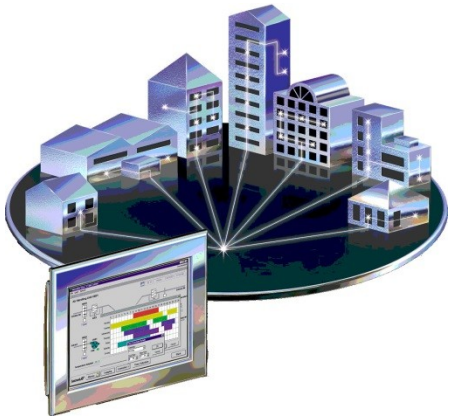
Programação com OpenMP



Programação Paralela Avançada - PPA

Mestrado em Computação Aplicação – MCA
Programa de Pós-Graduação em Computação Aplicada – PPGCA
Centro de Ciências Tecnólogicas - CCT
Universidade do Estado de Santa Catarina – UDESC

Profs Maurício A. Pillon e Guilherme P. Koslovski
Linha de Sistemas Computacionais
Grupo de Pesquisa de Redes de Computadore e Sistemas Distribuídos
Laboratório de Pesquisa LabP2D

Agenda: OpenMP

- O que é?
- Como usar?
- Funções auxiliares
- Variáveis de ambiente

OpenMP

- O que é?
  - API para criação de paralelismo em sistemas de memória compartilhada
- C, C++, FORTRAN
- O que NÃO é?
  - Feito para memória distribuída
  - Não garante a melhor utilização da memória compartilhada
  - Não verifica dependências ou *deadlocks*

Como compilar?

- #include <omp.h>
- gcc nome_do_arquivo.c –fopenmp –o nome_do_arquivo

OpenMP

- C/C++
  - Explora a diretiva "*#pragma*"
    - Permite a passagem de informações complementares ao compilador
  -

```
int a, b;

main() {
    // serial segment
    #pragma omp parallel num_threads (8) private (a) shared (b)
    {
        // parallel segment
    }
    // rest of serial segment
}
```

Sample OpenMP program

```
int a, b;
main() {
    // serial segment
    #pragma omp parallel num_threads (8) private (a) shared (b)
    {
        // parallel segment
    }
    // rest of serial segment
}
```

Sample OpenMP program

```
int a, b;
main() {
    // serial segment

    for (i = 0; i < 8; i++)
        pthread_create (........, internal_thread_fn_name, ...);

    for (i = 0; i < 8; i++)
        pthread_join (.......);

    // rest of serial segment

}

void *internal_thread_fn_name (void *packaged_argument) [
    int a;

    // parallel segment

}
```

Code inserted by the OpenMP compiler

Corresponding Pthreads translation

OpenMP

- *Clause List*
  - Condição de paralelização
    - *if(expressão condicional)*
  - Grau de concorrência
    - *num_threads(integer expression)*
  - Manipultação de dados
    - *private (variable list)*
    - *firstprivate (variable list)*
    - *shared (variable list)*
- *omp_get_thread_num()*

Exemplo

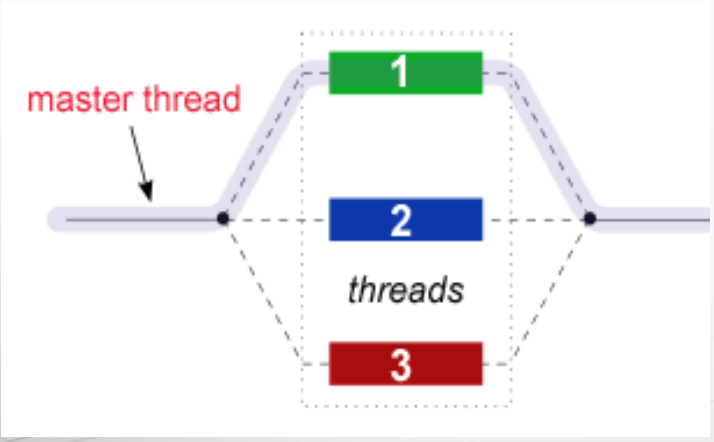
```
#include <omp.h>
int main (int argc, char
*argv[]) {
  printf("Hello World”);
}
```

Exemplo

```
#include <omp.h>
int main (int argc, char *argv[]) {
  int tid;
  /* Fork a team of threads giving them their own copies of
variables */
  #pragma omp parallel private(tid)
  {
    /* Obtain thread number */
    tid = omp_get_thread_num();
    printf("Hello World from thread = %d\n", tid);
  } /* All threads join master thread and disband */
}
```

OpenMP

- Diretiva "*master*"
  - *#pragma omp master*




```
#pragma omp parallel
{
  Work1();

  // This...
  #pragma omp master
  {
    Work2();
  }

  // ...is practically identical to this:
  if(omp_get_thread_num() == 0)
  {
    Work2();
  }

  Work3();
}
```

Exemplo

```
#include <omp.h>
int main (int argc, char *argv[]) {
  int nthreads, tid;
  /* Fork a team of threads giving them their own copies of
variables */
  #pragma omp parallel private(nthreads, tid)
  {
    /* Obtain thread number */
    tid = omp_get_thread_num();
    printf("Hello World from thread = %d\n", tid);

    /* Only master thread does this */
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);

  } /* All threads join master thread and disband */
}
```

