Ordonnancement et calcul parallèle

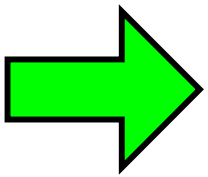
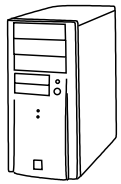
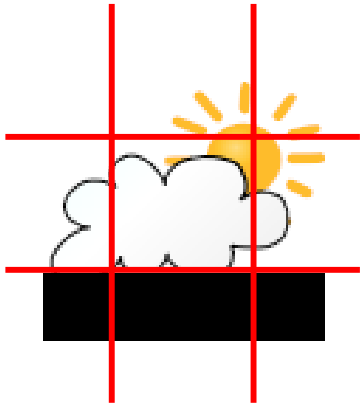
Grégory Mounié,
Laboratoire Informatique et Distribution (ID-IMAG),
Grenoble

23 nov. 2005

Plus vite, plus gros, pas trop cher

Motivation

Les besoins en puissance de calcul ne cessent de croître avec les modèles toujours plus compliqués, utilisés dans divers domaines (météo, rendu graphique, recherche opérationnelle)


Les grappes

Les machines de calculs parallèles sont actuellement un assemblage de matériels standards :

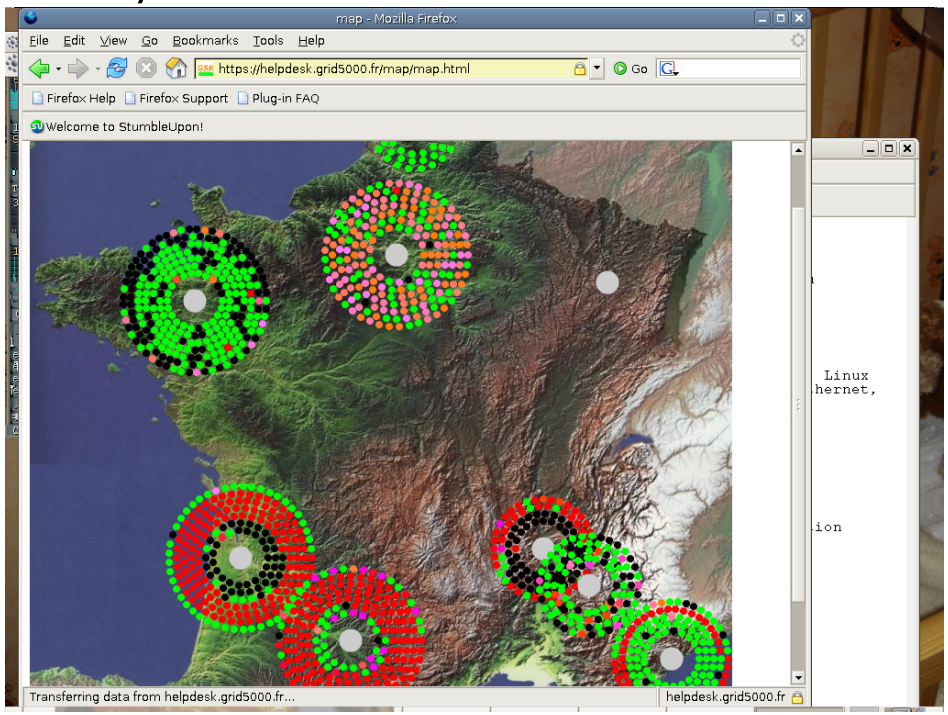
- des nœuds semblables à un PC,
- des réseaux dédiés (latence de quelques microseconde, débit en Gb/s) ou bien standards (débit en Gb/s)

Le top 500 (Novembre 2005)

- 1er : Blue Gene, 131072 processeurs, PowerPC 440 700 MHz (2.8 GFlops), 280 Teraflops soutenus, Linux
- 498ème : grappe IBM de 494 Xeon 2.8Ghz avec un réseau GigaEthernet, Linux (278ème 6 mois avant ! 129ème 1 an et demi avant !)

