Parallélisme, Algorithmes PRAM

Armelle Merlin

L.I.F.O
Laboratoire d'**I**nformatique **F**ondamentale d'**O**rléans

Transparents inspirés des cours de G. Hains et de B. Virot

Plan

Introduction

Définition du parallélisme

- Etude des problèmes concernant la programmation et l'exécution d'un ensemble de tâches qui se déroulent simultanément en interagissant.
- A plusieurs niveaux :
  - programme (systèmes d'exploitation multitâches)
  - processus, thread (applications réparties, programmes parallèles)
  - instruction (vectorisation)
  - à l'intérieur des instructions

Langages parallèles

Les plus utilisés

- ADA (Rendez-vous, gardes, types protected, select)
- MPI (Message Passing Interface) bibliothèque de fonctions C permettant la programmation des communications et des synchronisations entre des processus écrits en langage C ou C++
- JAVA langage objet qui gère l'exclusion mutuelle entre threads. Bien adapté pour la programmation d'applications réparties orientées Internet
- HPF (High Performance Fortran) et Fortran 90 : data-parallélisme.

Les architectures parallèles

Doit proposer

- la coordination des flots d'instructions (synchronisation)
- l'échange de données (communication)

Deux types d'architectures

- Synchrone : chaque unité exécute une instruction à chaque top d'une horloge globale
- Asynchrone : les unités peuvent exécuter indépendamment un ou plusieurs flots d'instructions

La classification de Flynn (1972)

Basée sur deux critères :

- La centralisation des données
- La centralisation du contrôle (flot d'instructions)

Architectures synchrones

Architectures synchrones

- Single Instruction Single Data
  Contrôle et données centralisées
  ordinateur séquentiel classique
- Single Instruction Multiple Data
  Contrôle centralisé et données distribuées
  - Flot de contrôle diffusé par un seul ordinateur séquentiel frontal
  - Tous les processeurs exécutent la même instruction sur les données de sa mémoire privée
  - Adaptée aux problèmes réguliers du type traitement d'images
- Mutliple Instruction Simple Data
  Pipelines

Architectures asynchrones

Architectures asynchrones

- Mutliple Instruction Multiple Data à mémoire répartie
  - Programmes différents sur des données différentes.
  - Pas d'horloge globale
  - Accès direct uniquement à la mémoire privée
- Mutliple Instruction Multiple Data à mémoire partagée
  - Pas d'horloge globale
  - Accès direct (à l'aide de bus) indifféremment à n'importe quel banc mémoire

Comment ?

Comment ?

- Rechercher le parallélisme intrinsèque d'un problème
- Concevoir des algorithmes parallèles efficaces pour le résoudre
- Estimer le coût de l'exécution de ces algorithmes sur une machine parallèle en fonction de la taille des données et du nombre de processeurs disponibles

Exemple de calcul de complexité

```
s:= 0;
for i=1 to n do   s:= s+x[i];
```

Q: Pourquoi cet algorithme est-il $O(n)$?
R: Parce qu'on peut compiler le corps en

```
LOAD R1 (i)
ADD  R1 R1 base_x
LOAD R2 (R1)
LOAD R3 (s)
ADD  R2 R2 R3
STORE R2 (s)
```

qui sera exécuté n fois.

Exemple de calcul de complexité

Hypthèse

On suppose ainsi que la machine exacte importe peu et que LOAD, STORE en temps $O(1)$.

