



Les architectures multiprocesseurs
hiérarchiques seront bientôt partout...

Mais sait-on vraiment les programmer ?

Raymond Namyst

Équipe LaBRI-INRIA RUNTIME



L'équipe Runtime

- **Domaine d'application = Calcul intensif et simulation**
 - Sismologie, Nucléaire, Dynamique moléculaire, Météo, Combustion, ...
 - Partenaires : CEA, EDF, IFP, Total, Bull, ...
 - ⇒ **Utilisation de machines « parallèles »**
- **Thème : « Conception de supports d'exécution performants pour architectures parallèles »**
 - **Ordonnancement de processus sur machines multiprocesseurs**
 - Communications sur réseaux rapides
 - Communications sur infrastructures à grande échelle



C'est quoi un support d'exécution ?

Applications de calcul intensif

Algorithmes, Bibliothèques spécialisées

Support d'exécution

Système d'exploitation

Matériel

Ça fait dynamiquement ce que l'on ne sait pas faire statiquement...



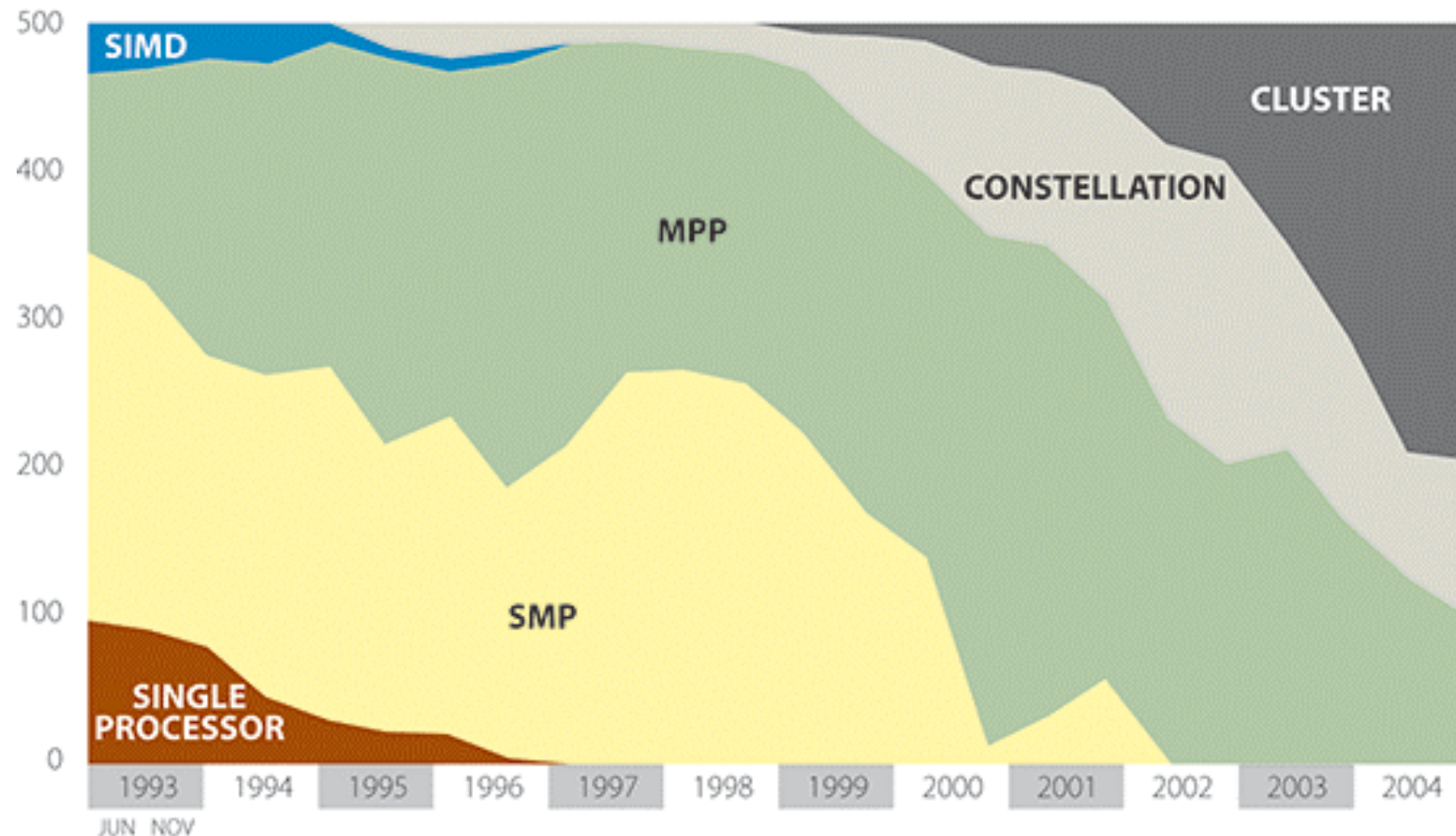
Les architectures de calcul contemporaines

Quelle est la tendance actuelle ?



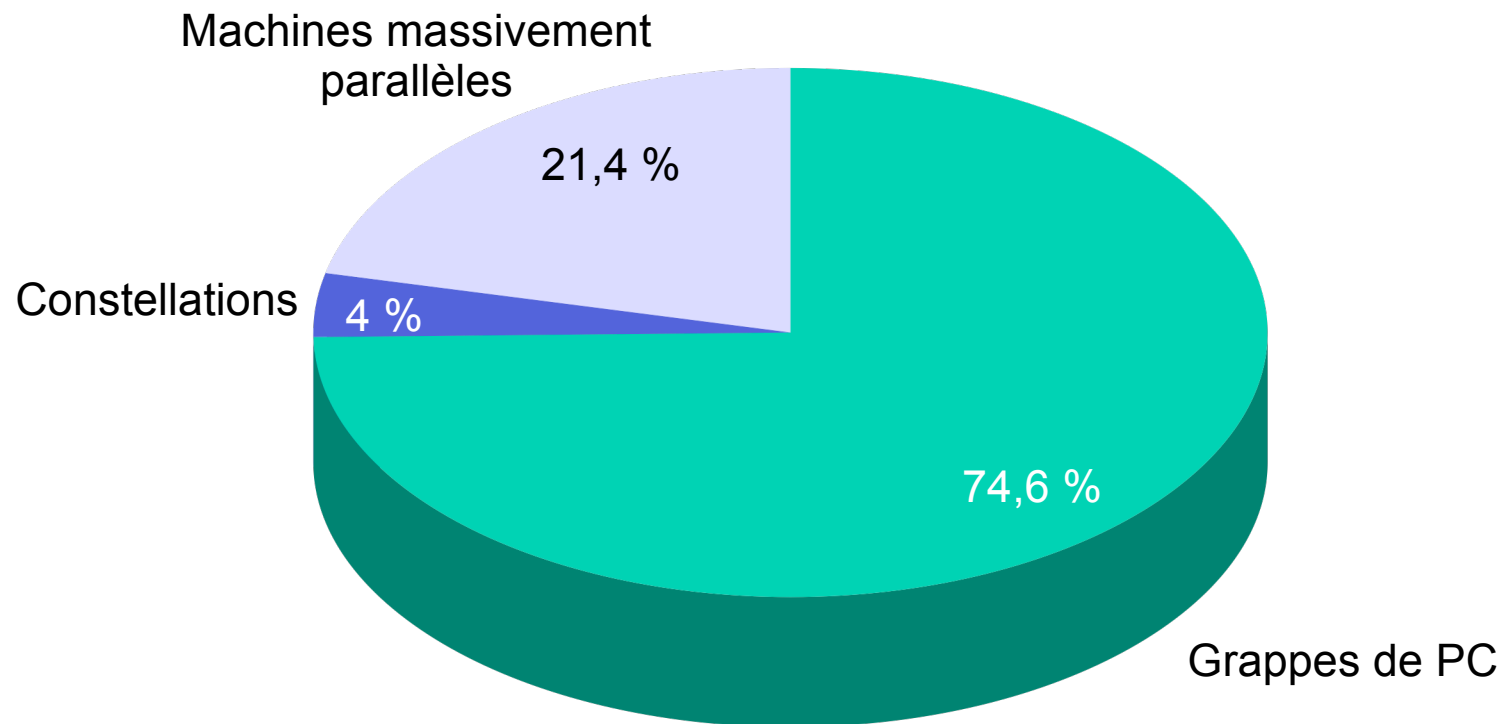
Physionomie des calculateurs parallèles contemporains

- <http://www.top500.org>





Physionomie des calculateurs parallèles contemporains



% de sièges au Top500 par catégorie d'architectures (juin 2007)



Les grappes de PC

- PC interconnectés par un (ou des) réseau(x) rapide(s)
 - Myrinet, Infiniband, Quadrics, ...
- Ces « PC » sont un peu survitaminés quand même
 - Beaucoup de mémoire
 - Plusieurs processeurs
 - Les processeurs sont « ordinaires »
 - Intel, AMD, IBM, Sun
 - C'est dans ce domaine que les évolutions sont les plus spectaculaires

1977 : Console Atari 2600

- 1,19 MHz



2006 : Sony PS3

- Processeur Cell 3.2 GHz
- Processeur graphique Nvidia RSX



2006 : Sony PS3

- Processeur Cell 3.2 GHz
- Processeur graphique Nvidia RSX

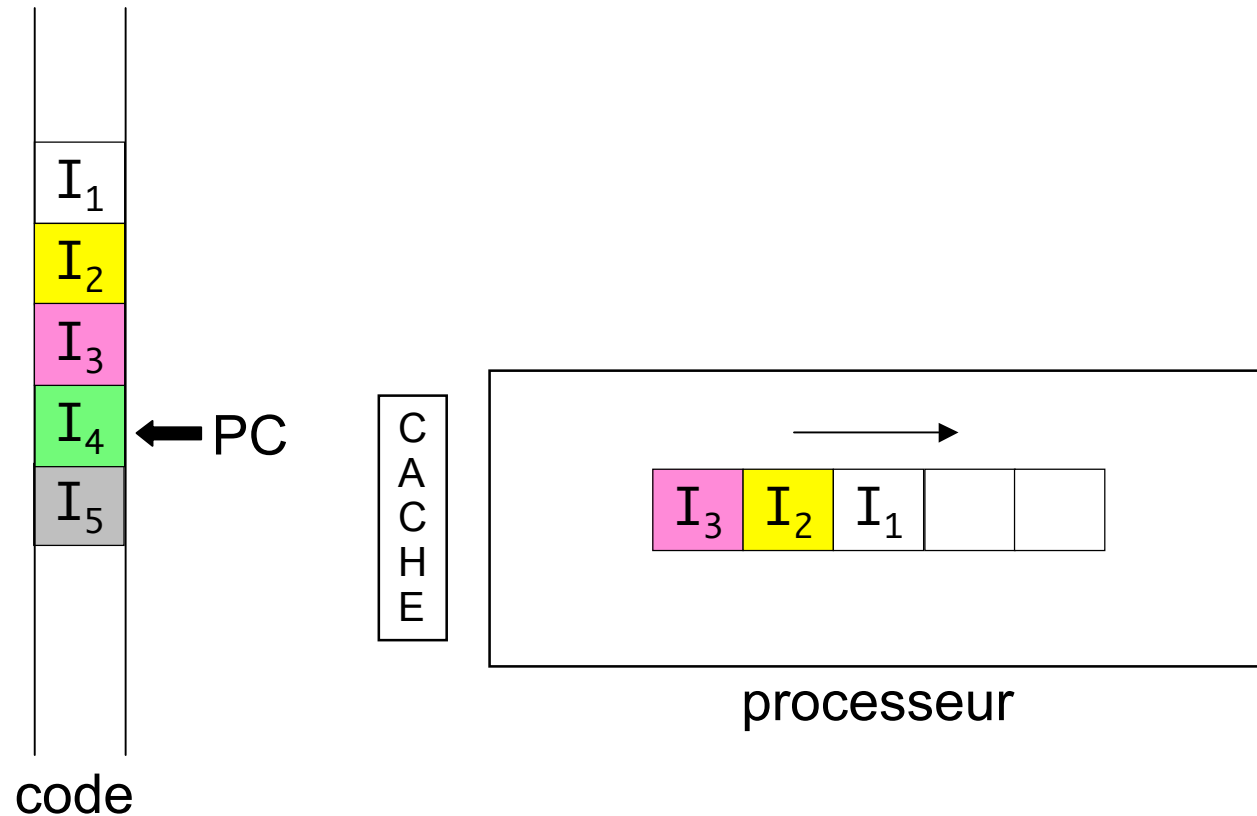


La PS3 calcule
des millions de fois
plus vite !





