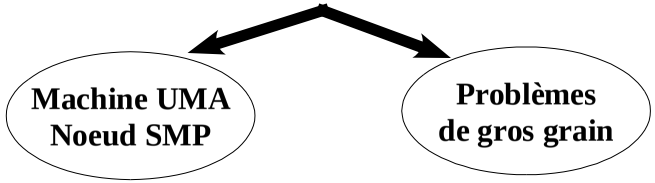
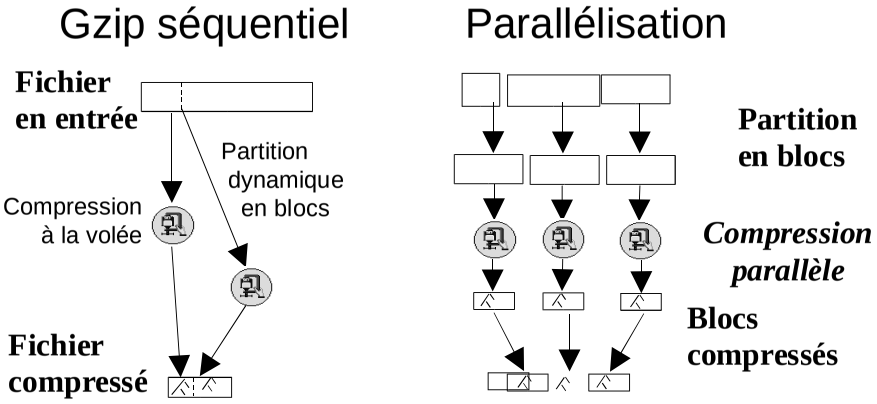
Cours 2. Algorithmes Parallèles sans communication



www−id.imag.fr/Laboratoire/Membres/Roch_Jean−Louis/perso.html/enseignement.html/

Comment paralléliser gzip ?



Comment choisir les blocs ?

Objectif cours 2: programme parallèle tel que

$T_p \approx (T_{seq}/ p) + \varepsilon$

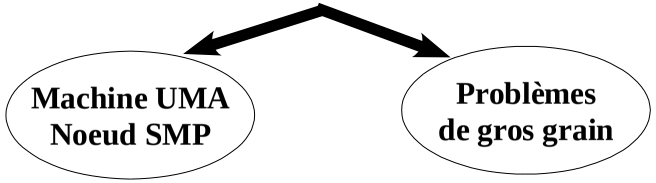
Base : algo parallèle + ordt “glouton” [cours 1]

$T_p < (T_1/ p) + T_\infty$

... mais $T_1 >> T_{seq}$ ☹

Techniques pour limiter le surcoût dû au parallélisme

urs 2. Algorithmes Parallèles sans communication



I. Introduction
II. ***Contrôle de la granularité***
III. Mise en oeuvre de l’ordonnancement
IV. Application: recherche arborescente
Contrôle de l’espace mémoire

Choix du seuil K

ompromis séquentiel/parallèle

xpérimentalement

héoriquement :

K le plus grand possible sans perte de parallélisme

- **K qui minimise $T_1^{(K)}$ avec $T_\infty^{(K)}=\Theta(T_\infty)$**

Exemple : produit itéré

1. Contrôle de la granularité

- Limiter le surcoût dû au parallélisme :
  - T_1 : temps de l'algorithme parallèle
  - T_{seq} : temps du « meilleur » algo séquentiel
  - **Objectif : $T_1 = T_{seq}$**
- Mais en gardant **T_∞ aussi petit que possible**

C. Leiserson : « A minicourse on multithreaded algorithms »
supertech.lcs.mit.edu/cilk/papers ftp://theory.lcs.mit.edu/pub/cilk/minicourse.ps.gz

J.L. Roch : « Parallel efficient algorithms and their programming »
www−id.imag.fr/~jlroch/perso.html/ps/polycop−algo−par.ps.gz p.4−22

Exemple 2 : Préfixe parallèle

trée : X[0..n−1] un tableau d'éléments
: loi associative

rtie : Π[0..n−1] avec Π[k] = X[0] * ... *X[k]

émarche :

1/ Algorithme séquentiel de référence : SeqPrefixe
2/ Parallélisation : algorithme ParPref1
3/ Adaptation de granularité : algorithme ParPrefK
4/ Choix de K : compromis théorie − pratique

