Introdução ao OpenMP

Fernando Silva
DCC - FCUP

Recursos

- OpenMP.org (http://www.openmp.org)
- Tutorial (http://www.llnl.gov/computing/tutorials/openMP/)
- Implementação gratuita
  - Omni OpenMP compiler project (http://phase.hpcc.jp/Omni/)

O que é o OpenMP?

- API para programação paralela explícita
  - paralelismo em memória partilhada
  - sobretudo paralelismo nos dados
- Constituída por:
  - directivas de compilação,
  - biblioteca de funções,
  - e variáveis de ambiente
- Objectivo – obter um standard de programação apoiado pelos grandes fabricantes de software e hardware

O que é o OpenMP (cont.)?

- Portável
  - versões para
    - C/C++ (ver. 1.0 em 1998)
    - e Fortran (versão 1.0 em 1997)
  - implementações para UNIX/Linux e Windows
- Significado:
  - *Open specifications for Multi Processing via collaborative work from software/hardware industry, academia and government*

Um Programa Simples em OpenMP

- Directivas de compilação ou pragmas
  - identificam pontos e formas de paralelização
- Ênfase na paralelização de ciclos

Programa sequencial

```
void main()
{
  double r[1000];

  for (int i=0; i<1000; i++) {
    large_computation(r[i]);
  }
}
```

Programa paralelo

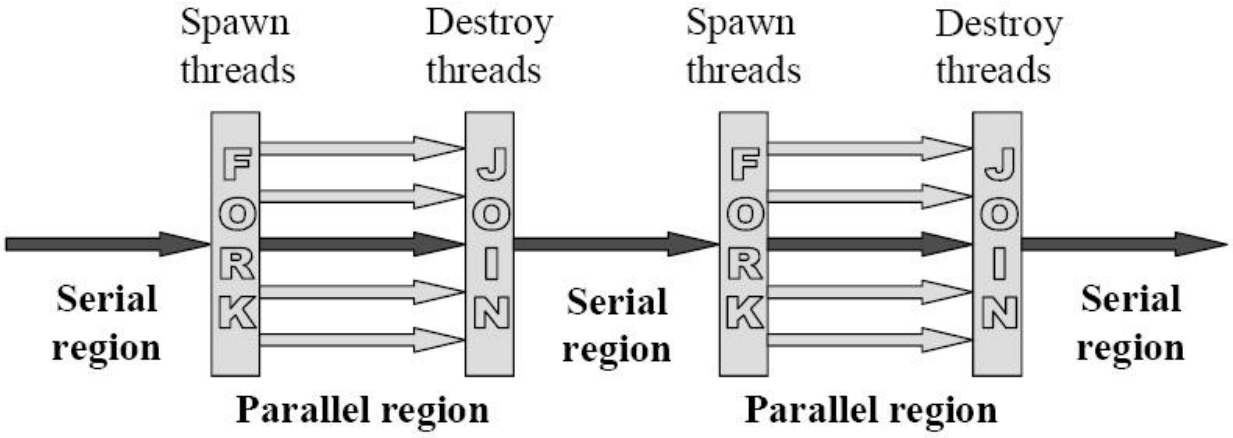
```
void main()
{
  double r[1000];

#pragma omp parallel for
  for (int i=0; i<1000; i++) {
    large_computation(r[i]);
  }
}
```


Modelo de Programação do OpenMP

- Paralelismo explícito
  - anotações feitas pelo programador
- Modelo de memória partilhada
  - baseado em threads
  - trabalho é dividido por threads de execução
    - variáveis podem ser
      - partilhadas por todos os threads
      - duplicadas (privadas) para cada thread
    - threads comunicam através das variáveis partilhadas
- Usar sincronização para evitar competição entre threads

OpenMP - Modelo de Execução

- Modelo de execução tipo fork-join
  - inicia execução com um processo (master thread)
  - no início do construtor paralelo – cria um team-of-threads
  - ao completar – o team-of-threads sincroniza numa barreira implícita
  - apenas o master continua a execução