Une évolution clé: le pipeline de calcul



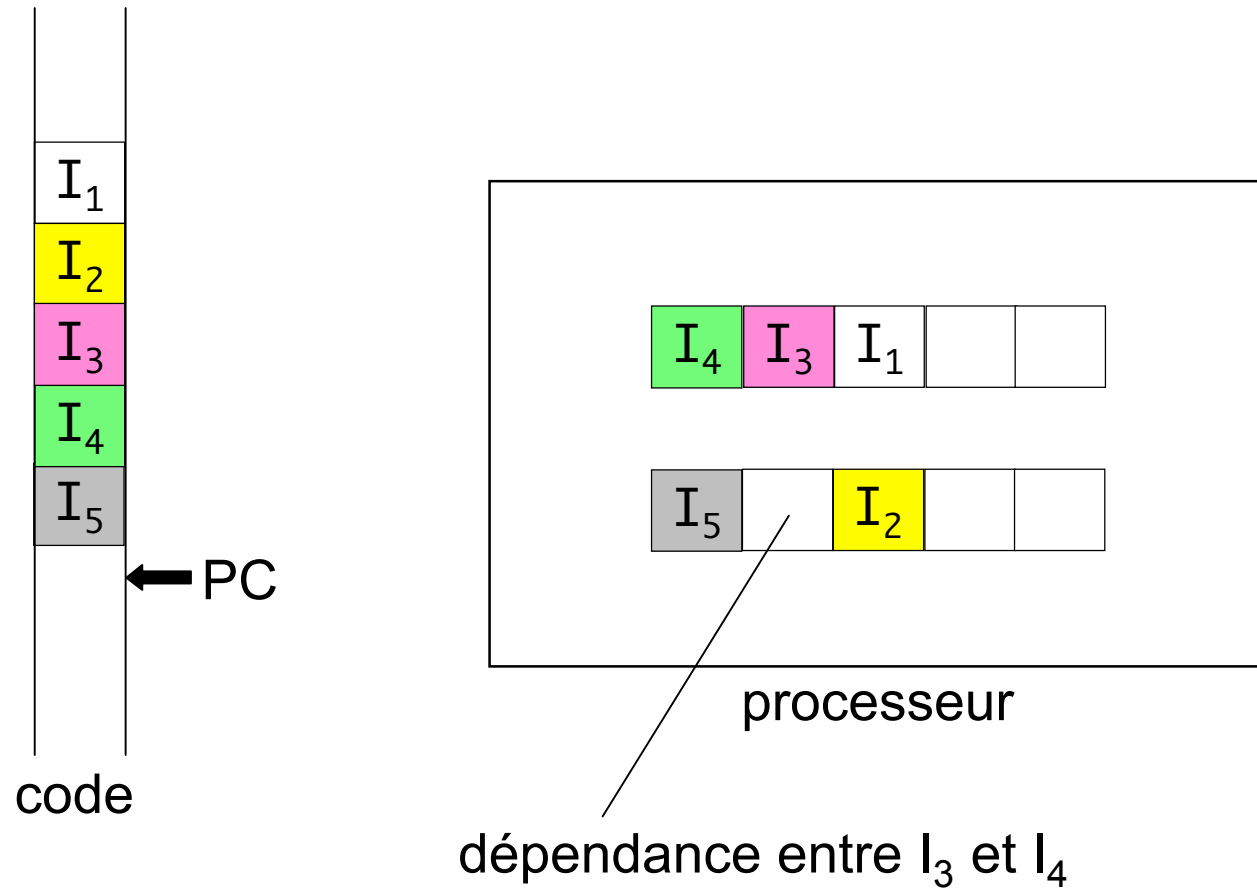


Les vertus de la finesse de gravure

- La fréquence augmente
 - En tout cas, c'était vrai jusqu'à il y a peu...
- Mais surtout, il y a de la place pour de nouveaux circuits
 - Caches
 - Exécution dans le désordre
 - Analyse de dépendances (renommage des registres)
 - Prédicateurs en tous genres
 - Exécution spéculative (branchements, préchargements)
 - Unités de calcul supplémentaires ?

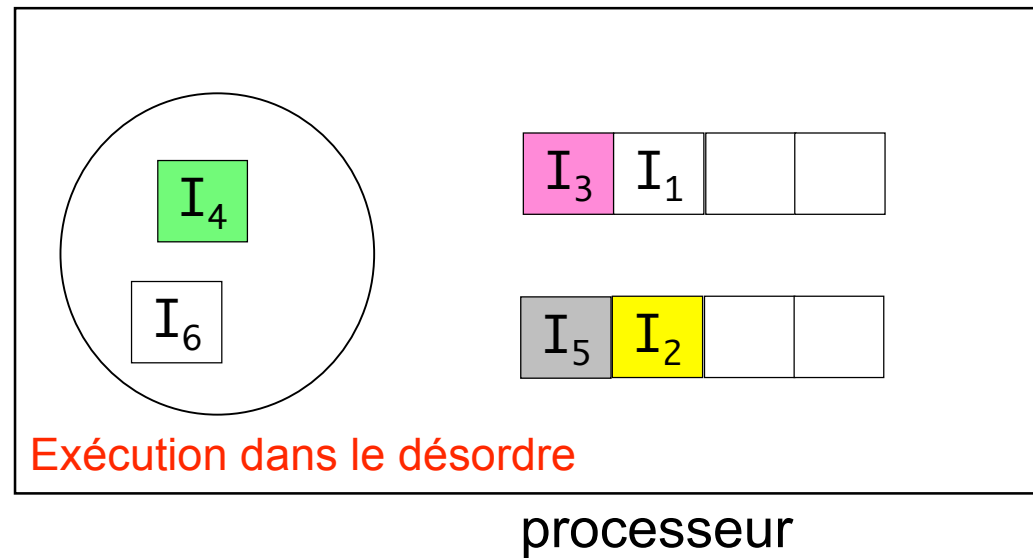
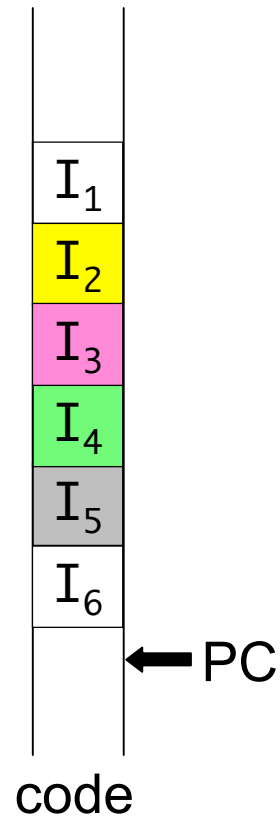


Processeurs « superscalaires »





En réalité,
c'est un peu plus compliqué...





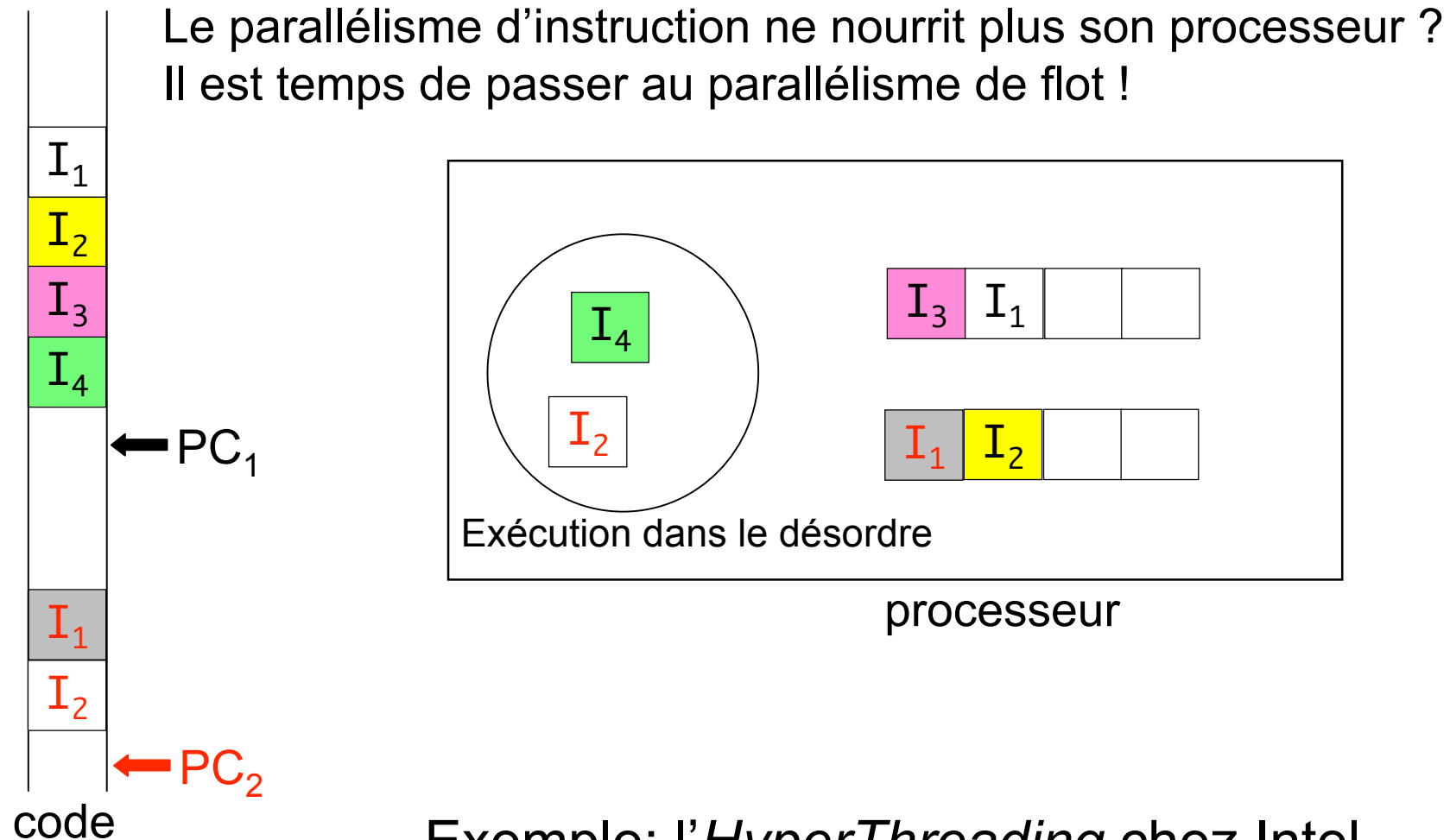
Les limites de l'« ILP »

- ILP = *Instruction Level Parallelism*
- Mais un flot séquentiel reste une suite d'instructions... souvent désespérément séquentielles !
 - Pas assez de parallélisme détectable « localement »
 - Les pipelines demeurent sous-utilisés
- Il serait vain d'ajouter encore des pipelines parallèles...