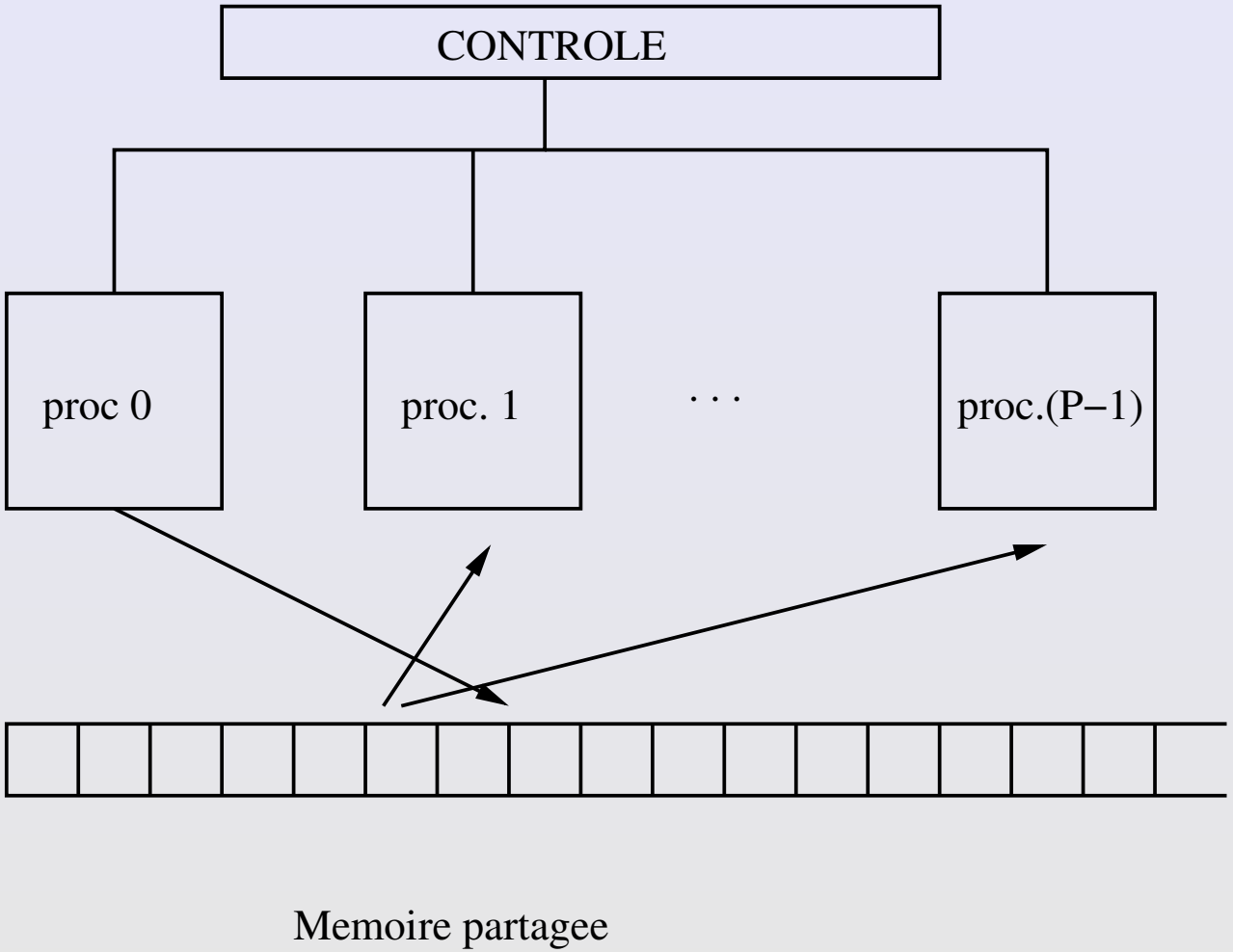
Le modèle PRAM (1)

Le modèle PRAM (1) : *parallel random-access machine*

Une machine PRAM modélise le fonctionnement d'une architecture MIMD à mémoire partagée.
On parle d'*accès direct* à la mémoire: *random access machine* ou RAM
Hypothèse du cas idéal :

- nombre illimité de processeurs
- lire et écrire dans la mémoire dans conflit d'accès
- unité de mesure de complexité = opération élémentaire sur une donnée scalaire
- synchrone (toutes les opérations élémentaires prennent une unité de temps)

Le modèle PRAM (2)



Le modèle PRAM(3)

Le modèle PRAM(3)

Une mémoire partagée par un ensemble de processeurs; chacun est une machine de type RAM. Les processeurs sont synchronisés. Ils exécutent collectivement un programme dont certaines instructions sont parallèles de type