C/C++ - Estrutura Geral do Código

```
#include <omp.h>
main() {
  int var1, var2, var3;

  parte_sequencial_1();

#pragma omp parallel private(var1,var2) shared(var3)
{
  parte_paralela(); // executada pelos threads
  ...
  // Todos os threads fazem o join com o master e terminam
}
  parte_sequencial_2();
}
```

OpenMP: Directivas C/C++

- **`#pragma omp`** *directive-name* [*clause, …*] *newline*

- *directive-name*
  - uma directiva OpenMP válida. Tem de aparecer depois do pragma e antes das claúsulas
- [*clause, …*]
  - Opcional. Podem aparecer em qualquer ordem e se necessário repetidas.
- *newline*
  - Obrigatório. Seguido do bloco estruturado que vai ser executado em paralelo.
- Exemplo:
  - **#pragma omp parallel default(shared) private(beta,pi)**

Compilação Condicional: _OPENMP

```
#ifdef _OPENMP
  bloco_código;
#endif
```

- Exemplo de código:
  - printf("%d processadores livres\n", omp_get_num_procs());
- Possibilita instrumentalizar o código

Syntax – parallel

- **#pragma omp parallel 'clause'**
  bloco_código;

- Indica que o bloco_código é para ser executado em paralelo.
- onde **'clause'** pode ser:
  - **if(*exp*)**
  - **private(*list*)**
  - **firstprivate(*list*)**
  - **num_threads(*int_exp*)**
  - **shared(*list*)**
  - **default(shared|none)**
  - **copyin(*list*)**
  - **reduction(*operator: list*)**

Claúsulas OpenMP

- As claúsulas private, shared, default e firstprivate possibilitam ao utilizador o controlo do âmbito das variáveis na região paralela.
- private (list)
  - as variáveis da lista ficam privadas a cada thread do team
  - não são incializadas
- firstprivate(list)
  - permite que as variáveis privadas sejam inicializadas
- shared (list)
  - as variáveis da lista são partilhadas por todos os threads
  - por defeito as variáveis são "shared"

Quantos threads?

- O número de threads na região paralela é determinado pelos seguintes factores (ordem de precedência):
  - claúsula **num_threads(int)**
  - função **omp_set_num_threads()**
  - variável ambiente OMP_NUM_THREADS
  - default - depende da implementação
- Os threads são numerados de 0 a N-1
- Por defeito, um programa com várias regiões em paralelo usará o mesmo número de threads para cada região, a menos que redefina como explicado.

Outras Claúsulas: reduction, copyin e if

- **reduction (operador:lista)** – permite operar sobre as variáveis da lista
- **copyin(list)** possibilita a atribuição do mesmo valor a variáveis THREADPRIVATE
- **if (exp)** – tem de avaliar como verdadeiro para que o team de threads seja criado, senão a execução será sequencial.

Programa Hello!

```
#include <omp.h>
main () {
  int nthreads, tid; // Fork team-threads, cada uma cópia das variáveis.

#pragma omp parallel private(nthreads, tid)
{                                          // Obtem e escreve thread-id
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);
                                           // apenas o master thread faz isto
  if (tid == 0){
      nthreads = omp_get_num_threads();
      printf("Number of threads = %d\n", nthreads);
  }
} /* Todos os theards juntam-se no master
}
```


Constructores de work-sharing

- Especificam regras de divisão de trabalho entre os threads, não cria os threads!
- Tipos de partilha:
  - **for** – partilha as iterações de um ciclo pelos threads da team (data paralelism).
  - **sections** – divide o trabalho em secções discretas, distintas, que são executadas pelos threads. Pode ser usado para paralelismo funcional.
  - **single** – serializa o código

