TÉCNICO LISBOA

Mestrado em Engenharia Eletrotécnica e de Computadores

Computação Paralela e Distribuída

Projeto

Simulação de Partículas

Ion Ciobotari 84075

Pedro Fernandes 84168

Ivan Andrushka 86291

5/4/2019

TÉCNICO LISBOA

Abordagem usada para a paralelização

Como se pode observar facilmente, pelo código fornecido o programa realizado revolve bastante à volta de ciclos. Assim é possível dividir o programa em ciclos, onde cada um corresponde a uma secção principal do mesmo, como se pode observar pela figura 1, em anexo.

Tendo em conta as dependências de dados que existem entre as várias iterações do ciclo **1**, o mesmo não é possível ser paralelizado. Para além disso, os restantes ciclos são todos dependentes uns dos outros. Ou seja, para executar o ciclo **5**, é necessário que os ciclos **2**, **3** e **4** já tenham sido executados, também por ordem, e para executar o ciclo **6** é necessário que o ciclo **1** tenha sido executado, pelo que não é possível executar esses ciclos paralelamente. No entanto, os ciclos **2** a **6** não têm dependências internas, pelo que a paralelização do projeto consiste na paralelização interna desses mesmos ciclos.

Decomposição usada na paralelização

Foi então decidido criar uma secção paralela que engloba os ciclos **2** a **5**. Esta secção executa os referidos ciclos paralelamente, da forma como foi mencionado anteriormente, e espera pela finalização do ciclo **5**. Com o final da execução do ciclo **1** é criada uma nova secção paralela que executa o ciclo **6** paralelamente.

Sincronização no acesso a variáveis

No ciclo **3**, são efetuados os somatórios auxiliares ao cálculo dos centros de massa. Nesse ciclo todas as *threads* analisam a que quadrante da *grid* pertence uma certa partícula e incrementam a massa e as coordenadas x e y do respetivo quadrante, de acordo com a respetiva partícula. Dado que todas as *threads* são executadas em simultâneo, é possível que múltiplas tentem aceder ao mesmo quadrante da *grid*. Pelo que assim existe uma secção crítica localizada nesse ciclo que tem que ser gerida, de forma a não se perder informação. Deste modo usa-se a cláusula *Atomic* nas variáveis partilhadas pelas *threads*.

No ciclo **5** existem um conjunto de variáveis locais que ao paralelizar o programa passam a ser partilhadas pelas diversas *threads*, o que pode levar a inconsistências no seu valor. Deste modo usa-se a cláusula *private*, que cria uma cópia de cada variável partilhada privado à *thread*, anulando assim todas as *data races* a tais variáveis.

Semelhante ao caso explicado anteriormente o ciclo **6**, também possui variáveis partilhadas pelas *threads*, nomeadamente a massa e coordenadas do centro de massa final. Estas variáveis são constantemente incrementadas durante o ciclo, pelo que usando a cláusula *reduction* com a operação de incrementar, é criada uma cópia de cada variável partilhada por *thread*. Posteriormente é aplicado o operador, de incrementação, às cópias criadas e o resultado final reescrito para a respetiva variável.

TÉCNICO LISBOA

Distribuição de cargas pelas *threads*

Tendo em conta que a paralelização do programa consiste na paralelização de ciclos, não foi dada muita atenção à distribuição de carga por *thread*. Isto deve-se ao facto de os ciclos executados serem bastante equilibrados, pelo que distribuir as iterações do mesmo pelas *threads* é suficiente para garantir uma distribuição de cargas aceitável.

Resultados observados

De modo a testar o desempenho dos programas desenvolvidos, os mesmos foram executados em 3 computadores diferentes e foram recolhidos os seus tempos de execução. A partir destes dados foram calculados valores para *Speedup's* e eficiências, que se encontram na tabela 1, em anexo. Os testes referidos na tabela referem-se aos testes fornecidos na página da cadeira. À medida que se avança nos testes o tamanho do problema aumenta, ou seja, o número de partículas do mesmo.

Observando a tabela, pode-se observar um *speedup* bastante reduzido para os primeiros três testes. Tendo em conta que a execução destes testes demora um tempo bastante reduzido, o tempo de criação e destruição de *threads,* na versão paralelizada, compõem grande parte do tempo total. Consequentemente, o *speedup* decresce com o número de *threads* nos primeiros testes como se pode verificar na tabela.

A partir do quarto teste, a parte do programa que é executada em paralelo começa a ser mais significativa, pelo que o tempo de criação e destruição de *threads* começa a influenciar pouco o tempo total de execução. Assim começam a surgir *speedups* superiores a 1, como se observa na tabela.

Para os últimos testes, em que o número de partículas tem valores elevados, a paralelização do programa começa a influenciar bastante o tempo do programa, dado que grande parte do mesmo é paralelizada. Verifica-se assim que o *speedup* começa a aumentar proporcionalmente com o número de partículas.

Como referido anteriormente, o programa paralelizado deve ser considerado para elevados números de partículas. Pelo que para a próxima análise considera-se apenas os últimos testes. Tendo em conta que os computadores usados possuem quatro *cores*, podemos verificar uma estabilização do valor do *speedup* para execuções com quatro e oito *threads*, dado que estas ocupam na totalidade o poder de processamento do computador.

