Módulo 10

OpenMP 1 : Introdução

O OpenMP (*Open Multi Processing*) permite a especificação do paralelismo explorável num dado programa através de diretivas. Todas as diretivas têm o formato:

```
#pragma omp <nome-directiva> [<cláusula>]
```

e a directiva aplica-se ao bloco de instruções que se lhe segue. Por exemplo:

```
#pragma omp parallel
{
  printf (“Hello world\n”);
}
```

cria um grupo de *threads* em que cada uma executará o bloco de instruções associado (`printf()`, neste exemplo).
O modelo de execução do OpenMP é do estilo *fork&join*, isto é, nos blocos sequenciais existe uma única *thread* (com ID 0), sendo que nos blocos paralelos é criado um conjunto de *threads* (*fork*) da qual a *thread* 0 também faz parte. No fim do bloco paralelo todas as *threads*, excepto a 0, terminam (*join*). O fim do bloco paralelo implica uma operação de sincronização, isto é, a *thread* 0 espera que todas as outras terminem (esta operação de sincronização é designada por barreira).

Uma das vantagens da utilização de directivas é que se o compilador não suportar OpenMP estas são ignoradas, sendo gerada uma versão sequencial do código.
Faça o *download* do ficheiro `/share/acomp/P10.zip` para sua directoria e faça o respectivo `unzip`. É criada a pasta `P10`.

Exercício 1 – Visualize o ficheiro pl10-1.c com o seguinte código:

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

main () {
  #pragma omp parallel
  {
    printf ("Hello world\n");
  }
  printf ("That's all, folks!\n");
}
```

a) Vamos construir o executável. Pretendemos usar o gcc 5.3.0, pelo que deve escrever na sua *shell*:

   `module load gcc/5.3.0`

   Compile este código sem usar o OpenMP:

   `gcc -O3 -march=ivybridge pl10-1.c -o pl10-1`

   Execute este programa (`sbatch pl10-1.sh 8`, o argumento da linha de comandos é o número de *threads* OpenMP a criar) e visualize o respectivo *output*. Quantas vezes foi escrita cada uma das frases? Porquê?

b) Recompile com o comando abaixo:

`gcc -O3 -march=ivybridge -fopenmp pl10-1.c -o pl10-1`

Execute este programa (`sbatch pl10-1.sh 8`) e visualize o respectivo *output*.. Quantas vezes foi escrita cada uma das frases? Porquê? Quantas *threads* foram criadas no bloco paralelo?

c) A função `int omp_get_thread_num (void)` permite a cada *thread* ter acesso ao seu ID. Declare uma variável "`int tid`" local ao bloco paralelo, isto é:

```
#pragma omp parallel
{
  int tid;
  …
}
```

e, dentro do bloco paralelo, faça cada *thread* ler o seu ID e imprimi-lo:

```
printf ("Hello world from thread %d\n", tid);
```

Note que ao declarar `tid` dentro do contexto (*scope*) do bloco paralelo garante que cada *thread* tem a sua própria instância **privada** da variável `tid`, logo cada *thread* vê apenas a sua própria cópia desta variável. Compile. Execute o programa várias vezes. O *output* é feito sempre pela mesma ordem? Porquê?

d) O número de *threads* criadas é, por omissão, igual ao número de processadores vistos pelo Sistema Operativo. Usando a função `int omp_get_num_procs (void)` faça com que a *thread* 0 (e apenas esta) imprima o número de processadores.

e) A função `int omp_get_num_threads (void)` permite ler o número de *threads* criadas. Deve ser invocada dentro do bloco paralelo. Altere o seu código para que apenas a *thread* 0 leia e imprima o número de *threads* criadas.

f) O OpenMP disponibiliza a função `double omp_get_wtime (void)` para medir o tempo, em segundos, decorrido desde um momento inicial qualquer. A diferença entre duas destas medições permite calcular o tempo decorrido para executar um determinado segmento de código. Altere o código para que a *thread* 0 meça o tempo imediatamente antes e depois do bloco paralelo e o imprima sem casas decimais em microssegundos:

```
double T1, T2;
…
T1 = omp_get_wtime();
…
T2 = omp_get_wtime();
printf ("That's all, folks! (%.0lf usecs)\n", (T2-T1)*1e6);
```

Resolução (LPS):

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

int main() {
  double T1, T2;

  T1 = omp_get_wtime();
  printf ("There are %d procs\n", omp_get_num_procs());
  #pragma omp parallel
  {
    int tid = omp_get_thread_num();

    printf ("Hello, world from thread %d!\n", tid);
    if (!tid) {
      printf ("There are %d threads in the team!\n", omp_get_num_threads());
    }
  }
  T2 = omp_get_wtime();
  printf("That's all, folks! (%.0f usecs)\n", (T2-T1)*1e6);
}
```

Exercício 2 – Visualize o ficheiro pl10-2.c com o código abaixo.

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

#define SIZE 100000000
float a[SIZE], c[SIZE];

main () {
  double T1, T2;
  float f;
  int i;

  for (i=0, f=1.f ; i<SIZE ; i++, f+=1.f) {
    a[i] = f;
  }
  T1 = omp_get_wtime();
  #pragma omp parallel for
    for (i=0 ; i<SIZE ; i++) {
      c[i] = powf (a[i], 3.f) + 10.f / a[i] - 100.f /(a[i] *a[i]);    }
  T2 = omp_get_wtime();
  printf ("That's all, folks! (%.0lf usecs)\n", (T2-T1)*1e6);  }
```

a) Compile este código usando o comando[1]:

`gcc -O3 -lm -fopenmp -march=ivybridge pl10-2.c -o pl10-2a`

Usando o comando `sbatch pl10-2a.sh` **<num-threads>** execute este programa para 1, 2, 4, 8 e 16 *threads*. Preencha a coluna 2a-Tempo da tabela abaixo.

b) Considerando o tempo obtido com o programa anterior e uma única *thread* como o tempo de referência preencha a coluna 2a-Ganho e comente os resultados.

c) O código dentro do ciclo `for` paralelo não vectoriza. Altere-o para que vectorize e crie novo executável:

`gcc -O3 -lm -fopenmp -march=ivybridge pl10-2.c -o pl10-2b`

Resolução (LPS) : remover a invocação da função powf()

```
    for (i=0 ; i<SIZE ; i++) {
      c[i] = a[i] * a[i] * a[i] + 10.f / a[i] - 100.f /(a[i] *a[i]);    }
```

d) Usando o comando `sbatch pl10-2b.sh` **<num-threads>** execute este programa para 1, 2, 4, 8 e 16 *threads* e preencha a coluna 2b-Tempo da tabela abaixo.

e) Usando o tempo para 1 *thread* da alínea 2a) como referência preencha a coluna 2b-Ganho. Qual o máximo ganho?

	2a		2b	
#threads	**Tempo (us)**	**Ganho**	**Tempo (us)**	**Ganho**
1	**10683905**	**1**	**231390**	**46**
2	**5350208**	**2**	**114669**	**93**
4	**2695273**	**3.9**	**59059**	**181**
8	**1329328**	**8**	**32585**	**328**
16	**735927**	**14.5**	**42075**	**254**

[1] Assume-se aqui que o módulo `gcc/5.3.0` foi carregado na alínea 1a . Se não repita o comando `module load`.