

**2. Algorithmique parallèle**

Introduction au parallélisme 1

---

---

---

---

---

---

---

---

**Plan**

---

1. Le modèle graphe de tâches
  1. Définition
  2. Exemples élémentaires
2. Applications et algorithmique
  1. Primitives et fonctions génériques
  2. Applications simples
  3. Applications difficiles
3. Scheduling
  1. Statique
  2. Dynamique

Introduction au parallélisme 2

---

---

---

---

---

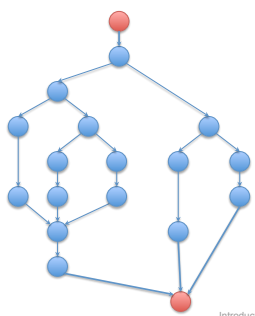
---

---

---

**Un Modèle générique du parallélisme: le DAG**

---



- Directed Acyclic Graph
- Nœud: une tâche
  - instruction, fonction, programme
  - Implémentée par processus, thread
- Arête: séquentialisation
- Donc acyclique
- Nœuds distingués début/fin

Introduction au parallélisme 3

---

---

---

---

---

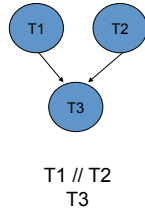
---

---

---

### Un Modèle générique du parallélisme: le DAG

- Un DAG définit un ordre partiel
  - $T1 \ll T2$  s'il existe un chemin de  $T1$  vers  $T2$
  - Ordre total : programme séquentiel
  - Ordre partiel : programme parallélisable



Introduction au parallélisme

4

---

---

---

---

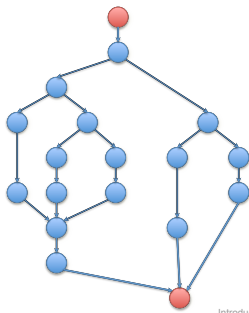
---

---

---

---

### Un Modèle générique du parallélisme: le DAG



- Directed Acyclic Graph
- La réalisation du parallélisme est vue uniquement comme un problème d'ordonnement

Introduction au parallélisme

5

---

---

---

---

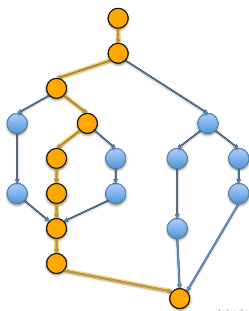
---

---

---

---

### Un Modèle générique du parallélisme: le DAG



- $T_{\infty}$  est la longueur pondérée du chemin critique (**span**, profondeur). C'est la borne inférieure du temps de calcul
- $T_1$  la **charge**, est la somme des temps d'exécution de tous les nœuds – exécution séquentielle. C'est la borne supérieure du temps de calcul.
- Indice de Parallélisme:  $T_1 / T_{\infty}$   
Quantité moyenne de travail par étape le long du chemin critique

Introduction au parallélisme

6

---

---

---

---

---

---

---

---

## CILK

Les exemples de cette partie seront donnés en CILK

<http://supertech.csail.mit.edu/cilk/>

- Langage adapté à l'algorithmique parallèle récursive
- Exécutif
- Développé au MIT puis industrialisé en CILK++
- Nous n'utiliserons que les constructions élémentaires, mais il y en a beaucoup d'autres.

Introduction au parallélisme

7

---

---

---

---

---

---

---

---

## Basic Cilk Keywords

```
cilk int fib (int n) {  
  if (n<2) return (n);  
  else {  
    int x,y;  
    x = spawn fib(n-1);  
    y = spawn fib(n-2);  
    sync;  
    return (x+y);  
  }  
}
```

Identifies a function as a *Cilk procedure*, capable of being spawned in parallel.

The named *child* Cilk procedure can execute in parallel with the *parent* caller.

Control cannot pass this point until all spawned children have returned.

© 2006 Charles E. Leiserson

Multithreaded Programming in Cilk — LECTURE 1

July 13, 2006 8

---

---

---

---

---

---

---

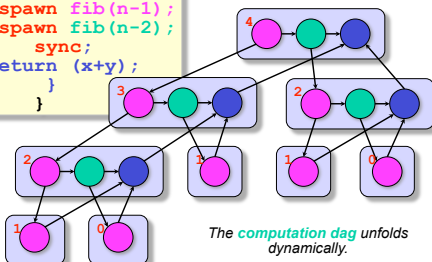
---

## Dynamic Multithreading

```
cilk int fib (int n) {  
  if (n<2) return (n);  
  else {  
    int x,y;  
    x = spawn fib(n-1);  
    y = spawn fib(n-2);  
    sync;  
    return (x+y);  
  }  
}
```

Example: fib(4)

"Processor oblivious"



© 2006 Charles E. Leiserson

Multithreaded Programming in Cilk — LECTURE 1

July 13, 2006 9

---

---

---

---

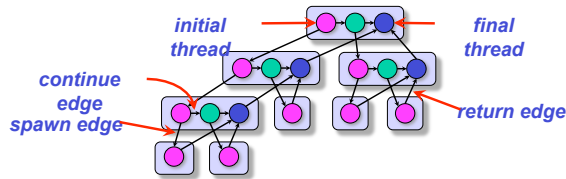
---

---

---

---

## Multithreaded Computation



- The dag  $G = (V, E)$  represents a parallel instruction stream.
- Each vertex  $v \in V$  represents a *(Cilk) thread*: a maximal sequence of instructions not containing parallel control (**spawn**, **sync**, **return**).
- Every edge  $e \in E$  is either a *spawn* edge, a *return* edge, or a *continue* edge.