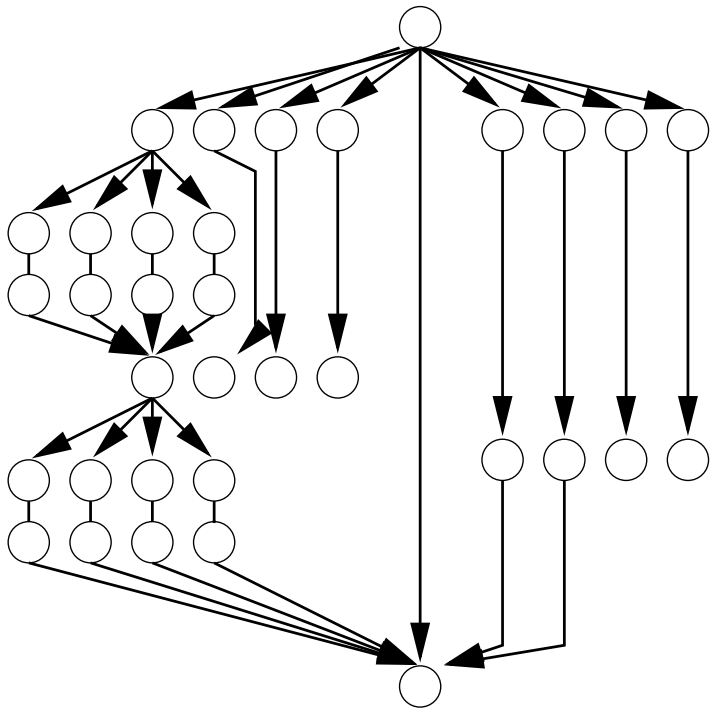
Grilles légère : l'exemple de Grid5000

Un nouveau challenge est l'interconnexion à grande échelle de ces grappes.
Grid5000 est une grappe d'expérimentations réunissant une 10 dizaines de site français.



Le graphe de tâches

Les applications sont représentées sous forme de graphes orientés sans cycle (DAG)



Le problème de l'ordonnancement

Definition

Le problème de l'ordonnancement Il consiste à choisir le lieu et la date d'exécution de chaque nœud du graphe.

Definition

Notations

m : le nombre de processeurs de la machine

p_i : le temps d'exécution de la tâche t_i

q_i : le nombre de processeurs utilisé par la tâche t_i

$W_i = p_i \times q_i$: le travail de la tâche t_i

Problème d'optimisation

Plusieurs critères d'optimisation en fonction de la date de terminaison C_i de la tâche i :

- date de fin de la dernière tâche (makespan) ($maxC_i$)
- le temps moyen de terminaison ($\sum C_i$)
- le stretch (en ligne) ($\max(C_i - r_i)$),

Même les problèmes les plus simples sont durs (longs) à résoudre optimalement si $P \neq NP$. On cherche donc des heuristiques produisant de "bonnes" solutions.

Rapport de compétitivité

Definition

Le *rapport de compétitivité* d'une heuristique est le rapport entre la valeur d'une solution construite par l'heuristique et la meilleure solution.

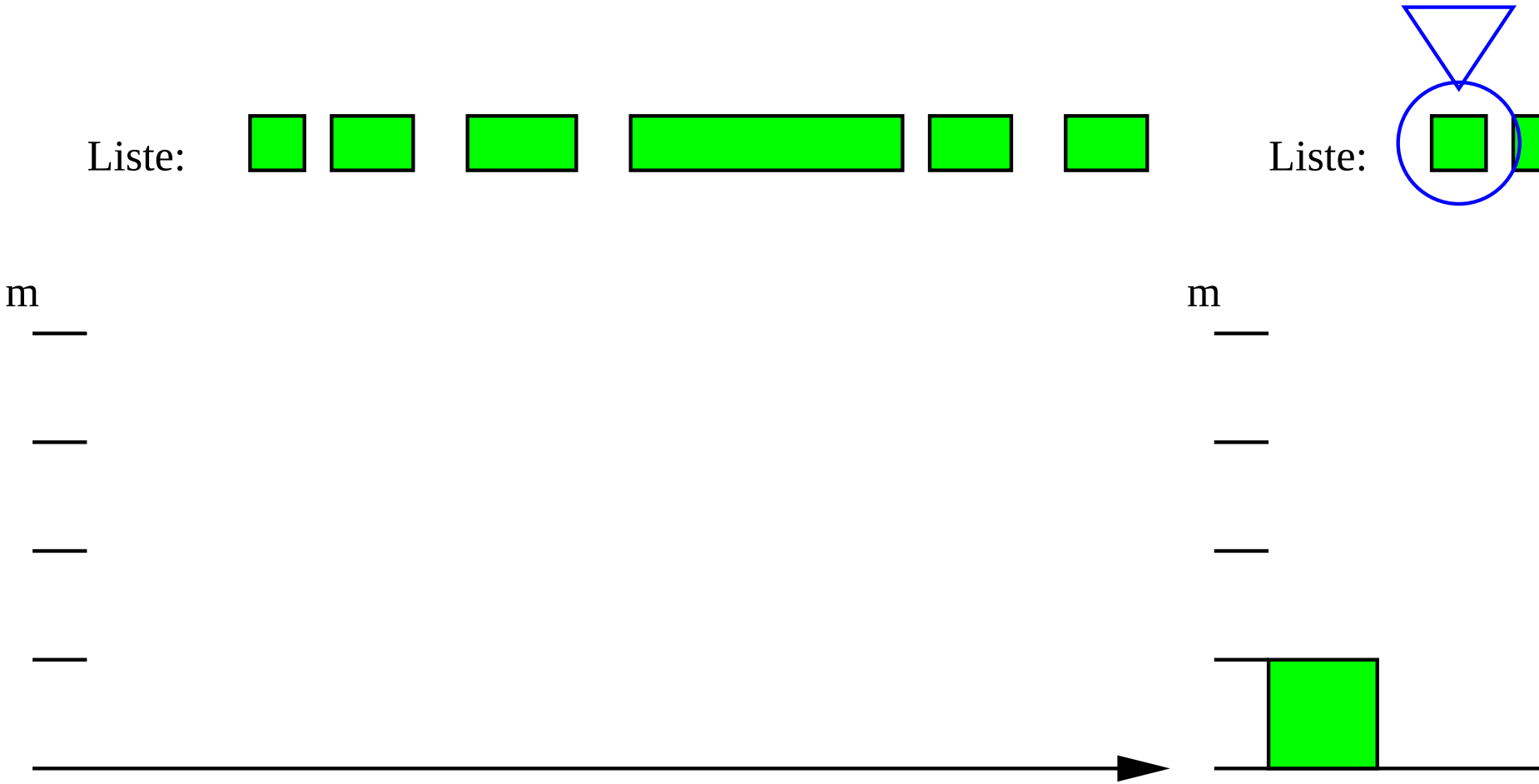
On peut calculer le rapport de compétitivité

