Introduction aux tâches en OpenMP

P. Fortin – UPMC / LIP6
ANR TaMaDi
27/10/2010

Bibliographie et remerciements

- *Tasking in OpenMP*, A. Duran, 5th International Workshop on OpenMP, 2009
- *Using OpenMP,* B. Chapman, G. Jost, R. van der Pas, MIT Press
- Norme OpenMP 3.0 (http://openmp.org)
- U.C. Berkeley CS267, J. Demmel et al. (http://www.cs.berkeley.edu/~demmel/cs267_Spr09)

Création des threads : les différentes possibilités (d'après CS267)

- Modèle *cobegin/coend* :

```
cobegin
     job1(a1);
     job2(a2);
coend
```

→ Nids de boucles avec OpenMP 2.5

- Modèle *fork/join* :

```
tid1 = fork(job1, a1);
job2(a2);
join tid1;
```

→ Threads POSIX

- Modèle *future* :

```
v = future(job1(a1));
… = …v…;
```

→ tâches (*tasks*) OpenMP 3.0 (2008), Cilk

→ boucles non bornées, algorithmes récursifs, schéma producteur/consommateur...

OpenMP tasks

(d'après A. Duran, *Tasking in OpenMP*)

- Pourquoi des tâches ?
  - Exemple : parcours de liste en parallèle avec OpenMP 2.5

```
void parcours_liste(liste l){
    element e = l → first;
    #pragma omp parallel firstprivate (e)
    while (e != NULL){
        #pragma omp single nowait
        traiter (e) ;
        e = e → next;
    }
}
```

- Peu naturel
- Mauvaises performances
- Pas composable

OpenMP tasks

- Pourquoi des tâches ? (suite)
  - Autre exemple : parcours d'arbre en parallèle avec OpenMP 2.5

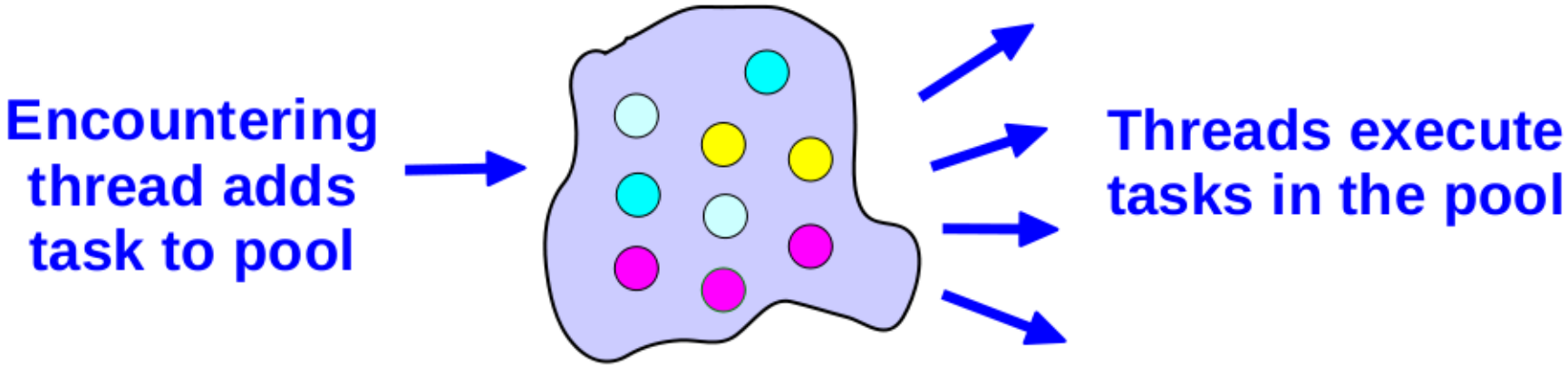
```
void parcours_arbre (arbre  *a){
#pragma omp parallel sections
  {
    #pragma omp section
    if (a → gauche) parcours_arbre ( a → gauche) ;
    #pragma omp section
    if ( a → droit) parcours_arbre ( a → droit) ;
  }
   traiter (a) ;
}
```

  - Trop de régions parallèles : surcoûts, synchronisations supplémentaires, pas toujours bien supporté par l'implémentation ...

OpenMP tasks

- Directive ***task*** :
  #pragma omp task [clauses]
  structured block
- Chaque thread qui « rencontre » la directive ***task*** crée une tâche/task → **exécution immédiate ou différée** par 1 des threads de la *team* (= ensemble des threads exécutant une région parallèle OpenMP)
- Hautement composable : imbrication possible dans
  - des régions parallèles
  - d'autres tâches
  - des directives « omp for », « omp sections », « omp single »

« OpenMP tasks » en image



Developer specifies tasks in application
Run-time system executes tasks

(An Overview of OpenMP 3.0, R. van der Pas, IWOMP2009)

OpenMP tasks : portée des variables et synchronisations

- Privilégier : ***firstprivate*** (attention aux pointeurs...)
- Attention aux variables (non *firstprivate*) dans la pile :
  - Le tâche mère ne doit pas les rendre invalides avant que les tâches filles ne s'en servent...
  - Solutions possibles :
    - Utiliser *firstprivate*
    - Déplacer les variables dans le tas (mais quand les « désallouer » ?)
    - Introduire des synchronisations
- Barrières OpenMP (**#pragma omp barrier**) → toutes les tâches créées par un thread de la team courante sont terminées à la sortie de la barrière
- Barrière de tâche : **#pragma omp taskwait**

  → La tâche courante attend alors la complétion de toutes ses tâches *filles* (seulement « filles directes », pas descendants).

Parcours de liste en OpenMP 3.0

```
void parcours_liste(liste l){
   element e = l → first;
   while (e != NULL){
      #pragma omp task firstprivate(e)
      traiter (e) ;
      e = e → next;
  }
#pragma omp taskwait // garantie ici la terminaison du parcours
}
```

```
liste l;
#pragma omp parallel
  parcours_liste(l);
```

```
liste l;
#pragma omp parallel
#pragma omp single
  parcours_liste(l);
```

```
liste l[N];
#pragma omp parallel
#pragma omp for
for (i=0;i<N;i++)
  parcours_liste(l[i]);
```

Optimisation : clause *untied*

- Par défaut : les tâches sont ***tied*** (liées)
  - Une tâche liée est toujours exécutée par le même thread
- Conséquence : performance/ordonnancement parfois non optimale
  - Exemple : parcours récursifs → déséquilibrage de charge
- Clause ***untied*** → la tâche n'est plus liée à 1 thread donné, mais alors
  - Éviter les variables *threadprivate*
  - Éviter l'utilisation de l'indice du thread dans les calculs de la tâche
  - Faire très attention aux verrous et sections critiques → interblocages possibles...

Optimisation : grain des tâches

- Facteur clé pour la performance :
  - Regrouper des tâches « fines » :
    - Clause ***if***
      #pragma omp task if (prof < PROF_MAX)
    - Instruction ***if*** :

```
void f(..., int prof){
…
#pragma omp task
 {
    ...
    if (prof < PROF_MAX)
                f(..., prof+1)
    else
                f_sequentielle(...)
 }
}
```