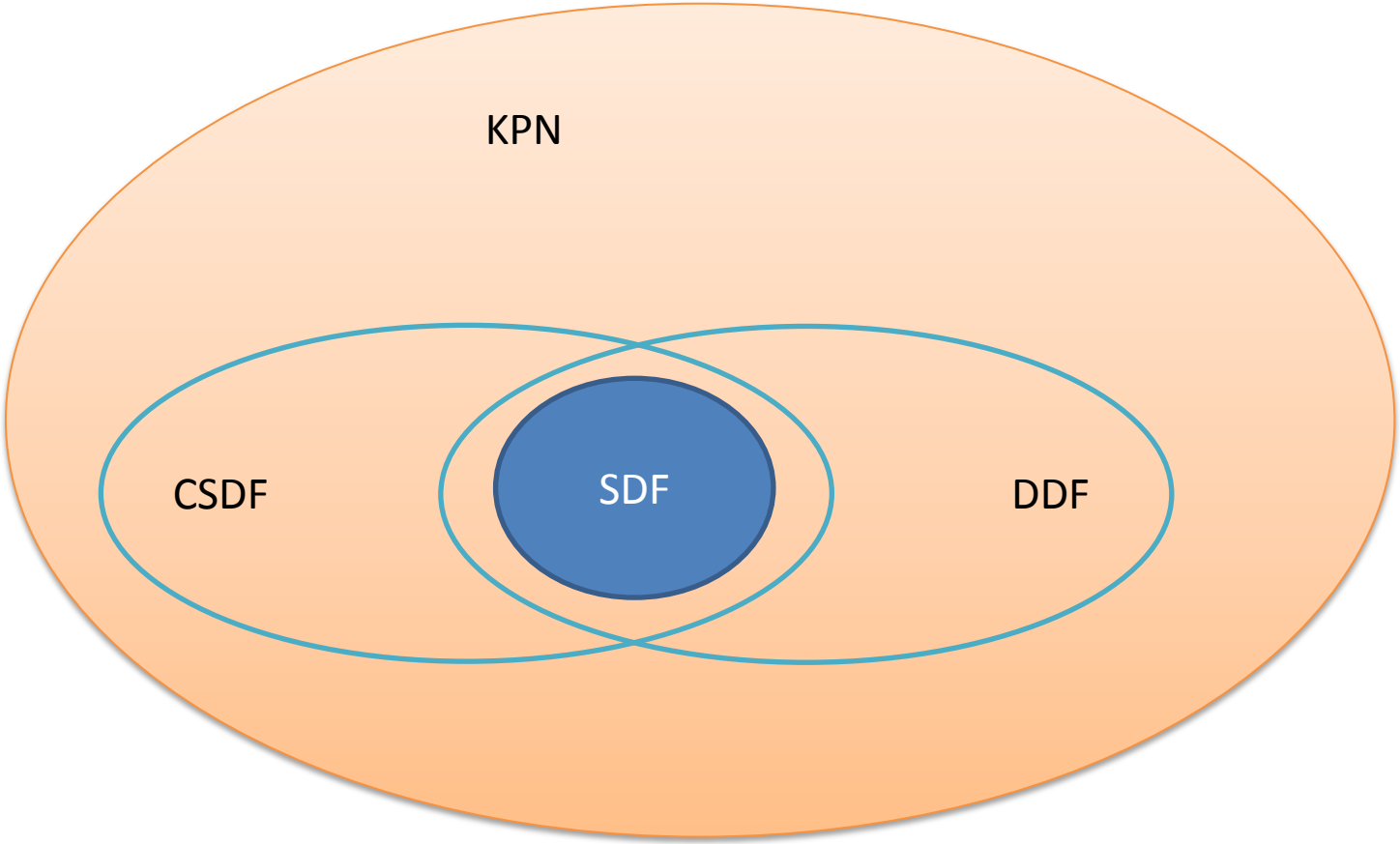
Dataflow pour l'Embarqué ?

