Systèmes concurrents

Philippe Quéinnec

Département Informatique et Mathématiques appliquées
ENSEEIHT

14 septembre 2012

Plan du cours

1. Introduction : problématique
2. Exclusion mutuelle
3. Synchronisation à base de sémaphores
4. Synchronisation à base de moniteur
5. Interblocage
6. API Java, Posix Threads
7. Problèmes génériques
8. Processus communicants – Ada
9. Transactions – mémoire transactionnelle
10. Synchronisation non bloquante
11. *Calcul parallèle : OpenMP, etc* [A.Buttari]

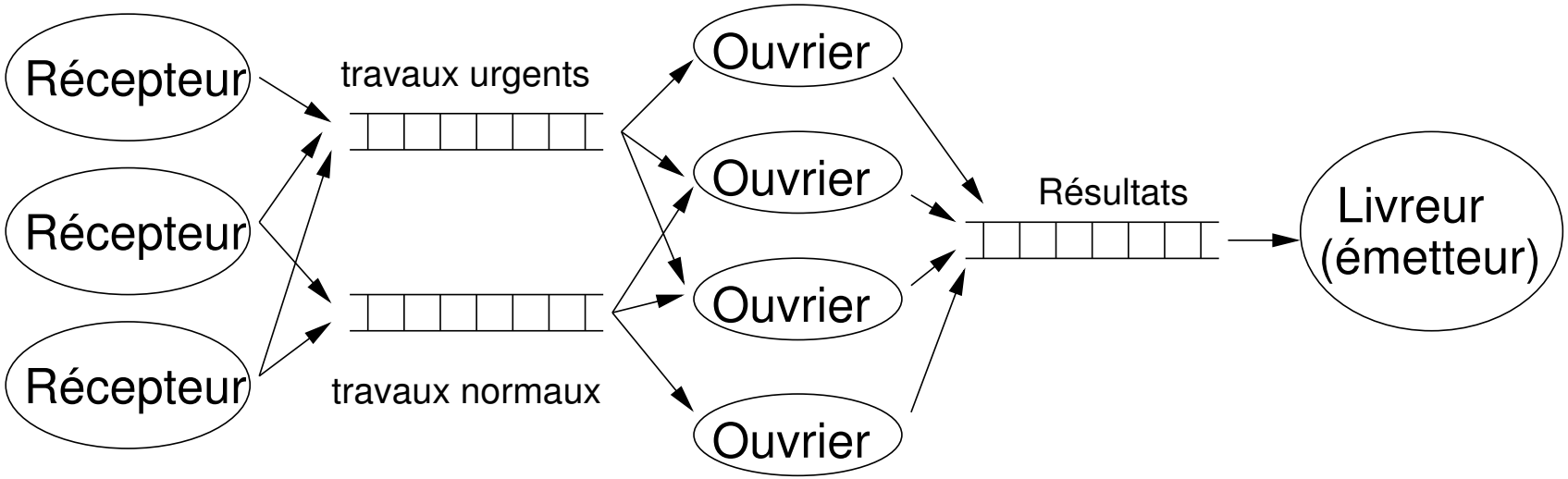
Première partie

Introduction

Plan

1. Activités concurrentes
   - Le problème
   - Un peu d'architecture
   - Communication & processus
2. Conception
   - Comment contrôler (réaliser) la composition ?
   - Comment décrire ?
   - Comment raisonner ?
3. Avantages/inconvénients

Le problème



- coopération : les activités « se connaissent »
- compétition : les activités « s'ignorent »
- vitesse d'exécution arbitraire

Vocabulaire

parallélisme (vrai parallélisme) : le fait que des activités s'exécutent simultanément. Possible uniquement sur un système multi-processeurs.

concurrence (ou parallélisme simulé, ou pseudo-parallélisme) : illusion que des activités s'exécutent simultanément.

asynchronisme : décrit le fait que des activités (ou événements) se produisent de manière indépendante.

synchronisation : apparaît quand il existe des dépendances entre les activités.