© 2006 Charles E. Leiserson Multithreaded Programming in Cilk — LECTURE 1 July 13, 2006 10

---

---

---

---

---

---

---

---

---

---

---

---

## Anatomie d'une application simplissime: additionner deux vecteurs

Séquentiel

```
void vadd (double *X, double *Y, double *Z,int n){
  int i; for (i=0; i<n; i++)
    Z[i]= X[i]+Y[i];}
```

Stratégie de parallélisation:

1. convertir la boucle en récursion

```
void vadd (double *X, double *Y, double *Z,int n) {
  if (n <= B) {
    int i; for (i=0; i<n; i++)
      Z[i]= X[i]+Y[i];}
  else {
    vadd(X, Y ,Z, n/2);
    vadd(X+n/2, Y+n/2 ,Z+n/2, n-n/2);}
}
```

Introduction au parallélisme

11

---

---

---

---

---

---

---

---

---

---

---

---

## Anatomie d'une application simplissime: additionner deux vecteurs

Séquentiel

```
void vadd (double *X, double *Y, double *Z,int n){
  int i; for (i=0; i<n; i++)
    Z[i]= X[i]+Y[i];}
```

Stratégie de parallélisation:

2. expliciter le parallélisme

```
cilk vadd (double *X, double *Y, double *Z,int n) {
  if (n <= B) {
    int i; for (i=0; i<n; i++)
      Z[i]= X[i]+Y[i];}
  else {
    spawn vadd(X, Y ,Z, n/2);
    spawn vadd(X+n/2, Y+n/2 ,Z+n/2, n-n/2);
    sync;}
}
```

Introduction au parallélisme

12

---

---

---

---

---

---

---

---

---

---

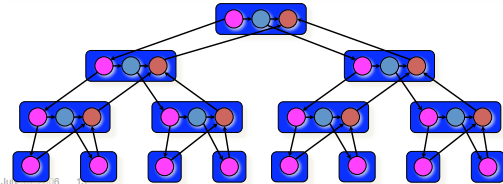
---

---

### Parallélisation récursive

```

cilk vadd (double *X, double *Y, double *Z, int n) {
  if (n <= B) {
    int i; for (i=0; i<n; i++)
      Z[i]= X[i]+Y[i];
  } else {
    spawn vadd(X, Y, Z, n/2);
    spawn vadd(X+n/2, Y+n/2, Z+n/2, n-n/2);
    sync;
  }
}
    
```




---

---

---

---

---

---

---

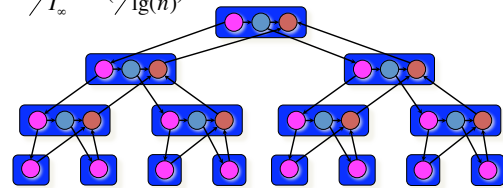
---

### Parallélisation récursive

$$T(1) = \Theta(n)$$

$$T_\infty = \Theta(\lg(n))$$

$$T_1/T_\infty = \Theta(n/\lg(n))$$



BASE  
July 13, 2008 14

---

---

---

---

---

---

---

---

### Avantages

- Stratégie de parallélisation:
1. Convertir la boucle en récursion
  2. Expliciter le parallélisme

- C'est une stratégie *Divide & Conquer* qui favorise la localité **ET** la répartition de charge
- La performance ne dépend pas de B au premier ordre
- Mais synchronisations inutiles

---

---

---

---

---

---

---

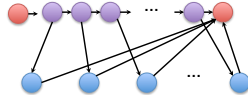
---

## Gestion explicite des tâches

```

cilk void vadd1 (double *X, double *Y, double *Z){
  int i; for (i=0; i<B; i++)
    Z[i]= X[i]+Y[i];
}
cilk void vadd (double *X, double *Y, double *Z){
  int j; for (j=0; j<N; j+=B) {
    spawn vadd1(X+j, Y+j, Z+j);
  }
  sync;
}
    
```

Aussi à la PVM, threads



$$T_{\infty} = B + N/B$$

$$T_1 = N$$

Conclusion ?

Réaliste : l'addition et le spawn sont au mieux du même ordre

---

---

---

---

---

---

---

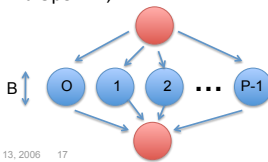
---

## Une autre vision du même algorithme

```

void vadd1 (double *X, double *Y, double *Z){
  int i; for (i=0; i<B; i++)
    Z[i]= X[i]+Y[i];
}
void vadd (double *X, double *Y, double *Z){
  int j; parfor (j=0; j<N; j+=B) {
    vadd1(X+j, Y+j, Z+j);
  }
}
    
```

• A la OpenMP, ...



$$T_{\infty} = B$$

$$T_1 = N$$

$$T_1/T_{\infty} = N/B$$

Conclusion ?

July 13, 2006 17

---

---

---

---

---

---

---

---

## Conclusion sur l'exemple simplissime

- L'approche Divide & Conquer est la plus robuste
- Le modèle par graphe de tâches dépend de finesse de la représentation des tâches
  - Granularité: degré de repliement du parallélisme maximal.
  - Réalisme: description des actions de gestion
- Le modèle par graphe de tâches représente assez naturellement la programmation en EAU, mais pas en EAM

---

---

---

---

---

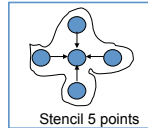
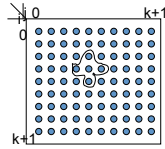
---

---

---

## Relaxation

- Calculer en chaque point la moyenne des 4 voisins
- Tant que la différence entre le tableau actuel et le précédent n'est pas négligeable



19 Introduction au parallélisme

---

---

---

---

---

---

---

---

## La relaxation : code séquentiel

```
double X[k+1], Y[k+1], err;
<random init Y>
```

```
while (err > epsilon) {
```

```
  for (i = 1; i <= k ; i++)
```

```
    for (j = 1; j <= k ; j++)
```

```
      X[i,j] = 0.25*(Y[i+1,j] + Y[i-1,j] + Y[i, j-1] + Y[i, j+1]);
```

```
  for (i = 1; i <= k ; i++)
```

```
    for (j = 1; j <= k ; j++)
```

```
      err += abs(X[i,j] - Y[i,j]);
```