- Les langages dataflow sont utiles pour les applications de traîtement du signal.
- Dataflow est proche d'une description d'un circuit, faciles à synthétiser vers du FPGA ou DSP. (VHDL et Verilog sont proches d'une sémantique dataflow).
- Bonnes propriétés:
  - Déterministe
  - Borné en mémoire
  - Facile à paralléliser

Historique de quelques langages dataflow

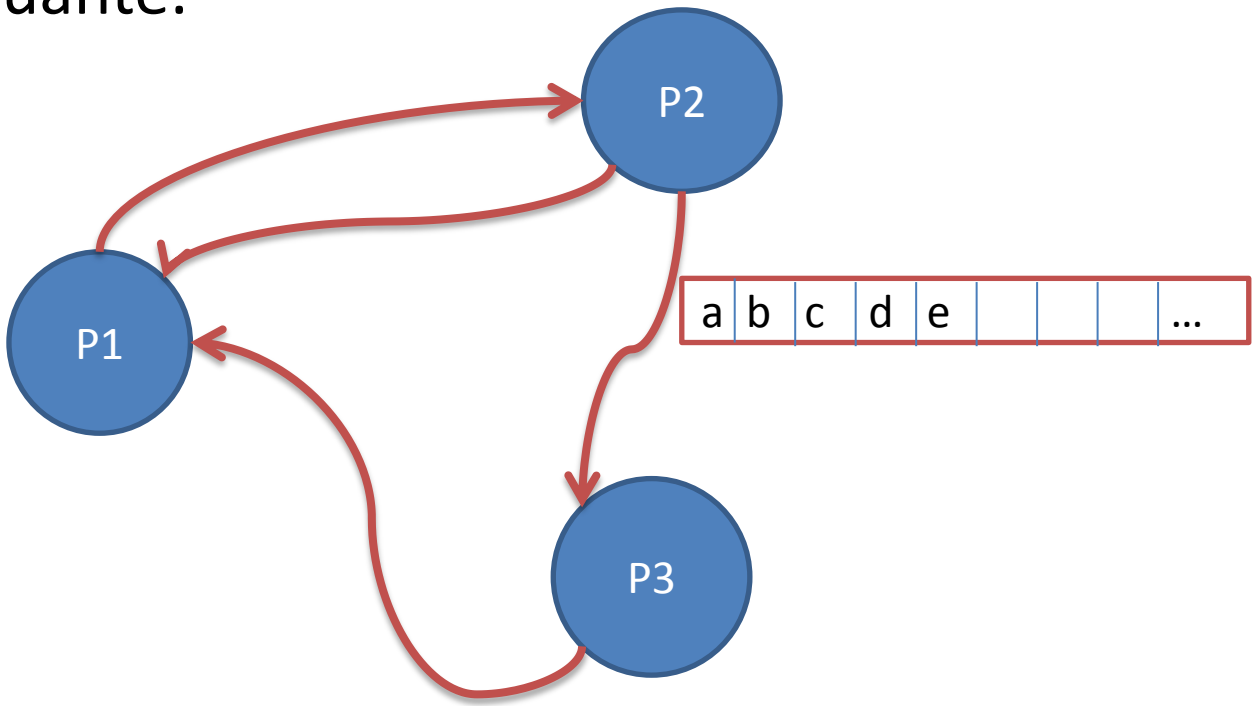
- 1977: Lucid s'appuie sur les flots pour raisonner mathématiquement sur la correction d'un programme.
- 1980: Lustre utilisé pour des logiciels critiques dans l'avionique (Airbus, Eurocopter, Dassault, Pratt), le ferroviaire et les centrales nucléaires.
- 1985: Sisal dans la lignée de Lucid basé sur la propriété d'affectation unique (single assignment) et son successeur SAC.
- 2002: StreamIt langage pour l'expression de parallélisme dataflow.

Modèle d'execution des langages dataflow



Kahn Process Network (1974)

- Processus communiquent à travers des canaux
- chaque canal est muni d'une file où les données en transit sont stockées
- l'écriture de données est asynchrone mais la lecture est bloquante.



