






CentraleSupélec  

SG6: High Performance Computing

## Multithreading with OpenMP

**Stéphane Vialle**

 universit  Paris-Saclay   Sciences et technologies de l'information et de la communication (STIC)

Stephane.Vialle@centralesupelec.fr  
<http://www.metz.supelec.fr/~vialle>

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

## Programmation OpenMP (par partage de m moire)

1. Principes d'OpenMP
2. D tails de syntaxe et de s mantique
3. Optimisations
4. Exemples de parall lisation
5. Bilan d'OpenMP
6. Mesures et repr sentation de performances

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

## Principes de base

**D velopper un code s quentiel :**

- conception
- implantation
- mise au point

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

↓

**Ajouter des directives de compilation parall les**

- **parall lisation incr mentale**
- peu de code suppl mentaire
- limit  au parall lisme pr sent dans le code initial

Exemple utilisant une « directive » OpenMP

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

---

---

---

---

---

---

---

---

---

---

Programmation OpenMP - principes

## Principes de base

**OpenMP : basé sur des *threads* et soumis aux mêmes limites**

Exploite une mémoire partagée, mais :

- pb de synchronisation
- pb de contention
- pb de *false sharing* (« *cache war* »)

→ Utiliser OpenMP facilite (souvent) l'implantation, mais ne dispense pas de résoudre les problèmes d'algorithmique parallèle en mémoire partagée.

---

---

---

---

---

---

---

---

Programmation OpenMP - principes

## Régions parallèles

```
main() {
  .....
  #pragma omp parallel
  {
    .....
  }
  .....
}
```

1th : code seq.

3th : code répliqué

1th : code seq.

**Ex sur une machine à 3 cœurs (!!) :**

- Création et destruction de 3 threads en début et fin de région parallèle
- « Join » (synchro) automatique sur la mort des 3 threads en sortie de région parallèle
- Syntaxe très simple et concise

---

---

---

---

---

---

---

---

Programmation OpenMP - principes

## Ctrl du parallélisme par directives

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

Code seq.

Code répliqué

Calculs répartis de même nature

Calcul seq.

Calculs « mutexés »

Code répliqué à durée variable -- Synchro -- Code répliqué

Code seq.

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

### Ctrl du parallélisme par directives

```

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

```

Code seq.

Code répliqué

Calculs répartis de natures différentes

Code répliqué

Code seq.

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

### Ctrl du parallélisme par directives

```

main() {
  .....
  #pragma omp parallel région parallèle
  {
    #pragma omp sections
    {
      #pragma omp section
      {
        while (...) {
          .....
          if (...)
            #pragma omp task
            { ..... }
        }
      }
      #pragma omp section
      {
        Ex: IO operations
        Write into a file...
        .....
      }
    }
  }
  .....
}

```

Code seq.

threads tasks

Calculs répartis : ensemble de « tâches » traitées au mieux par un ensemble de threads

Code seq.

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

### Ctrl du parallélisme par directives

```

main() {
  .....
  #pragma omp parallel région parallèle
  {
    #pragma omp sections
    {
      #pragma omp section
      {
        while (...) {
          .....
          if (...)
            #pragma omp task
            { ..... }
        }
      }
      #pragma omp section
      {
        Ex: IO operations
        Write into a file...
        .....
      }
    }
  }
  .....
}

```

Code seq.

threads tasks

Calculs répartis : ensemble de « tâches » traitées au mieux par un ensemble de threads

Code seq.

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

## Ctrl du parallélisme par directives

```

main() {
    .....
    #pragma omp parallel
    {
        #pragma omp sections
        {
            #pragma omp section
            {
                write(i,j);
                #pragma omp task
                .....
            }
            #pragma omp section
            {
                Ex: IO operations
                Write into a file..
                .....
            }
        }
    }
    .....
}

```

Code seq.

région parallèle

threads tasks

Calculs répartis : ensemble de « tâches » traitées au mieux par un ensemble de threads

Code seq.