Adapter la granularité

- Utiliser un algorithme séquentiel pour limiter le parallélisme
- Découpe récursive parallèle
- => Stopper la découpe récursive à un seuil K
- Exemple : produit itéré en Cilk

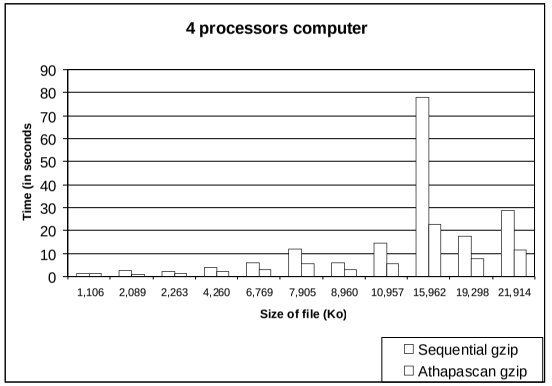
```
Cilk void ParProduit (int i, int j, int& res ) {
        if ( j−i < K ) { res = SeqProduit(i,j) ; }
        else {
            int r1, r2 ;
            spawn ParProduit ( i, (i+j)/2, r1 ) ;
            spawn ParProduit (  (i+j)/2+1; j , r2 ) ;
            sync ;
            res = r1 * r2 ;
        }
}
```

```
int  SeqProduit (int i, int j) {
        int r = 1;
        for (int s = i ; s <j; s++ ) r = r*s ;
        return r ;
 }
```

Exemple 3 : gzip

- TailleBloc := ... ;
  for(i=0; i<n/TailleBloc ; i++)
  Fork<gzip>(Fich[i*TailleBloc ... min (n, (i+1)*TailleBloc −1] ;

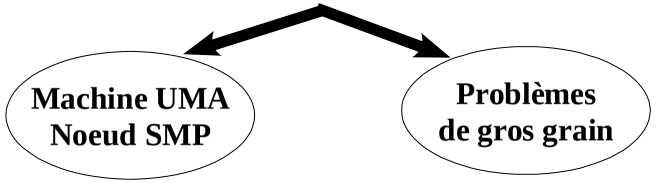
- Choix de K :
  - Expérimental
    - TailleBloc ~ 0.5 Mo
  - Théorique :
    - TailleBloc ~
  - Théorique :
    - TailleBloc ~



Exemple 4 : exercice
Tri par fusion

C. Leiserson : « A minicourse on multithreaded algorithms »
ftp://theory.lcs.mit.edu/pub/cilk/minicourse.ps.gz p 9−12

urs 2. Algorithmes Parallèles sans communication



Introduction

I. Contrôle de la granularité
   => ***Généralisation : algorithmes en cacade***

II. Mise en oeuvre de l'ordonnancement

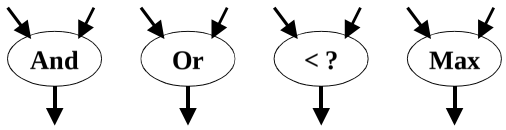
V. Application: recherche arborescente
   Contrôle de l'espace mémoire

Jeu : Calculer le maximum

: construire le circuit le plus rapide possible
r calculer le maximum :

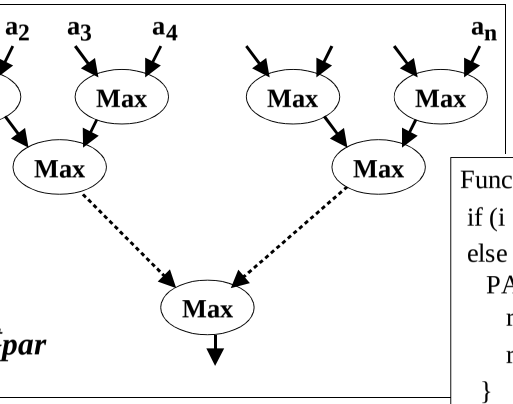
− Entrée : n éléments a_i distincts (ordre total <)
− Sortie : l'élément maximum

tes disponibles



té non−bornée :
less comm: multiple access (SDMA/FDMA/CDMA)
CW : Concurrent Read Concurrent Write

