Computação de Alto Desempenho 2014/15

Draft da resolução do 1º Teste 2013/14

1.

a) É necessário assegurar a coerência das caches porque uma cache L1 é privada a cada CPU e a escrita numa das suas linhas faz com que o seu conteúdo seja divergente em relação às caches L1 dos outros CPUs, e como tal é invalidada por hardware.

b) O false sharing consiste num problema de desempenho resultante da invalidação de uma linha da cache de um dado CPU, como resultado de uma atualização periódica de dados não partilhados existentes nessa linha, por parte do código em execução noutra CPU. Assim, embora a primeiro CPU não necessite de aceder a esses dados, toda a linha é invalidada resultando numa degradação do desempenho.

No código OpenMP existe false sharing porque os dois inteiros x e y cabem numa linha da cache. Assim, apesar de cada um dos threads atualizar apenas uma das duas variáveis (ou x ou y), tal resulta na invalidação da linha da cache do CPU que suporta a execução do outro thread. Na realidade, o problema só é relevante se o código de atualização estivesse no contexto de um ciclo.

c) O "false sharing" poderia ser evitado definindo espaço adicional (não usado) entre as duas variáveis de modo a garantir que x e y não se situariam na mesma linha da cache. Outra possibilidade é definir, para cálculos intermédios (e.g. no contexto de um ciclo de execução), variáveis privadas para a cada thread, garantindo-se assim que elas são reservadas em zonas diferentes da memória/pilha.

2.

$$\text{Speedup} = T_{\text{sequencial}}/T_{\text{paralelo}}$$

$$T_{\text{paralelo}} = T_{\text{sequencial}}/n$$

$$= \underset{(20\%)}{T_{\text{parte-sequencial}}} + \underset{(80\%)}{T_{\text{parte-paralelizavel}}/n}$$

como o programa pode ser executado de uma forma embaraçosamente paralela (desprezando overheads da paralelização e considerando que se n tende para infinito, $T_{\text{parte-paralelizavel}}/n$ tende para zero):

$$T_{\text{paralelo}} = T_{\text{parte-sequencial}}$$

Assim,

$$\begin{aligned} \text{Speedup} &= T_{\text{sequencial}}/T_{\text{parte-sequencial}} \\ &= 1/0.2 = 5 \end{aligned}$$

3. **a)** Uma possível solução da aplicação das quatro fases da metodologia de Foster:

Particionamento: cada task faz uma operação de soma de cada elemento das matrizes A e B, dado que são operações independentes.

Comunicação: As tasks são independentes, mas tem de ser ter em consideração o problema de *false sharing*.

Agregação: Cada task (resultado da aglomeração) deve corresponder a muitas somas, dado o overhead da criação de threads. Para além disso, e considerando o modo como o compilador de C arruma os vetores de dados na memória, é conveniente que a mesma task faça um processamento por linhas de modo a tirar partido da cache.

Mapeamento: Divisão do trabalho por dois blocos de linhas consecutivas, minorando o problema de *false sharing*.

b) Ver código *q3.c* na página do clip.

4. Ver código *q4.c* na página do clip.

5.

a)

```
Get input data;
# pragma omp parallel
for each timestep {
    # pragma omp single
    if (timestep output) Print positions and velocities of particles;

    # pragma omp for
    for each particle q
        Compute total force on q;

    # pragma omp for
    for each particle q
        Compute position and velocities on q;
}
```

b) Considerando o algoritmo reduzido, a solução em a) apresenta problemas de concorrência na atribuição de valores ao vector de forças, dado que o thread que calcula um valor para *q*, preenche também a posição de outro corpo *q'* com um valor simétrico. A solução encontra-se no ficheiro *omp_nbody_red.c* na página do clip.

Adicionalmente, poderia ser definida explicitamente uma partição cíclica do trabalho dado que o processamento das primeiras forças requer mais cálculos.