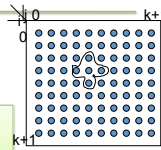
```
  for (i = 1; i <= k ; i++)
```

```
    for (j = 1; j <= k ; j++)
```

```
      Y[i,j] = X[i,j];
```

Parallèle

Introduction au parallélisme




---

---

---

---

---

---

---

---

## La relaxation : code séquentiel

```
double X[k+1], Y[k+1], err;
```

```
<random init Y>
```

```
while (err > epsilon) {
```

```
  for (i = 1; i <= k ; i++)
```

```
    for (j = 1; j <= k ; j++)
```

```
      X[i,j] = 0.25*(Y[i+1,j] + Y[i-1,j] + Y[i, j-1] + Y[i, j+1]);
```

```
  for (i = 1; i <= k ; i++)
```

```
    for (j = 1; j <= k ; j++)
```

```
      err += abs(X[i,j] - Y[i,j]);
```

```
  for (i = 1; i <= k ; i++)
```

```
    for (j = 1; j <= k ; j++)
```

```
      Y[i,j] = X[i,j];
```

Parallèle ?

Introduction au parallélisme

---

---

---

---

---

---

---

---

## Plan

1. Le modèle graphe de tâches
  1. Définition
  2. Exemples élémentaires
2. Applications et algorithmique
  1. Primitives et fonctions génériques
  2. Applications simples
  3. Applications difficiles
3. Scheduling
  1. Statique
  2. Dynamique

Introduction au parallélisme

22

---

---

---

---

---

---

---

---

## Primitives

- Problème: calculer et communiquer des valeurs conceptuellement uniques lorsque
  - En espace d'adressages multiples, les données sont distribuées
  - En espace d'adressage unique, les données sont privées
- Parallélisme de données

Introduction au parallélisme

23

---

---

---

---

---

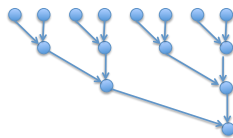
---

---

---

## La réduction

- $s = f(x_0, \dots, x_{n-1})$  où  $f$  est « associative et commutative » : addition, produit max, min, ...



- Réalisations optimisées spécifiques de chaque architecture, visibles à travers des constructions du langage
  - OpenMP: `reduction(+:result)`
  - `MPI_reduce`

$S(n, n) = ?$   
 $S(n, p) = ?$

Introduction au parallélisme

24

---

---

---

---

---

---

---

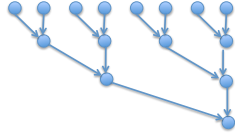
---



## La réduction

- Où est le résultat ?

- OpenMP: partout, plus précisément dans une variable partagée
- MPI\_reduce : sur un seul processeur
- Si on veut l'avoir sur tous les processeurs, MPI\_Allreduce
- Pourquoi ?



Introduction au parallélisme

25

---

---

---

---

---

---

---

---

## Sémantique du parallélisme et traitement d'erreurs

« Notes on collective operations: The reduction functions (MPI\_Op) do not return an error value. As a result, if the functions detect an error, all they can do is either call MPI\_Abort or silently skip the problem. Thus, if you change the error handler from MPI\_ERRORS\_ARE\_FATAL to something else, for example, MPI\_ERRORS\_RETURN, then no error may be indicated. The reason for this is the performance problems in ensuring that all collective routines return the same error value. » Manuel MPI

Introduction au parallélisme

26

---

---

---

---

---

---

---

---

## Gather et Scatter : compacter et décompacter des données

Gather	$A[i] = B[L[i]]$
Scatter	$B[L[i]] = A[i]$

- En espaces d'adressages multiples
  - Gather : récupérer un tableau à partir de tableaux répartis sur processeurs
  - Scatter : distribuer un tableau aux processeurs
  - Dans quel ordre ?
- En espace d'adressage unique, compacter/décompacter un tableau dans un autre
  - En parallèle, quelle sémantique si L n'est pas injective ? Modèles PRAM
  - Quelle relation avec le tri ?

Introduction au parallélisme

27

---

---

---

---

---

---

---

---

## Applications élémentaires

- Retour sur la relaxation
- Produit de matrices
- Relaxation de Gauss-Seidel
  - Et remplissage dynamique d'un tableau
- Merge Sort (en TD)

Introduction au parallélisme

28

---

---

---

---

---

---

---

---

## Relaxation : un schéma générique pour

- Résolution numérique d'EDP par méthode des éléments finis – « toute » la simulation numérique traditionnelle
- Certaines applications de traitement d'images
- En général, méthodes de point fixe lorsque le voisinage n'est pas trop grand

