# **Computação Paralela e Distribuída, 2019**

## Relatório (*OpenMP*) - Grupo 18:

Beatriz Marques, 80809

Carlos Carvalho, 81395

Diogo Nunes, 85184

# Introdução

O objetivo da primeira entrega é paralelizar (num modelo de memória partilhada) a versão sequencial do projeto “simulador de partículas” com o uso da ferramenta *OpenMP*.

Por forma a realizar este trabalho, o grupo tomou os seguintes passos:

1. Identificação das secções de código computacionalmente pesadas;
2. Iterativamente sobre cada uma das secções supramencionadas:
   1. Implementação da respetiva paralelização;
   2. Verificação dos resultados mediante as alterações;
   3. Análise dos resultados produzidos (relatório da ferramenta *OpenMP Profiler*), sobre casos computacionalmente pesados;
   4. Conclusão sobre a relevância das alterações;
3. Análise e conclusão dos resultados finais produzidos (relatório da ferramenta *OpenMP Profiler*) sobre os casos computacionalmente pesados.

As entradas seguintes do relatório têm por base os passos descritos acima.

# Identificação e decomposição em secções paralelizáveis

Pela formalização do problema, as únicas direções de escalabilidade a considerar são o número de partículas do sistema (***n***), o número de células (***c***) e o número de *time steps* (***t***).

Por natureza, a computação de cada *time step* não pode ser paralelizada, já que o *time step* ***t*** depende do anterior, ***t-1***. Por esta razão, o foco do trabalho está sobre as duas restantes variáveis.

Da versão sequencial do grupo, as seguintes secções foram identificadas como computacionalmente pesadas (respetivo custo de execução entre parêntesis retos):

1. Inicialização das partículas [**n**]
2. Criação da matriz de células [**c**]
3. Inicialização da matriz de células [**c2**]
4. Para cada *time step* **t**,
   1. Determinação do centro de massa de cada célula [**n + c**]
   2. Computação da força gravítica aplicada a cada partícula [**9n**]
   3. Cálculo da nova velocidade e posição de cada partícula [**n**]
   4. Inicialização dos valores para a próxima iteração [**n + c**]
5. Cálculo do centro de massa do sistema [**n**]
6. Libertação de memória [**c**]

# Para cada secção, implementação da respetiva paralelização e resolução de problemas de sincronização

Das secções identificadas anteriormente, a (1.) inicialização de partículas não deve ser paralelizada devido à utilização de funções *random* (*thread safe*). É também de notar que, em cada uma destas secções, é executado exatamente o mesmo conjunto de procedimentos ou, para cada partícula ou, para cada célula, dependendo do contexto. Isto significa que é necessária muito pouca sincronização de acessos à memória. Tomando partido deste *data parallelism*, cada uma das secções (exceções descritas à frente) pode ser paralelizada utilizando a diretiva:

#pragma omp parallel for

Existem apenas duas secções de exceção à afirmação acima: (4.1.) determinação do centro de massa de cada célula e (5.) o cálculo do centro de massa do sistema. A razão de exceção está no facto de ser necessário obter, para cada célula, os valores cumulativos de ***x****,* ***y*** e ***m*** (respetivamente, posição bidimensional e massa) de cada partícula que nesta está inserida. É, portanto, necessário que cada *thread* tenha uma cópia local destas variáveis, execute a operação cumulativa sobre as partículas que lhe são designadas, e que no fim, seja efetuada uma operação de *reduction* para obter o resultado correto (desta vez, cumulativo da cópia local de cada *thread*).

No caso (5.) a implementação é relativamente direta já que se está a calcular o centro de massa de todo o sistema (entenda-se, uma única célula de lado 1). Desta forma, a diretiva utilizada foi:

#pragma omp parallel for reduction(+ : cmx, cmy, cmm)[[1]](#footnote-1)

No caso (4.1.), como é necessário efetuar o cálculo descrito acima para **c** células, é utilizado um *array* de estruturas que mantém a informação de cada uma destas células. A diretiva de *reduction* do *OpenMP*, de momento, não suporta nativamente a operação de redução sobre estruturas complexas - para ultrapassar esta limitação, foi necessário definir a seguinte operação de redução:

#pragma omp declare reduction(cm : cell\_t : \

omp\_out.x = omp\_out.x + omp\_in.x, \

omp\_out.y = omp\_out.y + omp\_in.y, \

omp\_out.m = omp\_out.m + omp\_in.m, \

omp\_out.npar = omp\_out.npar + omp\_in.npar)[[2]](#footnote-2)

Com esta operação definida e declarada, a implementação da paralelização resume-se à seguinte diretiva:

#pragma omp for reduction(cm: cell)[[3]](#footnote-3)

É de notar que cada secção mencionada como computacionalmente pesada, que passou a ser executada em paralelo, tem uma *barrier* (implícita) no final, o que obriga as *threads* dessa respetiva secção a esperarem umas pelas outras a fim do programa poder continuar a ser executado. Tiramos proveito desta implementação pois o conjunto destas secções deve ser executado sequencialmente.