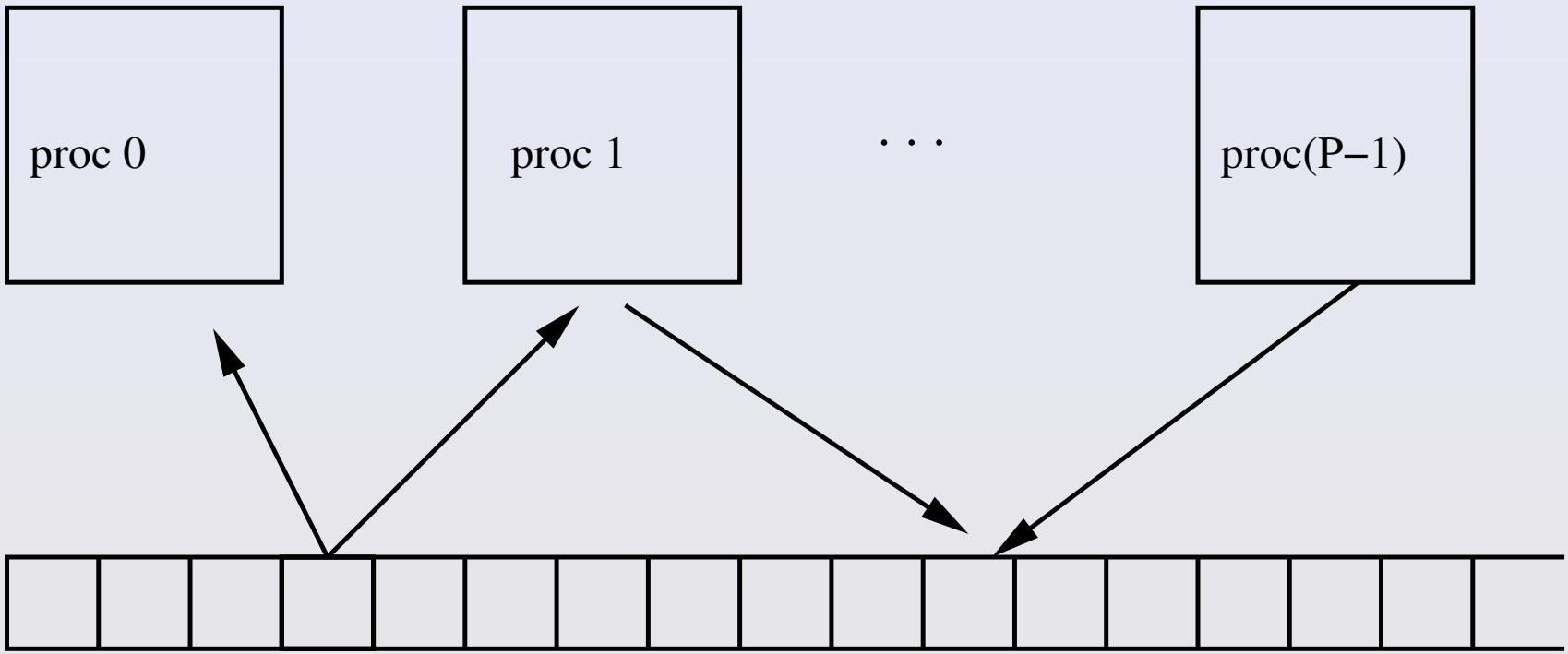
```
par proc= 0 to (P-1) do ....
```

Protocoles d'accès

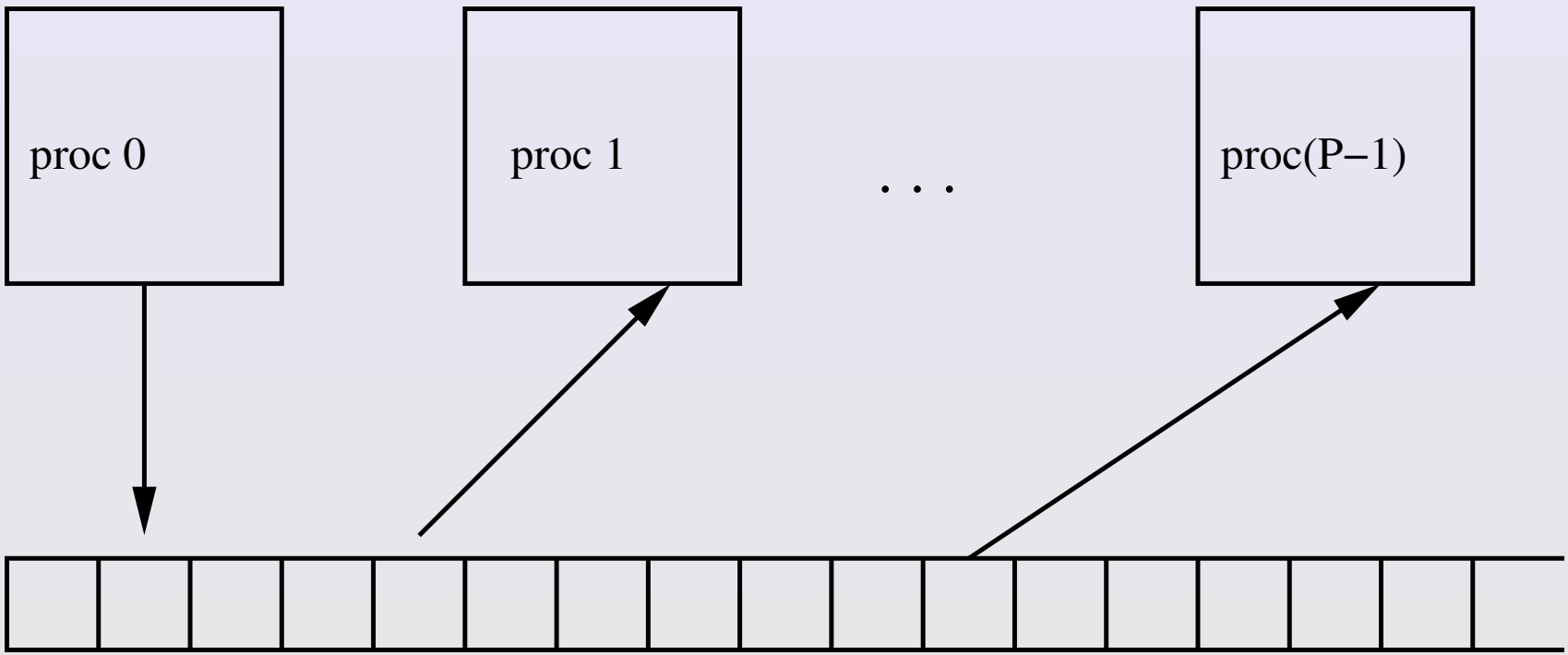
Protocoles d'accès

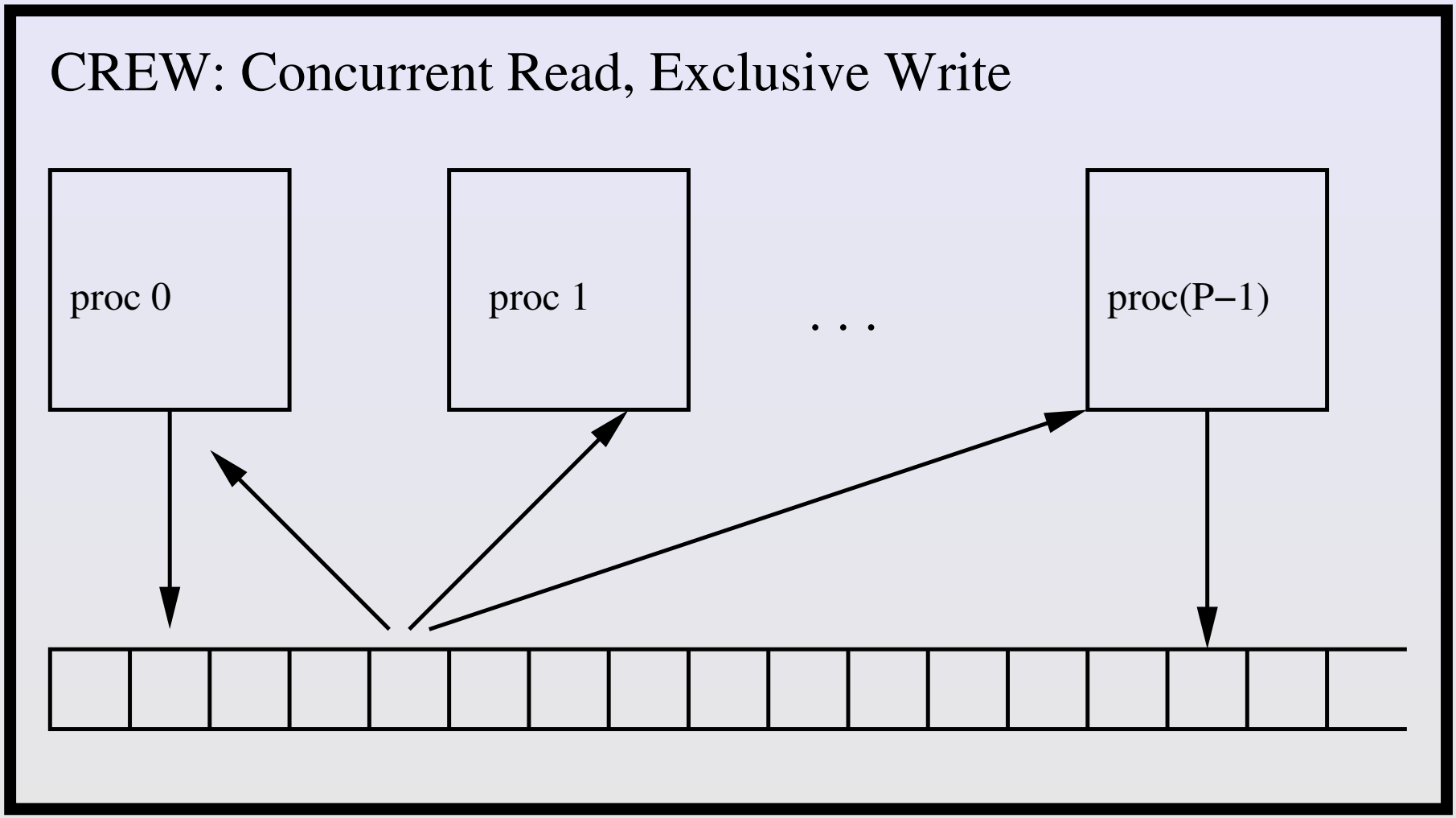
Une instruction PRAM synchronise les P processeurs. Pour que l'action sur la mémoire partagée soit bien définie, il faut fixer un protocole d'accès mémoire.

CRCW: Concurrent Read, Concurrent Write



EREW: Exclusive Read, Exclusive Write




CREW: Concurrent Read, Exclusive Write
proc 0
proc 1
. . .
proc(P−1)

Mesures de complexité