- au pire
- en moyenne (souvent fonction de la distribution statistique des instances)
- asymptotique

Ordonnancement de liste

- Une liste ordonnée des tâches (priorité)
- On exécute la tâche prête qui démarre le plus tôt, en respectant la priorité de la liste.

Exemple d'ordonnancement de listes



Ordonnancement de tâches séquentielles indépendantes

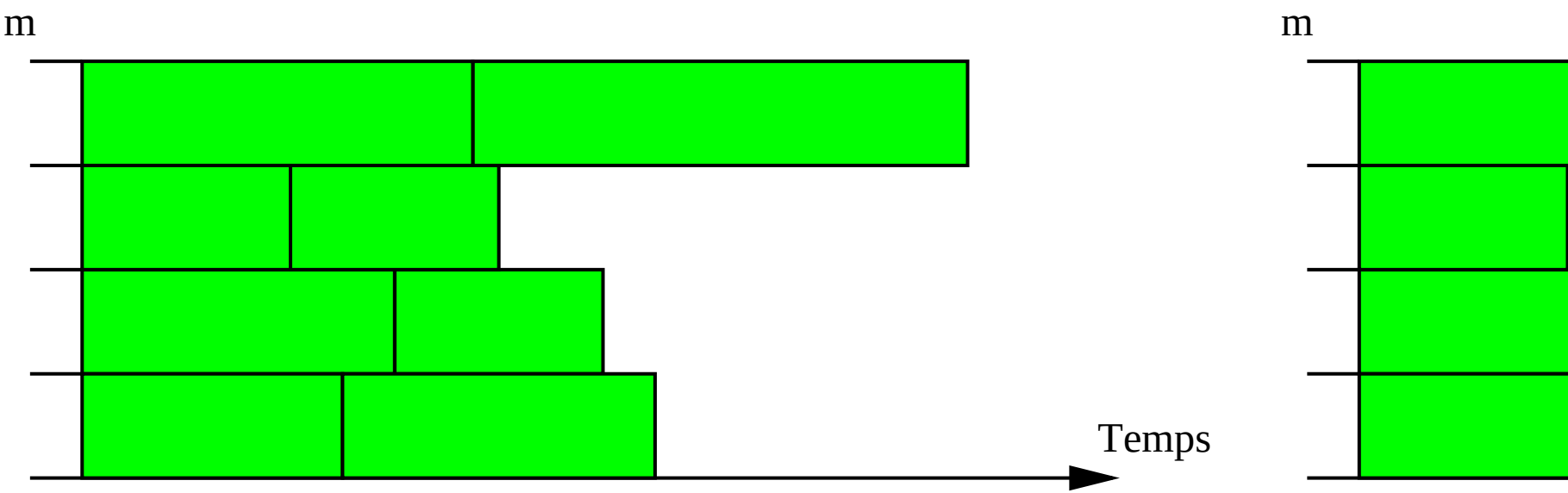
Definition ($P|q = 1|C_{max}$)

Soit T un ensemble de tâches t_1, t_n séquentielles indépendantes, et m le nombre de processeurs de la machine. Le but est de minimiser C_{max}