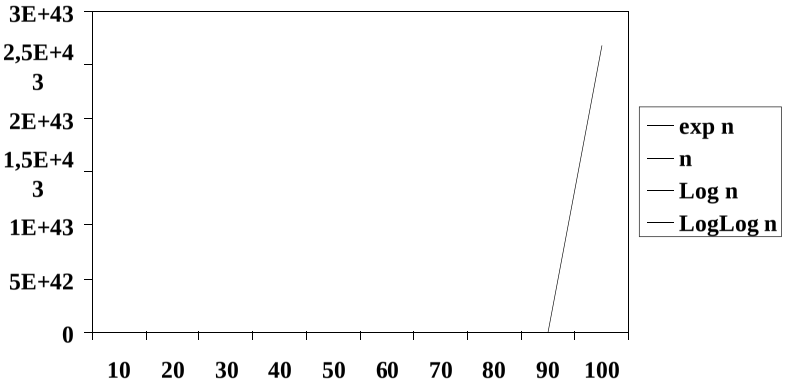
Circuit parallèle




```
Function max2(a_i .. a_k ) {
 if (i == k) return a_i ;
 else {
   PARALLEL {
     rl = max2(a_i .. a_(i+k)/2 ) ;
     rh = max2(a_(i+k)/2+1 .. a_k);
   }
 return Max( rl, rh ) ;
}
```

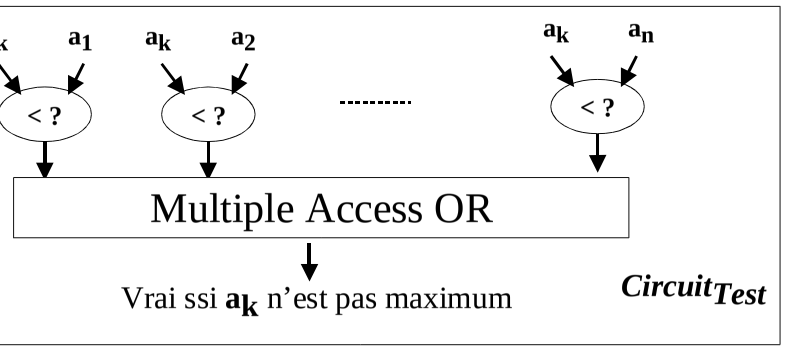
$\log_2 n$ #procs = n portes

Coûts et algorithmes ultra−rapides



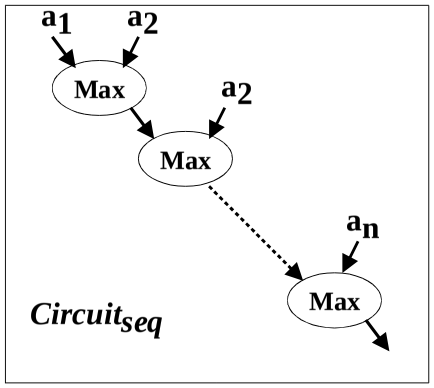
Un circuit ultra−rapide r tester si un élément a_k est le max

$= \text{Max}(a_1 .. a_n) \Leftrightarrow a_k > a_i \quad \forall i=1..n, i /= k$



$T_1(n) = O(1)$ ☺ #portes = O(n) ☹

Circuit séquentiel de base

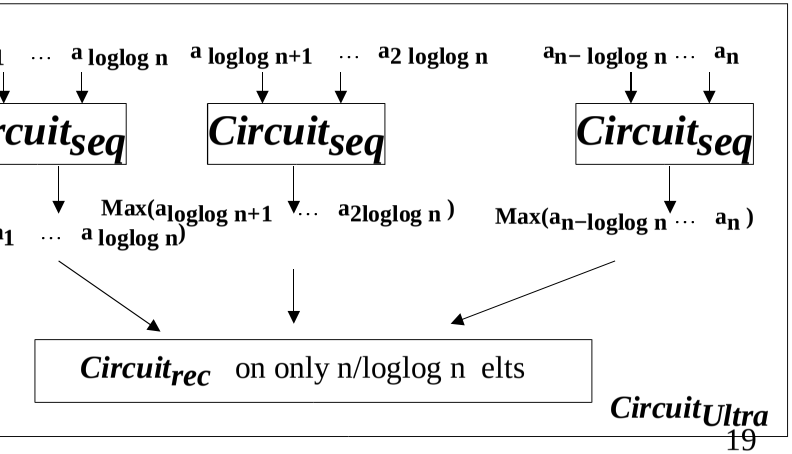



```
res := a_1 ;
For i :=2 .. n do
   res := Max ( res, a_n )
;
Return res;
```

