Módulo 10

Processamento paralelo

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 conjunto de *threads* em que cada uma executará o `printf()`.
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.

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.

Exercício 1 – Crie um 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) Compile este código sem usar o OpenMP:

```
gcc -O3 pl10-1.c -o pl10-1
```

b) Execute este programa. Quantas vezes foi escrita cada uma das frases? Porquê?

c) Recompile com o comando abaixo:

```
gcc -O3 -fopenmp pl10-1.c -o pl10-1
```

Execute este programa. Quantas vezes foi escrita cada uma das frases? Porquê? Quantas *threads* foram criadas no bloco paralelo?

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** imprima o número de processadores.

e) A função `int omp_get_thread_num (void)` permite a cada *thread* ter acesso ao seu ID. Declare uma variável

```
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 é necessário que cada *thread* tenha a sua própria cópia **privada** da variável ***tid***. Para que tal seja garantido acrescente a cláusula private() à directiva OpenMP. Esta cláusula especifica que serão criadas tantas cópias dessa variável quanto o número de *threads*, e que cada *thread* vê a sua própria cópia:

```
#pragma omp parallel private (tid)
```

Compile e execute o programa várias vezes. O *output* é feito sempre pela mesma ordem? Porquê?

f) 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.

g) O número de *threads* efectivamente criado pode ser alterado definindo, na linha de comando do Sistema Operativo, a variável de ambiente `OMP_NUM_THREADS`. Manipule esta variável (veja exemplo abaixo) e execute várias vezes o programa

```
export OMP_NUM_THREADS=8
```

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 em segundos usado 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);
```

Exercício 2 – Crie um ficheiro pl10-2a.c com o código abaixo. Note que a directiva

```
#pragma omp parallel for
```

faz com que o OpenMP distribua de forma estática e uniforme as iterações do ciclo ***for*** pelas *threads* criadas. Isto é, caso sejam criadas 4 *threads* cada uma vai executar 500000 iterações do ciclo para diferentes intervalos consecutivos do índice ***i***.

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

#define SIZE 2000000

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:

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

Manipulando a variável `OMP_NUM_THREADS` execute este programa (várias vezes) para 1, 2, 4 e 8 *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 ciclo dentro do ciclo ***for*** paralelo não vetoriza. Altere-o de forma a que vectorize (use a opção `-ftree-vectorizer-verbose=2` do `gcc` se quiser informações do vetorizador).

d) Manipulando a variável `OMP_NUM_THREADS` execute este programa para 1, 2, 4 e 8 *threads*. Preencha a coluna **2b-Tempo** da tabela abaixo. Usando como referência o tempo para 1 *thread* apenas da alínea 2a) preencha a coluna **2b-Ganho**. Qual o máximo ganho? Que comentário lhe suscita?

	2a		**2b**	
#threads	**Tempo** (us)	**Ganho**	**Tempo** (us)	**Ganho**
1				
2				
4				
8				