Formalisme

- On associe a un canal ses traces avec l'ordre préfixe:

$$T_1 \leq T_2 \leq \cdots \leq T_n$$

abcdefg….

$$T_1 = \{ab\} \qquad T_2 = \{abcd\} \qquad T_3 = \{abcdefg\}$$

Formalisme

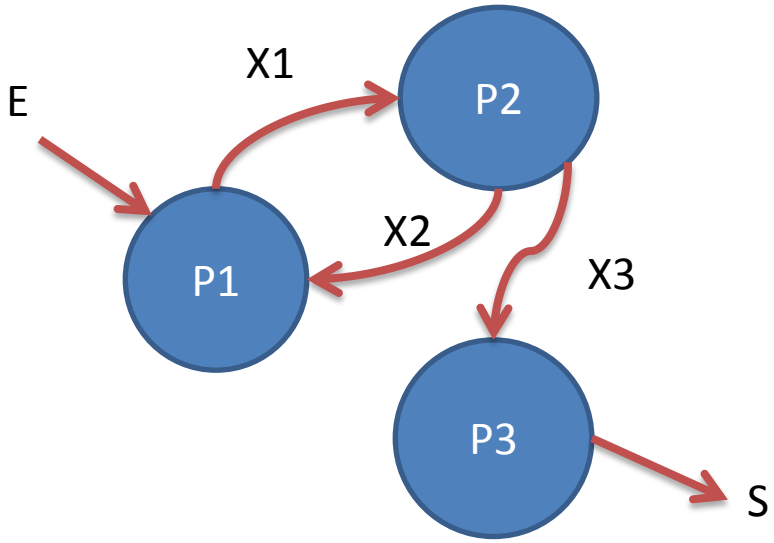
- On appelle une suite croissante S = $T_1 \leq T_2 \leq \cdots$ (possiblement infinie) un ordre partiel complet.
- Une telle suite admet toujours une borne sup. notée $\lim S$ telle que:
  - $\forall x \in S, x \leq \lim S$
  - lim S est le plus petit des majorants de S

Monotonie et Continuité

- Les processus de Kahn sont continus:
  - $limf(S) = f(limS)$
- La continuité implique la monotonicité:
  - Si $x \ll y$ alors $f(x) \ll f(y)$
- Monotonicité:
  - exécutable de manière itérative, un nœud peut commencer à calculer avant d'avoir reçu toutes les données en entrée puisque les entrées futures n'affecteront que les sorties futures
- Continuité:
  - interdit à un noeud de consommer une séquence infinie en entrée avant de produire des données.

Déterminisme

- On peut transformer un KPN en un système d'équations récursives:



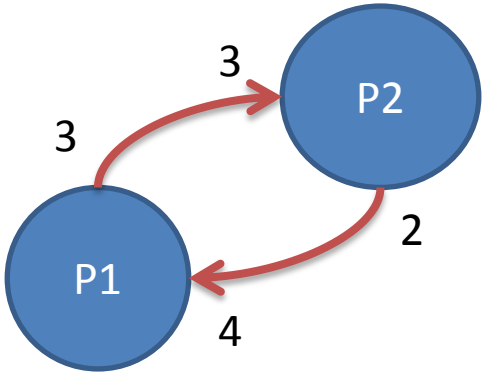
P1(E, X2) = X1
P2(X1) = {X3, X2}
P3(X3) = S

- Scott montre qu'un tel système admet une solution unique qui est le point fixe du système.

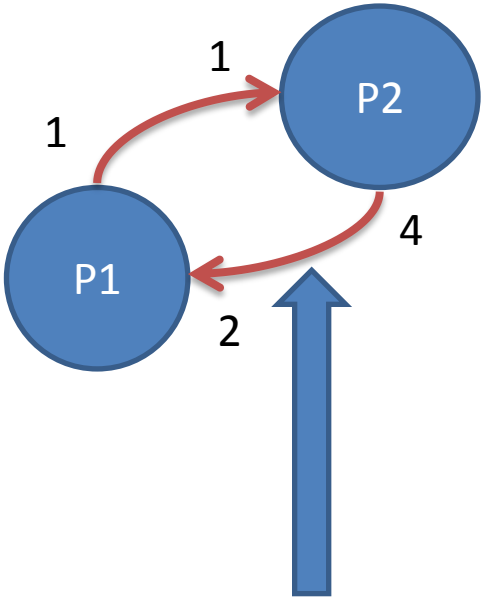
Kahn Process Network (1974)