Theorem

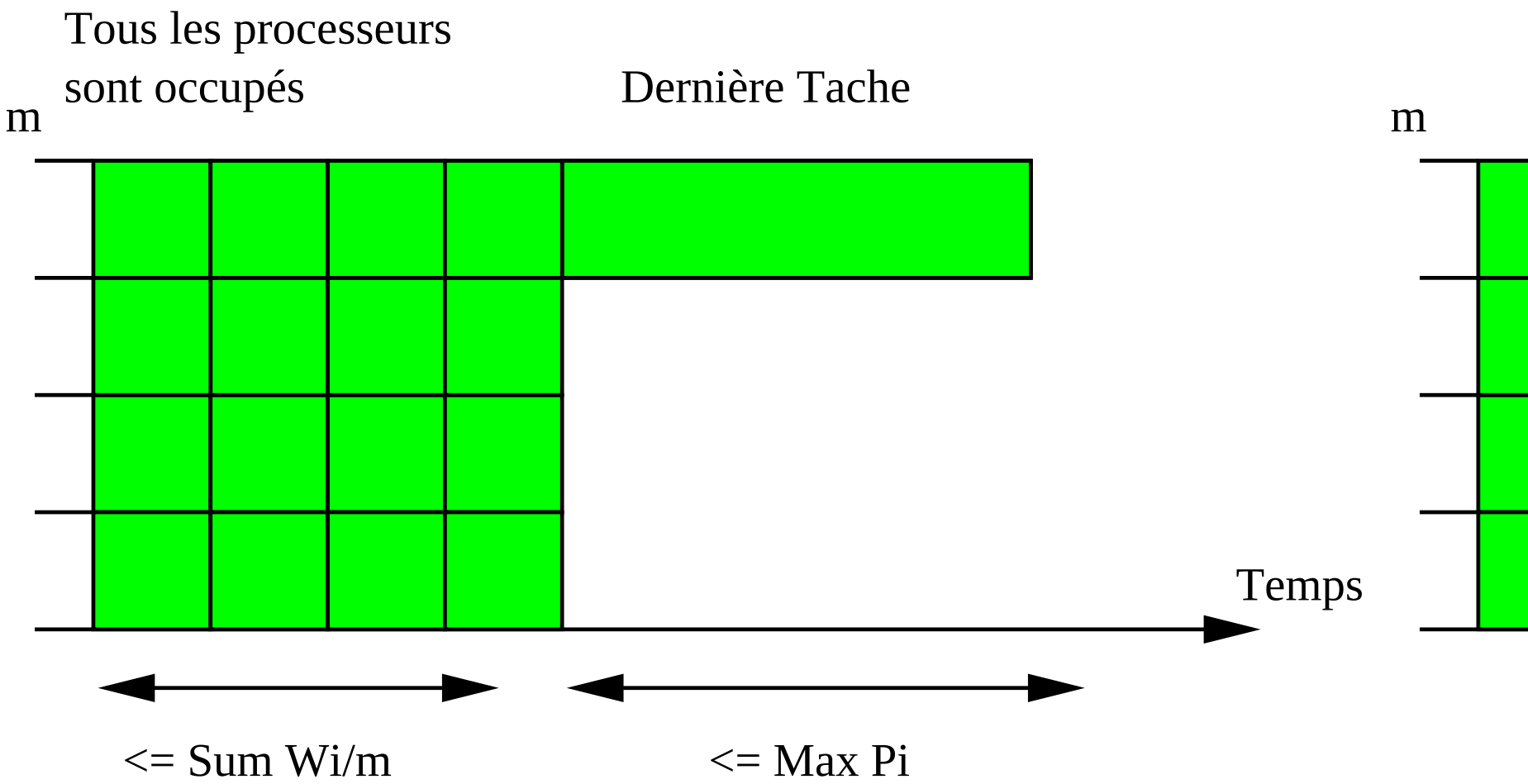
[Graham 1966] Pour le problème des tâches indépendantes, tout algorithme de liste à un rapport de compétitivité inférieure à $2 - \frac{1}{m}$.

Tâches indépendantes [Graham 1966]