Différence avec la programmation séquentielle

- pluralité d'activités « simultanées »
  - explosion de l'espace d'états
  - $\Rightarrow$ nécessité de méthodes et d'outils (conceptuels et logiciels) pour le raisonnement et le développement
- interdépendance des activités
  - logique : production/utilisation de résultats intermédiaires
  - chronologique : disponibilité des résultats

Composants matériels

- Processeurs
- Mécanisme d'interconnexion
- Mémoire et caches

Processeur

Vision simpliste : à chaque cycle, le processeur exécute une instruction machine à partir d'un flot séquentiel (le code).
En pratique :

- pipeline : plusieurs instructions en cours dans un même cycle : obtention, décodage, exécution, écriture du résultat
- superscalaire : plusieurs unités d'exécution
- instructions vectorielles
- réordonnancement (out-of-order)

Interconnexion

- Bus
  - médium à diffusion
  - interconnecte des processeurs entre eux
  - interconnecte les processeurs et la mémoire
  - interconnecte les processeurs et des unités d'E/S
- Mini réseaux locaux (parallélisme massif)
- Réseaux locaux classiques (système réparti)

Mémoire

La mémoire et le processeur sont éloignés : un accès mémoire est considérablement plus lent que l'exécution d'une instruction (peut atteindre un facteur 100 dans un ordinateur, 10000 en réparti).
Principe de localité :

temporelle si on utilise une adresse, on l'utilisera probablement de nouveau dans peu de temps

spatiale si on utilise une adresse, on utilisera probablement une adresse proche dans peu de temps

⇒ conserver près du CPU les dernières cases mémoire accédées
⇒ Cache : mémoire rapide proche du processeur

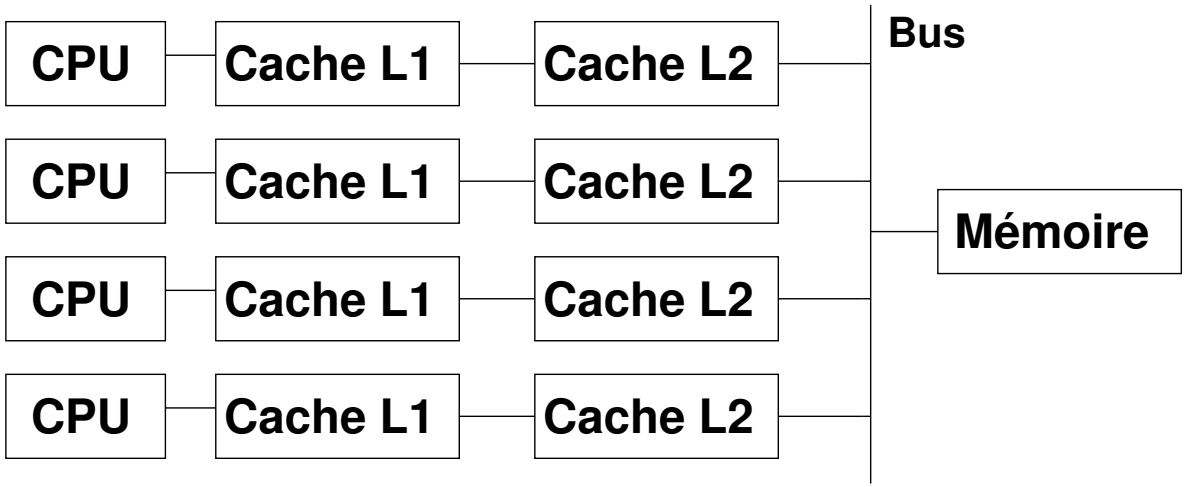
Plusieurs niveaux de caches : de plus en plus gros, de moins en moins rapides (actuellement 3 niveaux).