...à moins que ?



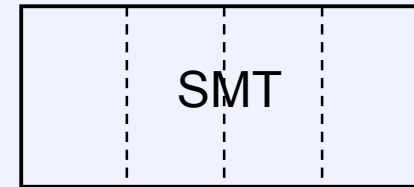
Processeurs multiprogrammés (ou *SMT*, 1990)





Quelles implications pour les programmeurs ?

- Le système d'exploitation « voit » une machine multiprocesseurs
 - Pour en tirer parti, il faut paralléliser son application à l'aide de threads/processus
- Mais cette vision est abusive !
 - Partage du cache
 - Partage de certaines unités de traitement
 - Progression des *threads* non équitable
- Peut-on augmenter infiniment le degré de multiprogrammation de ces processeurs ?
 - Non, pour des raisons de **complexité** et de **points de contention**
- Mais alors que fait-on avec la place qu'il reste sur les puces ?

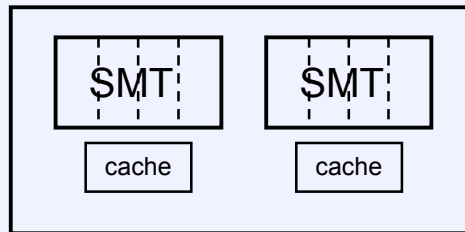


Processeur multiprogrammé



Les processeurs multicœurs

= plusieurs processeurs gravés sur une même puce

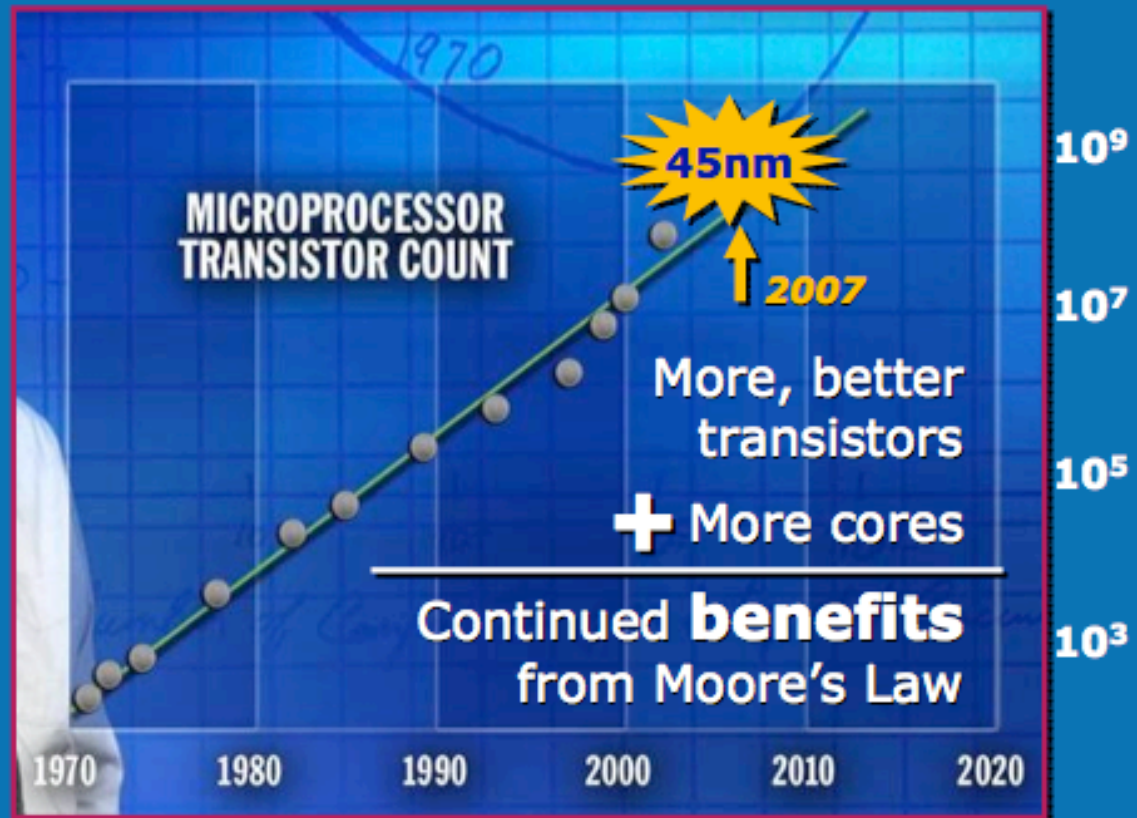


Puce bi-cœurs avec SMT 4 voies

- Caches externes parfois partagés
- **Tendance nette chez les constructeurs**
 - *Intel core 2 Duo, Itanium 2 Montecito, démonstration d'une puce à 80 cœurs !*
 - *AMD dual-core Opteron*
 - *IBM Power5*
 - *SUN Niagara*



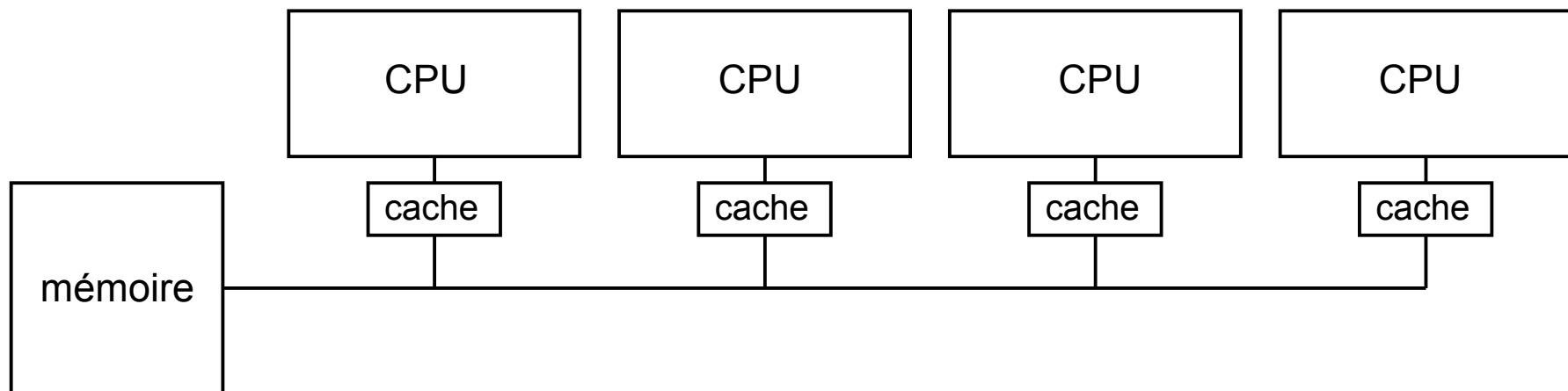
Gordon Moore a toujours raison...





Et les machines multiprocesseurs ?

- Elles sont majoritairement « à mémoire commune »
 - Mécanismes de cohérence des caches
- Il suffit donc de relier plusieurs processeurs à (au moins) un banc mémoire

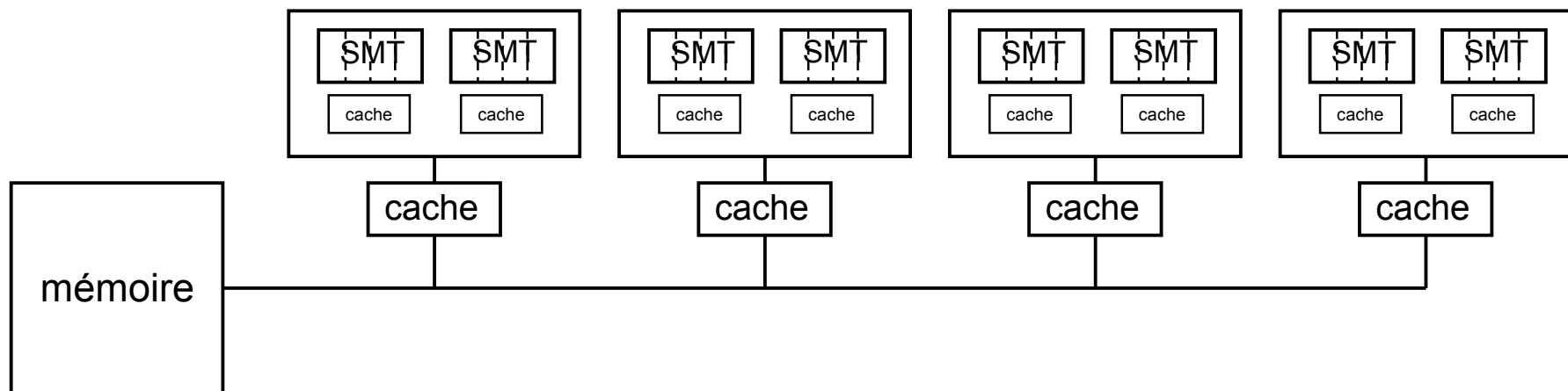


Architecture à accès mémoire uniforme



Avec des processeurs multicœurs...