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

## Ctrl du parallélisme par directives

```

main() {
    .....
    #pragma omp parallel
    {
        #pragma omp sections
        {
            #pragma omp section
            {
                write(i,j);
                #pragma omp task
                .....
            }
            #pragma omp section
            {
                Ex: IO operations
                Write into a file..
                .....
            }
        }
    }
    .....
}

```

Code seq.

région parallèle

threads tasks

Calculs répartis : ensemble de « tâches » traitées au mieux par un ensemble de threads

Code seq.

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

## Ctrl explicite du parallélisme

Parallélisation d'un appel de fonction séquentielle :

```

main() {
    .....
    f_lib(0, N, SharedTable);
    .....
}

```

Code parallèle

```

main() {
    .....
    #pragma omp parallel
    {
        // Lower boundary of the thread
        int inf = N/omp_get_num_threads() * omp_get_thread_num();
        // Upper boundary of the thread
        int sup = N/omp_get_num_threads() * (omp_get_thread_num() + 1);
        // Call to the sequential library function
        f_lib(inf, sup, SharedTable);
    }
    .....
}

```

Code séquentiel

omp\_get\_num\_threads() : nb of threads in the current region  
omp\_get\_thread\_num() : rank of the thread

Code répliqué MAIS avec des paramètres spécifiques à chaque thread

Rmq : il faut que le code de la fonction soit « ré-entrant »

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

## Ctrl explicite du parallélisme

```
main() {
  .....
  #pragma omp parallel
  {
    switch (omp_get_thread_num()) {
    case 0 :
      ..... // calcul sur le GPU
      break;
    default :
      ..... // calcul sur les cœurs CPU
      break;
    }
  }
  .....
}
```

Hyp : 3 threads OpenMP créés sur une machine à 3 cœurs CPU...

...et l'un des cœurs CPU est dédié au pilotage d'un GPU

Faire accomplir au thread 0 (qui existe toujours) une tâche spéciale

- Pilotage d'un GPU en parallèle de calculs sur les autres cœurs CPU
- IO disque en parallèle de calculs sur les autres cœurs CPU...

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

## Région parallèles de directives orphelines

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

Région parallèle explicite

Région parallèle d'une directive « orpheline »

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

**Directive orpheline : installe sa propre région parallèle minimale**

- plus simple et plus compacte
- **attention** : provoque la création et la mort automatique de threads à chaque exécution de la directive  
→ possible pb de perf si on répète/enchaîne ces directives

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

## Programmation OpenMP (par partage de mémoire)

1. Principes d'OpenMP
2. Détails de syntaxe et de sémantique
3. Optimisations
4. Exemples de parallélisation
5. Bilan d'OpenMP
6. Mesures et représentation de performances

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

## Parallel for

OpenMP peut paralléliser des boucles de calcul :

- avec une syntaxe légère
- si les itérations sont indépendantes !

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

```
for (i = 1; i < N; i++) {
  tab2[i] = 1 + (tab1[i-1])2
}
```

Algorithme parallélisable

```
for (i = 1; i < N; i++) {
  tab[i] = 1 + (tab[i-1])2
}
```

Algorithme séquentiel !

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

## Parallel for

OpenMP peut paralléliser des boucles de calcul :

- avec une syntaxe légère
- si les itérations sont indépendantes !
- chaque thread créé devra posséder SES propres compteurs de boucles

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

Chaque thread à **SA plage de valeur** pour son compteur (SES itérations) et chaque thread **incrémente SON** compteur.

**private(i)** précise que le compteur global *i* devra être remplacé par des compteurs locaux *i* dans chaque corps de thread

Le *i* local masquera alors le *i* global dans la boucle de chaque thread.

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

## Parallel for

OpenMP peut paralléliser des boucles de calcul :

- avec une syntaxe légère
- si les itérations sont indépendantes !
- chaque thread créé devra posséder SES propres compteurs de boucles

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

On peut aussi préciser la clause **private(i)** dès la création de la région parallèle

Le *i* local masque alors le *i* global dans tous le corps de chaque thread

---

---

---

---

---

---

---

---

---

---

CentralesSupélec

Programmation OpenMP - principes

## Parallel for

OpenMP peut paralléliser des boucles de calcul :

- avec une syntaxe légère
- si les itérations sont indépendantes !
- chaque thread créé devra posséder SES propres compteurs de boucles

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

On peut enfin déclarer des compteurs dans les boucles...  
... qui deviennent automatiquement privés à chaque thread créé !