- Ce modèle garantit:
  - Déterminisme d'exécution
- Ce modèle ne permets pas:
  - La détection automatique d’interblocages
  - de borner statiquement la taille maximale atteinte par les files des canaux

Exemple d'interblocage



Exemple en mémoire non bornée

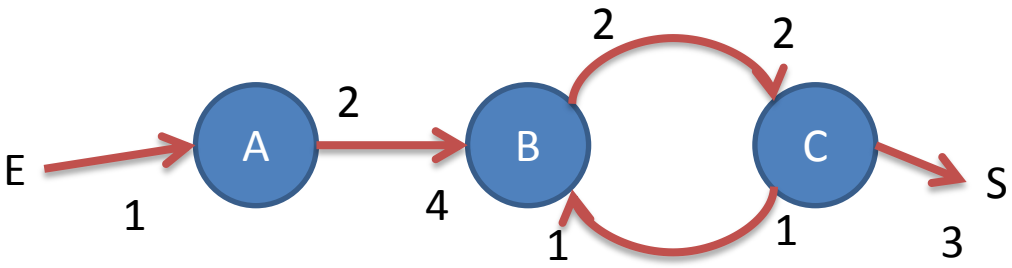


Sur cette file d'attente un nombre infini d'éléments s'accumule.

Synchronous Data Flow (SDF)

- SDF est plus contraint que Kahn:
  - Les processus consomment et produisent un nombre de données fixes sur chaque entrée/sortie.
- SDF garanti les propriétés suivantes:
  - Déterminisme
  - Détection automatique des interblocages
  - Exécution en mémoire bornée

SDF : Ordonnancements



Un ordonnancement est un ordre d'exécution du graphe SDF.
Un ordonnancement fini comporte un nombre fini d'exécutions:
A A A B B C

Un SDF possède un ordonnancement stationnaire ssi il existe un ordonnancement fini pour lequel le nombre d'éléments en attente sur les files est le même avant et après son exécution.

A A A B C est-il stationnaire ?

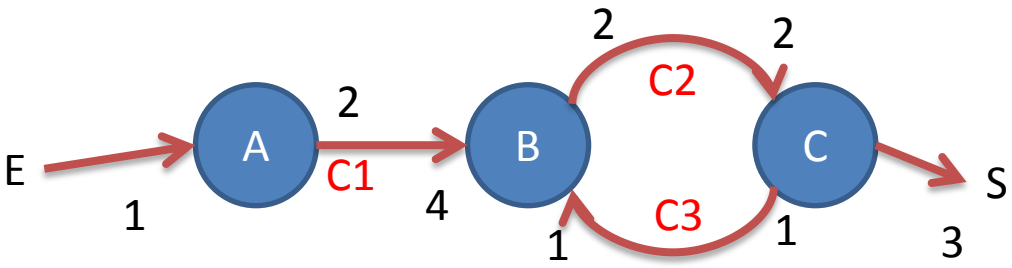
A A B C est-il stationnaire ?

SDF: Consistance

- Un ordonnancement stationnaire peut s'exécuter un nombre infini de fois en mémoire bornée.
  - En effet le nombre d'élements sur les canaux est invariant après une exécution. Donc
    - Exécuter 1 fois l'ordonnancement
    - Exécuter n fois l'ordonnancement

    nécessite la même mémoire.
- Un SDF est consistant lorqu'il adment un ordonnancement stationnaire non vide.

Trouver un ordonnancement stationnaire ?



Supposons qu'il existe un ordonnancement stationnaire. Soir R_N le nombre d'exécutions du nœud N dans cet ordonnancement.