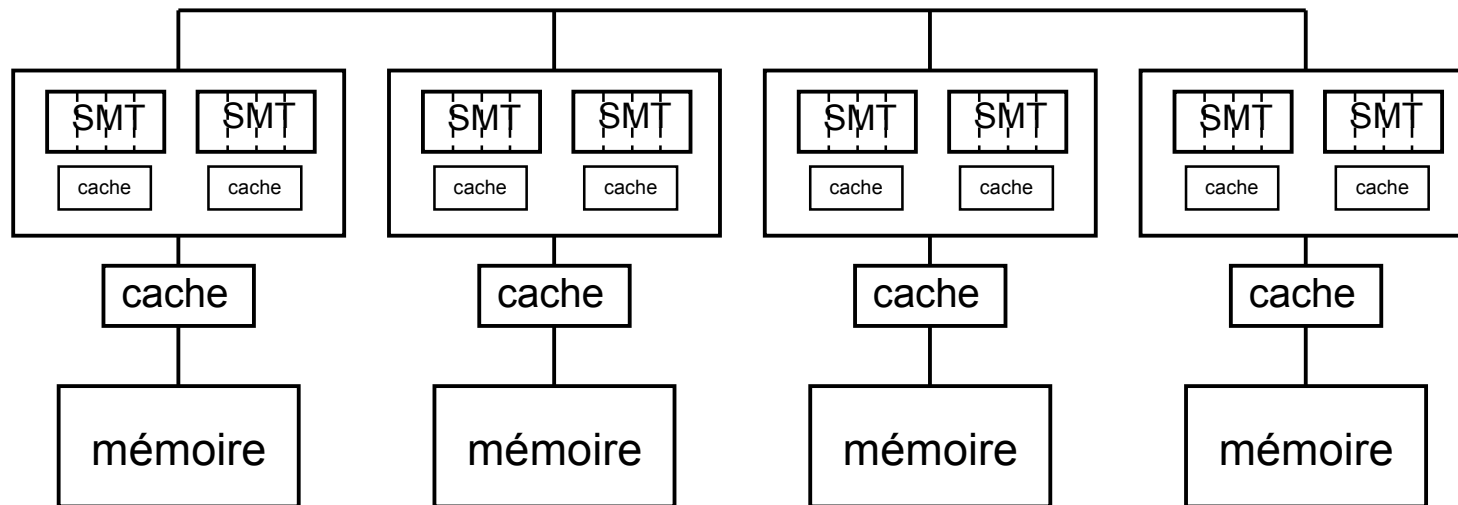
- Les échanges entre processus deviennent non uniformes
 - Données, synchronisations





On peut hiérarchiser davantage

- Mémoire directement attachée aux puces
 - Ex: AMD Opteron
 - Intel va également s'engager dans cette voie prochainement

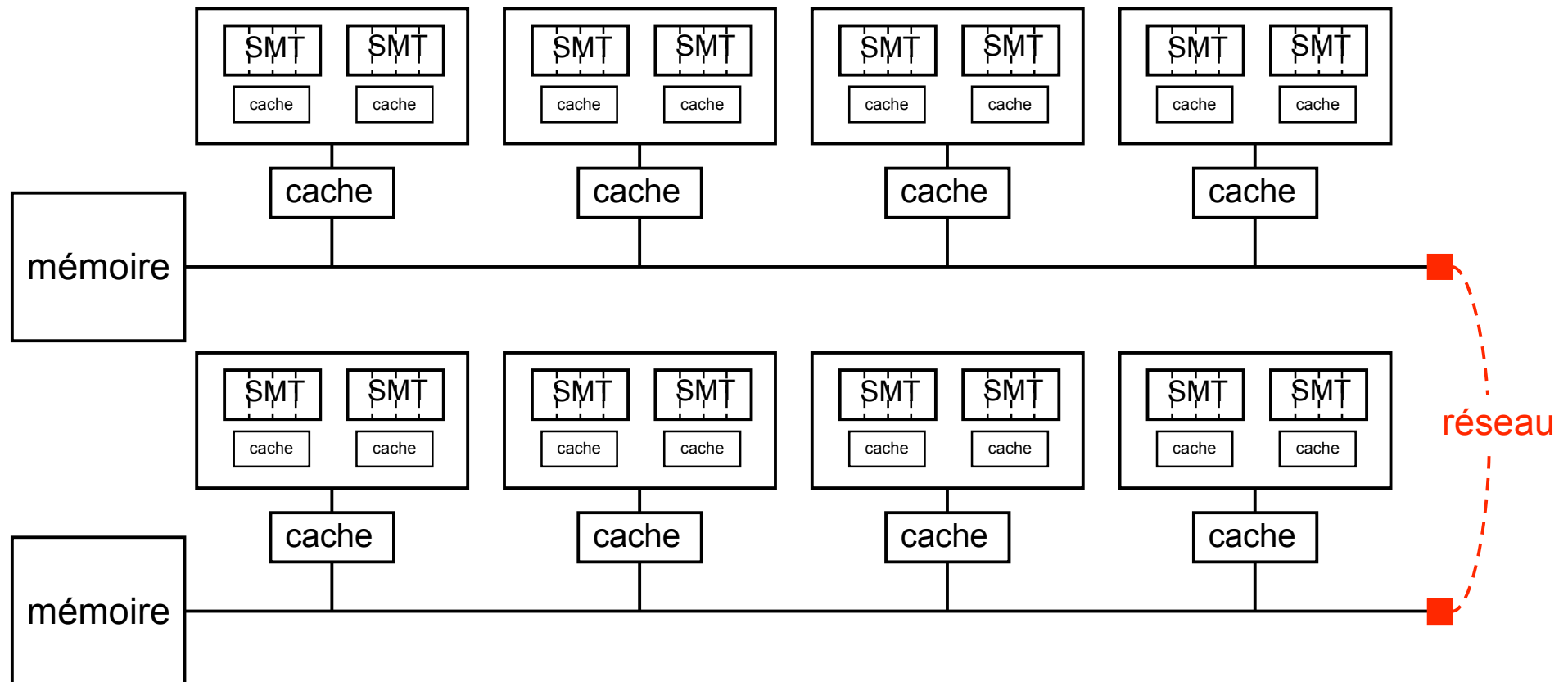


Architecture à accès mémoire non uniforme (NUMA)



Vers des architectures hiérarchiques complexes

- Vers des machines de plusieurs centaines de processeurs
 - Tout le monde n'est pas directement connecté à tout le monde



Ex: machine Tera10 du CEA/DAM



Comment exploiter efficacement
de telles architectures...

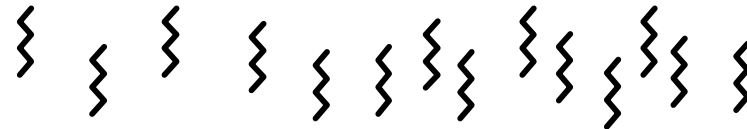
...de manière portable ?





Comment programme-t-on ces machines ?

- En créant explicitement des processus légers (*threads*)
 - Ordonnés par le système ou par un support d'exécution
- En recompilant une application parallèle distribuée sur un support ad-hoc
 - Remplacement des communications par des accès en mémoire partagée
- En reposant sur un compilateur de langage parallèle
 - Gestion de threads transparente et portable grâce un langage de haut niveau



Support exécutif

Système d'exploitation

Machine multiprocesseurs



Le standard OpenMP

- Extension de langages permettant de paralléliser des programmes séquentiels
 - Objectif = parallélisation incrémentale + portabilité
 - Directives de compilation (C, C++, Fortran) + routines spécifiques
 - <http://www.openmp.org>
 - V1.0 en 1997-1998, V2.5 en mai 2005
 - Plusieurs propositions d'extensions NUMA sont candidates pour la version 3.0
- Modèle de programmation
 - Type « Fork-Join », parallélisation des blocs et des boucles



Parallélisation des boucles

```
int main()  
{  
    int i;  
    double m[N];  
  
    #pragma omp parallel for  
    for(i=0; i<N; i++)  
        m[i] = f(i);  
  
    do_something(m);  
}
```

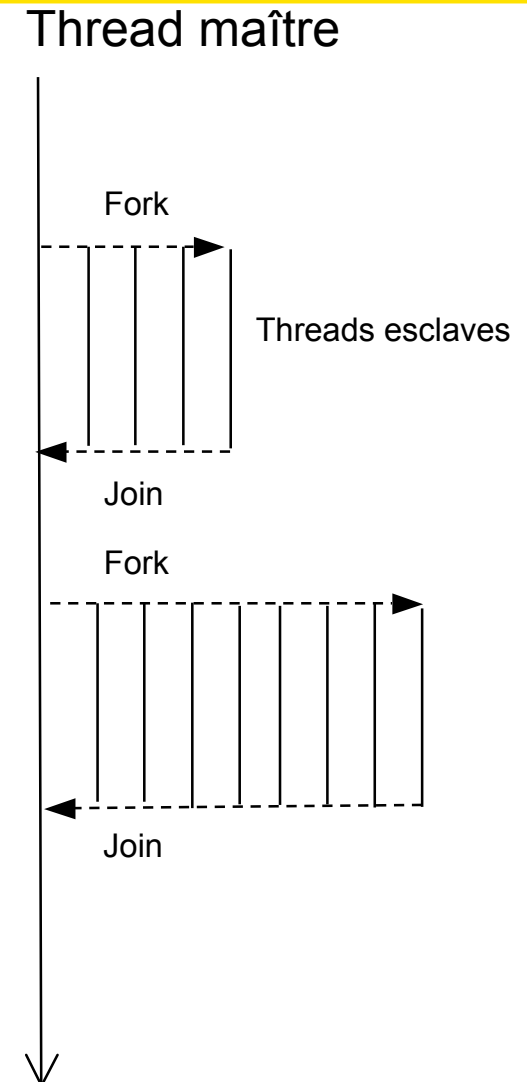
- Le nombre de threads générés par le compilateur est contrôlable

- En pratique, les utilisateurs le fixent systématiquement...



À propos du modèle fork/join

- Thread maître
 - Exécute le code séquentiel
 - Fork : création de threads esclaves
- Threads esclaves
 - Exécutent la portion parallèle
 - Join : destruction des esclaves et retour du contrôle au maître
- Le nombre d'esclaves peut varier d'une région parallèle à l'autre
- Technique bien adaptée à la parallélisation incrémentale d'un code





Comment ordonnancer des threads sur des machines hiérarchiques ?

Des bulles pour guider l'ordonnancement





Qu'entend-t-on par « ordonnancer efficacement » ?

- **Maximiser**
 - Le rendement de chaque processeur
 - la localité des accès mémoire
- **Minimiser**
 - le nombre de processeurs inactifs
 - le surcoût de l'ordonnanceur lui-même
- **Pour cela, il faut**
 - Renoncer à capturer une vision globale de l'ordonnancement
 - Enrichir la spécification des contraintes de placement/ordonnancement
 - Comment exprimer les affinités threads/mémoire ?
 - Comment décrire le comportement des threads (calcul, E/S) ?



Approche 1 : ordonnancement prédéterminé

- Utilisée dans les environnements spécialisés
- Procédure en deux phases
 - Prétraitement: calcul
 - du placement mémoire
 - de l'ordonnancement
 - Exécution: threads et données fixés selon ce prétraitement
- Performances : excellentes pour des problèmes réguliers
- Problème
 - Insuffisant/pas toujours possible pour les problèmes fortement irréguliers...
 - C'est en grande partie ceux-là qui nous intéressent !