Mesures de complexité

L'algorithme PRAM sert à minimiser T (le temps de calcul), mais pas à n'importe quel prix P (le nombre de processeurs).
Le *travail* d'un algo. PRAM =
no. d'instructions exécutées $\leq T * P$
Exemple : Résoudre un problème NP-complet en temps polynomial PRAM avec $P \approx 2^n$ processeurs.

- Décrivez l'algo. polynomial pour 3-SAT.
- En l'an 20YY on pourra s'acheter autant de processeurs que de mots de mémoire vive aujourd'hui. Estimez la taille de 3-SAT qu'un PC-PRAM pourra résoudre.

Algo. PRAM vs algo. séquentiel

Temps et travail parallèle vs temps séquentiel

S'il existe un algorithme PRAM de complexité $(T, P) \in O(f(n), g(n))$ pour un certain problème, alors il existe un algorithme séquentiel de complexité $O(f(n) * g(n))$ pour ce problème.

Algo. PRAM vs algo. séquentiel

Exemple

Un algo. PRAM pour trier n nombres en $P = n^2$ et $T \in O(\log n)$.
Cela implique un travail de $O(n^2 \log n)$ et correspond à un algorithme séquentiel de cette complexité : encore pire qu'un tri-bulle!
Cet algorithme, même s'il est rapide, n'est pas vraiment efficace.

