

CNAM de Lorraine
Applications concurrentes : conception et outils (ACCOV/NFP103)

5 - Ouverture : OpenMP

Stéphane Vialle
Stephane.Vialle@supelec.fr
http://www.metz.supelec.fr/~vialle

Programmation OpenMP (par partage de mémoire)

1. Principes de programmation en OpenMP
2. Exemples de parallélisation
3. Mesures de performances
4. Bilan d'OpenMP

Programmation OpenMP

1 - Principes de programmation

1. Principes de base
2. Syntaxe et sémantique
3. Optimisations

Programmation OpenMP - principes

Principes de base

Développer un code séquentiel :

```
Initialisation();
for (i=0; i<N; i++)
    Calcul(i);
Autre_calcul();
```

Ajouter des directives de compilation parallèles

- parallélisation incrémentale
- peu de code supplémentaire
- limité au parallélisme présent dans le code initial

```
Initialisation();
#pragma omp parallel for private(i)
for (i=0; i<N; i++)
    Calcul(i);
Autre_calcul();
```

Programmation OpenMP - principes

Principes de base

Parallélisation par directives

SIMD & Vectoriel	MIMD	MIMD - ShM
1980	1993	1998 2002
Des sols existent	Echec!	OpenMP-1 OpenMP-2

Aujourd'hui : OpenMP est intégré à de nombreux compilateurs C++

OpenMP : basé sur des threads

- nécessite une mémoire partagée
- pb de synchronisation
- pb de false sharing
- pb de contention
- pb de ShM

Programmation OpenMP - principes

Principes de base

OpenMP-1 vs OpenMP-2

OpenMP-1	OpenMP-2
- 1998	- 2002
- Régions parallèles	- Régions parallèles
- Parallel for	- Parallel for
- Parallel sections avec des « options »	- Parallel sections ... avec de nouvelles « options »
- Peu de composition possible des directives	- Beaucoup de compositions possibles des directives
- Simple	- Devient (très) compliqué !

OpenMP-2 utilisé au maximum est-il plus simple que les P-Threads ?

Programmation OpenMP

1 - Principes de programmation

1. Principes de base
 2. **Syntaxe et sémantique**
 3. Optimisations

Programmation OpenMP - principes

Syntaxe et sémantique

```

main() {
  .....
  #pragma omp parallel
  {
    .....
    #pragma omp sections
    {
      #pragma omp section
      { ..... }
      #pragma omp section
      { ..... }
    }
  }
  .....
}

```

Sequential code
 Replicated code
 Parallel and different units of work
 Sequential code

Programmation OpenMP - principes

Syntaxe et sémantique

```

main() {
  #pragma omp parallel
  {
    #pragma omp for private(i)
    for (i=0; i<N; i++)
      .....
  }
  #pragma omp single
  { ..... }
  #pragma omp critical
  { ..... }
  #pragma omp barrier
  .....
}

```

Work repartition

Programmation OpenMP - principes

Syntaxe et sémantique

Directives « orphelines » vs « région parallèle »

```

main() {
  #pragma omp parallel
  {
    #pragma omp for private(i)
    for (i=0; i<N; i++)
      .....
  }
}

```

Région parallèle explicite
 Peut s'écrire plus simplement sans région parallèle explicite :

```

main() {
  int i;
  #pragma omp parallel for private(i)
  for (i=0; i<N; i++)
    .....
}

```

Plus simple et compact, mais parfois moins efficace (voir plus loin) !

Programmation OpenMP - principes

Syntaxe et sémantique

Variables « shared » et « private » :

```

#define N 1000

double Tab1[N], Tab2[N];

main() {
  int i;
  double alpha = 2.0;
  .....
  #pragma omp parallel for private(i) \
    shared(Tab1,Tab2) \
    shared(alpha)
  for (i=0; i<N; i++) {
    double coef = pow(Tab1[i],alpha);
    Tab2[i] *= coef;
  }
}

```

Voir aussi `firstprivate` et `lastprivate`

Programmation OpenMP - principes

Syntaxe et sémantique

Reduction :

Le « `parallel for` » permet aussi de faire une réduction

```

#pragma omp parallel for private(i) \
  shared(x, y, n) \
  reduction(+: a, b)

for (i=0; i<n; i++) {
  a = a + x[i];
  b = b + y[i];
}

```

Finalement la somme des `x[i]` est disponible dans `a`, et la somme des `y[i]` dans `b` : deux variables scalaires.