Approche 2 : ordonnancement opportuniste

- Divers algorithmes gloutons
 - Une, plusieurs, une hiérarchie de listes de tâches
- Utilisés par les systèmes d'exploitation
 - Linux, BSD, Solaris, Windows, ...
- Redistribution des threads ou des régions mémoire
 - Par observation de compteurs de performance matériels
- Performances : variables
 - Pas de connaissance directe des affinités, par exemple



Approche 3 : ordonnancement négocié

- Extensions de langages
 - OpenMP, HPF, UPC, ...
- Approche portable
 - Adaptation automatique à la machine
- Problèmes
 - Expressivité limitée (e.g. pas de support NUMA)
 - Implémentations pas encore prêtes pour les machines hiérarchiques
- Extensions du système d'exploitation
 - Ex: libnuma
- Problème
 - Placement statique
 - Réécriture de la stratégie de placement en fonction de l'architecture



Quelle stratégie choisir ?

- Pas de solution universelle
 - Selon quels critères prendre des décisions d'ordonnancement ?
 - Affinités entre threads ?
 - Occupation mémoire ?
 - Consommation bande passante mémoire ?
 - Comment l'ordonnanceur pourrait-il deviner le comportement de l'application ?
- Mais de l'espoir quand même !
 - Le programmeur de l'application a souvent la réponse à ces questions
- Comment lui « donner le contrôle » de manière portable ?
 - Tout au long de l'exécution...

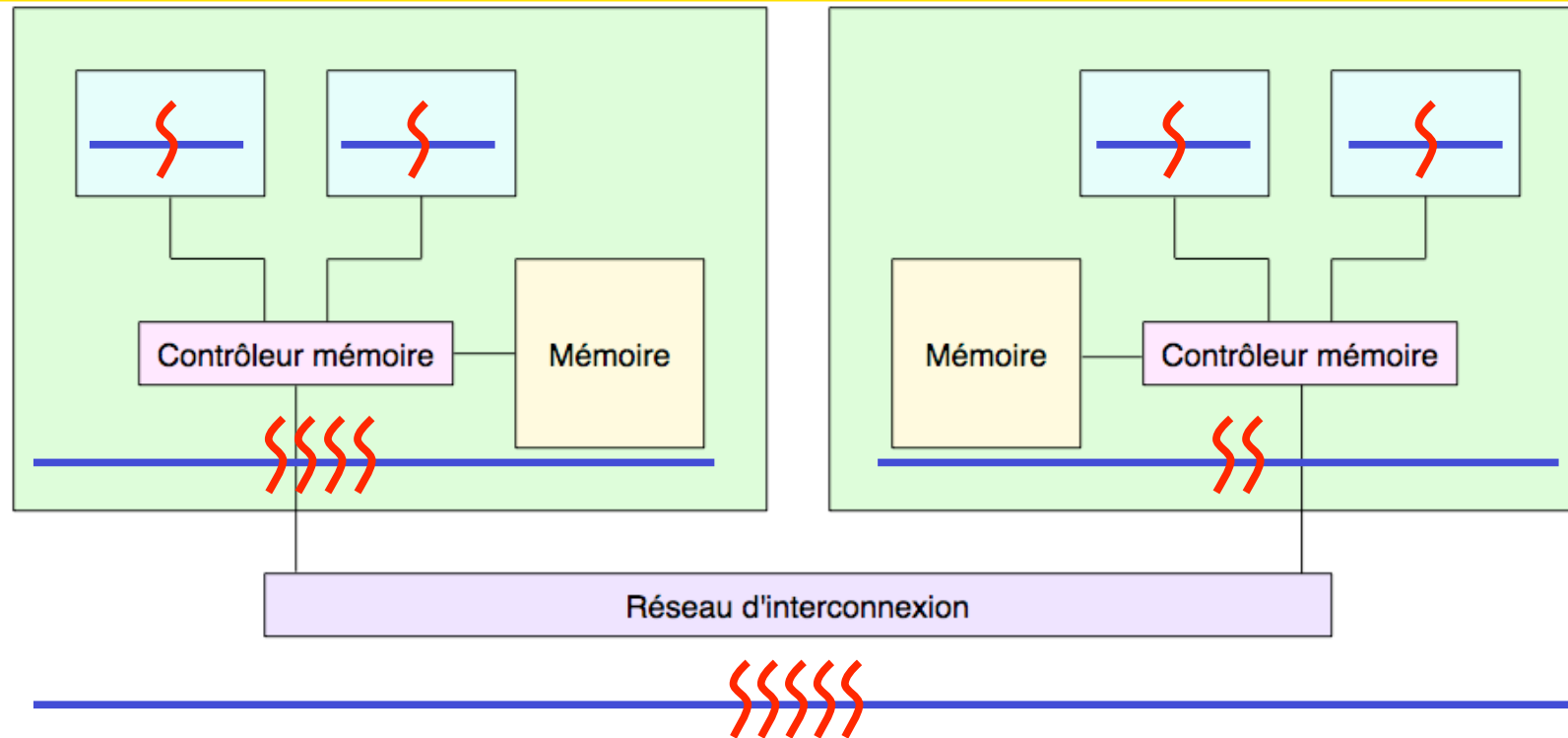


Idée

- Fournir des abstractions de haut niveau pour guider l'ordonnancement
 - Permettre les approches prédéterminée + négociée + opportuniste
 - Exprimer des relations entre threads, mémoire, périphériques d'E/S, etc.
- Permettre le développement de stratégies d'ordonnancement
 - Utilisation d'outils de primitives de haut niveau
 - Factorisation et masquage du « savoir faire » (très) technique

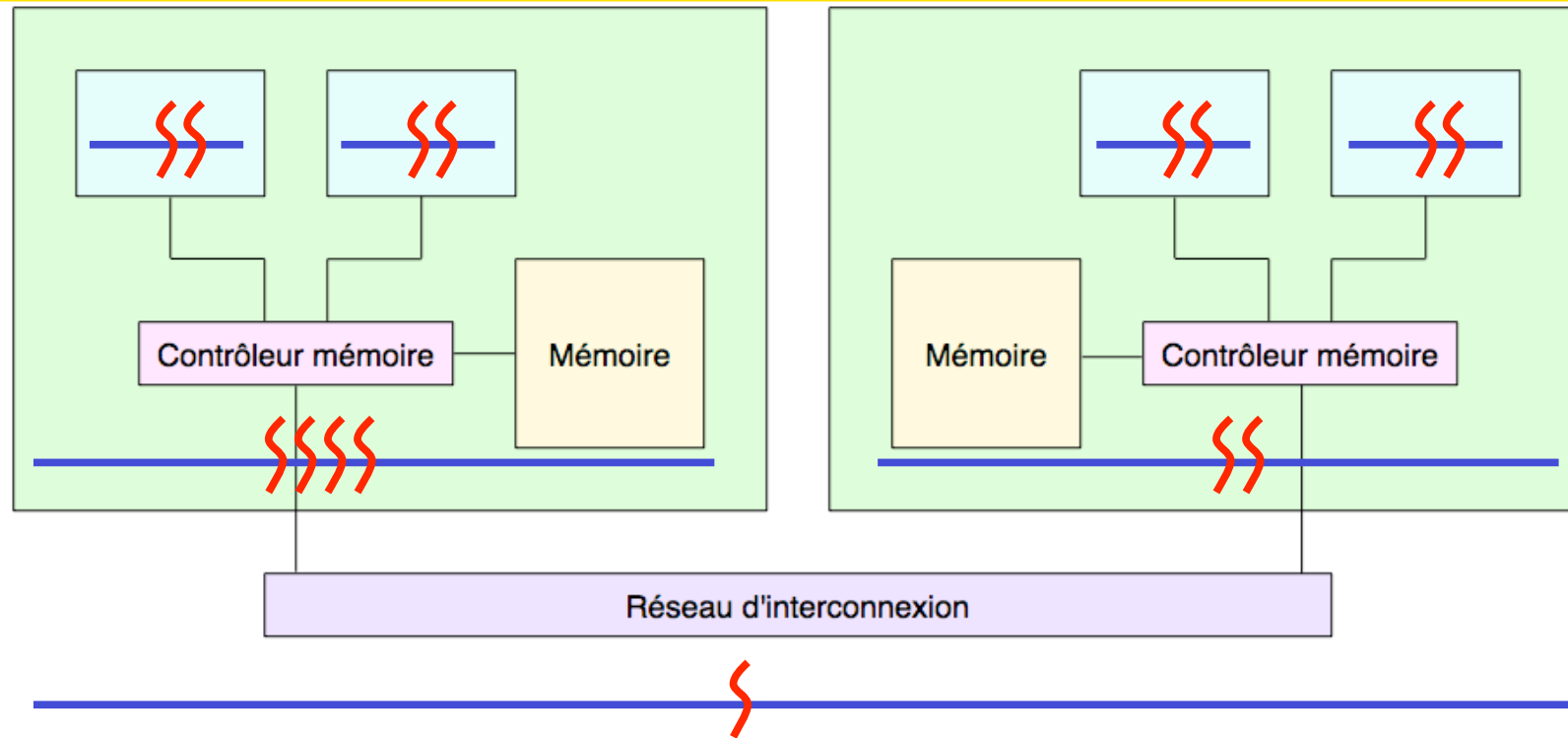


Modélisation de l'architecture





Modélisation de l'architecture

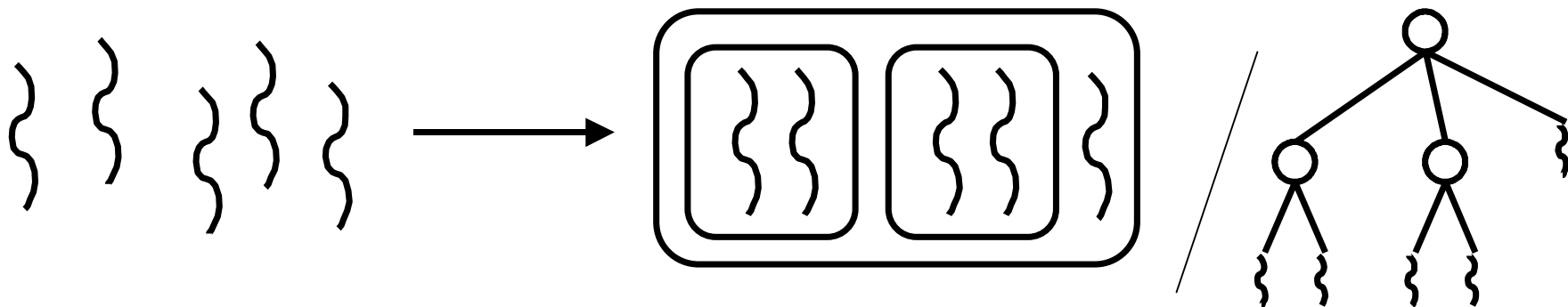




Notion de bulle pour exprimer des affinités

Réification de la structure des applications

- Partage de données
- Opérations collectives
- ...