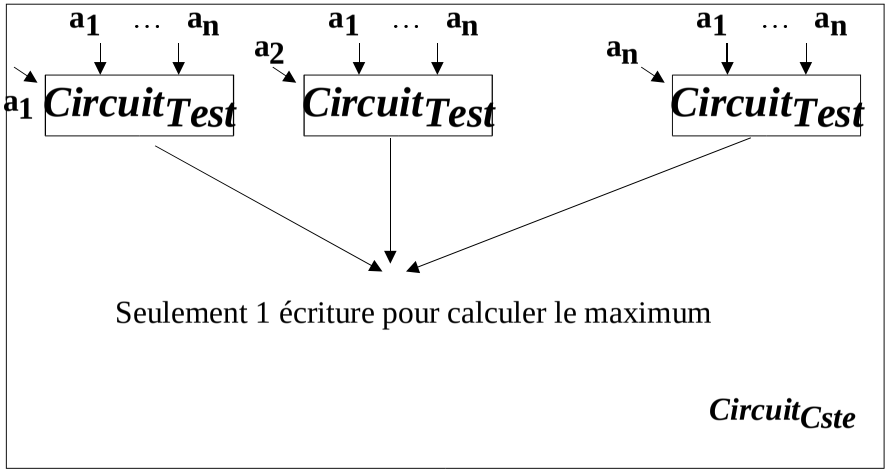
$T_1 = n$ #procs = n portes

Réduction du nombre de portes

Granularité: utiliser un algo économique
pour réduire le nombre de portes



Application: calcul du maximum



$T_1(n) = O(1)$ ☺ **#portes = $O(n^2)$** ☹

onclusion : un algo ultra−rapide

orithme final : **temps = loglog n #portes=n**

nique utilisée : « cascading »

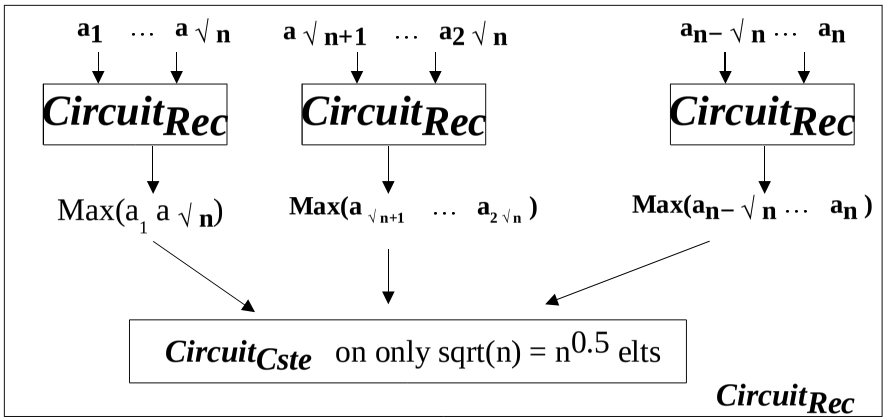
élange de 3 algorithmes pour construire un
mpromis plus performant :
temps et nombre de portes

nique importante en parallélisme (adaptation
granularité)... et en génie logiciel

-> nombreuses applications : [ATLAS, FFTW, ...]

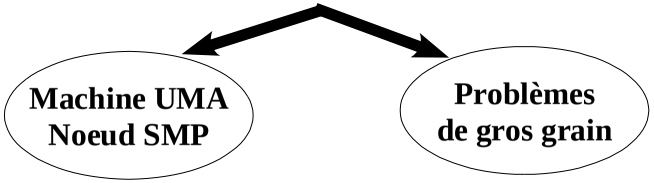
Un circuit recursif ultra−rapide

- Granularité : utiliser l'algo rapide pour accélérer



- Time(n) = Time($n^{0.5}$) + O(1) = loglog n
- #procs (n) = $n^{0.5}$.#procs($n^{0.5}$) = n loglog n

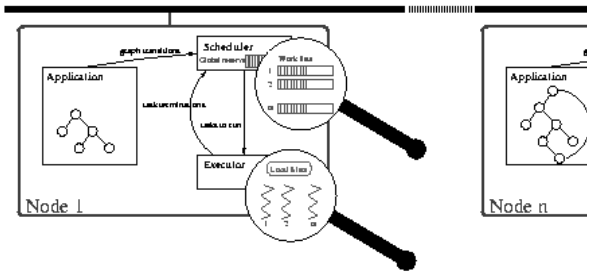
Cours 2. Algorithmes Parallèles
sans communication