---

---

---

---

---

---

---

---

---

---

CentralesSupélec

Programmation OpenMP - principes

## Parallel for

### Variables privées et partagées dans les threads

```
#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;
        coef = pow(Tab1[i], alpha);
        Tab2[i] *= coef;
    }
}
```

Chaque thread possèdera une variable locale « i » qui masquera la variable initiale

Comportement par défaut

Variable locale à la boucle parallélisée → chaque thread en possèdera une

Les compteurs de boucles déclarés en amont doivent devenir privés.

Le compilateur OpenMP peut le faire sur des codes simples!...  
...**mais non portable!!** → **privatiser explicitement les compteurs**

---

---

---

---

---

---

---

---

---

---

CentralesSupélec

Programmation OpenMP - principes

## Parallel for

### Variables privées et partagées dans les threads

```
#define N 1000
double Tab1[N], Tab2[N];
main() {
    double alpha = 2.0;
    .....
    #pragma omp parallel for shared(Tab1, Tab2) \
    shared(alpha)
    for (int i = 0; i < N; i++) {
        double coef; // var locale : privée
        coef = pow(Tab1[i], alpha);
        Tab2[i] *= coef;
    }
}
```

Variable locale à la boucle parallélisée → chaque thread possèdera son compteur de boucle

Les compteurs de boucles déclarés dans les boucles sont automatiquement privés à chaque thread

→ **La meilleure solution !**

---

---

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

## Parallel for

Gestion des compteurs des **nids de boucles** :

- $i$  aura une plage de valeurs différente dans chaque thread  
→ chaque thread doit avoir **SON  $i$**
- $j, k$  et  $l$  auront les mêmes plages de variation, mais :
  - chaque thread avance à son rythme
  - chaque thread incrémente ses compteurs
  - chaque thread doit avoir SES  $j, k, l$ .

```
double sharedTable[I][J][K][L];
main() {
  int i,j,k,l;
  .....
  #pragma omp parallel
  {
    .....
    #pragma omp for private(i,j,k,l)
    for (i=0; i<I; i++)
      for (j=0; j<J; j++)
        for (k=0; k<K; k++)
          for (l=0; l<L; l++) {
            f(i,j,k,l,sharedTab);
          }
    .....
  }
}
```

---

---

---

---

---

---

---


---

CentralesSupélec Programmation OpenMP - principes

## Parallel for

Gestion des compteurs des **nids de boucles** :

- Finalement, avec des déclarations de **compteurs locaux**...  
...tout est beaucoup plus simple!



```
double sharedTable[I][J][K][L];
main() {
  .....
  #pragma omp parallel
  {
    .....
    #pragma omp for
    for (int i=0; i<I; i++)
      for (int j=0; j<J; j++)
        for (int k=0; k<K; k++)
          for (int l=0; l<L; l++) {
            f(i,j,k,l,sharedTab);
          }
    .....
  }
}
```

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

## Parallel for

Exemple de gestion de différents types de variables

```
#define NbC 100
#define N 1000
double Tab[2][N];
main() {
  int i, c;
  int idx = 0;
  double a = 2.0;
  for (c = 0; c < NbC; c++){
    for (i = 0; i < N; i++){
      double coef =
        pow(Tab[idx][i],a);
      Tab[1-idx][i] *= coef;
    }
    idx = 1 - idx;
  }
  printf("%d",idx);// 0 ou 1
}

#define NbC 100
#define N 1000
double Tab[2][N];
main() {
  int i, c;
  int idx = 0;
  double a = 2.0;
  #pragma omp parallel
  private(i,c,idx) \
  shared(Tab,a)
  {
    idx = 0; // NE PAS OUBLIER
    for (c = 0; c < NbC; c++){
      #pragma omp for
      for (i = 0; i < N; i++){
        double coef =
          pow(Tab[idx][i],a);
        Tab[1-idx][i] *= coef;
      }
      idx = 1 - idx;
    }
  }
  printf("%d",idx) // 0
}
```