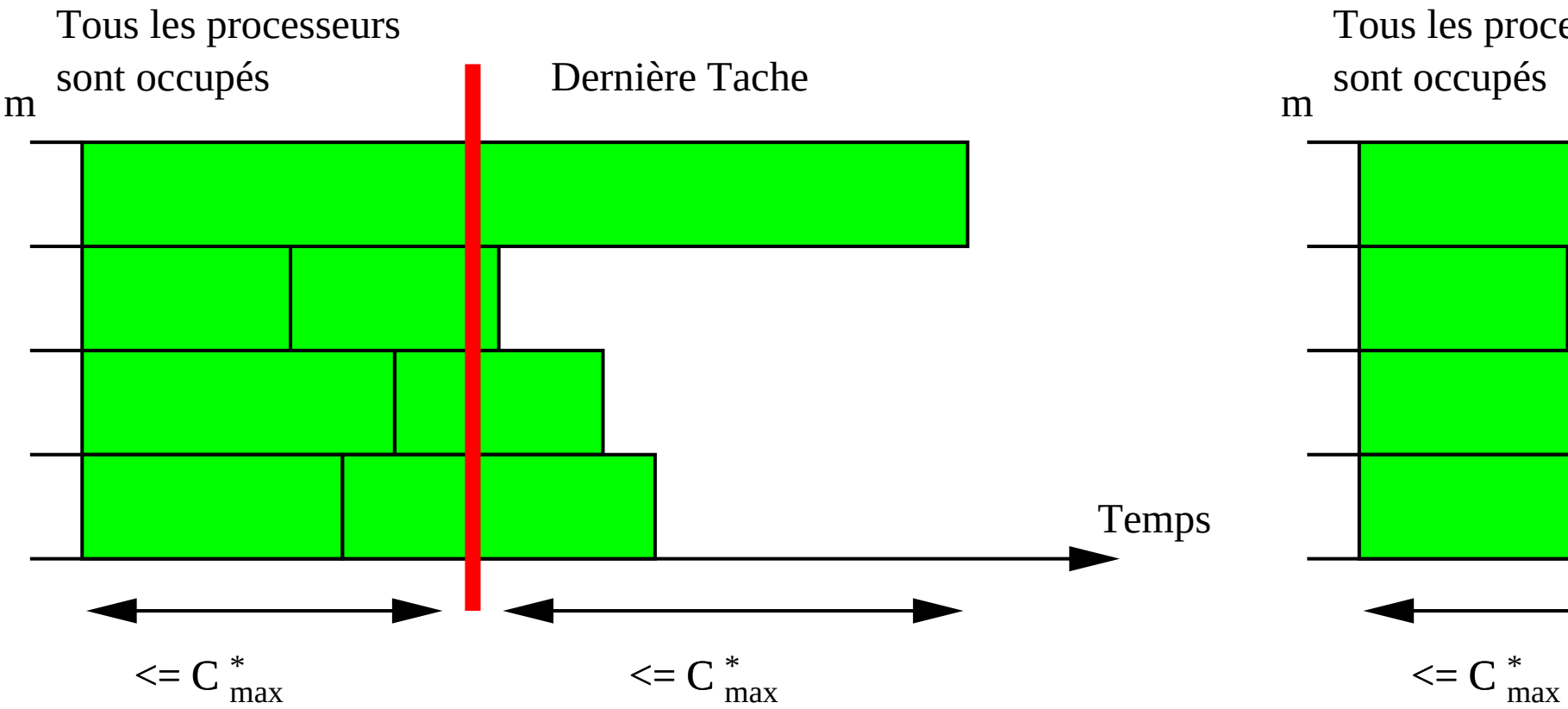


Tâches indépendantes [Graham 1966]

La valeur de $2 - \frac{1}{m}$ est atteinte :



$2 \neq 1 + 1$


Tous les processeurs
sont occupés
m
Dernière Tache
Temps
<= C *max
<= C *max
Tous les proce
sont occupés
m
<= C *max

Somme des travaux et chemin critique

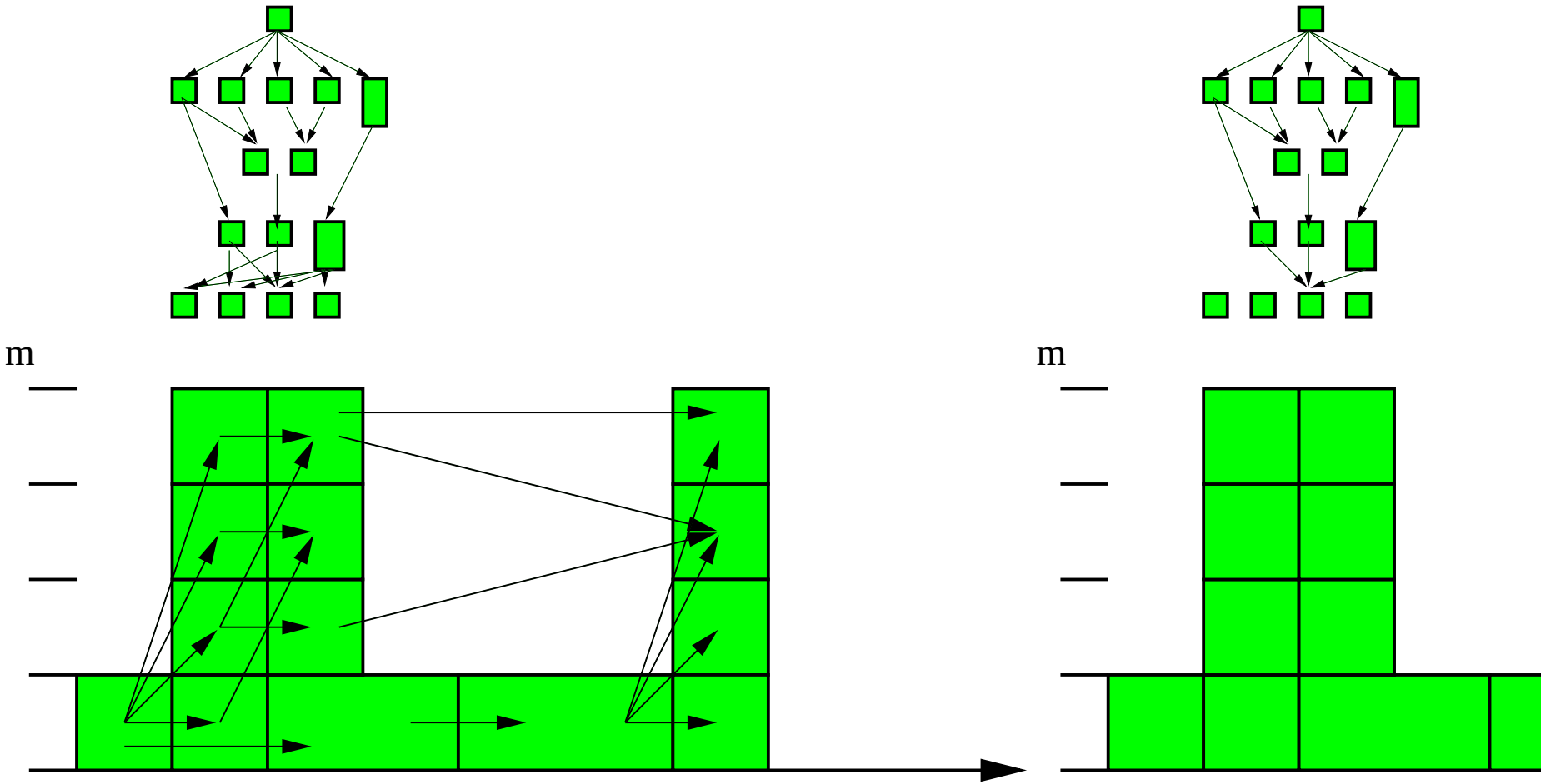
- Si un critère domine l'autre, la parallélisation est "simple" (Seti@Home).
- Les communications jouent aussi un rôle prépondérant, dans la plupart des applications réelles.
- Si le coût unitaire des communications est plus grand que le coût de calcul alors, il n'existe pas (pour l'instant) d'algorithmes avec une garantie de performance fixé : la garantie est dépendante du rapport $\frac{\text{coût de communication}}{\text{coût de calcul}}$