$$F(X) = X$$

Sous des conditions assez générales, l'itération

$$X_{n+1} = F(X_n)$$

à partir de  $X_0$  arbitraire converge vers le point fixe

Introduction au parallélisme

29

---

---

---

---

---

---

---

---

## Master Method

Problème : résoudre

$$T(n) = aT(n/b) + f(n)$$

Avec  $a \geq 1$ ,  $b > 1$  et  $f(n)$  asymptotiquement positive

**Idée: comparer  $f(n)$  et  $n^{\log_b a}$**

Introduction au parallélisme

30

---

---

---

---

---

---

---

---

### Master Method – Cas 1

$$T(n) = aT(n/b) + f(n)$$

$$n^{\log_b a} \gg f(n),$$

spécifiquement  $f(n) = O(n^{\log_b a - \epsilon})$  pour  $\epsilon > 0$

$$\text{alors } T(n) = \Theta(n^{\log_b a})$$

Exemple :  $a = 4, b = 2, f(n) = \Theta(n)$  ou  $\Theta(1)$

$$n^{\log_b a} = n^2$$

$$T(n) = \Theta(n^2)$$

Introduction au parallélisme

31

---

---

---

---

---

---

---

---

### Master Method – Cas 2

$$T(n) = aT(n/b) + f(n)$$

$$n^{\log_b a} \approx f(n),$$

spécifiquement  $f(n) = \Theta(n^{\log_b a} \lg^k n)$  pour  $k \geq 0$

$$\text{alors } T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$$

Exemple :  $a = 1, f(n) = \Theta(\lg n)$  ou  $\Theta(1)$

$$n^{\log_b a} = 1$$

$$T(n) = \Theta(\lg^2 n) \text{ ou } \Theta(\lg n)$$

Introduction au parallélisme

32

---

---

---

---

---

---

---

---

### Master Method – Cas 3

$$T(n) = aT(n/b) + f(n)$$

$$n^{\log_b a} \ll f(n),$$

spécifiquement  $f(n) = \Omega(n^{\log_b a + \epsilon})$  pour  $\epsilon > 0$

et  $af(n/b) \leq cf(n)$  pour  $0 < c < 1$

$$\text{alors } T(n) = \Theta(f(n))$$

Introduction au parallélisme

33

---

---

---

---

---

---

---

---

### Produit de matrices

$$\begin{matrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{matrix} = \begin{matrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{matrix} \times \begin{matrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \dots & b_{nn} \end{matrix}$$

**C**                      **A**                      **B**

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

July 14, 2006 34

---

---

---

---

---

---

---

---

### Programme séquentiel naïf

```
<init C>
for (i= 0; i < n; i++)
  for (j = 0 ; j < n; j++)
    for (k = 0; k <n; k++)
      C[i,j] = A[i,k]*B[k,j];
```

Introduction au parallélisme

35

---

---

---

---

---

---

---

---

### Programme parallèle naïf

```
<init C>
parfor (i= 0; i < n; i++)
  parfor (j = 0 ; j < n; j++) {
    parfor (k = 0; k <n; k++)
      temp[i,j,k] = A[i,k]*B[k,j];
    C[i,j] = Sum_reduce (3, temp);
  }
```

En réalité, temp est réalisé soit comme une variable privée (EAU), soit comme une variable locale (EAM)

Introduction au parallélisme

36

---

---

---

---

---

---

---

---

## Parallélisation naïve

- Parallélisme non borné
  - Lancer les  $n^3$  calculs en parallèle
  - Lancer les  $n^2$  réductions en parallèle

$$T_1 = \Theta(n^3)$$

$$T_\infty = \Theta(\lg(n))$$

$$T_1/T_\infty = \Theta(n^3/\lg(n))$$

- MAIS problème de localité

Introduction au parallélisme

37

---

---

---

---

---

---

---

---

## Produit de matrices par blocs – vision récursive

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \times \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$
$$= \begin{pmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{pmatrix} + \begin{pmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{pmatrix}$$

8 multiplications de matrices  $(n/2) \times (n/2)$ .

1 addition de 2 matrices  $n \times n$ .

July 14, 2006 38

---

---

---

---

---

---

---

---

## Produit de matrices par blocs – vision récursive

```
void Mult(*C, *A, *B, n) {
    double *T = Cilk_alloca(n*n*sizeof(double));
    < base case & partition matrices >
    spawn Mult(C11, A11, B11, n/2);
    spawn Mult(C12, A11, B12, n/2);
    spawn Mult(C22, A21, B12, n/2);
    spawn Mult(C21, A21, B11, n/2);
    spawn Mult(T11, A12, B21, n/2);
    spawn Mult(T12, A12, B22, n/2);
    spawn Mult(T22, A22, B22, n/2);
    spawn Mult(T21, A22, B21, n/2);
    sync;
    spawn Add(C, T, n);
    sync;
    return;
}
```

July 14, 2006 39

---

---

---

---

---

---

---

---

### Addition de matrices

```
void Add(*C, *T, n) {
  <base case & partition matrices>
  spawn Add(C11, T11, n/2);
  spawn Add(C12, T12, n/2);
  spawn Add(C21, T21, n/2);
  spawn Add(C22, T22, n/2);
  sync;
  return;
}
```

$$A_1(n) = 4A(n/2) + \Theta(1) = \Theta(n^2) \quad \text{CAS 1}$$

$$A_\infty(n) = A_\infty(n/2) + \Theta(1) = \Theta(\lg n) \quad \text{CAS 2}$$

July 14, 2006 40

---

---

---

---

---

---

---

---

---

---

### Complexité du produit de matrices par blocs récursif

```
cilk void Mult(*C, *A, *B, n) {
  double*T = Cilk_allocate(n*n*sizeof(double));
  h base case & partition matrices i
  8 { spawn Mult(C11, A11, B11, n/2);
    spawn Mult(C12, A11, B12, n/2);
    :
    spawn Mult(T21, A22, B21, n/2);
    sync;
    spawn Add(C, T, n);
    sync;
    return;
  }
```

$$\begin{aligned} M_1(n) &= 8M_1(n/2) + A_1(n) + \Theta(1) \\ &= 8M_1(n/2) + \Theta(n^2) \quad \text{CAS 1} \\ &= \Theta(n^3) \text{ car } \log_2 8 = 3 \text{ et } n^2 \ll n^3 \end{aligned}$$

---

---

---

---

---

---

---

---

---

---

### Complexité du produit de matrices par blocs récursif

```
cilk void Mult(*C, *A, *B, n) {
  double*T = Cilk_allocate(n*n*sizeof(double));
  h base case & partition matrices i
  8 { spawn Mult(C11, A11, B11, n/2);
    spawn Mult(C12, A11, B12, n/2);
    :
    spawn Mult(T21, A22, B21, n/2);
    sync;
    spawn Add(C, T, n);
    sync;
    return;
  }
```

$$\begin{aligned} M_\infty(n) &= M_\infty(n/2) + \lg n + \Theta(1) = \Theta(\lg^2(n)) \\ M_1(n) / M_\infty(n) &= \Theta(n^3 / \lg^2(n)) \quad \text{CAS 2} \\ &\text{Meilleur en pratique} \end{aligned}$$

---

---

---

---

---

---

---

---

---

---

### Construction de tableau

Remplir un tableau  $n \times n$  avec  
 $A[i, j] = f ( A[i, j-1], A[i-1, j], A[i-1, j-1] )$ .