Diffusion (*broadcast*)

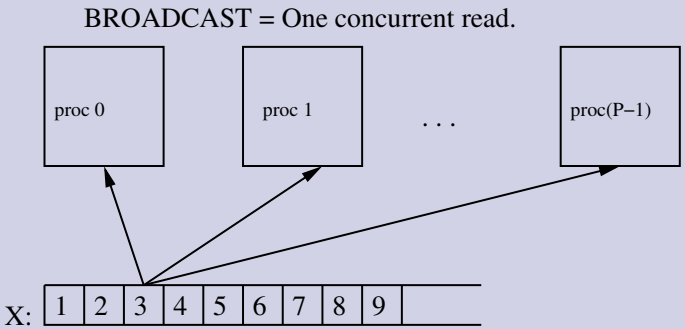
Diffusion (*broadcast*)

En CREW : lecture simultanée par les P processeurs.

```
par i= 0 to (n-1) do
  Operation_i(X[3])
end_par
```

Diffusion(*broadcast*)

Diffusion (*broadcast*)



$T = 1,\ P = n,\ T * P = n$

Comparaison avec l'équivalent séquentiel ?

La réduction associative (*fold*)

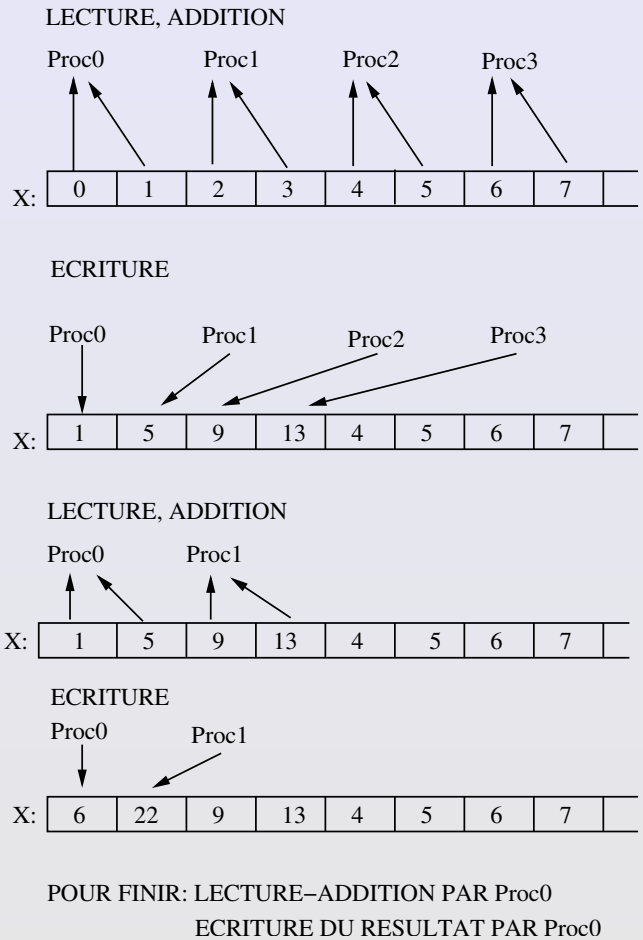
La réduction associative (*fold*)

Etant donnés n valeurs $x_0, x_1, \ldots, x_{n-1}$, et une opération binaire *associative* $\oplus$, le calcul de réduction consiste à en calculer la combinaison $x_0 \oplus (x_1 \oplus \ldots (x_{n-2} \oplus x_{n-1}))$.
Quelles opérations sont associatives parmi :
$+, -, *, /, \min, \max, \wedge, \vee, \cap, \cup,$ `concat` ? L'associativité permet d'utiliser n'importe quel parenthésage. Un parenthésage bien équilibré comme

$$((x_0 \oplus x_1) \oplus (x_2 \oplus x_3)) \ldots (x_{n-2} \oplus x_{n-1})$$

donne lieu à un algorithme parallèle rapide.

La réduction associative (*fold*) pour $n = 8$



La réduction associative (*fold*) pour $n = 8$

La réduction associative (*fold*)

- T, P, comparaison avec séquentiel ?
- Pseudo-code récursif pour fold binaire.