---

---

---

---

---

---

---

---



CentralesSupélec Programmation OpenMP - principes

## Parallel for

Exemple de gestion de différents types de variables

```

#define NbC 100
#define N 1000
double Tab[2][N];
main() {
  int i, c;
  int idx = 0;
  double a = 2.0;
  for (c = 0; c < NbC; c++){
    for (i = 0; i < N; i++){
      double coef =
        pow(Tab[idx][i], a);
      Tab[1-idx][i] *= coef;
    }
    idx = 1 - idx;
  }
  printf("%d", idx); // 0 ou 1
}

```

```

#define NbC 100
#define N 1000
double Tab[2][N];
main() {
  int i, c;
  int idx = 0;
  double a = 2.0;
  #pragma omp parallel
  private(i,c)
  shared(Tab,a,idx)
  {
    for (c = 0; c < NbC; c++){
      #pragma omp for
      for (i = 0; i < N; i++){
        double coef =
          pow(Tab[idx][i], a);
        Tab[1-idx][i] *= coef;
      }
      #pragma single
      { idx = 1 - idx; }
    }
  }
  printf("%d", idx); // 0 ou 1
}

```

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

## Parallel for

Propagation de valeurs aux variables privées : clause *firstprivate*

```

#define NbC 100
#define N 1000
double Tab[2][N];
main() {
  int i, c;
  int idx = 0;
  double a = 2.0;
  for (c = 0; c < NbC; c++){
    for (i = 0; i < N; i++){
      double coef =
        pow(Tab[idx][i], a);
      Tab[1-idx][i] *= coef;
    }
    idx = 1 - idx;
  }
  printf("%d", idx); // 0 ou 1
}

```

```

#define NbC 100
#define N 1000
double Tab[2][N];
main() {
  int i, c;
  int idx = 0;
  double a = 2.0;
  #pragma omp parallel
  private(i,c)
  firstprivate(idx)
  shared(Tab,a)
  {
    for (c = 0; c < NbC; c++){
      #pragma omp for
      for (i = 0; i < N; i++){
        double coef =
          pow(Tab[idx][i], a);
        Tab[1-idx][i] *= coef;
      }
      idx = 1 - idx;
    }
  }
  printf("%d", idx); // 0
}

```

Voir aussi « lastprivate »

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - principes

## Parallel reduce

Def : une « réduction » passe d'un vecteur de dimension  $n$  à un vecteur de dimension  $m$  ( $< n$ )

- c'est toujours une opération délicate en parallélisme
- en OpenMP : *parallel for + clause reduction*
- ex : réduction d'un vecteur dans un scalaire

```

#define N 1000
double x[N];
double Sx = 0.0;
double MoyX;
#pragma omp parallel for shared(x) \
reduction(+: Sx, Sy)
for (int i = 0; i < N; i++) {
  Sx = Sx + x[i]; // local Sx
}
// global Sx = sum of all local Sx
MoyX = Sx/N; // global Sx

```

Cohérence de l'opérateur et des opérations : à la charge du développeur

---

---

---

---

---

---

---

---

CentraleSupélec Programmation OpenMP - principes

## Parallel reduce

**Def :** une « réduction » passe d'un vecteur de dimension  $n$  à un vecteur de dimension  $m$  ( $< n$ )

- OpenMP peut **réduire dans un vecteur** a une ou plusieurs dimensions

```
double Mark[Ncourse][Nstudent];
double Weight[Ncourse];
double Score[Nstudent];
#pragma omp parallel for reduction(+: Score)
for (int c = 0; c < Ncourse; c++)
  for (int s = 0; s < Nstudent; s++)
    Score[s] += Mark[c][s]*Weight[c];
```

OpenMP récents

Le tableau `Mark[] []` est accédé dans un ordre favorable au cache  
 → on conserve cet ordre

Threads plus « gros » si on parallélise la boucle en `c` plutôt que celle en `s`  
 → on tente de paralléliser la boucle en `c`

→ Il faut faire une **réduction dans le tableau `Score`**  
 → Une tableau `Score` dans chaque thread : peut excéder la RAM !