Syntax – for

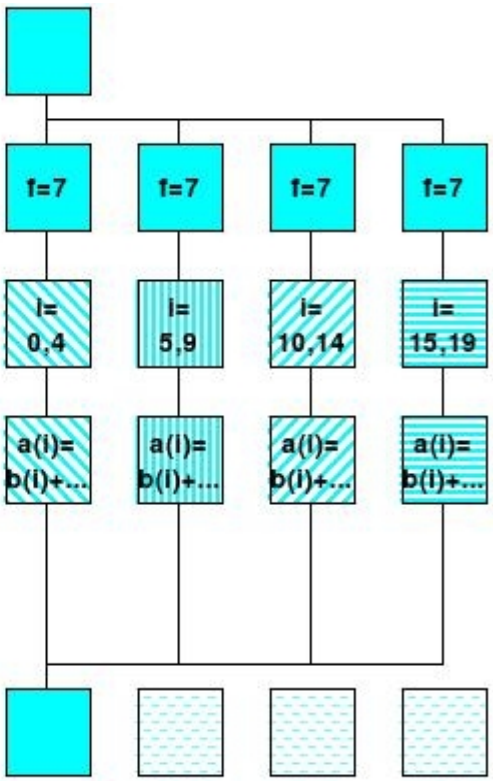
- `#pragma omp for 'clause'`
  `{ ciclo_for(); }`

- **onde 'clause' pode ser:**
  - **private(*list*)**
  - **firstprivate(*list*)**
  - **lastprivate(*list*)**
  - **reduction(*operator: list*)**
  - **ordered**
  - **schedule(*type*)**
  - **Nowait**
- O compilador distribui as iterações pelos threads

Exemplo - for

C / C++:

```
#pragma omp parallel private(f)
{
    f=7;

#pragma omp for
    for (i=0; i<20; i++)
        a[i] = b[i] + f * (i+1);

} /* omp end parallel */
```



Claúsulas schedule

- como dividir as iterações do ciclo pelos threads.
- **static** – as iterações são agrupadas em conjuntos (chunks), estaticamente atribuídos aos threads.
- **dynamic** – as iterações sõa agrupadas em bocados (chunks) e são dinamicamente distribuídos pelos threads; quando um termina, recebe dinamicamente outro chunk.
- **guided** – indica o número mínimo de iterações a agrupar numa tarefa;
- **runtime** – a decisão é tomada em tempo de execução a partir da var. OMP_SCHEDULE

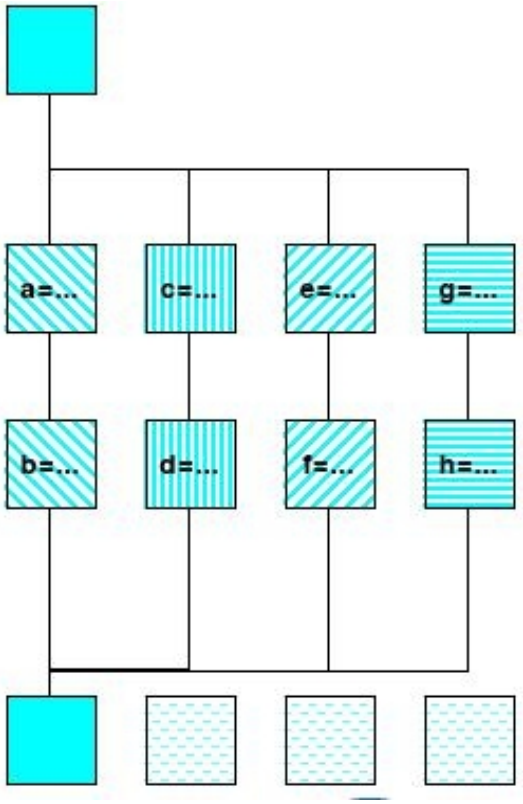
Exemplo do uso da directiva for

```
#include <omp.h>
#define CHUNKSIZE 100
#define N 1000
main() {
  int i, chunk;
  float a[N], b[N], c[N];
  /* Algumas inicializacoes */
  for (i=0; i < N; i++)
     a[i] = b[i] = i * 1.0;
  chunk = CHUNKSIZE;
#pragma omp parallel shared(a,b,c,chunk) private(i) {
#pragma omp for schedule(dynamic,chunk) nowait
  for (i=0; i < N; i++)
     c[i] = a[i] + b[i];
}  // fim da secção paralela/
}
```

Exemplo - sections

C / C++:

```
#pragma omp parallel
{
#pragma omp sections
 {{ a=...;
    b=...; }
#pragma omp section
   { c=...;
     d=...; }
#pragma omp section
   { e=...;
     f=...; }
#pragma omp section
   { g=...;
     h=...; }
 } /*omp end sections*/
} /*omp end parallel*/
```