$P|q_i = 1, prec|C_{max}$ [Graham 1969]

Theorem

Algorithme de liste avec précédence Tout algorithme de liste a une garantie de performance meilleure que $2 - \frac{1}{m}$

Ordonnancement avec relation de précédence



$P|q_i|C_{max}$

Cas typique : ordonnancement de jobs parallèle dans une grappe.

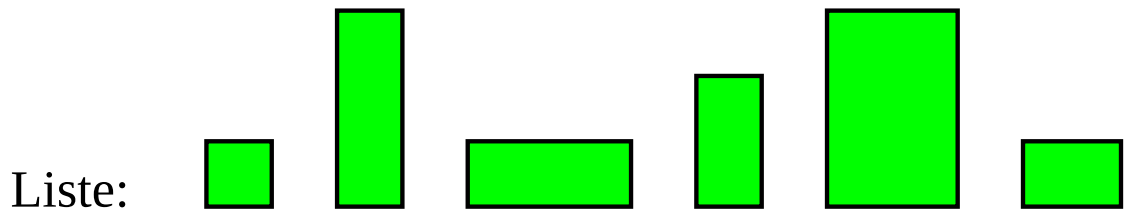
Theorem

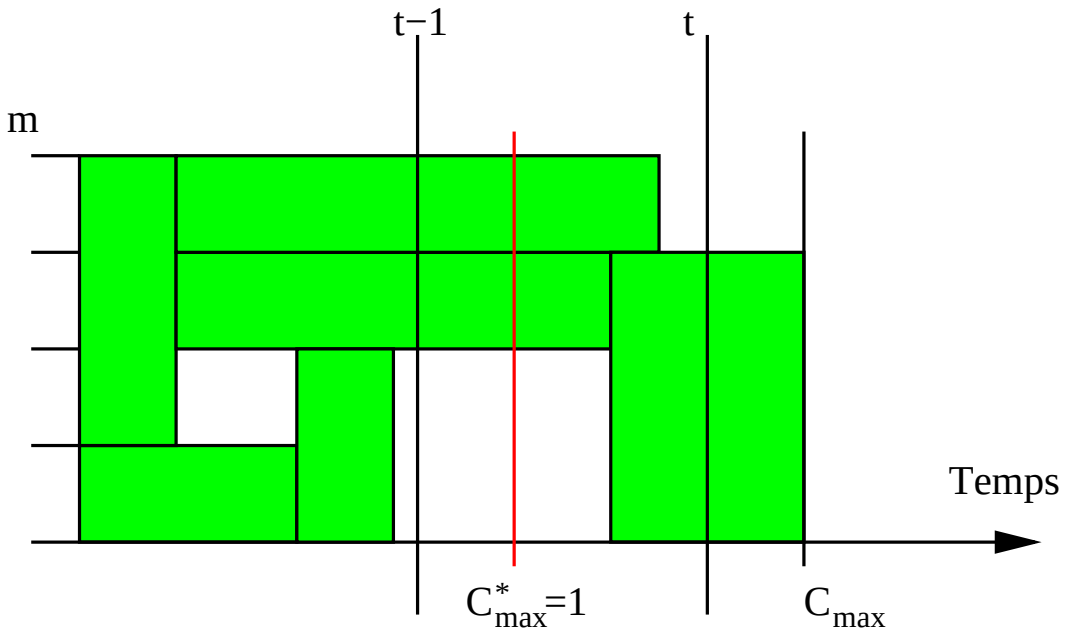
Tâches multiprocesseur Tout algorithme de liste à une garantie de performance meilleure que 2.