---

---

---

---

---

---

---

---

---

---

CentraleSupélec Programmation OpenMP - principes

## Contrôle de l'ampleur du parallélisme

Par défaut OpenMP *inonde* le nœud : un thread par cœur logique !  
 Mais on peut spécifier le nbr de threads créés

**Sol 1 :** spécifier le nbr de threads à chaque entrée en région parallèle

```
#pragma omp parallel num_threads(10)
{
  .....
}
.....
#pragma omp parallel for num_threads(10)
for ...
.....
#pragma omp parallel for num_threads(2)
for ...
```

Impose le nombre de threads créés **juste sur une portion de code** OpenMP  
 Commande à re-spécifier à chaque région parallèle (non robuste)

---

---

---

---

---

---

---

---

---

---

CentraleSupélec Programmation OpenMP - principes

## Contrôle de l'ampleur du parallélisme

Par défaut OpenMP *inonde* le nœud : un thread par cœur logique !  
 Mais on peut spécifier le nbr de threads créés

**Sol 2 :** spécifier le nbr de threads jusqu'à une nouvelle spécification

```
omp_set_num_threads(10);
#pragma omp parallel {
  .....
}
.....
#pragma omp parallel for
for ...
.....
omp_set_num_threads(2);
#pragma omp parallel for
for ...
```

Impose le nombre de threads :  
 • pour le reste du programme  
 • jusqu'à une nouvelle spécification

---

---

---

---

---

---

---

---

---

---

CentraleSupélec Programmation OpenMP - principes

## Contrôle de l'ampleur du parallélisme

Par défaut OpenMP *inonde* le nœud : un thread par cœur logique !  
Mais on peut spécifier le nbr de threads créés

Sol 3 : en mixant les deux types de spécifications

```

omp_set_num_threads(10);
#pragma omp parallel {
  .....
}

#pragma omp parallel for
for ...

.....

#pragma omp parallel for num_threads(2)
for ...

```

Impose le nombre de threads pour tout le reste du programme  
Mais une nouvelle spécification locale l'efface momentanément

---

---

---

---

---

---

---

---

---

---

CentraleSupélec Programmation OpenMP - principes

## Qu'imprime ce programme ?

```

omp_set_num_threads(10);
printf("Thread %d/%d\n",
      omp_get_thread_num(),
      omp_get_num_threads());

#pragma omp parallel num_threads(2)
{
  printf("Thread %d/%d\n",
        omp_get_thread_num(),
        omp_get_num_threads());

  #pragma omp for private(i,j)
  for (i = 0; i < 4; i++)
    for (j = 0; j < 2; j++)
      print("%d-%d-%d\n",
            omp_get_thread_num(), i, j);
}

printf("Thread %d/%d\n",
      omp_get_thread_num(),
      omp_get_num_threads());

```

Thread 0/1

Thread 0/2 ou Thread 1/2  
Thread 1/2 Thread 0/2

th = 0 :	th = 1
0-0-0	1-2-0
0-0-1	1-2-1
0-1-0	1-3-0
0-1-1	1-3-1

Thread 0/1

---

---

---

---

---

---

---

---

---

---

CentraleSupélec Programmation OpenMP - principes

## Mesure du temps écoulé

```

start = omp_get_wtime();
#pragma omp parallel \
  private(cycle,i,j)
{
  ..... // PARALLEL COMPUTATIONS
}
finish = omp_get_wtime();
duration = finish - start;
gigaflops = NbrOPs/duration/1E9;
.....

```

Début de la mesure du temps

Fin de la mesure de temps

Déduction de la vitesse de calcul

omp\_get\_wtime() :

- Portable sous Linux et Windows
- Mesure le temps qui passe (le « wall clock »)
- Est assez précise
- Est très facile à utiliser

---

---

---

---

---

---

---

---

---

---

CentraleSupélec Programmation OpenMP - principes

## Synchronisation par « locks » explicites

Exemple d'utilisation des fonctions d'OpenMP :

```

int compteur = 0;
omp_lock_t Lock;
omp_init_lock(&Lock);
#pragma omp parallel
{
.....
omp_set_lock(&Lock);
compteur++ // accès à une rsrc critique
omp_unset_lock(&Lock);
.....
}
.....
omp_destroy_lock (&Lock);

```

Déclaration d'un verrou  
Initialisation du verrou  
Utilisation du verrou  
Libération du verrou

Permet de réaliser une synchronisation par verrous, qui est portable entre Windows et Linux.