Exemplo

```
#include <omp.h>
int main (int argc, char *argv[]) {
  int nthreads, tid;
  /* Fork a team of threads giving them their own copies of
variables */
  #pragma omp parallel private(nthreads, tid)
  {
    /* Obtain thread number */
    tid = omp_get_thread_num();
    printf("Hello World from thread = %d\n", tid);
    /* Only master thread does this */
    #pragma omp master
    {
      nthreads = omp_get_num_threads();
      printf("Number of threads = %d\n", nthreads);
    }
  } /* All threads join master thread and disband */
}
```

OpenMP

Diretiva "*for*"

```
#pragma omp parallel for num_threads(8)
{
    /* compute for here */
}
```

Diretiva "*reduction*"

```
#pragma omp parallel reduction(+: sum)
num_threads(8)
{
    /* compute local sums here */
}
/*sum here contains sum of all local instances of
sums */
```

Exemplo

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

  /* Some initializations */
  n = 100;
  for (i=0; i < n; i++)
    a[i] = b[i] = i * 1.0;
  sum = 0.0;

  for (i=0; i < n; i++)
    sum = sum + (a[i] * b[i]);

  printf("   Sum = %f\n",sum);
}
```

Exemplo

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

  /* Some initializations */
  n = 100;
  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);
}
```

OpenMP

- Diretiva "*schedule*" indica a distribuição da carga entre as threads
- *omp for schedule (scheduling_class [, parameter])*

(static, [, step]): distribuição estática entre todas as threads com tamanho step
(dynamic [, step]): distribuição dinâmica (cíclica) de blocos de tamanho step
(guided [, step]): distribuição estática entre as threads mas o tamanho do bloco diminui automaticamente durante a execução
(runtime): o escalonamento será definido em tempo de execução

Exemplo

```
/* static scheduling of matrix multiplication loops */
#pragma omp parallel default(private) shared (a, b, c,
dim) \
num_threads(4)
#pragma omp for schedule(static)
for (i = 0; i < dim; i++) {
    for (j = 0; j < dim; j++) {
        c(i,j) = 0;
        for (k = 0; k < dim; k++) {
            c(i,j) += a(i, k) * b(k, j);
        }
    }
}
```

OpenMP

- Diretiva "*nowait*", para que o programa não espere o fim da execução de cada thread da diretiva "*for*"

```
#pragma omp parallel
{
  #pragma omp for nowait
    for (i= 0; i< nmax; i++)
      if (isEqual(name, current_list[i])
        processCurrentName(name);
  #pragma omp for nowait
    for (i= 0; i< mmax; i++)
      if (isEqual(name, past_list[i])
        processPastName(name);
}
```

OpenMP

- Diretiva "*sections*" informa trechos de código que não são iterativos

```
#pragma omp sections [clause list]
{
  #pragma ompsection
    /* structured block */
  #pragma ompsection
    /* structured block */
...
}
```

Exemplo

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    {
      taskA();
    }
    #pragma omp section
    {
      taskB();
    }
    #pragma omp section
    {
      taskC();
    }
  }
}
```

OpenMP

- Sincronização de execução
  - *#pragma omp barrier*
  - *#pragma omp single*
  - *#pragma omp critical*
  - *#pragma omp ordered*

OpenMP

- Execução aninhada

```
omp_set_nested(1);
#pragma omp parallel for
for(int y=0; y<25; ++y)
{
  #pragma omp parallel for
  for(int x=0; x<80; ++x)
  {
    tick(x,y);
  }
}
```

Funções de OpenMP

- Conjunto de funções auxiliares que em combinação com as diretivas permitem o controle da execução.

```
/* thread and processor count */
void omp_set_num_threads (int num_threads);
int omp_get_num_threads ();
int omp_get_max_threads ();
int omp_get_thread_num ();
int omp_get_num_procs ();
int omp_in_parallel();
```

Funções de OpenMP

- Conjunto de funções auxiliares que em combinação com as diretivas permitem o controle da execução.

```
/* controlling and monitoring thread creation */
void omp_set_dynamic (int dynamic_threads);
int omp_get_dynamic ();
void omp_set_nested (int nested);
int omp_get_nested ();
/* mutual exclusion */
void omp_init_lock (omp_lock_t *lock);
void omp_destroy_lock (omp_lock_t *lock);
void omp_set_lock (omp_lock_t *lock);
void omp_unset_lock (omp_lock_t *lock);
int omp_test_lock (omp_lock_t *lock);
```

Variáveis de ambiente em OpenMP

- OMP_NUM_THREADS
  - Especifica o número padrão de threads criadas em uma região paralela
- OMP_SET_DYNAMIC
  - Determina se o número de threads pode ser alterado dinamicamente
- OMP_NESTED
  - Habilita a execução com nested parallelism
- OMP_SCHEDULE
  - Seleciona o tipo de agendamento

Conclusão

Threads explícitas

vs.

Programação baseada em diretivas