I. Introduction
II. Contrôle de la granularité
III. ***Mise en oeuvre de l'ordonnancement***
IV. Application: recherche arborescente
Contrôle de l'espace mémoire

Principe

- Lancement de p threads
  - 1 thread == processeur virtuel
- Chaque thread :
  - While (not fin)
    - Attendre une tache t prete
    - L'exécuter
- Liste centralisée + verrou :
  - $T_p < (T_1/p) + T_\infty + O(\#sync)$



oût pour réaliser l'ordonnancement

rôle :

éation/placement/entrelacement des tâches

estion de la mémoire, mouvements de données

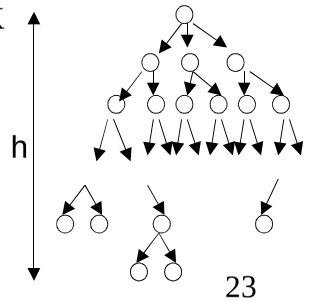
éemptivité, réactivité

nple : algorithme récursif d'exploration

donnancement glouton : théorique = OK

ais réalisation ???

- Nombre de tâches ~ T_1 …
- Mémoire ~ 2^h … en séquentiel ~ h



Work−stealing
Distribuer les synchronisations

ste de taches prêtes par processeur :

Lorsqu'un thread crée une tâche, il l'ajoute à sa liste locale.

rsqu'un processeur est inactif :

Si il n'y a plus de tache prête localement

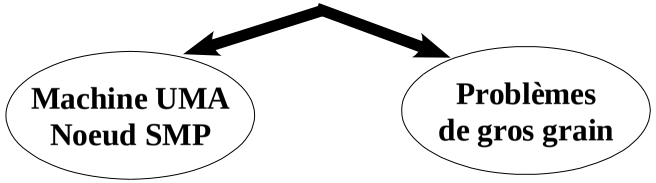
- Choix d'un processeur victime
- Lui voler une tâche prête

oix : exemple [Cilk]

Processeur victime : au hasard

Accès à la liste des tâches prêtes : verrou arithmétique

urs 2. Algorithmes Parallèles sans communication



. Introduction
I. Contrôle de la granularité
II. Mise en oeuvre de l'ordonnancement
V. ***Application: recherche arborescente***
Contrôle de l'espace mémoire

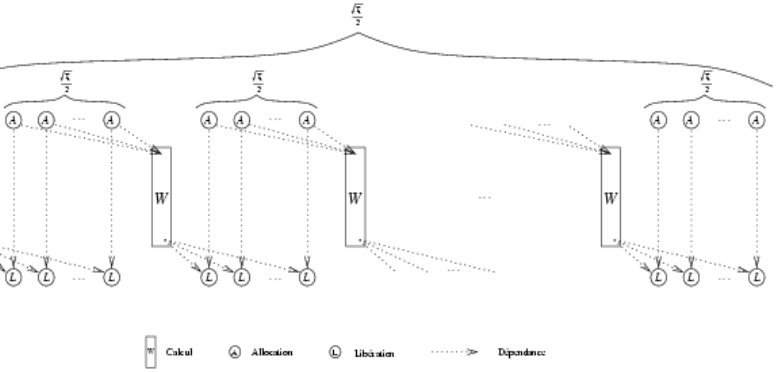
Work−stealing
Minimiser le surcoût d'ordt

- => Minimiser le nombre de vols
- Programmes récursifs = graphe « série−parallèle »
  - Principe : exécution localement d'abord
  - => sur chaque proc. : les vols sont sur un chemin critique

$$\#vols < p.T_\infty$$

- Optimisation : création locale dégénérée en appel de fonction
  - Exemple : produit itéré