Cache

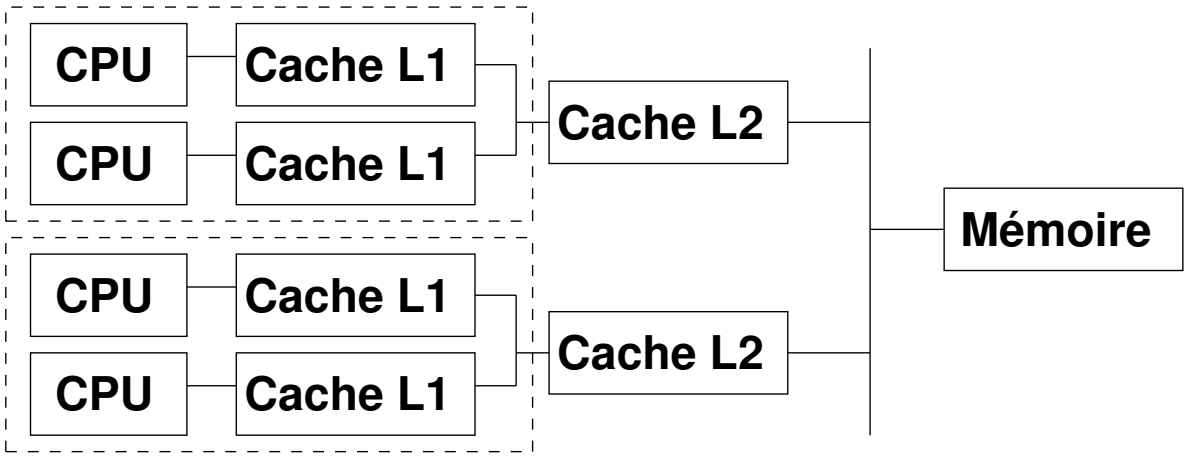
- Différents principes de placement dans le cache (associatif, mappé, k-associatif...)
- Différentes stratégies de remplacement (LRU - least recently used...)
- Différentes stratégies d'invalidation - cohérence mémoire

Architecture multi-processeurs

Multi-processeurs « à l'ancienne » :

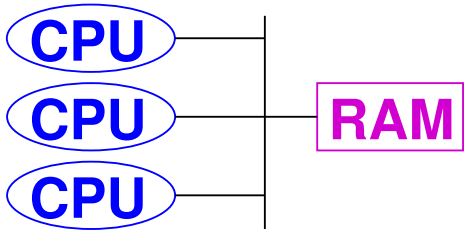


Multi-processeurs multi-cœurs :

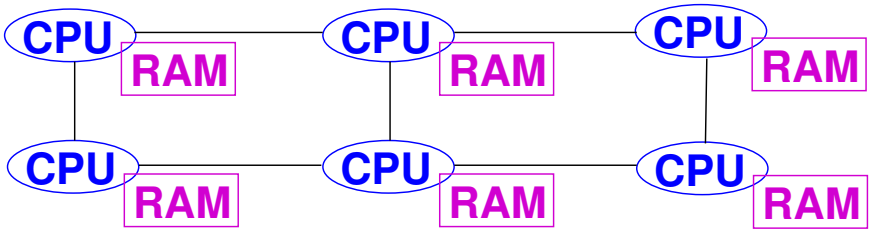


Architecture multi-processeurs

SMP Symmetric multiprocessor : une mémoire + un ensemble de processeurs



NUMA Non-Uniform Memory Access : un graphe d'interconnexion de {CPU+mémoire}



CC-NUMA Cache-Coherent Non-Uniform Memory Access

Écritures en mémoire

Comme fonctionne l'écriture d'une case mémoire en présence de caches ?

Write-Through diffusion sur le bus à chaque valeur écrite

- \+ visible par les autres processeurs ⇒ invalidation des valeurs passées
- \+ la mémoire et le cache sont cohérents
- − trafic inutile : écritures répétées, écritures de variables privées au thread

Write-Back diffusion uniquement à l'éviction de la ligne

- \+ trafic minimal
- − cohérence cache - mémoire - autres caches ?

Cohérence mémoire

Si un processeur écrit la case d'adresse a_1, quand les autres processeurs verront-ils cette valeur ? Si plusieurs écritures consécutives en a_1, $a_2 \ldots$, sont-elles vues dans cet ordre ? Et les lectures indépendantes d'une écriture ?

Règles de cohérence mémoire

Cohérence séquentielle le résultat d'une exécution parallèle est le même que celui d'une exécution séquentielle qui respecte l'ordre partiel de chacun des processeurs.

Cohérence PRAM (pipelined RAM ou fifo) les écritures d'un même processeur sont vues dans l'ordre où elles ont été effectuées ; des écritures de processeurs différents peuvent être vues dans des ordres différents.

Cohérence « lente » (slow consistency) : une lecture retourne *une* valeur précédemment écrite, sans remonter dans le temps.

Cohérence Mémoire – exemple

Init : $x = 0 \wedge y = 0$

Processeur P1	Processeur P2
(1) $x \leftarrow 1$	(a) $y \leftarrow 1$
(2) $t1 \leftarrow y$	(b) $t2 \leftarrow x$

Un résultat $t1 = 0 \wedge t2 = 0$ est possible en cohérence PRAM et slow, impossible en cohérence séquentielle.