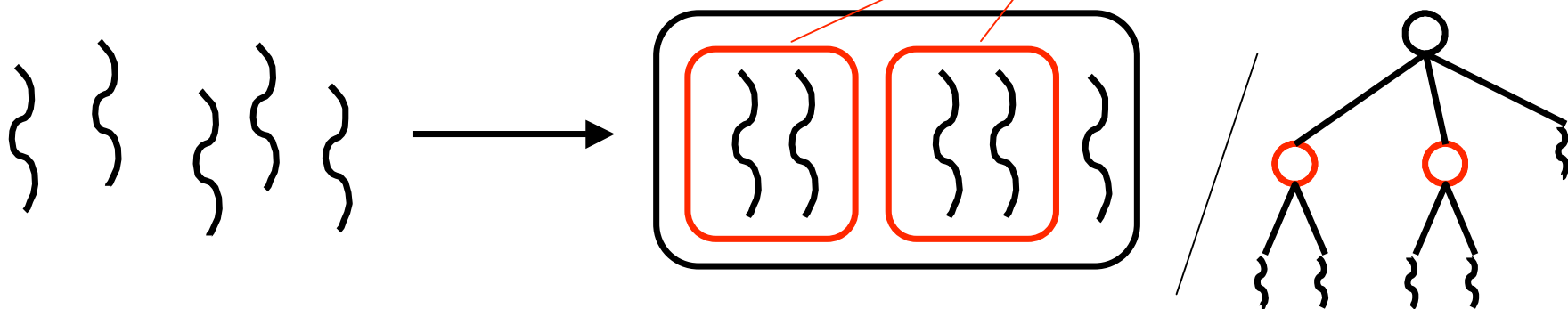


Notion de bulle pour exprimer des affinités

Réification de la structure des applications

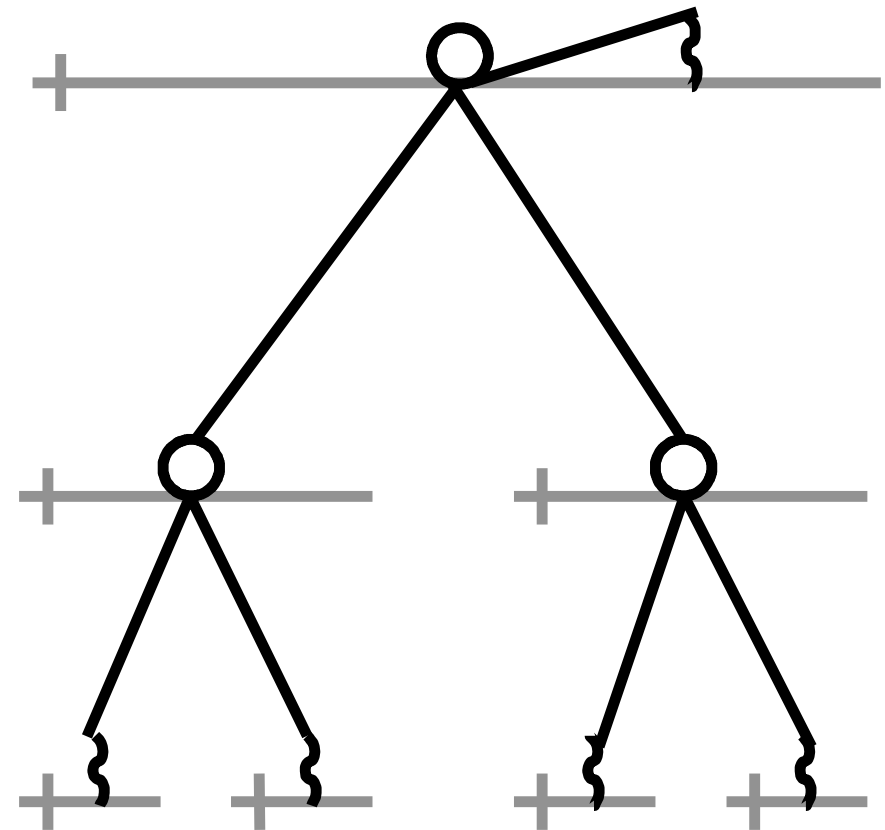
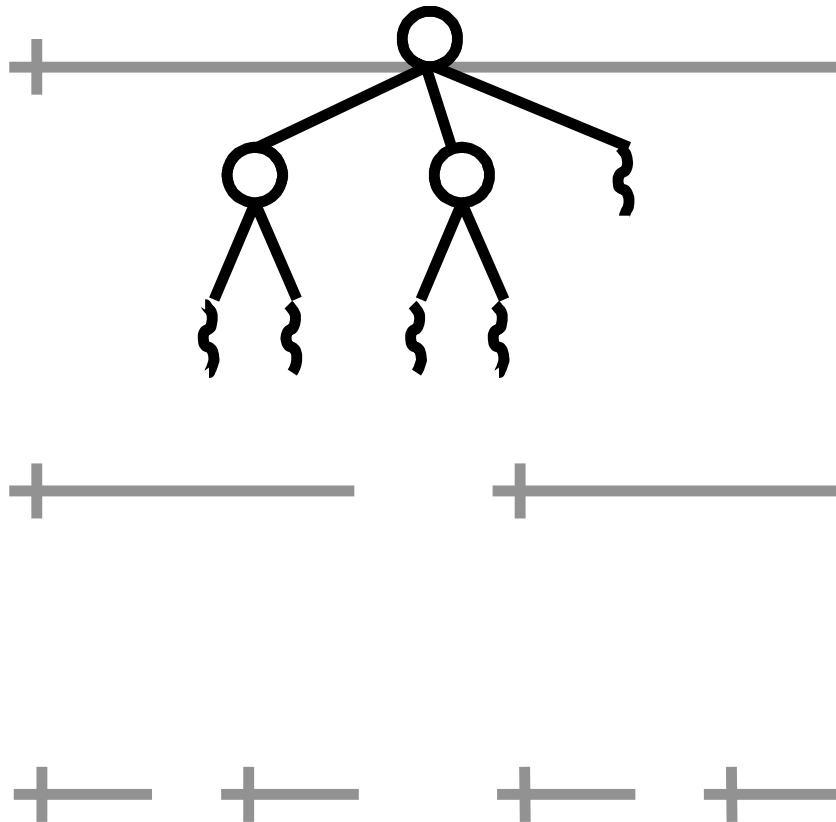
- Partage de données
- Opérations collectives
- ...

Certaines bulles peuvent être moins élastiques que d'autres





Exemples de répartitions possibles de bulles et threads





Informations attachées aux bulles

- Attributs statiques
 - Coefficient d'élasticité
 - Priorités
- Attributs collectés dynamiquement (accumulés récursivement)
 - Nombre de threads actifs
 - Taux de « *cache hit* » ou d'accès mémoire distants
 - Segments mémoire alloués



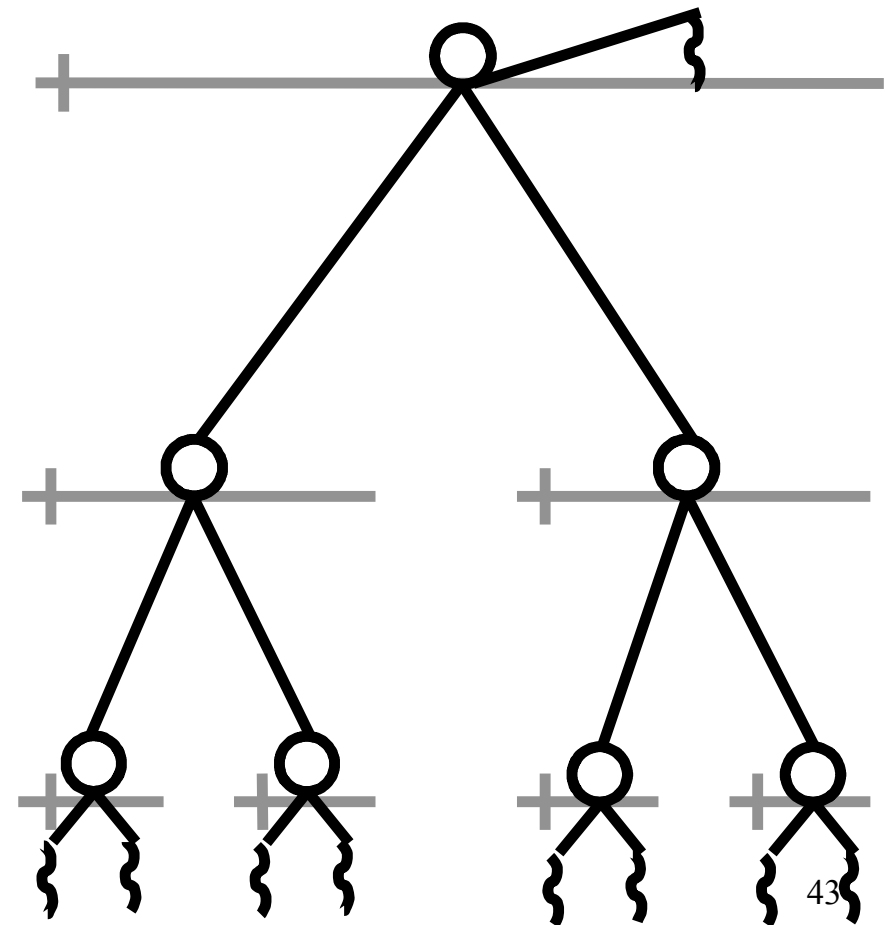
Principe d'exécution

- Lorsque le programme génère des bulles
 - Elles sont d'abord placées
 - sur la liste supérieure
 - ou à l'endroit où s'exécute le thread créateur
 - Puis une fonction de répartition est appelée pour « descendre » leur contenu en direction des processeurs
- Un processeur inactif
 - Cherche des threads à exécuter du bas vers le haut
 - Explore « virtuellement » le contenu des bulles récursivement
 - Caches de threads utilisés pour accélérer la recherche
- De nombreux événements d'ordonnancement déclenchent des actions de rééquilibrage