---

---

---

---

---

---

---

---

---

---

CentraleSupélec Programmation OpenMP - principes

## Fonctions disponibles en OpenMP

*Execution Environment Functions*

```

omp_set_num_threads
omp_get_num_threads
omp_get_max_threads
omp_get_thread_num
omp_get_num_procs

```

```

omp_in_parallel
omp_set_dynamic
omp_get_dynamic
omp_set_nested
omp_get_nested

```

*Lock Functions*

```

omp_init_nest_lock
omp_destroy_nest_lock
omp_set_nest_lock
omp_unset_nest_lock
omp_test_nest_lock

```

```

omp_init_lock
omp_destroy_lock
omp_set_lock
omp_unset_lock
omp_test_lock

```

*Timing Routines*

```

omp_get_wtime
omp_get_wtick

```

---

---

---

---

---

---

---

---

---

---

CentraleSupélec Programmation OpenMP - principes

## Evolution d'OpenMP

### OpenMP-1 vs OpenMP-2 (ou 2.5 ou 3.1 ou 4.0)

	OpenMP-1.0	OpenMP-2.0
- 1998	- 1998	- 2002
- Régions parallèles	- Régions parallèles	- Régions parallèles
- <b>Parallele for</b>	- <b>Parallele for</b>	- <b>Parallele for</b>
- <b>Parallele sections</b> avec des « options »	- <b>Parallele sections</b> ... nouvelles « options »	- <b>Parallele sections</b> ... nouvelles « options »
- Peu de composition possible des directives	- Beaucoup de compositions possibles des directives	- Beaucoup de compositions possibles des directives
- <b>Simple</b>	- <b>Devient plus compliqué !</b>	- <b>Devient plus compliqué !</b>

OpenMP-5.0  
OpenMP-4.0  
OpenMP-3.1  
OpenMP-2.5

De plus en plus de fonctionnalités ...

Documentation d'OpenMP...	OpenMP -1.0	OpenMP -2.0	OpenMP -2.5	OpenMP -3.0	OpenMP -3.1	OpenMP -4.0
	1998	2002	2005	2008	2011	2012
	85p	106p	250p	326p	354p	380p

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

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

## Programmation OpenMP (par partage de mémoire)

1. Principes d'OpenMP
2. Détails de syntaxe et de sémantique
- 3. Optimisations**
4. Exemples de parallélisation
5. Bilan d'OpenMP
6. Mesures et représentation de performances

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

### Réutilisation des threads

**Définition d'une région parallèle qui 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++)
      .....
    }
  }
}
```

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

### Suppression des synchro inutiles

**Suppression de barrières de synchronisation systématiques :**

```
#pragma omp sections nowait
#pragma omp for nowait
```

Par défaut : un **wait** est présent à la fin de chaque directive OpenMP  
→ **barrière de synchronisation** de tous les threads de la région

**On peut l'enlever en ajoutant **nowait** :**

- si les threads n'ont pas besoin d'être resynchronisés à cet endroit du code !
- si il y a matière à gagner du temps : enchaîner avec des calculs de tailles différentes et espérer gagner au final

---

---

---

---

---

---

---

---

CentraleSupélec  
Programmation OpenMP - principes

## Optimisation de l'ampleur du parallélisme

**Eviter les créations de threads inutiles :**

1 - Limitation au nombre de tâches à exécuter (< nbr de rsrce)

Ex: 4 cœurs disponibles

```

.....
omp_set_num_threads(2);
#pragma omp sections
{
  #pragma omp section
  {.....}
  #pragma omp section
  {.....}
}

```

Seulement 2 threads créés

2 - Optimisation du parallélisme selon le couple calcul/machine

```

#pragma omp parallel for num_threads(nth)
for (int i = 0; i < N; i++) {
  ..... // Accès données et calculs
}

```

Selon les calculs et les accès aux données, le nbr optimal de tâches peut varier d'une architecture à l'autre

---

---

---

---

---

---

---

---

---

---

CentraleSupélec  
Programmation OpenMP - principes

## Equilibrage de charge

**Paralléliser en équilibrant la charge**

Quelle boucle paralléliser dans ce « nid de boucles » ?