rallélisme et explosion mémoire



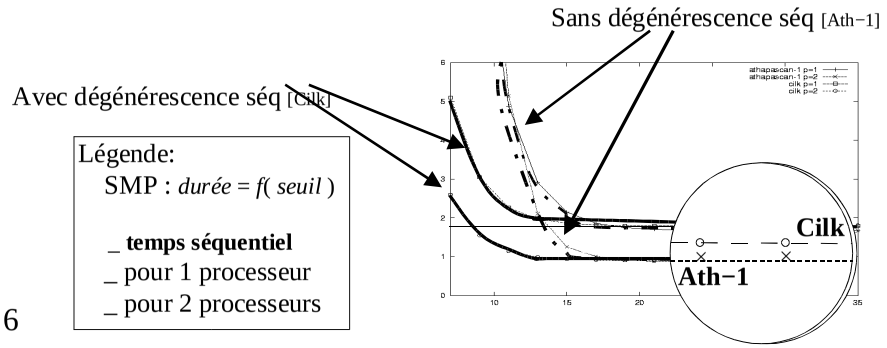
ofe C. Leiserson : « Space−efficient scheduling of multithreaded
omputations » SIAM Journal of computing, vol 27, Fév. 1998 p. 207−229
kar : « Space−efficient scheduling for parallel
ultithreaded computations » thèse − www−2.cs.cmu.edu/~girija/publications.html
e : « Athapascan−1 : Interprétation distribuée du flot de donnée d'un
ogramme parallèle » Thèse de doctorat INPG P 113−127 :
ww−id.imag.fr/~jlroch/perso.html/ps/these−galilee_pages113−151.ps.gz

Ordonnancement SMP −Conclusion

- Théorique : si programme série−parallèle [Blumofe98]

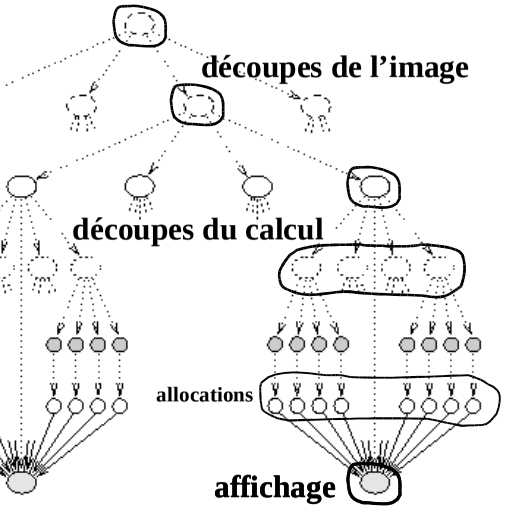
$$T_p < (T_1/p) + T_\infty + \boldsymbol{p.T_\infty}$$

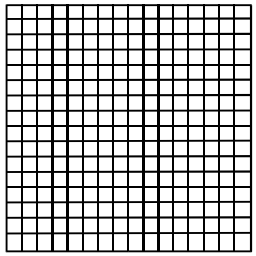
- Pratique : contrôle automatique de granularité
  - Exemple : arbre ~10^6 tâches de grain 1



Et la consommation mémoire ?

Calcul récursif : Découpe en 4





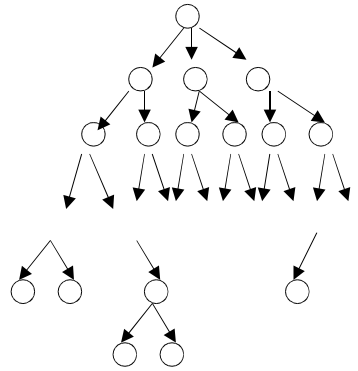
= log*n* + un morceau d'image = 78 ko

Un exemple

```
oid explore( nœud  n, …) {
if … {
  for (s =n.first();  s <n.last(); n++)
  {
     dopar explore( s, …) ;
}
```

tâche

h



T_1 : temps séquentiel

T_∞ = O(h)

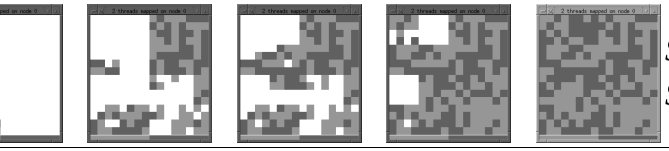
S_1 = O(h)