Le calcul des préfixes (*scan*)

Le calcul des préfixes (*scan*)

Etant donnés n valeurs $x_0, x_1, \ldots, x_{n-1}$, et une opération binaire *associative* $\oplus$, il s'agit de calculer toutes les réductions de préfixes de la liste soit :

$$(x_0), (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}).$$

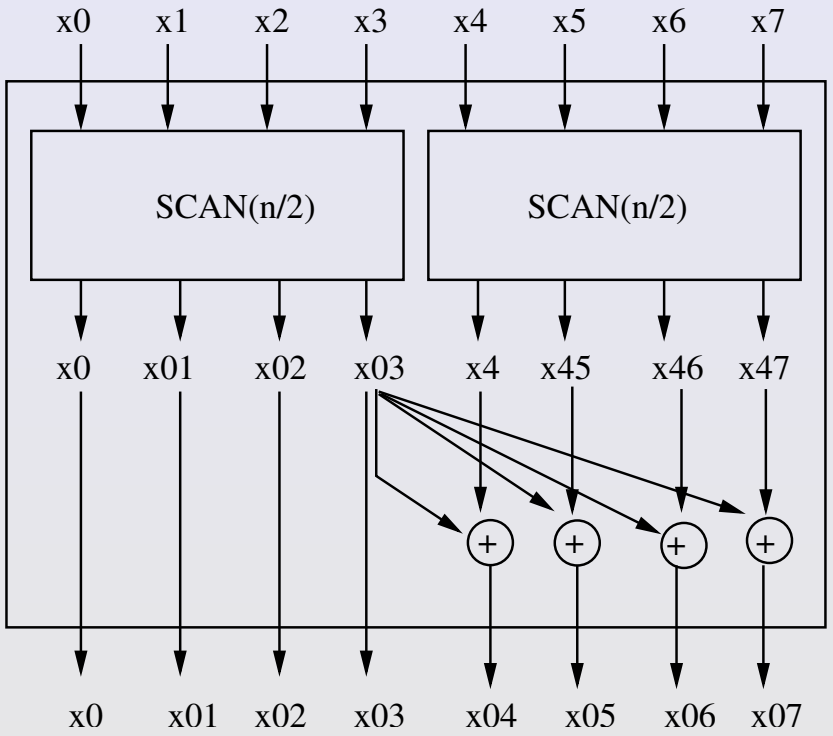
Ce calcul est à la base de nombreux algorithmes parallèles en arithmétique, en géométrie, pour les SGBD, etc.

- Décrire un algorithme naïf utilisant n *fold*s en parallèle.
- Donner sa complexité (T, P) et comparer avec le cas séquentiel.

Le calcul des préfixes (*scan*) pour $n = 8$

APPROCHE RECURSIVE BINAIRE DU SCAN

SCAN(n):



Le calcul des préfixes (*scan*) pour $n = 8$

Le calcul des préfixes (*scan*)

- Donner un pseudo-code récursif.
- Complexité (T, P) ?
- Comparaison avec le cas séquentiel ?

Le théorème de Brent

Le théorème de Brent

Soit A un algorithme PRAM de complexité en temps T et qui effectue un travail m. Alors il existe un algorithme PRAM équivalent sur P processeurs et de complexité en temps

$$O(m/P + T).$$

Le théorème de Brent

Preuve

Soit $m(i)$ le nombre d'instructions exécutées à l'étape i de l'algorithme. On peut simuler cette phase avec p processeurs en temps parallèle $m(i)/p + O(1)$. La somme de ces temps sur i donne la complexité de la simulation complète.

Algorithmes par blocs (*coarse-grain*)

Algorithmes par blocs (*coarse-grain*)

On peut appliquer le théorème de Brent pour réduire le nombre de processeurs de tout algorithme parallèle.
Cela peut mener à P=1, algorithme séquentiel.
Ou mener à une valeur plus réaliste de P.
En pratique cette méthode permet de cacher la latence des réseaux de communication (ou de la mémoire partagée).

Fold par blocs

Fold par blocs

- Travail parallèle trop important vs séquentiel.
- Calcul du bon facteur de réduction.
- Nouvel algorithme : réduction séquentielle locale, suivie d'un fold classique.

Scan par blocs

Scan par blocs

- Travail parallèle trop important vs séquentiel.
- Calcul du bon facteur de réduction.
- Nouvel algorithme : scan séquentiel local, algorithme classique, re-diffusion locale.