Módulo 11

OpenMP – operações colectivas

Considere o programa apresentado abaixo:

```
#include <stdio.h>
#include <omp.h>
#define S 10000000
int a[S];

main () {
  double T1, T2;
  int i;
  int sum=0, f, aux;

  for (i=0, f=-5000000.; i<S ; i++, f++) a[i] = f;
  T1 = omp_get_wtime();
  for (i=0 ; i<S ; i++) {
    sum += a[i];
  }
  T2 = omp_get_wtime();
  printf ("sum = %d\t%.0lf usecs\n", sum, (T2-T1)*1e6);
  printf ("That's all, folks!\n");
}
```

Crie um ficheiro pl11.c com este código e compile-o usando o comando
`gcc -O3 -fopenmp pl11.c -o pl11`
execute-o e anote na tabela o resultado obtido bem como o respectivo tempo de execução.

Note que este programa realiza o que se chama uma redução. Aplica a mesma operação a todos os elementos de um conjunto de dados. Esta operação, adição neste exemplo, resulta num único valor, a soma acumulada de todos os elementos do vector.

Exercício 1 – Reescreva o programa acima usando a directiva *parallel for* do OpenMP. Note que o vector ***a*** é partilhado por todas as *threads*, mas a variável ***i*** deve ser privada para cada uma (isto é, cada thread tem a sua cópia de ***i***).
O que deverá acontecer com ***sum***? Deve ser partilhada ou privada?
Experimentemos com a versão privada:

```
#pragma omp parallel for shared(a) private (i,sum)
for (i=0 ; i<S ; i++) {
    sum += a[i];
}
```

Execute este programa e anote o tempo de execução na tabela. O resultado obtido está correcto? Porquê?

Exercício 2 – Experimente agora indicar que a variável ***sum*** é partilhada. Execute este programa e anote o tempo de execução na tabela. O resultado obtido está correcto? Se executar o programa várias vezes e/ou com diferentes números de threads (`export`

OMP_NUM_THREADS=...) obtém sempre o mesmo resultado? Porquê? E se executar apenas com uma thread o resultado é correcto? Porquê?

Exercício 3 – O OpenMP permite indicar que uma determinada sequência de código pertence a uma região crítica, isto é, apenas uma *thread* a pode executar em cada instante. Modifique o programa de acordo com as instrução abaixo, execute-o, anote o resultado obtido bem como o respectivo tempo de execução e discuta estes resultados.

```
#pragma omp parallel for shared(a,sum) private (i)
for (i=0 ; i<S ; i++) {
    #pragma omp critical
    sum += a[i];
}
```

Exercício 4 – Uma alternativa mais eficiente é permitir a cada *thread* usar a sua própria variável de acumulação e posteriormente acumular todas estas somas locais. Execute o código que se segue e anote os resultados na tabela. Note que a cláusula usada para a variável `init_now` é `firstprivate`: o OpenMP atribuirá acada thread uma cópia privada de `init_now` inicializada com o valor que esta tem antes do bloco paralelo; com a cláusula `private` as variáveis não são inicializadas.

```
int tid, init_now=1;
float mysum[16];
...
for (i=0 ; i<16 ; i++) mysum[i] = 0;
#pragma omp parallel for shared(a,mysum) firstprivate (first) private (i,tid)
for (i=0 ; i<S ; i++) {
    if (init_now) {tid = omp_get_thread_num(); init_now=0;}
    mysum[tid] += a[i];
}
for (i=0 ; i<16 ; i++) sum += mysum[i];
...
```

Exercício 5 – A redução é uma operação colectiva (envolvendo todas as *threads*) tão comum que o OpenMP inclui uma cláusula específica para esta situação. Modifique o programa de acordo com as instrução abaixo, execute-o, anote o resultado obtido bem como o respectivo tempo de execução e discuta estes resultados.

```
#pragma omp parallel for shared(a) private (i) reduction (+:sum)
for (i=0 ; i<S ; i++) {
   sum += a[i];
}
```

Versão	Tempo (usecs)	Resultado	Correcto?
Sequencial			
Exercício 1			
Exercício 2			
Exercício 3			
Exercício 4			
Exercício 5			