Plate-forme pour le développement d'ordonnanceurs

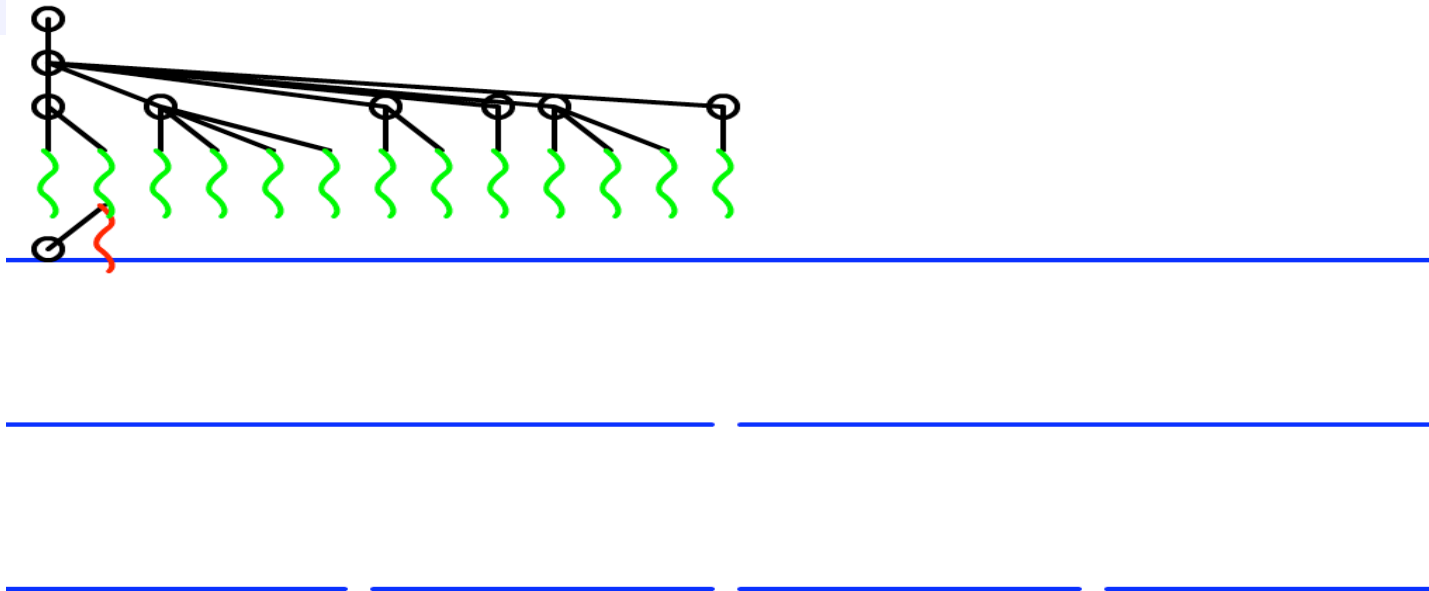
- Basé sur un ordonnanceur générique
- Points d'appels
 - Oisiveté d'un processeur
 - Timeslice « par bulle »
 - Création/Terminaison
 - Threads « démon »
- Base d'ordonnanceurs
 - Répartition par affinité
 - Éclatement et répartition équilibrée
 - Vol de travail
 - Gang scheduling





Analyse du comportement post-mortem

- Génération de trace légère pendant l'exécution
 - Bibliothèque FxT (Fast User/Kernel Traces) co-développée avec UNH
- Conversion de la trace en animation flash





Discussion

- Le modèle des bulles n'est pas « universel »
 - On ne peut pas exprimer n'importe quelle relation avec des arbres de bulles...
 - Ex: pas d'intersections entre bulles
- Mais il est bigrement effectif !
 - Adapté aux schémas de décomposition récursifs
 - Divide & Conquer
 - Ex: OpenMP
 - Algorithmes de répartition polynomiaux dans les cas simples



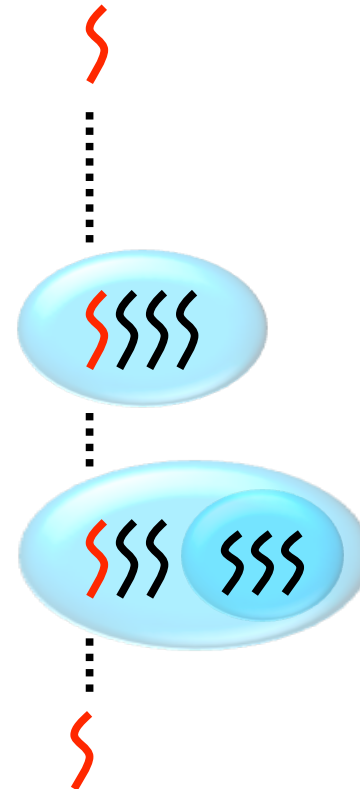
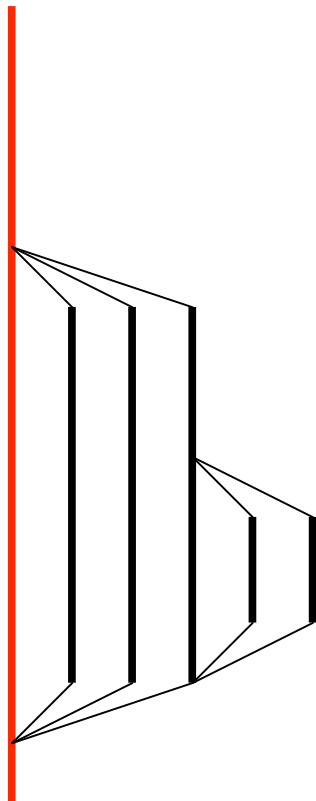
Des bulles dans OpenMP

Premières expériences avec GNU OMP



Idée : modifier un compilateur OpenMP pour utiliser BubbleSched

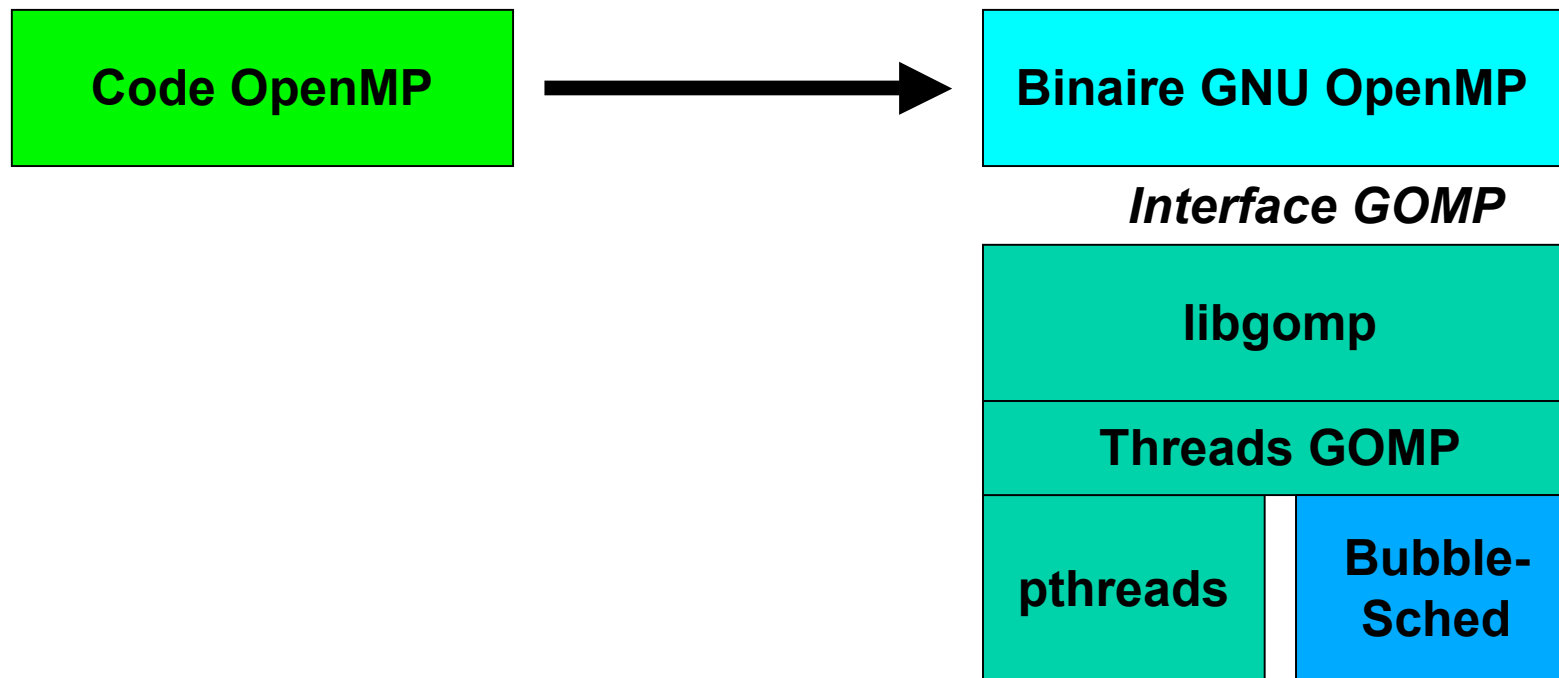
- Supporter davantage de threads que de processeurs
 - Meilleur contrôle du grain, effets de cache, équilibrage de charge





Mise en œuvre dans GOMP

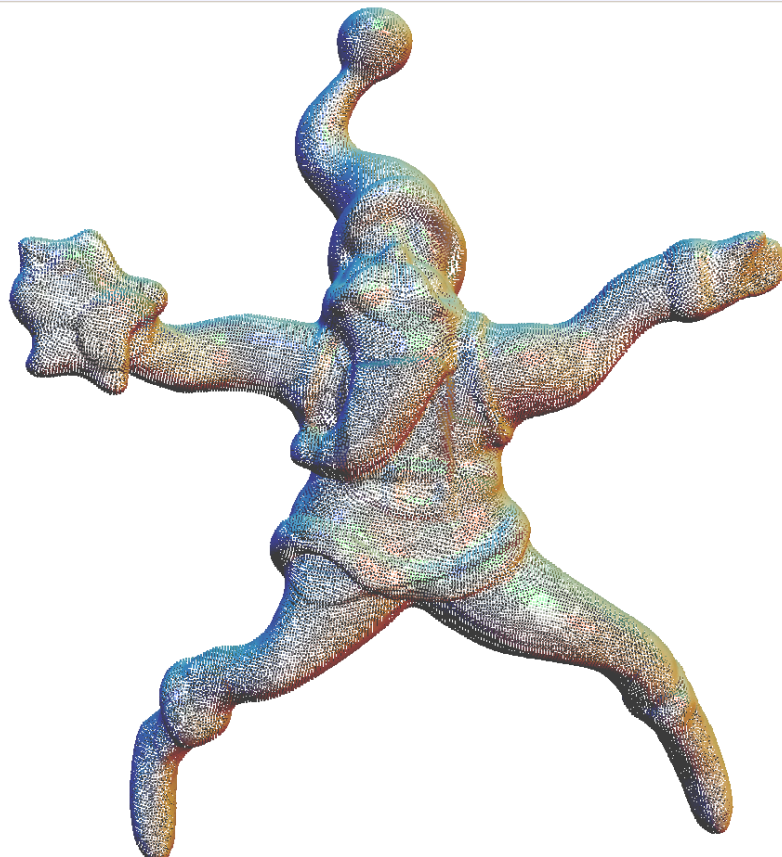
- Portage de GNU OpenMP sur les threads Marcel/BubbleSched
- Compatibilité binaire avec les applications OpenMP existantes