Programmation OpenMP - principes

Syntaxe et sémantique

OpenMP possède aussi quelques fonctions :

Execution Environment Functions

omp_set_num_threads	omp_in_parallel
omp_get_num_threads	omp_set_dynamic
omp_get_max_threads	omp_get_dynamic
omp_get_thread_num	omp_set_nested
omp_get_num_procs	omp_get_nested

Lock Functions

omp_init_nest_lock	omp_init_lock
omp_destroy_nest_lock	omp_destroy_lock
omp_set_nest_lock	omp_set_lock
omp_unset_nest_lock	omp_unset_lock
omp_test_nest_lock	omp_test_lock

Timing Routines

omp_get_wtime
omp_get_wtick

Programmation OpenMP

1 - Principes de programmation

1. Principes de base
2. Syntaxe et sémantique
3. Optimisations

Programmation OpenMP - principes

Optimisations

Définition d'une région parallèle : évite de re-crée les threads

```

main() {
    for (c=0; c<NbCycle; c++){
        #pragma omp parallel for private(i)
        for (i=0; i<N; i++)
            .....
    }
}

main() {
    #pragma omp parallel private(c)
    {
        for (c=0; c<NbCycle; c++){
            #pragma omp for private(i)
            for (i=0; i<N; i++)
                .....
        }
    }
}

```

Programmation OpenMP - principes

Optimisations

Réglage du *scheduling* des `#pragma omp for` :

- taille des tâches : nombre d'itérations de boucle prises en charge
- type de répartition des tâches : `static`, `dynamic`, ...

Calculs réguliers équilibrés

Calculs irréguliers déséquilibrés

Programmation OpenMP - principes

Optimisations

Suppression des barrières de synchronisation implicites :

```

#pragma omp sections nowait
#pragma omp for nowait

```

Limitation du nombre de *threads* créés :

```

omp_set_num_threads(#val);
#pragma omp sections
{
    #pragma omp section
    .....
    #pragma omp section
    .....
}

```

Ex: 3 *threads* créés
2 *threads* utilisés

Programmation OpenMP

2 – Exemples de parallélisation

1. Produit de matrices
2. Relaxation de Jacobi
3. Tri à bulles
4. Quick-sort

Programmation OpenMP - exemples

Parallélisation du produit de matrices

Principe et implantation du produit de matrices carrées denses

- Parallélisation très simple : distribution de la boucle externe (boucle sur les lignes de la matrice)
- Ne pas oublier les déclarations des variables « **private** » :

```

...
#pragma omp parallel for private(i,j,k,temp)
for (i=0; i<N; i++) {
    for (j = 0; j<N; j++) {
        temp = 0.0;
        for (k=0; k<N; k++)
            temp += A[i][k] * B[k][j];
        C[i][j] += temp;
    }
}
...

```

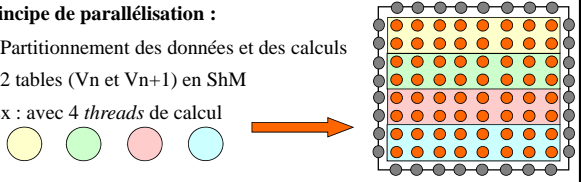
Programmation OpenMP - exemples

Parallélisation de la relaxation de Jacobi

Principe de parallélisation :

- Partitionnement des données et des calculs
- 2 tables (V_n et V_{n+1}) en ShM

Ex : avec 4 *threads* de calcul



Implantation OpenMP :

- Parallélisation de la boucle sur les lignes
- Utilisation d'une région parallèle plus globale (plus efficace)

```

#pragma omp parallel private(c,...)
{
    for (c=0; c<NbCycle; c++){
        #pragma omp for private(i,j)
        for (i=0; i<N; i++)
            for (j=0; j<N; j++)
                V[i][j] = ... ..
    }
}
.....

```

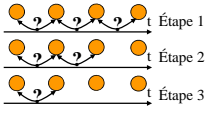
Programmation OpenMP - exemples

Parallélisation du bubble-sort

Principe du bubble-sort « odd-even » :

Pb : l'algorithme de base du *bubble-sort* est fortement séquentiel !