0	1	0	1	0	1	0
10	11	12	13	14	15	16
20	41	64	89	116	145	176
30	91	196	349	554	815	1136
40	161	448	993	1896	3265	5216
50	251	860	2301	5190	10351	18832
60	361	1472	4633	12124	27665	56848

Complexité  
séquentielle :  $n^2$

July 14, 2006 43

---

---

---

---

---

---

---

---

### Applications

- Relaxation de Gauss-Seidel
  - Même applications que la relaxation de Jacobi
  - Converge beaucoup plus vite
- Programmation dynamique
  - Edit distance
  - Alignement de séquences
  - ...

Introduction au parallélisme

44

---

---

---

---

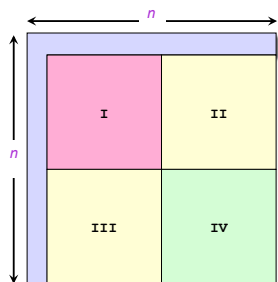
---

---

---

---

### Construction Récursive



```
spawn I ;
sync ;
spawn II ;
spawn III ;
sync ;
spawn IV ;
sync ;
```

DAG ?

July 14, 2006 45

---

---

---

---

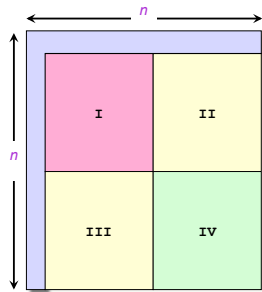
---

---

---

---

### Construction récursive



$$T_{\infty}(n) = 3T_{\infty}(n/2)$$

$$T_{\infty}(n) = \Theta(n^{\lg(3)})$$

$$T_1/T_{\infty} = n^2/n^{\lg(3)}$$

$$\approx n^{0.42}$$

Pour  $n = 1000$ ,

$$T_1/T_{\infty} \approx 17$$

---

---

---

---

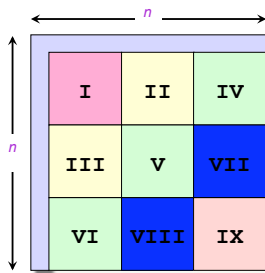
---

---

---

---

### Plus de parallélisme



```

spawn I;
sync;
spawn II;
spawn III;
sync;
spawn IV;
spawn V;
spawn VI;
sync;
spawn VII;
spawn VIII;
sync;
spawn IX;
sync;
    
```

July 14, 2006 47

---

---

---

---

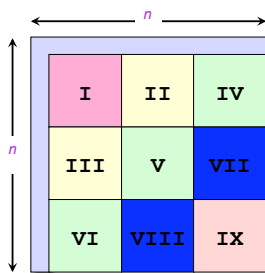
---

---

---

---

### Plus de parallélisme



$$T_{\infty}(n) = 5T_{\infty}(n/3)$$

$$T_{\infty}(n) = n^{\lg_3(5)}$$

$$T_1/T_{\infty} = n^2/n^{\lg_3(5)}$$

$$\approx n^{0.54}$$

Pour  $n = 1000$ ,

$$T_1/T_{\infty} \approx 40$$

July 14, 2006 48

---

---

---

---

---

---

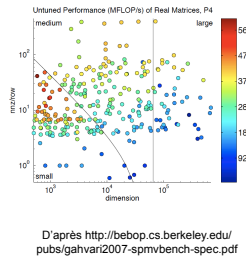
---

---



## Calculs creux

- Toutes les structures de données représentables par une matrice d'adjacence où les zéros sont **très** dominants
  - Maillages
  - Certains Graphes
- On ne peut pas/ne veut pas
  - Stocker les 0
  - Effectuer les calculs inutiles
- Problème de localité



Introduction au parallélisme 49

---

---

---

---

---

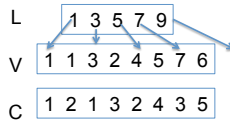
---

---

---

## Représentation CRS

- Compressed Row Storage
- V: valeurs non nulles
- L: L[i] = indice dans V du début de la ligne i
- C: C[k] = numéro de colonne de V[k]

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 3 & 0 & 2 & 0 & 0 \\ 0 & 4 & 0 & 5 & 0 \\ 0 & 0 & 7 & 0 & 6 \end{bmatrix}$$


Pour  $L[i] \leq k < L[i+1]$   
 $V[k] == A[i][C[k]]$

Introduction au parallélisme 50

---

---

---

---

---

---

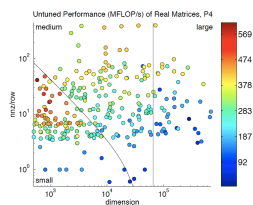
---

---

## Produit matrice-vecteur creux

- Accès par indirection

for i= 1, n  
 for k = L[i], L[i+1] -1  
 $y[i] += V[k]*x[C[k]]$



Introduction au parallélisme 51

---

---

---

---

---

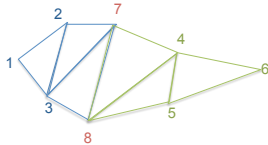
---

---

---

## Décomposition de domaines

- Renuméroter les sommets
- Matrice creuse avec une structure localisée



$$\begin{matrix}
 D1 & D2 & I \\
 D1 & \begin{bmatrix} A1 & 0 & C1 \\ 0 & A2 & C2 \\ B1 & B2 & S1+S2 \end{bmatrix} \\
 D2 & \\
 I & 
 \end{matrix}$$