Idée

Compter les ressources utilisées à chaque instant

FIFO n'est pas un algorithme de liste





$$\forall t, 1 \leq t \leq C_{max}, Q(t) + Q(t-1) > m$$

$$m \geq \int_1^{C_{max}} Q(t) + Q(t-1) > (C_{max} - 1)m$$

Tâches modelables (*moldable, malleable*)

Nombre de processeurs

Une véritable application parallèle peut s'exécuter sur un nombre quelconque de processeurs.

Choix du nombre de processeurs

On peut donc laisser le système choisir ce nombre "au mieux".

Performances d'une application parallèle

Definition

Hypothèses de monotonie

- le temps d'exécution d'une tâche est une fonction décroissante du nombre de processeurs
- le travail d'une tâche est une fonction croissante du nombre de processeurs

Idée

Trouver le meilleur compromis entre le chemin critique et la somme des travaux

Pour faire mieux que 2 il faut être plus fin par rapport à la structure du problème et de la solution optimale :

- Toutes les tâches ont temps d'exécution plus petit que C^*_{max}
- La somme des travaux est plus petite que $m \times C^*_{max}$
- Un processeur exécute au plus une seule tâche i avec $p_i > \frac{C^*_{max}}{2}$

Theorem (Tâches modelable)

Il existe un algorithme avec une garantie de performance de $\frac{3}{2} + \epsilon$.

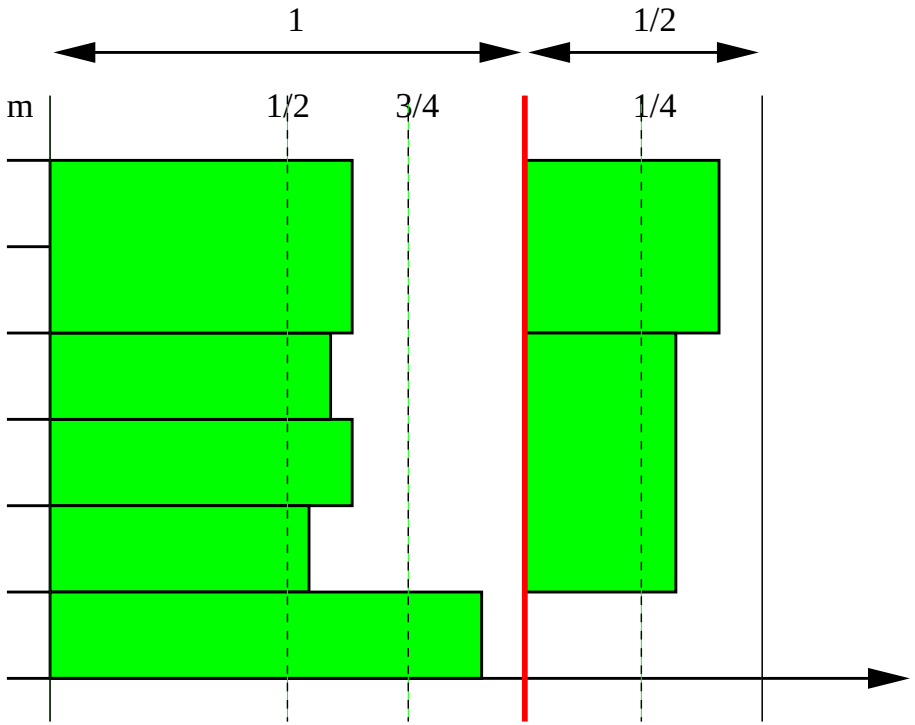
Changer de point de vue

Definition (Approximation duale avec une garantie de ρ)

- Choisir une valeur λ
- Choisir une heuristique qui calcule un ordonnancement en moins de $\rho\lambda$ si $\lambda \geq C_{max}$
- Faire une dichotomie sur la valeur de λ

Dans la suite, nous supposerons que $\lambda = 1$.

Structure de base en deux étagères de 1 et de 1/2



Les tâches dont le temps d'exécution sur 1 processeur est plus petit que 1/2 seront placés à la fin.

Choisir l'étagère de chaque tâche

Il n'y a que m processeurs utilisé par des tâches de durée supérieure à $1/2$. On peut donc faire un sac-à-dos optimal en temps pseudo polynomial nm (programmation dynamique), pour choisir si une tâche à une durée supérieure ou pas à $1/2$.