Système d'équations récurrentes de consommation:

C1 : $2.R_A = 4.R_B$
C2 : $2.R_B = 2.R_c$
C3 : $1.R_c = 1.R_B$

Sous forme matricielle $\Gamma = \begin{bmatrix} 2 & -4 & 0 \\ 0 & 2 & -2 \\ 0 & -1 & 1 \end{bmatrix}$

Le vecteur nul est toujours solution de ce système. Pourquoi ?

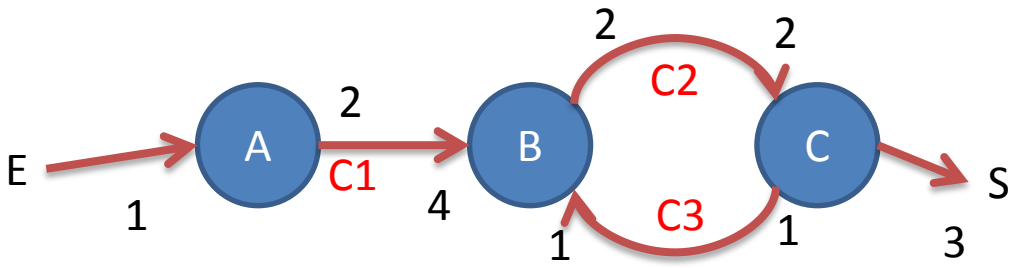
Trouver un ordonnancement stationnaire ?

- (Lee87) $rg(\Gamma) \geq n - 1$
  où n est le nombre de noeuds.
- Cela laisse deux possibilités:
  - $rg(\Gamma)$ = 0, seul l'ordonnancement nul est stationnaire, le graphe est inconsistent.
  - $rg(\Gamma)$ = n-1, il existe des solutions de la forme

    r = [r_1, r_2 , …, r_N]

    La plus petite solution, r/pgcd(r_1, r_2 , …, r_N) est appellée ordonnancement stationnaire minimal.

Trouver un ordonnancement stationnaire ?



$$\Gamma = \begin{bmatrix} 2 & -4 & 0 \\ 0 & 2 & -2 \\ 0 & -1 & 1 \end{bmatrix}$$

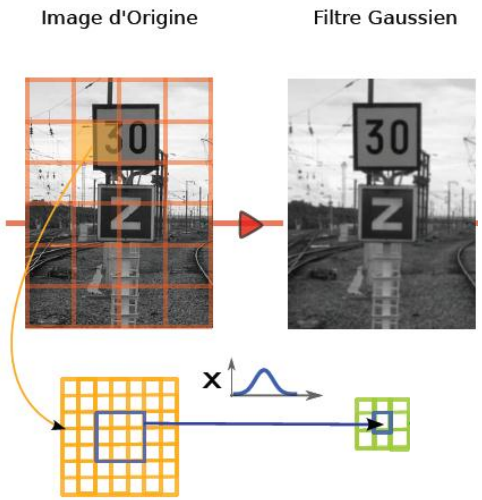
r = [2 1 1] est l'ordonnancement stationnaire minimal.

Vivacité

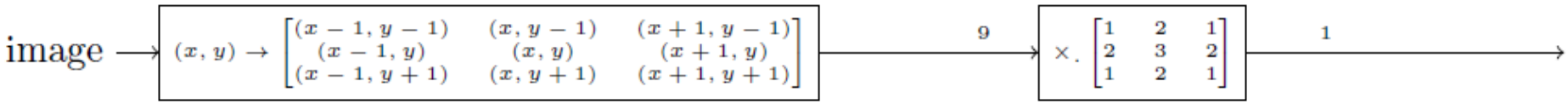
- Un SDF est vivace ssi il adment un ordonnancement exécutable un nombre infini de fois sans interblocage.
- Un SDF est correct s'il est vivace est consistant.

- Pour vérifier la vivacité d'un SDF il suffit de simuler une exécution de l'ordonnancement stationnaire minimal. Voyez vous pourquoi ?

Examples de programmes SDF : Filtre Gaussien



$$\mathbf{G} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix} \otimes \mathbf{image}$$



Multiplication de matrices

$$C_{y_0,x_1} = A_{y_0,x_0} \times B_{y_1,x_1}$$