Activité

Activité/processus/tâches/threads/processus légers/...

- exécution d'un programme séquentiel
- entité logicielle
- exécutable par un processeur
- interruptible et commutable

La communication

Mémoire partagée

Système centralisé multi-tâches

- communication implicite, résultant de l'accès à des variables partagées
- processus anonymes (pas d'identification nécessaire)

Messages

Système réparti

- communication explicite par transfert de données
- désignation nécessaire du destinataire (l'activité ou un intermédiaire)
- synchronisation implicite

Activités

	processus légers	processus communicants	processus Unix
communication	mémoire	message	fichier/tube/signal
mémoire partagée	oui	non	non (sauf...)
désignation explicite	non	(du canal)	non/indirecte/oui
primitives de synchro	évoluées	(communication)	pauvres

Rq : un processus unix est plus une unité d'allocation que d'exécution.

Plan

1 Activités concurrentes
- Le problème
- Un peu d'architecture
- Communication & processus

2 Conception
- Comment contrôler (réaliser) la composition ?
- Comment décrire ?
- Comment raisonner ?

3 Avantages/inconvénients

Contrôler

Concevoir une application concurrente

- contrôler un ensemble d'activités concurrentes
- contrôler la progression et les interactions de chaque activité
- assurer leur protection réciproque

Moyen

attente (par blocage – suspension – de l'activité)

Contrôler

Hypothèses de bon comportement des activités : un protocole définit les interactions possibles :

- Opérations et paramètres autorisés.
- Séquences d'actions autorisées.
  Un ouvrier ne doit pas déposer plus de résultats qu'il n'a pris de travaux.

Décrire

Compteurs d'événements

Compter les actions ou les changements d'états et les relier entre eux.

Exemple

$$\#\text{nb de travaux soumis} = \#\text{nb travaux effectués} + \#\text{nb travaux en cours} + \#\text{nb travaux en attente}$$

Décrire

Triplets de Hoare

précondition/action/postcondition

Exemple

$\{t = \text{nb travaux en attente} \wedge t > 0 \wedge r = \text{nb résultats}\}$
ouvrier effectue un travail
$\{\text{nb travaux en attente} = t - 1 \wedge \text{nb résultats} = r + 1\}$

Sérialisation : $\dfrac{\{p\}A_1; A_2\{q_{12}\},\ \{p\}A_2; A_1\{q_{21}\}}{\{p\}A_1 \parallel A_2\{q_{12} \vee q_{21}\}}$

Indépendance :

$$\frac{A_1 \text{et } A_2 \text{ sans interférence},\ \{p\}A_1\{q_1\},\ \{p\}A_2\{q_2\}}{\{p\}A_1 \parallel A_2\{q_1 \wedge q_2\}}$$

Décrire

Propriétés temporelles

Linéaires : pour toutes les exécutions possibles,
à tout moment d'une exécution.

Arborescentes : pour certaines exécutions possibles,
à tout moment d'une exécution.

Exemple

- Sûreté : rien de mauvais ne se produit
  Deux ouvriers ne peuvent jamais prendre le même travail.
- Vivacité : quelque chose de bon finit par se produire
  Un travail déposé finit par être pris par un ouvrier.
- Possibilité : *deux travaux déposés consécutivement peuvent être exécutés séquentiellement par le même ouvrier.*

Modèle : l'entrelacement

Raisonner sur tous les cas parallèles est trop complexe
$\Rightarrow$ on raisonne sur des exécutions séquentielles obtenues par entrelacement des instructions des différentes activités.

Deux activités $P = p_1; p_2$ et $Q = q_1; q_2$. L'exécution concurrente $P \| Q$ est vue comme (équivalente à) l'une des exécutions :
$p_1; p_2; q_1; q_2$ ou $p_1; q_1; p_2; q_2$ ou $p_1; q_1; q_2; p_2$ ou $q_1; p_1; p_2; q_2$ ou $q_1; p_1; q_2; p_2$ ou $q_1; q_2; p_1; p_2$

Nombre d'entrelacements : $\frac{(p+q)!}{p!\ q!}$

Attention

- Ne pas oublier que c'est un modèle simplificateur (vraie concurrence, cohérence mémoire...)
- Il peut ne pas exister de code séquentiel équivalent au code parallèle.