- Surcoût important, amorti car structure statique ré-exploitée

Introduction au parallélisme

52

---

---

---

---

---

---

---

---

## Parallélisation du produit matrice-vecteur en décomposition de domaines

$$\begin{bmatrix} Y1 \\ Y2 \\ Y3 \end{bmatrix} = \begin{bmatrix} A1 & 0 & C1 \\ 0 & A2 & C2 \\ B1 & B2 & S1+S2 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \\ X3 \end{bmatrix}$$

En parallèle, calcul local optimisé (cache). X3 doit être dupliqué

$$\begin{bmatrix} Y1 \\ Z3 \end{bmatrix} = \begin{bmatrix} A1 & C1 \\ B1 & S1 \end{bmatrix} \begin{bmatrix} X1 \\ X3 \end{bmatrix} \quad \begin{bmatrix} Y2 \\ T3 \end{bmatrix} = \begin{bmatrix} A2 & C2 \\ B2 & S2 \end{bmatrix} \begin{bmatrix} X2 \\ X3 \end{bmatrix}$$

Mise à jour aux interfaces (petit)  
 $Y3 = T3 + T3$

Introduction au parallélisme

53

---

---

---

---

---

---

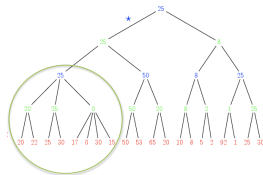
---

---

## Branch-And-Bound

Exemple: arbre max-min

- Parallélisme naïf
  - Développement de l'arbre
  - Heuristique d'évaluations
  - Remontée



Introduction au parallélisme

54

---

---

---

---

---

---

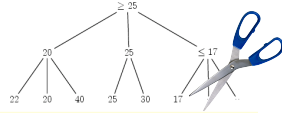
---

---

## Branch-And-Bound

Exemple: arbre max-min

- Parallélisme naïf
  - Développement de l'arbre
  - Heuristique d'évaluations
  - Remontée
- Coupures alpha-beta
- Problème d'équilibrage de charge



Introduction au parallélisme

55

---

---

---

---

---

---

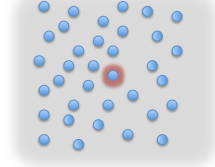
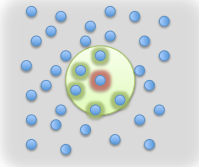
---

---

## Applications irrégulières dynamiques

Le voisinage évolue dans le temps

- Voisinage local –  $N$  interactions
- Dynamique moléculaire
- Voisinage global –  $N^2$  interactions
- $N$ -corps



Introduction au parallélisme

56

---

---

---

---

---

---

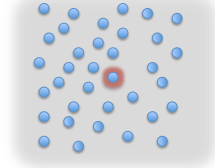
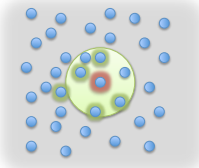
---

---

## Applications irrégulières dynamiques

Le voisinage évolue dans le temps

- Voisinage local –  $N$  interactions
- Dynamique moléculaire
- Voisinage global –  $N^2$  interactions
- $N$ -corps



Introduction au parallélisme

57

---

---

---

---

---

---

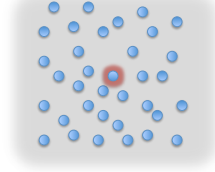
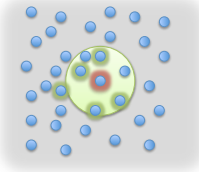
---

---

## Applications irrégulières dynamiques

### Le voisinage évolue dans le temps

- Voisinage local –  $N$  interactions
- Dynamique moléculaire
- Voisinage global –  $N^2$  interactions
- $N$ -corps



Introduction au parallélisme

58

---

---

---

---

---

---

---

---

## Plan

1. Le modèle graphe de tâches
  1. Définition
  2. Exemples élémentaires
2. Applications et algorithmique
  1. Primitives et fonctions génériques
  2. Applications simples
  3. Applications difficiles
3. Scheduling
  1. Statique
  2. Dynamique

Introduction au parallélisme

59

---

---

---

---

---

---

---

---

## Scheduling = placement-ordonnement

Avec  $P$  processeurs, réaliser un ordre partiel compatible avec celui défini par le DAG.

« Replier » le parallélisme illimité sur  $P$  ressources

- Objectif : Minimisation du temps total d'exécution (*makespan*)  
 $T_p \rightarrow$  équilibrage de charge
- Contrainte : minimisation des surcoûts, au premier ordre localité spatiale et temporelle voir chapitres programmation
- Qui ? Placement des calculs
- Quand ? Ordonnement
  - Intra-code: synchronisation (OpenMP, HPF, Cilk - PAR(SEQ)), communication (MPI, PVM - PAR(SEQ)), structures de contrôles (HPF - SEQ(PAR))
  - Extérieur: Exécutif (Condor, Cilk)

Introduction au parallélisme

60

---

---

---

---

---

---

---

---

### Scheduling statique

- Off-line, éventuellement paramétré par le nombre de processus et le numéro de processus
- Parallélisme de données / itératif: les temps de calcul sont supposés identiques
- Cas régulier : Block, cyclic, cyclic (k), ...
- Cas irrégulier : ad hoc – par exemple bisection récursive

Introduction au parallélisme

61

---

---

---

---

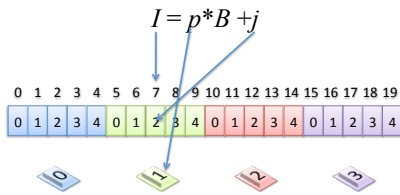
---

---

---

---

### Distribution bloc



$I$  indice global.  $B$  taille du bloc.  $B = N/P$   
 $p$  numéro de processeur.  $p = I/B$   
 $j$  indice local.  $j = I \bmod P$

