Parallélisme: utilisation directe des threads; exemple avec boost

Thierry Dumont

29 novembre 2008

Le problème :

Retour au TP. “Réaction–Diffusion”.

Les problèmes “réactifs (R_h) sont autant de problèmes indépendants que de nœuds sur la grille. Paralléliser ?

Le problème :

Retour au TP. “Réaction–Diffusion”.

Les problèmes “réactifs (R_h) sont autant de problèmes indépendants que de nœuds sur la grille. Paralléliser ?

Mémoire partagée (machine multicœurs).

Le problème :

Retour au TP. “Réaction–Diffusion”.

Les problèmes “réactifs (R_h) sont autant de problèmes indépendants que de nœuds sur la grille. Paralléliser ?

Mémoire partagée (machine multicœurs).

Gestion directe des processus légers (threads).

- problème numérique,
- programmation des threads.

Problème numérique

- la mauvaise idée : découper l'ensemble des nœuds en paquets de même taille (autant de paquets que de processus légers (autant que de processeurs)).

Problème numérique

- la mauvaise idée : découper l'ensemble des nœuds en paquets de même taille (autant de paquets que de processus légers (autant que de processeurs)).
  Adaptation du pas $=>$ temps calculs très variables.

Problème numérique

- la mauvaise idée : découper l'ensemble des nœuds en paquets de même taille (autant de paquets que de processus légers (autant que de processeurs)). Adaptation du pas $=>$ temps calculs très variables.
- la bonne idée : découper l'ensemble des nœuds en un grand nombre de paquets (exemple : un paquet= une ligne de la grille).

Problème numérique

- la mauvaise idée : découper l'ensemble des nœuds en paquets de même taille (autant de paquets que de processus légers (autant que de processeurs)). Adaptation du pas $=>$ temps calculs très variables.
- la bonne idée : découper l'ensemble des nœuds en un grand nombre de paquets (exemple : un paquet= une ligne de la grille).

1. Une liste des paquets de nœuds à traiter est fabriquée ;

Problème numérique

- la mauvaise idée : découper l'ensemble des nœuds en paquets de même taille (autant de paquets que de processus légers (autant que de processeurs)). Adaptation du pas $=>$ temps calculs très variables.
- la bonne idée : découper l'ensemble des nœuds en un grand nombre de paquets (exemple : un paquet= une ligne de la grille).

1. Une liste des paquets de nœuds à traiter est fabriquée ;
2. Chaque thread va *aller chercher* un paquet à traiter,

Problème numérique

- la mauvaise idée : découper l'ensemble des nœuds en paquets de même taille (autant de paquets que de processus légers (autant que de processeurs)). Adaptation du pas $=>$ temps calculs très variables.
- la bonne idée : découper l'ensemble des nœuds en un grand nombre de paquets (exemple : un paquet= une ligne de la grille).

1. Une liste des paquets de nœuds à traiter est fabriquée ;
2. Chaque thread va *aller chercher* un paquet à traiter,
3. Le paquet est retiré de la liste... jusqu'à ce que la liste soit vide.

droits et devoirs des threads

Un thread a tous les droits vis à vis de la mémoire... pas de gendarme !

droits et devoirs des threads

Un thread a tous les droits vis à vis de la mémoire... pas de gendarme !

- pas de problème si les threads partagent une zone de mémoire en lecture seule

droits et devoirs des threads

Un thread a tous les droits vis à vis de la mémoire... pas de gendarme !

- pas de problème si les threads partagent une zone de mémoire en lecture seule
- danger en écriture ! Chaque thread *doit* accéder à la mémoire partagée en écriture... de manière disciplinée.

Ici :

- les paquets correspondent à des nœuds distincts : pas de problème pour les traiter.

Ici :

- les paquets correspondent à des nœuds distincts : pas de problème pour les traiter.
- la liste des paquets à traiter doit être protégée de sorte qu'un seul thread la modifie à un moment donné. Notion de lock.

Ici :

- les paquets correspondent à des nœuds distincts : pas de problème pour les traiter.
- la liste des paquets à traiter doit être protégée de sorte qu'un seul thread la modifie à un moment donné. Notion de lock.

Threads Posix normalisés : programmation pénible.

Ici :

- les paquets correspondent à des nœuds distincts : pas de problème pour les traiter.
- la liste des paquets à traiter doit être protégée de sorte qu'un seul thread la modifie à un moment donné. Notion de lock.

Threads Posix normalisés : programmation pénible.

Bibliothèques facilitant l'utilisation : Phyton-threads, BOOST (C++).

Boost

```
#include <boost/thread/mutex.hpp>
#include <boost/thread/thread.hpp>
```

Dans `Data2`

```
boost::mutex mutex;

patch Data2::tget(bool& finished)
{
  // la vue scoped_lock de mutex sera détruite
  // à la sortie de cette routine, permettant
  // à un  autre thread d'y entrer.
  boost::mutex::scoped_lock scoped_lock(mutex);
  finished=Pile.empty();
  patch P;
  if(!finished)
    {
      P=Pile.top();
      Pile.pop();
      return P;
    }
}
```

les threads

```
Reaction( paramètres )

{
  Patch P;
  bool finished;
  P=tget(finished);
  while(!finished)
   {
    ....resoudre les equtions du paquet P
    ....stocker les resultats
    P=tget(finished);
   }
}
```

Le controle des threads

Routine (paramètre `nthreads`) qui contient :

```
boost::thread_group thrds;


for (int i=0; i < nthreads; ++i)
      thrds.create_thread(Reaction());


thrds.join_all();
```