Atendendo agora à segunda metade da tabela, é possível confirmar a utilidade da paralelização para problemas de tamanho elevado. Nos primeiros testes a eficiência do programa é bastante reduzida. À medida que se avança nos testes e aumenta o número de partículas, a eficiência aumenta também.

Verificou-se também que a eficiência para execuções com quatro *threads* é superior a oito *threads*. Tendo em conta que não existe grandes variações de *speedups* nos mesmos, é possível concluir que a utilização de quatro *threads* é mais vantajosa para a execução do programa. Conclui-se também que a aplicação de *hyper-threading* nos computadores utilizados não é ideal.

Anexo

(1) //ciclo que percorre as iterações temporais
for(t = 0; t < time_stamps; t++) //linha 58

(2) //ciclo que inicializa o vetor grid no início de cada iteração temporal
for(i = 0; i < ncside * ncside; i++) //linha 64

(3) //ciclo que calcula o somatório usado no cálculo dos centros de massa
for(i = 0; i < n_part; i++) //linha 73

(4) //ciclo que calcula os centros de massa
for(i = 0; i < ncside * ncside; i++) //linha 88

(5) //ciclo que calcula a força resultante, aceleração, velocidade e posição das partículas
for(p = 0; p < n_part; p++) //linha 98

(6) //ciclo que calcula o centro de massa total
for(i=0; i<n_part; i++) //linha 187

Figura 1 – Esquema com secções do programa

Tabela 1 - Speedup e Eficiência do programa em diferentes computadores

		PC Lab				PC 1				PC 2			
	Teste	1 thread	2 threads	4 threads	8 threads	1 thread	2 threads	4 threads	8 threads	1 thread	2 threads	4 threads	8 threads
Speedup	1	1.0000	1.0000	0.0833	0.0313	0.9375	0.8824	0.8333	0.8333	0.7059	0.7500	0.8000	0.3333
	2	1.0000	1.0000	0.0417	0.0139	1.0000	0.9412	0.9412	0.8889	0.9286	0.9286	0.8667	0.4194
	3	1.0000	1.0000	0.1250	0.0377	1.0000	0.9412	0.9412	0.8889	0.8750	0.8750	0.7778	0.4375
	4	0.9512	0.9512	0.3750	0.3223	0.9545	1.0769	1.3125	1.3548	0.9545	1.1667	1.3404	1.1887
	5	0.9901	1.1976	1.3793	1.2121	0.9618	1.2583	1.5567	1.9868	1.0249	1.4845	1.8581	2.4000
	6	0.9643	1.3622	1.6271	1.6924	0.9950	1.3449	1.7800	2.3059	0.9536	1.3297	1.7136	2.2782
	7	0.9676	1.4075	1.8739	1.8838	0.9519	1.4007	1.9166	2.3128	0.9651	1.4538	1.9738	2.5378
	Teste												
Eficiência [%]	1	100.00	50.00	2.08	0.39	93.75	44.12	20.83	10.42	70.59	37.50	20.00	4.17
	2	100.00	50.00	1.04	0.17	100.00	47.06	23.53	11.11	92.86	46.43	21.67	5.24
	3	100.00	50.00	3.13	0.47	100.00	47.06	23.53	11.11	87.50	43.75	19.44	5.47
	4	95.12	47.56	9.38	4.03	95.45	53.85	32.81	16.94	95.45	58.33	33.51	14.86
	5	99.01	59.88	34.48	15.15	96.18	62.92	38.92	24.84	102.49	74.23	46.45	30.00
	6	96.43	68.11	40.68	21.16	99.50	67.24	44.50	28.82	95.36	66.48	42.84	28.48
	7	96.76	70.38	46.85	23.55	95.19	70.03	47.92	28.91	96.51	72.69	49.34	31.72

Tabela 2 – Tempos de execução do programa no PC Lab

PC Lab						
	Teste	Sequencial	1 thread	2 threads	4 threads	8 threads
Tempo de execução [s]	1	0.001	0.001	0.001	0.012	0.032
	2	0.001	0.001	0.001	0.024	0.072
	3	0.002	0.002	0.002	0.016	0.053
	4	0.039	0.041	0.041	0.104	0.121
	5	0.200	0.202	0.167	0.145	0.165
	6	18.490	19.175	13.574	11.364	10.925
	7	192.578	199.027	136.822	102.770	102.231

Tabela 3 – Tempos de execução do programa no PC 1

PC 1						
	Teste	Sequencial	1 thread	2 threads	4 threads	8 threads
Tempo de execução [s]	1	0.001	0.001	0.017	0.018	0.018
	2	0.001	0.001	0.017	0.017	0.018
	3	0.002	0.002	0.017	0.017	0.018
	4	0.039	0.041	0.039	0.032	0.031
	5	0.200	0.202	0.120	0.097	0.076
	6	18.490	19.175	10.308	7.788	6.012
	7	192.578	199.027	102.215	74.698	61.904

Tabela 4 – Tempos de execução do programa no PC 2

PC 2						
	Teste	Sequencial	1 thread	2 threads	4 threads	8 threads
Tempo de execução [s]	1	0.001	0.001	0.016	0.015	0.036
	2	0.001	0.001	0.014	0.015	0.031
	3	0.002	0.002	0.016	0.018	0.032
	4	0.039	0.041	0.054	0.047	0.053
	5	0.200	0.202	0.194	0.155	0.120
	6	18.490	19.175	18.258	14.167	10.656
	7	192.578	199.027	178.281	131.316	102.131