Introduction au parallélisme

62

---

---

---

---

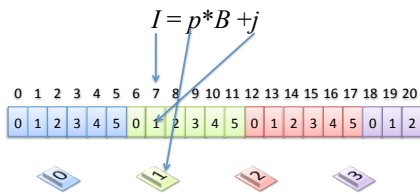
---

---

---

---

### Distribution bloc



$I$  indice global.  $B$  taille du bloc.  $P = \lceil N/B \rceil$   
 $p$  numéro de processeur.  $p = I/B$   
 $j$  indice local.  $j = I \bmod P$

Introduction au parallélisme

63

---

---

---

---

---

---

---

---

### Distribution cyclique

---

$$I = j * P + p$$

$I$  indice global  
 $p$  numéro de processeur.  $p = I \bmod P$   
 $j$  indice local.  $j = I/P$

Introduction au parallélisme 64

---

---

---

---

---

---

---

---

### Distribution cyclique

---

$$I = j * P + p$$

$I$  indice global  
 $p$  numéro de processeur.  $p = I \bmod P$   
 $j$  indice local.  $j = I/P$

Introduction au parallélisme 65

---

---

---

---

---

---

---

---

### Distribution bloc-cyclique – cyclic(B)

---

$$I = b * B + j = B * (c * P + p) + j$$

$I$  indice global  
 $p$  numéro de processeur.  $p = I \bmod P$   
 $j$  indice local.  $j = I/P$

Introduction au parallélisme 66

---

---

---

---

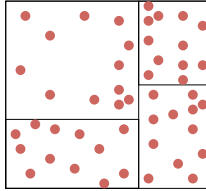
---

---

---

---

## Bisection récursive



Les éléments de calcul sont caractérisés par une donnée nD

Souvent position spatiale

Introduction au parallélisme

67

---

---

---

---

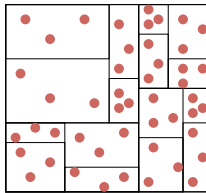
---

---

---

---

## Bisection récursive



Les éléments de calcul sont caractérisés par une donnée nD

Souvent position spatiale

Introduction au parallélisme

68

---

---

---

---

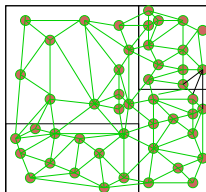
---

---

---

---

## Bisection récursive



Applicable aussi à un maillage  
Et plus généralement à toute structure de données

- irrégulière
- creuse
- organisée spatialement (localité)

Introduction au parallélisme

69

---

---

---

---

---

---

---

---

## Limites du scheduling statique

### Parallélisme de contrôle

Pour un problème modérément général et réaliste

$P$  processeurs identiques

$M$  tâches indépendantes de durée  $t_i$

La minimisation du makespan est un problème NP-complet

Scheduling dynamique

Introduction au parallélisme

70

---

---

---

---

---

---

---

---

## Scheduling dynamique

- Motivations supplémentaires et plus réalistes :
  - La durée des calculs n'est pas prévisible
  - La puissance des machines n'est pas connue
- Scheduling on-line : décidé à l'exécution
- Questions
  - Surcoûts : gestion des processus ou des threads, des échanges d'information
  - Robustesse à l'irrégularité des temps d'exécution

Introduction au parallélisme

71

---

---

---

---

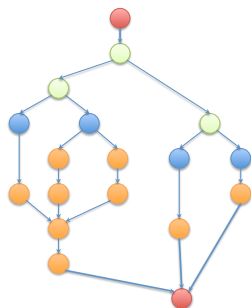
---

---

---

---

## Scheduling glouton (greedy)



- Parallélisme de contrôle
- Exécuter toute tâche prête dès que possible
- (Relativement) facile à implémenter
- Distance à l'optimal ?

Introduction au parallélisme

72

---

---

---

---

---

---

---

---



### Scheduling glouton (greedy)

#### P processeurs

- Exécuter tout ce qui peut l'être dès que possible
- Phase pleine: le nombre de processus actifs est  $\geq P$ 
  - Choix des processus activés = algorithmique du scheduling
- Phase incomplète: le nombre de processus actifs est  $< P$

Introduction au parallélisme

73

---

---

---

---

---

---

---

---

### Scheduling glouton (greedy)

**Théorème [Graham '68].**

**Pour tout ordonnancement glouton**

$$T_p \leq T_1/P + T_\infty$$

*Preuve*

- Le nombre de phases pleines est au plus  $T_1/P$  puisque chaque pas de chaque phase pleine effectue un travail  $P$ ,
- Le nombre de phases incomplètes est au plus  $T_\infty$  puisque chaque pas de chaque phase incomplète effectue un travail 1 le long du chemin critique

Introduction au parallélisme

74

---

---

---

---

---

---

---

---

### Scheduling glouton (greedy)

**Théorème [Graham '68].**

**Pour tout ordonnancement glouton**

$$T_p \leq T_1/P + T_\infty$$

**Corollaire 1 Tout ordonnancement glouton est au plus à un facteur 2 de l'optimal.**

*Preuve*

Si  $T_p^*$  est l'optimal (à P processeurs)

$$T_p^* \geq T_1/P \text{ et } T_p^* \geq T_\infty$$

D'où

$$T_p \leq 2T_p^*$$

Introduction au parallélisme

75

---

---

---

---

---

---

---

---

## Scheduling glouton (greedy)

*Théorème [Graham '68].*

Pour tout ordonnancement glouton

$$T_p \leq T_1/P + T_\infty$$

**Corollaire 2** Si  $P \ll T_1/T_\infty$  l'accélération est quasi-linéaire

*Preuve*

$$T_1/P \gg T_p \text{ donc } T_1/P + T_\infty \approx T_1/P$$

$T_1/PT_\infty$ : parallel slackness

Introduction au parallélisme

76

---

---

---

---

---

---

---

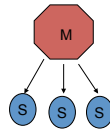
---

## Implémentation du scheduling dynamique

Centralisé: Maître-Esclave

*Maître*

- Gère la distribution des tâches : glouton, GSS, ...
- Effectue les opérations globales par exemple réduction



*P Esclaves*

- Exécutent un bloc, et redemandent du travail dès que terminé

Introduction au parallélisme

77

---

---

---

---

---

---

---

---

## Maître-Esclave à grain fin

- Pure self-scheduling : coût de synchronisation excessif
- Idée : Au début, on alloue des blocs de grande taille pour diminuer les coûts de synchronisation, puis des blocs de taille décroissante pour ajuster progressivement l'équilibrage de charge
  - Guided self-scheduling (GSS): chaque esclave reçoit  $1/P$  du batch restant
  - Factoring (FSS): durant chaque phase, chaque esclave reçoit  $1/P$  de la moitié du batch restant

Introduction au parallélisme

78

---

---

---

---

---

---

---

---

### Maître-Esclave à grain fin

$N = 256, P = 4$

Statique : 64, 64, 64, 64

PSS : 1,1,1,1,1,1,1...