Raisonner

Contrôler les effets des interactions

- isoler (raisonner indépendamment) $\Rightarrow$ modularité
- contrôler/spécifier l'interaction
- schémas connus d'interaction (design patterns)

Plan

1. Activités concurrentes
   - Le problème
   - Un peu d'architecture
   - Communication & processus
2. Conception
   - Comment contrôler (réaliser) la composition ?
   - Comment décrire ?
   - Comment raisonner ?
3. Avantages/inconvénients

Avantages/inconvénients

+ utilisation d'un système multi-processeurs.
+ utilisation de la concurrence naturelle d'un programme.
+ modèle de programmation naturel, en explicitant la synchronisation nécessaire.
– surcoût d'exécution (synchronisation, implantation du pseudo-parallélisme).
– surcoût de développement : nécessité d'expliciter la synchronisation, vérifier la réentrance des bibliothèques, danger des variables partagées.
– surcoût de mise-au-point : debuggage souvent délicat (pas de flot séquentiel à suivre) ; effet d'interférence entre des activités, interblocage...

Parallélisme et performance

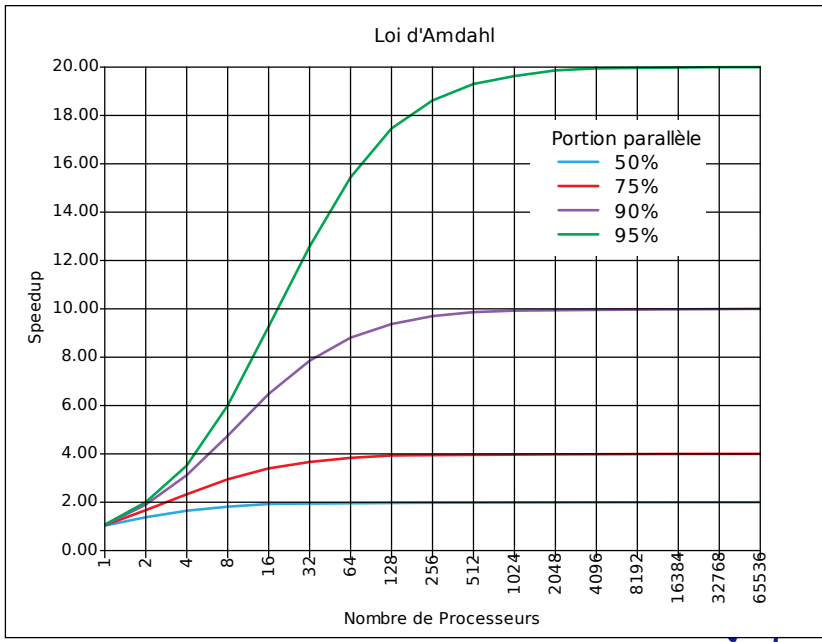
Mythe du parallélisme

« Si je remplace ma machine mono-processeur par une machine à N processeurs, mon programme ira N fois plus vite »

Soit un système composé par une partie p parallélisable + une partie $1-p$ séquentielle.

CPU	durée	$p = 40\%$	$p = 80\%$
1	$p + (1-p)$	100	100
4	$\frac{p}{4} + (1-p)$	70	40
8	$\frac{p}{8} + (1-p)$	65	30
16	$\frac{p}{16} + (1-p)$	62,5	25
∞		60	20

Loi d'Amdahl : maximal speedup $= \frac{1}{1-p}$



(source : wikicommons)

Parallélisme et performance

Mythe de la performance

« Si je remplace ma machine par une machine N fois plus rapide, mon programme traitera des problèmes N fois plus grands dans le même temps »

Pour un problème de taille n soluble en temps T, taille de problème soluble dans le même temps sur une machine N fois plus rapide :

complexité	$N = 4$	$N = 16$	$N = 1024$
$O(n)$	$4n$	$16n$	$1024n$
$O(n^2)$	$\sqrt{4}n = 2n$	$\sqrt{16}n = 4n$	$\sqrt{1024}n = 32n$
$O(n^3)$	$\sqrt[3]{4}n \approx 1.6n$	$\sqrt[3]{16}n \approx 2.5n$	$\sqrt[3]{1024}n \approx 10n$
$O(e^n)$	$\ln(4)n \approx 1.4n$	$\ln(16)n \approx 2.8n$	$\ln(1024)n \approx 6.9n$

En supposant en outre que tout est 100% est parallélisable et qu'il n'y a aucune interférence !