```

void ActStock(double sqrtDt)
{
  int StkIdx, yIdx, xIdx; // Loop indexes
  #pragma omp parallel private(StkIdx,yIdx,xIdx)
  {
    #pragma omp for
    for (StkIdx = 0; StkIdx < NbStocks; StkIdx++) {
      Parameters t *parPt = &par[StkIdx];
      for (yIdx = 0; yIdx < Ny; yIdx++) // Process each
        for (xIdx = 0; xIdx < Nx; xIdx++) { // "trajectory"
          float call;
          // - First pass
          call = ..... // Calcul de Monte Carlo
          // - The passes that remain
          for (int stock = 1; stock <= StkIdx; stock++)
            call = ..... // Calcul de Monte Carlo
          // Copy result in the global GPU memory
          TabStockCPU[StkIdx][yIdx][xIdx] = call;
        }
      }
    }
  }
}

```

On tente de paralléliser la boucle la plus externe (corps + conséquent)  
Mais les premières itérations de la boucle externe durent moins longtemps...

---

---

---

---

---

---

---

---

---

---

CentraleSupélec  
Programmation OpenMP - principes

## Equilibrage de charge

**Paralléliser en équilibrant la charge**

Quelle boucle paralléliser dans ce « nid de boucles » ?

```

void ActStock(double sqrtDt)
{
  int StkIdx, yIdx, xIdx; // Loop indexes
  #pragma omp parallel private(StkIdx,yIdx,xIdx)
  {
    for (StkIdx = 0; StkIdx < NbStocks; StkIdx++) {
      Parameters t *parPt = &par[StkIdx];
      #pragma omp for
      for (yIdx = 0; yIdx < Ny; yIdx++) // Process each
        for (xIdx = 0; xIdx < Nx; xIdx++) { // "trajectory"
          float call;
          // - First pass
          call = ..... // Calcul de Monte Carlo
          // - The passes that remain
          for (int stock = 1; stock <= StkIdx; stock++)
            call = ..... // Calcul de Monte Carlo
          // Copy result in the global GPU memory
          TabStockCPU[StkIdx][yIdx][xIdx] = call;
        }
      }
    }
  }
}

```

On tente de paralléliser la boucle la plus externe (corps + conséquent)  
Mais ici, pour un bon équilibrage de charge statique : il faut paralléliser la 2<sup>ème</sup> boucle (et non pas la première)

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

## Distributions équilibrées de boucles

Réglage du **scheduling** des **parallel for** :

`#pragma omp for private(...) schedule(mode, size) ...`

- mode de répartition : `static`, `dynamic`, `guided`, `auto`, `runtime`
- taille des tâches : nombre d'itérations traitées

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

Programmation OpenMP - principes

## Distributions équilibrées de boucles

Réglage du **scheduling** des **parallel for** :

`#pragma omp for private(...) schedule(mode, size) ...`

- mode de répartition : `static`, `dynamic`, `guided`, `auto`, `runtime`
- taille des tâches : nombre d'itérations traitées

**schedule (guided, chunk\_size) :**  
le nombre d'itérations de chaque tâche est fixé (sauf pour la dernière), mais pas le mode d'affectation des tâches (static/dyna.)

**schedule (auto) :**  
le découpage des boucles en tâches est délégué au compilateur

**schedule (runtime) :**  
le découpage des boucles en tâches et leur affectation aux threads, se fera à l'exécution seulement

Suppose un compilateur ou un middleware *intelligent*

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

## Programmation OpenMP (par partage de mémoire)

1. Principes d'OpenMP
2. Détails de syntaxe et de sémantique
3. Optimisations
4. Exemples de parallélisation
5. Bilan d'OpenMP
6. Mesures et représentation de performances

---

---

---

---

---

---

---

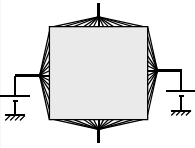
---

---

---

CentralesSupélec Programmation OpenMP - exemples

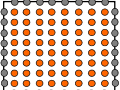
## Parallélisation de la relaxation de Jacobi



Calcul des lignes de potentiel dans une plaque diélectrique :

$$\frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0$$

- Discrétisation et équation aux différences
- Itération jusqu'à la convergence ( $V^{n+1}_{i,j} - V^n_{i,j} < \epsilon$ )



- Condition aux limites : V fixé
- $V^{n+1}_{i,j} = \frac{V^n_{i-1,j} + V^n_{i+1,j} + V^n_{i,j-1} + V^n_{i,j+1}}{4}$  « stencil »

---

---

---

---

---

---

---

---

---

---

CentralesSupélec Programmation OpenMP - exemples

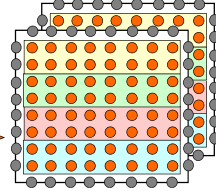
## Parallélisation de la relaxation de Jacobi

**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 :

- on donne une zone mémoire continue à chaque thread
- on partitionne la première dimension du tableau



**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++) {
    int current = c%2;
    int futur = (c+1)%2;
    #pragma omp for private(i,j)
    for (i = 0; i < N; i++)
      for (j = 0; j < N; j++)
        V[futur][i][j] = ... ..
  }
}