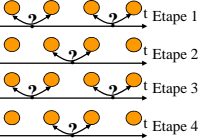
→ Modifier l'algorithme pour qu'il contienne du parallélisme potentiel !



↓

→ Bubble-sort « odd-even » :

- Même complexité
- Comparaisons indépendantes



Programmation OpenMP - exemples

Parallélisation du bubble-sort

Implantation du bubble-sort « odd-even »

- Définition d'une région globale (+ efficace)
- Parallélisations distinctes des 2 boucles de tri.

```

#pragma omp parallel private(step)
{
    for (step = N; step > 0; step--) {
        if (step % 2 == 0) {
            #pragma omp for private(i, buff)
            for (i = 0; i < N-1; i += 2)
                if (Tab[i] > Tab[i+1]) {
                    buff = Tab[i];
                    Tab[i] = Tab[i+1];
                    Tab[i+1] = buff;
                }
        } else {
            #pragma omp for private(i, buff)
            for (i = 1; i < N-1; i += 2)
                if (Tab[i] > Tab[i+1]) {
                    buff = Tab[i];
                    Tab[i] = Tab[i+1];
                    Tab[i+1] = buff;
                }
        }
    }
}

```

Programmation OpenMP - exemples

Parallélisation du bubble-sort

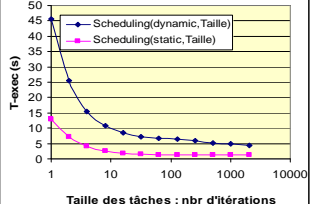
Tentative d'optimisation de l'implantation OpenMP

- Étapes de calcul régulières : « comparaisons-échanges »
- Mais contenus différents

```

if (Tab[i] > Tab[i+1])
    exchange(i, i+1)
else
    rien!

```



- Essais d'optimisation :
 - scheduling dynamique
 - granularité des tâches
- Tests sur SGI-VWS-540 KAI-OpenMP / WNT
- Pas d'améliorations pour cette application.

Programmation OpenMP - exemples

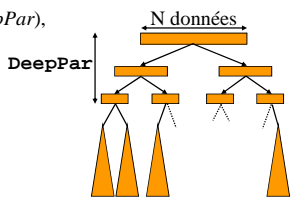
Parallélisation du quick-sort

Principe de parallélisation simple du Quick-Sort

Appels récursifs → Créations récursives de *sections* parallèles

→ « *nested parallelism* » nécessaire

→ limiter la profondeur de la parallélisation (variable *DeepPar*), sinon trop de *threads*!

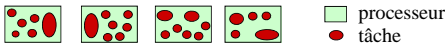


Programmation OpenMP - exemples

Parallélisation du quick-sort

Optimisation :

- Pb d'équilibrage de charge :
 - selon les pivots les *sections* sont plus ou moins importantes
 - solution possible : **équilibrage statistique !**
Nb Sections >> Nb Processeurs



• Il faut donc créer beaucoup plus de *threads* qu'il n'y a de processeurs mais sans saturer les processeurs de *threads* !

Programmation OpenMP – exemples

Parallélisation du quick-sort

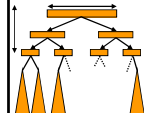
Implantation d'une parallélisation simple du Quick-Sort

- Test & sections (OpenMP-1)

ou bien :

- Sections conditionnelles (OpenMP-2)