Évaluation à l'aide d'une application irrégulière

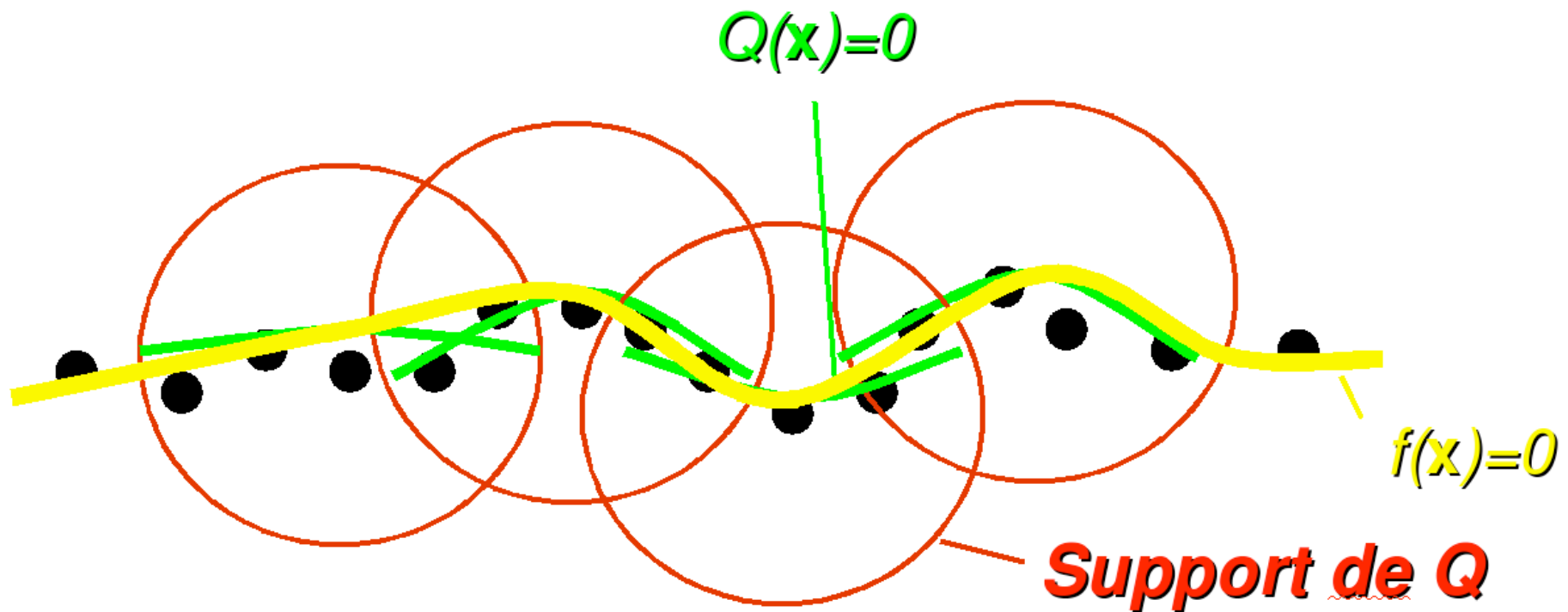
- Reconstruction de surface implicite
 - Objectif : trouver une fonction mathématique approximant la surface d'un objet définie par un nuage de points





Principe de l'algorithme séquentiel

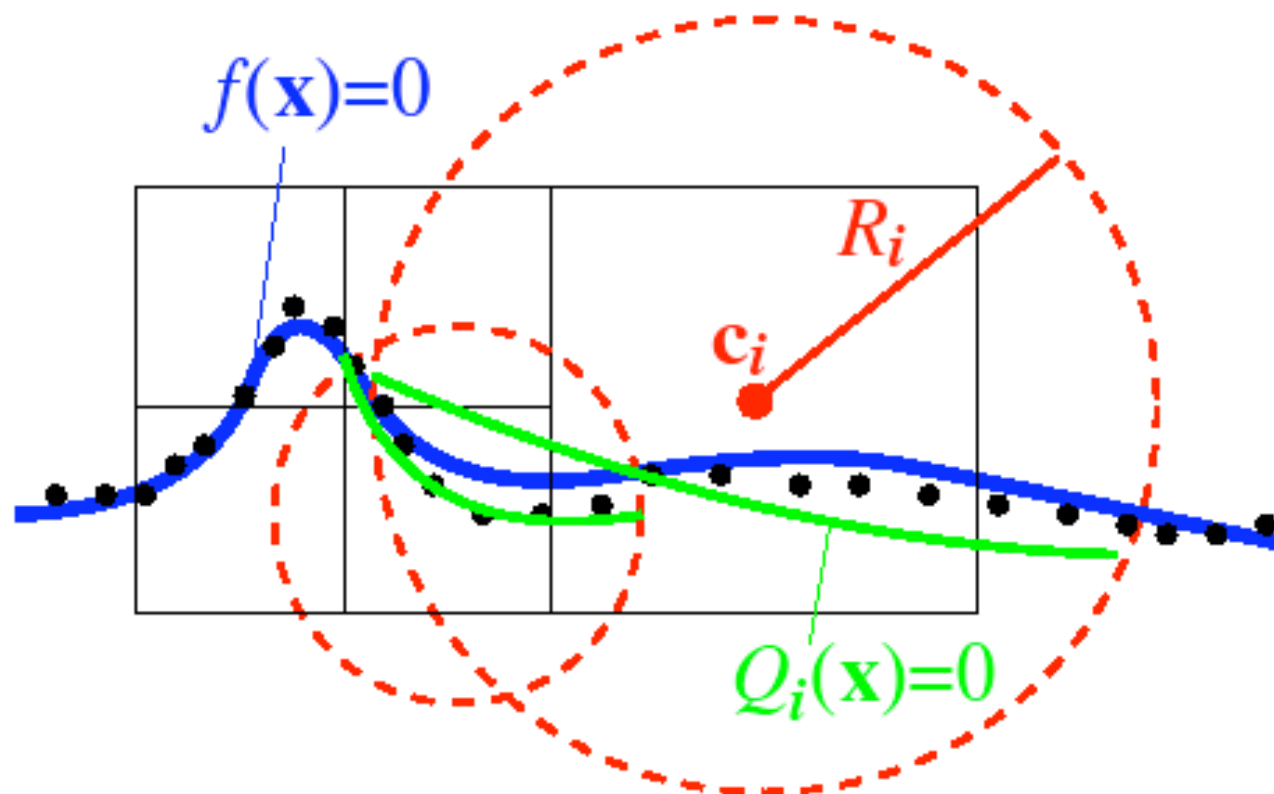
- Approximer la surface par des fonctions quadriques « locales »





Principe de l'algorithme séquentiel

- Subdiviser l'espace lorsque la précision est insuffisante
(ici en 2D)



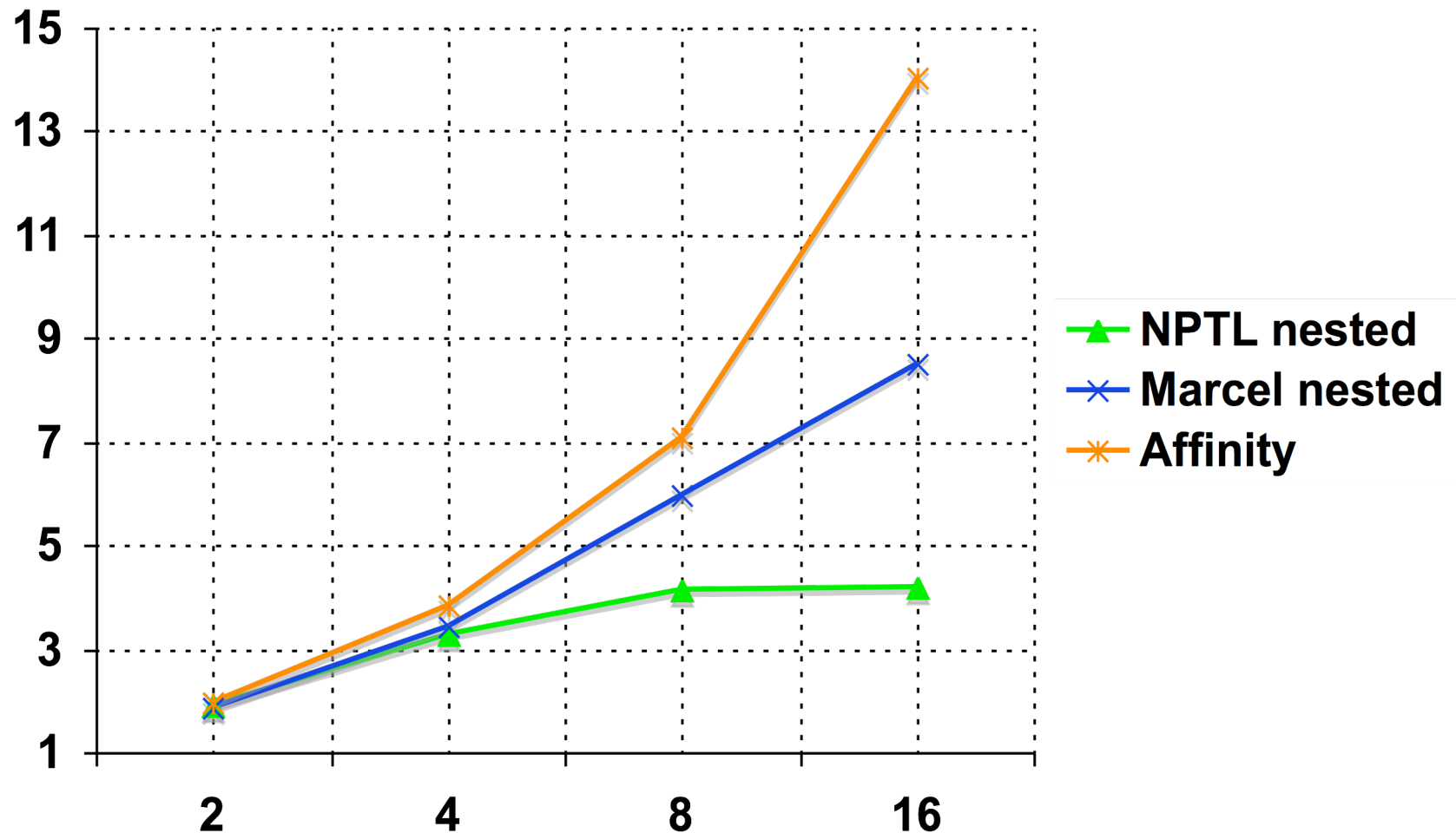


Parallélisation en OpenMP

```
void Node::compute() {  
  
    computeApprox();  
  
    if(_error > _max_error){  
        splitCell();  
  
        #pragma omp parallel for  
        for(int i=0; i<8; i++)  
            _children[i]->compute();  
    }  
}
```



Accélération sur 16 cœurs





Conclusion et perspectives

Du pain sur la planche...



Conclusion

- Les constructeurs de microprocesseurs nous donnent du fil à retordre !
 - Gestion du parallélisme (bientôt massif !) reportée au niveau logiciel
 - Puissance soutenue de plus en plus éloignée de la puissance théorique !
- Les supports exécutifs doivent suivre
 - Gérer efficacement un grand nombre de flots d'exécution simultanés
 - Permettre le contrôle de l'ordonnancement
 - Inciter le programmeur à exhiber le plus de parallélisme possible
- Mais le chemin est encore long !
 - Majorité de programmes écrits en Fortran+MPI
 - Imposer un nouveau modèle de programmation, c'est long !



Et demain ?

- Améliorer l'articulation entre compilateurs, supports d'exécution et matériel
 - Extraction d'informations
 - Quantification des affinités, schémas de synchronisation
 - Collecte d'informations
 - Compteurs de performance, accès mémoire/cache
- Maîtriser les évolutions matérielles
 - Architectures multicœurs hétérogènes
 - Ex: Cell
 - Architectures dépourvues de cohérence de cache
 - Accélérateurs spécialisés
 - GPU