# Balanceamento de carga

Balanceamento de carga é alcançado através da distribuição equilibrada de carga de trabalho entre as várias *threads* e/ou através da utilização da diretiva *schedule*.

As zonas identificadas como paralelizáveis são maioritariamente *loops* com tarefas semelhantes em cada iteração. Isto resulta numa distribuição de carga suficientemente equilibrada entre as várias *threads* pelo que não se justifica o uso de *schedule*. Existe apenas uma situação em que a diretiva *schedule* poderia ser vantajosa: quando existem células com zero partículas - neste caso, a execução dos *loops* irá ser bastante mais rápida aquando do tratamento destas células, eventualmente criando um desequilíbrio na carga de trabalho entre *threads*. Visto a improbabilidade deste caso com o aumento do número de partículas, concluiu-se que as vantagens não justificariam o *overhead* da utilização de *schedule*.

# Resultados de performance e conclusão

Os resultados experimentais foram obtidos numa máquina com 8 GB de RAM, 256 KB de cache L2 e 2 *cores* físicos. Os dados experimentais incluem os tempos de execução do programa paralelo (sobre um input computacionalmente pesado com 108 partículas, fornecido pelo corpo docente) com 1, 2, 4 e 8 *threads* ativas, e o respetivo *speedup*. Como base de comparação para o cálculo do *speedup* foi utilizada a versão paralela do código, com apenas 1 *thread* activa, que simula a versão do código em formato sequencial.

Em teoria o *speedup* máximo e ideal é igual ao número de *cores* (físicos) ativos e disponíveis. Desta forma, pelas características da máquina referida acima, é esperado obter um *speedup* próximo de 2. Na prática, obter o valor este valor ideal é irrealista, já que o programa em teste não é o único a correr na máquina alvo e existe todo o *overhead* do sistema operativo.

Tendo em conta os dados obtidos (resumidos no gráfico em anexo), verificamos que o *speedup* máximo obtido está próximo do valor 2 e, para além disso, confirmamos também uma clara influência negativa do *overhead* envolvido na utilização de *threads* virtuais, pelo facto de que quando se aumenta o número de *threads* ativas para um número maior do que 2 (número de *cores* físicos) o *speedup* nunca ultrapassa 2 e , se, por exemplo, o programa fosse testado com 16 *threads* (16 - 2 = 14 *threads* virtuais), observar-se-ia um aumento significativo do tempo de execução e uma consequente redução do *speedup*.

Por último, é relevante mencionar que os resultados produzidos pela ferramenta *OpenMP Profiler* indicam uma *parallel coverage* de aproximadamente 99.5 %.

# Anexo – Tabela de dados experimentais e gráfico

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Threads** | **Tempo (s)** | **Speedup** | **Tempo (experimental s)** | | | | | |
| 1 | 82,539 | 1 | 82,247 | 80,77 | 87,088 | 86,147 | 77,564 | 81,415 |
| 2 | 50,305 | 1,641 | 52,157 | 54,394 | 44,976 | 60,783 | 44,673 | 44,848 |
| 4 | 43,596 | 1,893 | 41,31 | 45,395 | 37,041 | 50,33 | 45,628 | 41,873 |
| 8 | 42,891 | 1,924 | 38,393 | 56,508 | 36,67 | 39,93 | 44,846 | 40,996 |

1. Onde ***cmx*** e ***cmy*** representam a posição bidimensional do centro de massa e ***cmm*** o valor cumulativo das massas das partículas consideradas para o cálculo. [↑](#footnote-ref-1)
2. Onde a estrutura ***cell\_t*** contém as variáveis ***x*** e ***y*** que definem a posição bidimensional da célula, ***m*** a soma cumulativa das massas das partículas que nesta estão inseridas, e ***npar*** o seu número de partículas. [↑](#footnote-ref-2)
3. Onde ***cm*** é a operação de redução sobre estruturas definida acima e ***cell*** é a variável que representa a célula em questão na iteração do *loop*. [↑](#footnote-ref-3)