GSS : 64,48,36,27,20,...

FSS : 32,32,32,32,8, 8, 8, 8, 4, 4, 4, 2,...

Introduction au parallélisme

79

---

---

---

---

---

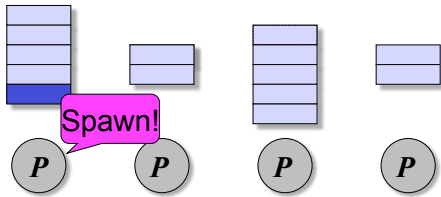
---

---

---

### Cilk's Work-Stealing Scheduler

Each processor maintains a *work deque* of ready threads, and it manipulates the bottom of the deque like a stack.



© 2006 Charles E. Leiserson Multithreaded Programming in Cilk — LECTURE 1 July 13, 2006 80

---

---

---

---

---

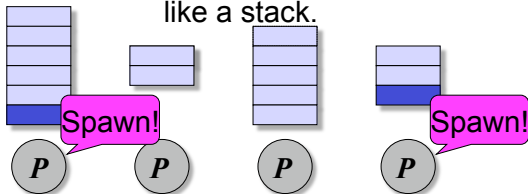
---

---

---

### Cilk's Work-Stealing Scheduler

Each processor maintains a *work deque* of ready threads, and it manipulates the bottom of the deque like a stack.



© 2006 Charles E. Leiserson Multithreaded Programming in Cilk — LECTURE 1 July 13, 2006 81

---

---

---

---

---

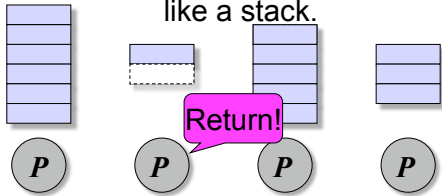
---

---

---

### Cilk's Work-Stealing Scheduler

Each processor maintains a **work deque** of ready threads, and it manipulates the bottom of the deque like a stack.



---

---

---

---

---

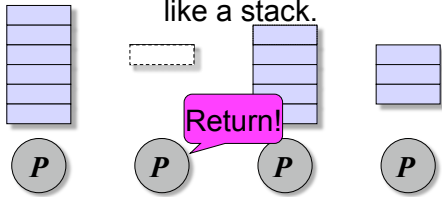
---

---

---

### Cilk's Work-Stealing Scheduler

Each processor maintains a **work deque** of ready threads, and it manipulates the bottom of the deque like a stack.



---

---

---

---

---

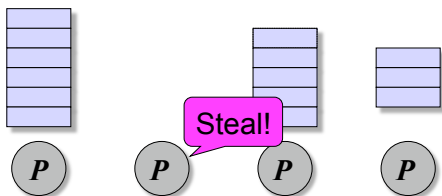
---

---

---

### Cilk's Work-Stealing Scheduler

Each processor maintains a **work deque** of ready threads, and it manipulates the bottom of the deque like a stack.



When a processor runs out of work, it **steals** a thread from the top of a **random** victim's deque.



---

---

---

---

---

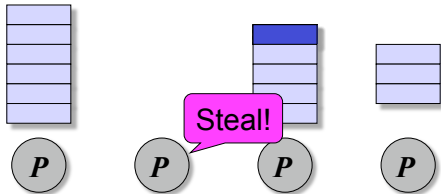
---

---

---

### Cilk's Work-Stealing Scheduler

Each processor maintains a *work deque* of ready threads, and it manipulates the bottom of the deque like a stack.



When a processor runs out of work, it *steals* a thread from the top of a *random* victim's deque.

---

---

---

---

---

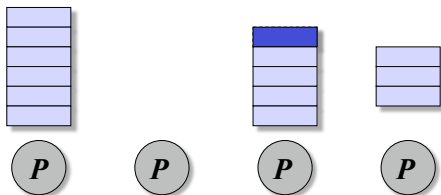
---

---

---

### Cilk's Work-Stealing Scheduler

Each processor maintains a *work deque* of ready threads, and it manipulates the bottom of the deque like a stack.



When a processor runs out of work, it *steals* a thread from the top of a *random* victim's deque.

---

---

---

---

---

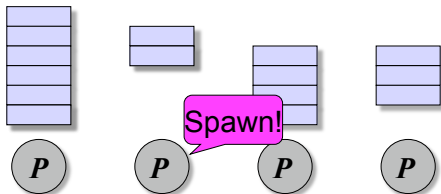
---

---

---

### Cilk's Work-Stealing Scheduler

Each processor maintains a *work deque* of ready threads, and it manipulates the bottom of the deque like a stack.



When a processor runs out of work, it *steals* a thread from the top of a *random* victim's deque.

---

---

---

---

---

---

---

---

## Placement/ordonnement dynamique

---

- Réparti : Vol de travail
  - Peut être prouvé optimal pour une large classe d'applications
  - Implémentation délicate : gestion de verrous en espace d'adressage unique, protocole en espaces d'adressages multiples
  - <http://supertech.lcs.mit.edu/cilk/>

---

---

---

---

---

---

---

---