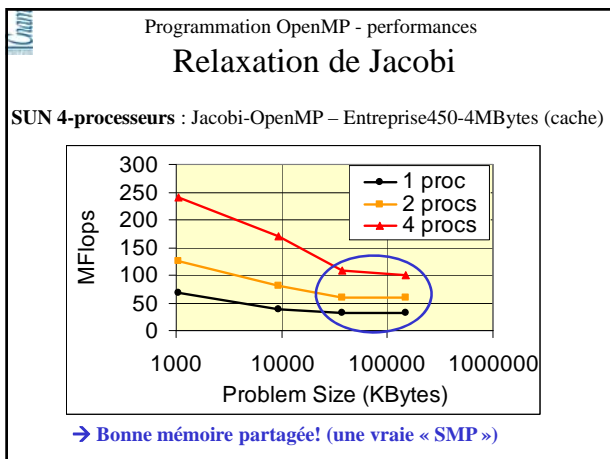
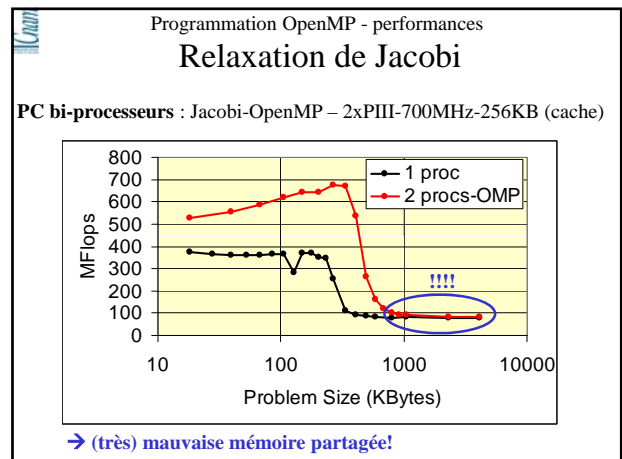
```
void quicksort(int q, int r, int deep)
{
    if (q < r) {
        // split the table ... classic quick-sort op
        if (deep < DEEPPAR) {
            #pragma omp parallel
            {
                #pragma omp sections nowait
                {
                    #pragma omp section
                    quicksort(q,s-1,deep+1);
                    #pragma omp section
                    quicksort(s+1,r,deep+1);
                }
            }
        }
        else {
            quicksort(q,s-1,deep+1);
            quicksort(s+1,r,deep+1);
        }
    }
}
```



Programmation OpenMP

3 – Mesures de performances

1. Relaxation de Jacobi
2. Bubble-sort
3. Quick-sort



Programmation OpenMP - performances

Bubble-sort

Performances du bubble-sort en OpenMP:

Sur PC 4-processeurs (SGI-VWS540) – 4xPII-Xeon-450MHz :

$S(4) = 3.7$, $e(4) > 92\%$

→ Simple et efficace !

→ pas de problème d'accès à la mémoire partagée ... !

Programmation OpenMP - performances

Quick-sort

Performances du quick-sort en OpenMP :

Sur PC 4-processeurs (SGI-VWS540) – 4xPII-Xeon-450MHz :

Nested parallelism : non disponible en 2001

- Eclatement de l'arbre de tri sur une profondeur de 1!
- **Utilisation de 2 processeurs seulement en 2001!**

$S(2) = [1.35 ; 1.84], \quad e(2) = [67\% ; 92\%]$

En 2004 : *nested parallelism* disponible en OpenMP-2

Il existe des algorithmes de quick-sort parallèle beaucoup plus Efficace (voir chapitre sur les tris parallèles)

Programmation OpenMP

4 – Bilan de la programmation en OpenMP

Programmation OpenMP

Bilan

Deux démarches possibles (très différentes) :

- récupérer et compléter les algorithmes et les codes séquentiels
- concevoir de nouveaux algorithmes parallèles et pour OpenMP

Expérimentations :

<p>Relaxation de Jacobi :</p> <ul style="list-style-type: none"> • $S(4) = 3.9, \quad e(4) > 98\%$ • Récupération de code séquentiel
<p><i>Quick-Sort</i> :</p> <ul style="list-style-type: none"> • $S(2) = [1.35 ; 1.84], \quad e(2) = [67\% ; 92\%]$ • Récupération de code séquentiel, résultats incertains
<p><i>Bubble-Sort</i> :</p> <ul style="list-style-type: none"> • $S(4) = 3.7, \quad e(4) > 92\%$ • Conception et implantation d'un nouvel algorithme

Programmation OpenMP

Bilan

Stratégies :

I - Algorithmes réguliers contenant un fort parallélisme naturel :

- conserver l'algorithme classique
- compléter l'implantation séquentielle

→ T_{dev} faible, Performances élevées

OK

II - Algorithmes irréguliers ou contenant peu de parallélisme naturel :

conservation de l'algorithme
ajouts au code séquentiel

→ T_{dev} faible, Perfs moyennes

conception d'un nouvel algo
nouvelle implantation

→ T_{dev} élevé, Perfs élevées

OpenMP ne dispense pas de connaître l'algorithmique parallèle !

Programmation OpenMP

FIN