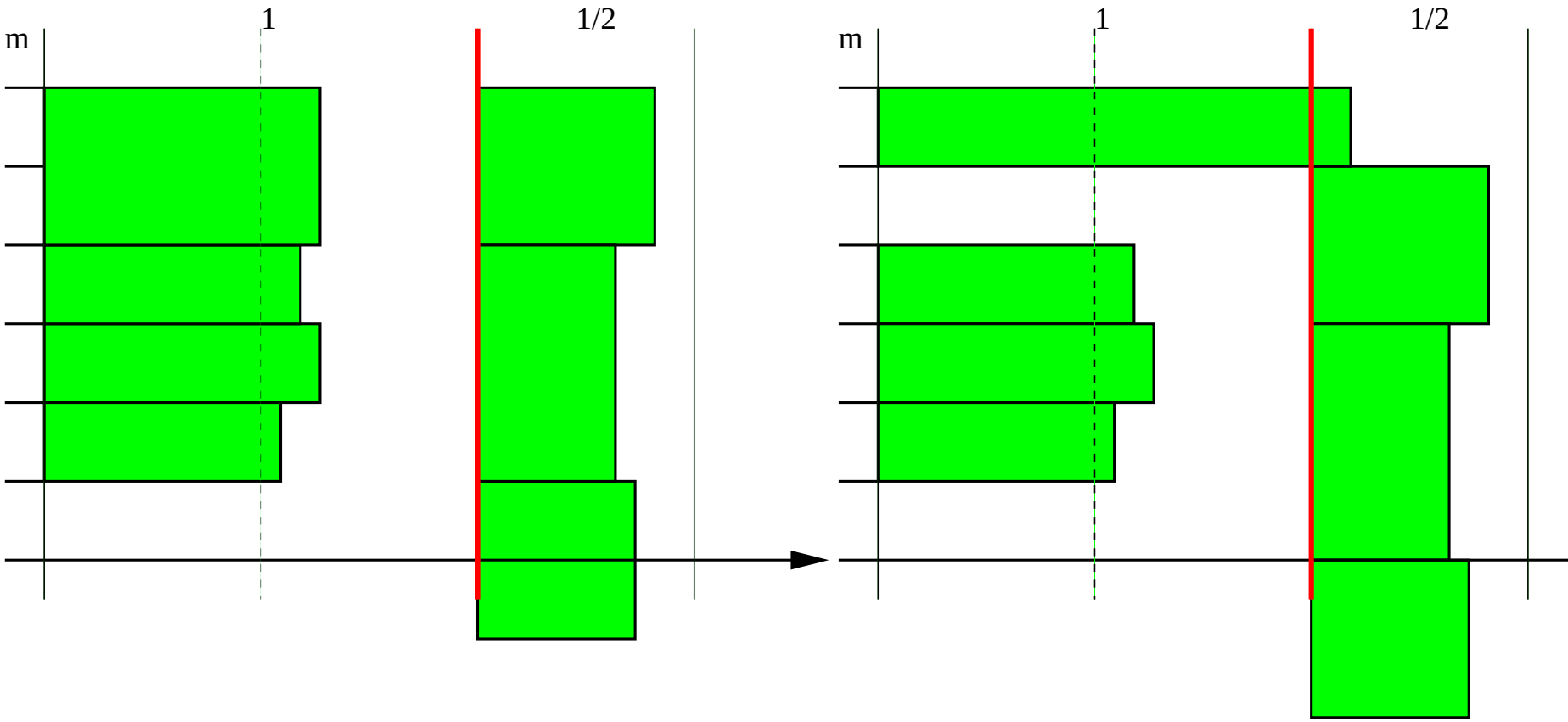
Sac-à-dos

Le choix de tâche minimisant le travail total est calculable en temps $O(nm)$

Ensuite il faut rendre la solution réalisable

Idée

Garantir une utilisation des processeurs à plus de 1 ou bien $3/4 + 1/4$.


m
1
1/2
m
1
1/2

Les petites tâches

Chaque petite tâche peut être inserée entre les deux étagères sur un des processeurs chargé à moins de 1.

Conclusion

Beaucoup d'heuristiques pour ordonnancer des applications paralllèles s'appuient sur deux bornes inférieures fondamentales :

- le chemin critique du graphe
- la somme des travaux

Pour obtenir des solutions heuristiques avec une garantie de performance il est souvent nécessaire de prendre en compte finement la structure du problème étudié.

Perspectives

De nombreux problèmes d'ordonnancement sont encore au coeur de la recherche autour du calcul parallèle :

- l'ordonnancement hiérarchique pour les grilles
- les ordonnancements à grand temps de communication
- l'optimisation de critères plus interactifs ou proches des besoins de certains utilisateurs : Stretch, contraintes temps réel,
- l'optimisation de plusieurs critères : minimiser simultanément le makespan et le temps moyen de terminaison des tâches.