```

---

---

---

---

---

---

---

---

---

---

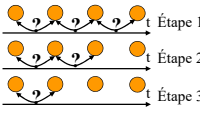
CentralesSupélec Programmation OpenMP - exemples

## Parallélisation du bubble-sort

**Principe du bubble-sort « odd-even » :**

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

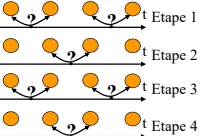
→ Modifier l'algorithme pour qu'il contienne du parallélisme potentiel !



↓

Bubble-sort « odd-even » :

- Même complexité
- Comparaisons indépendantes
- Parallélisable !




---

---

---

---

---

---

---

---

---

---



CentralesSupélec 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;
        }
    }
  }
}
  
```

---

---

---

---

---

---

---

---

---

---

CentralesSupélec 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*!

Algorithme parallèle peu efficace !  
 → Il existe des algorithmes de quick-sort parallèle beaucoup plus performants

---

---

---

---

---

---

---

---

---

---

CentralesSupélec 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

■ processeur  
● tâche

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

---

---

---

---

---

---

---

---

---

---

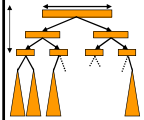
CentraleSupélec

Programmation OpenMP – exemples

## Parallélisation du quick-sort

**Implantation d'une parallélisation simple du Quick-Sort**

Région parallèle & sections & réplication de code (OpenMP-1)



```

void quicksort(int q, int r, int deep)
{
    if (q < r) {
        ..... // split the table ...
        // classic quick-sort op

        if (deep < MAX) {
            #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);
        }
    }
}

```

---

---

---

---

---

---

---

---

---

---

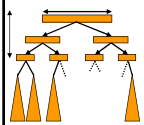
CentraleSupélec

Programmation OpenMP – exemples

## Parallélisation du quick-sort

**Implantation d'une parallélisation simple du Quick-Sort**

Région parallèle & clause conditionnelle & sections (OpenMP-2)



```

void quicksort(int q, int r, int deep)
{
    if (q < r) {
        ..... // split the table ...
        // classic quick-sort op

        #pragma omp parallel if(deep < MAX)
        {
            #pragma omp sections nowait
            {
                #pragma omp section
                quicksort(q,s-1,deep+1);
                #pragma omp section
                quicksort(s+1,r,deep+1);
            }
        }
    }
}

```

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

## Programmation OpenMP (par partage de mémoire)

1. Principes d'OpenMP
2. Détails de syntaxe et de sémantique
3. Optimisations
4. Exemples de parallélisation
5. **Bilan d'OpenMP**
6. Mesures et représentation de performances

---

---

---

---

---

---

---

---

---

---

CentraleSupélec 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 !**

---

---

---

---

---

---

---

---

CentraleSupélec

## Multithreading with OpenMP

# Questions ?

---

---

---

---

---

---

---

---