Syntax – sections

- **`#pragma omp sections 'clause'`**

```
{ #pragma omp section newline
   código();
  #pragma omp section newline
   código();
   };
```

- onde **`'clause'`** pode ser:
  - **`private(`*`list`*`)`**
  - **`firstprivate(`*`list`*`)`**
  - **`lastprivate(`*`list`*`)`**
  - **`reduction(`*`operator: list`*`)`**
  - **`nowait`**

Exemplo - sections

```
#include <omp.h>
#define N 1000
main() {
  int i, chunk;
  float a[N], b[N], c[N];
  // Some initializations
  for (i=0; i < N; i++)
    a[i] = b[i] = i * 1.0;
#pragma omp parallel shared(a,b,c) private(i) {
#pragma omp sections nowait {
#pragma omp section
  for (i=0; i < N/2; i++) c[i] = a[i] + b[i];
#pragma omp section
  for (i=N/2; i < N; i++) c[i] = a[i] + b[i];
} // fim de secções
}
```


Constructores de Sincronização

- omp master – especificar uma região que será executada apenas pelo master (os outros ignoram)
- omp critical – especifica uma região crítica de código que deve ser executada apenas por um thread de cada vez.
- omp barrier – quando esta directiva é alcançada por um thread, este espera até que os restnates cheguem ao mesmo ponto.

Constructores de Sincronização (cont.)

- omp atomic - especifica um endereço de memória para actualização atómica.
- omp flush – identifica um ponto de sincronização no qual é necessário providenciar uma visão consistente da memória.
- omp ordered - especifica que as iterações devem ser executadas pela mesma ordem, como se fossem executadas seuqencialmente.

Funções OpenMP

- `void omp_set_num_threads (int)`
  - invocada na parte sequencial.
- `int omp_get_num_threads (void)`
  - retorna o número de threads activos
- `int omp_get_max_threads (void)`
  - retorna o número máximo de threads permitidos
- `int omp_get_thread_num (void)`
  - retorna o ID do thread (entre `0` e `t-1`)
- `int omp_get_num_procs(void)`
  - retorna o número de processadores disponíveis para o programa

Funções OpenMP (cont)

- `void omp_init_lock(omp_lock_t*)`
  - Inicializa um lock associado à variável de lock
- `void omp_destroy_lock(omp_lock_t*)`
- `void omp_set_lock(omp_lock_t*)`
  - espera até conseguir o lock
- `void omp_unset_lock(omp_lock_t*)`
  - liberta o lock
- `double omp_get_wtime(void)`
  - retorna o o num. segundos decorridos (elapsed time)
- `double omp_get_wtick(void)`
  - retorna os segundos decorridos entre chamadas sucessivas

Exemplo – Redução

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[]) {
  int   i, n;
  float a[100], b[100], sum;

  n = 100; /* Some initializations */
  for (i=0; i < n; i++)
    a[i] = b[i] = i * 1.0;
  sum = 0.0;
  #pragma omp parallel for reduction(+:sum)
    for (i=0; i < n; i++)
      sum = sum + (a[i] * b[i]);
  printf("   Sum = %f\n",sum);
}
```

Exemplos – funções OpenMP

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[]){
int nthreads, tid, procs, maxt, inpar,
    dynamic, nested;

/* Start parallel region */
#pragma omp parallel private(nthreads, tid) {
  /* Obtain thread number */
  tid = omp_get_thread_num();

  /* Only master thread does this */
  if (tid == 0) {
    printf("Thread %d getting info...\n", tid);
    /* Get environment information */
    procs = omp_get_num_procs();
    nthreads = omp_get_num_threads();
    maxt = omp_get_max_threads();
    inpar = omp_in_parallel();
    dynamic = omp_get_dynamic();
    nested = omp_get_nested();

    /* Print environment information */
    printf("Number of processors = %d\n", procs);
    printf("Number of threads = %d\n", nthreads);
    printf("Max threads = %d\n", maxt);
    printf("In parallel? = %d\n", inpar);
    printf("Dynamic threads? = %d\n", dynamic);
    printf("Nested parallelism? = %d\n", nested);
    }
  }  /* Done */
}
```