Slide 1

INF7440 Conception et analyse d'algorithmes :

Introduction

Paradigme = "Modèle théorique de pensée qui oriente la recherche et la réflexion scientifique"
(Le Petit Larousse 1997)

Paradigme de programmation $\approx$ Façon d'aborder un problème de programmation, généralement à l'aide d'un type de langage qui supporte bien certains mécanismes d'abstractions

- Paradigme procédurale $\Rightarrow$ procédures et sous-routines
- Paradigme orienté objets $\Rightarrow$ classe d'objets
- Paradigme fonctionnel $\Rightarrow$ valeurs et fonctions (mathématiques)

Paradigme comme façon de voir le monde

"*Quand le seul outil qu'on connait est le marteau, on voit des clous partout!*"

Slide 2

Soit l'algorithme suivant (maximum parmi une série de nombres) :

```
PROCEDURE max( s: sequence{integer}; i, j: nat ): integer
# PRECONDITION
#   i <= j & i IN domain(s) & j IN domain(s)
DEBUT
   max <- s[i]
   POUR k <- i+1 A j FAIRE
      SI s[k] > max ALORS
        max <- s[k]
      FIN
   FIN
   RETOURNER( max )
FIN
```

Quel sera le temps d'exécution pour une séquence S de longueur n?
Plus précisément, quelle est la complexité asymptotique de cet algorithme?

Question : Est-il possible de faire mieux, c'est-à-dire d'obtenir un temps d'exécution qui soit inférieur à $O(n)$ pour une séquence de longueur $n = 2^k$? Si oui, comment?

Slide 3

Solution alternative :

```
PROCEDURE max( s: sequence{integer}; i, j: nat ): integer
# PRECONDITION
#   i <= j & i IN domain(s) & j IN domain(s)
DEBUT
  SI i = j ALORS
    RETOURNER( s[i] )
  SINON
    m <- (i+j) / 2
    max1 <- max( s[i..m] )
    max2 <- max( s[m+1..j] )
    SI max1 > max2 ALORS
      RETOURNER( max1 )
    SINON
      RETOURNER( max2 )
    FIN
  FIN
FIN
```

Question : Quel sera le temps d'exécution de cette algorithme pour une séquence `S` de longueur n ($n = 2^k$)?

Slide 4

Sortons maintenant du cadre habituel où tout s'exécute de façon *séquentielle*.

Supposons que les appels de fonction se fassent *en parallèle*.

```
PROCEDURE max( s: sequence{integer}; i, j: nat ): integer
DEBUT
  SI i = j ALORS
    RETOURNER( s[i] )
  SINON
     m <- (i+j) / 2
     EN PARALLELE
       max1 <- max( s[i..m] )
       max2 <- max( s[m+1..j] )
     FIN
     SI max1 > max2 ALORS
       RETOURNER( max1 )
     SINON
       RETOURNER( max2 )
     FIN
  FIN
FIN
```

Question : Quel sera alors le temps d'exécution, en fonction de $n = 2^k$?

Slide 5

Autre exemple : trouver la fin d'une liste chaînée

Soit une liste chainée séquentielle (voir figure au tableau).

Supposons que chaque noeud de la liste soit sur un processeur différent.

Supposons que chaque noeud possède un pointeur vers le noeud `suivant`.

Question : Quelle sera la complexité asymptotique du temps d'exécution pour que chaque pointeur `suivant` indique le dernier élément de la liste?

Slide 6

Soit l'algorithme suivant :

```
TANTQUE il existe un noeud tel que suivant != null
                                ET suivant->suivant != null FAIRE
  POUR chacun des processeurs EN PARALLELE FAIRE
    SI (suivant != null) ET (suivant->suivant != null) ALORS
      tmp <- suivant->suivant
      suivant <- tmp
    FIN
  FIN
FIN
```

Question : Quelle est alors la complexité asymptotique du temps d'exécution, pour une liste de longueur $n = 2^k$?

Slide 7

Introduction à la programmation concurrente et parallèle

Ce qu'est la programmation concurrente et parallèle

Programme séquentiel

= programme défini par une *séquence* d'actions

$\Rightarrow$ une instruction après l'autre

= Un processus, une tâche, un *thread*

thread = fil d'exécution

Slide 8

Programme concurrent

= programme qui contient *plusieurs* processus (ou *thread*s) qui *coopèrent*

Coopération $\Rightarrow$ Communication, échange d'information

Deux principales façons de communiquer :

- Par l'intermédiaire de variables *partagées*
- Par l'échange de messages et de signaux

Niveau matériel (*hardware*)

Différentes classes de matériel

- Machine uniprocesseur : boîte contenant un seul processeur
- Multi-processeurs : boîte contenant plusieurs processeurs
  - Communication via un *bus*
- Multi-ordinateurs : plusieurs boîtes inter-connectées
  - Communication via un réseau

Slide 9

Différents types d'applications concurrentes

Application multi-contextes (*multi-threaded***)** = contient plusieurs *thread*s

Note importante : On considère généralement un *thread* comme étant un processus léger (*lightweight thread*)

Utilisations : Pour mieux organiser/structurer une application (plus grande modularité)

- Système d'exploitation multi-tâches
- Fureteurs multi-tâches
- Interface personnes-machines vs. application
- . . .

Slide 10

Application parallèle = chaque processus s'exécute sur son propre processeur

Utilisations : Pour résoudre plus rapidement un problème ou pour résoudre un problème plus gros

- Prévisions météorologiques
- Prospection minière
- Physique moderne
- Bio-informatique (génomique)
- . . .

Slide 11

Application distribuée = les processus communiquent entre eux par l'intermédiaire d'un réseau ($\Rightarrow$ délais plus longs)

Utilisations :

- Serveurs de fichiers
- Accès à distance à des banques de données

Slide 12

Emphase du cours INF7440 :
Conception *et analyse des algorithmes séquentiels et parallèles*