$T_{exec(p)} \sim T_1 / p$

$S_{exec(p)} \sim S_1$ *O(1)* **???**

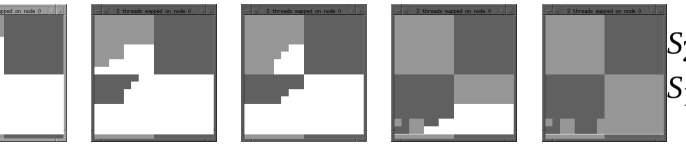
Ordre d'exécution des tâches $p = 2$

□ *p0*
□ *p1*



S_2 = 548ko
S_1 = 78ko ☹

S_1 = log*n* + un morceau d'image = 78 ko



S_2 = 143ko
S_1 = 78ko

Programme série−parallèle

- Work−stealing:
  - Si inactif : tirage au hasard d'un proc victime Pv
  - La tâche volée est la plus ancienne sur Pv
- Chaque processeur vole des tâches sur un chemin critique : espace mémoire < S1 / proc

$$S_p < p.S_1$$

- Exemple :
  - Cilk : Joueur d'échecs Socrates
  - Athapascan : Visualisateur Mandelbrot

nalyse théorique du coût mémoire

bitraire [Threads]

Ordonnancement **arbitraire** des tâches $S_p \leq \#T\, S_1$

2 : « séquentiel local »

pour graphes série−parallèle [Blumofe98] $S_p \leq pS_1$

Ordre de **référence** suivi **localement**

Parcours en **profondeur**

Athapascan−1 : pour tout graphe [Géalilée 99]

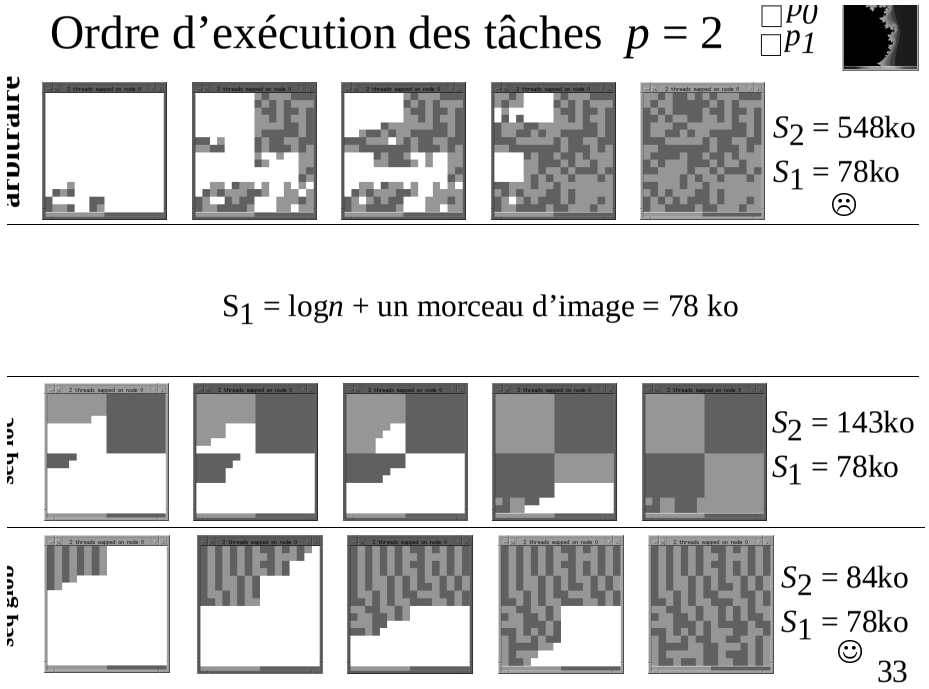
: « séquentiel global »

pour tout graphes [Narlikar2000] $S_p \leq S_1 + p.hOT_\infty$

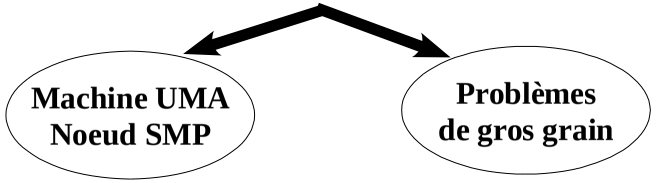
Ordre de **référence** suivit de manière globale

Athapascan−1 : graphes dynamiques

Ordre d'exécution des tâches $p = 2$



urs 2. Algorithmes Parallèles sans communication



Conclusion

- Contrôler la granularité
- Limiter le surcoût dordonnancement
- Maîtriser l'espace mémoire requis

Consommation mémoire en fonction du nombre de processeurs