Programação de Alto Desempenho

Atividade 5 - Programação Paralela em Sistemas de Mem. Compartilhada

Lucas Santana Lellis - 69618
PPGCC - Instituto de Ciência e Tecnologia
Universidade Federal de São Paulo

I. Introdução

Nesta atividade foram realizados experimentos relacionados com a implementação de técnicas de programação paralela em sistemas de memória compartilhada utilizando OpenMP. Cada experimento foi realizado 5 vezes, e os resultados apresentados são a média dos resultados obtidos em cada um deles, sendo calculado o speedup pela fórmula

$$speedup(P) = \frac{\text{Tempo para 1 thread}}{\text{Tempo para P threads}}$$

e a eficiência pela fórmula

$$eficiencia(P) = \frac{speedup(P)}{P}$$

Todos os programas foram feitos em C, com otimização -O3, utilizando a biblioteca PAPI para estimar o tempo total de processamento, e o número de cache misses.

As especificações da máquina utilizada estão disponíveis na Tabela I.

Tabela I: Especificações da Máquina

CPU	Intel Core i5 - 3470
Cores	4
Threads	4
Clock	3.2 GHz
Cache L3	6144 KB
Cache L2	256 KB * 4
Hardware Counters	11
RAM	8 Gb
SO	Fedora 23
Kernel	4.7.4
GCC	5.3.1

II. Experimento I - Multiplicação de Matrizes

Neste experimento foi feita a implementação da multiplicação de matrizes com vetorização e blocagem para utilização efetiva de cache, utilizando OpenMP.

A. Avaliando resultado da multiplicação

Utilizando o software R, foi possível conferir que a multiplicação das matrizes de tamanho 8x8 e com blocagem de tamanhos 2x2, 4x4 e 8x8, foi realizada com sucesso para até 4 threads.

Nas Figuras 1 e 2 estão as matrizes de entrada, foi feita a multiplicação dessas matrizes em um outro software confiável (R), e o resultado está na Figura 3, enfim, para todos os testes executados, obtivemos a mesma saída, representada na Figura 4.f Assim, comparando os resultados é fácil observar que a multiplicação está sendo feita corretamente.

```
77.26 54.95 97.36 30.08 77.49 52.89 61.91 09.30
93.01 98.07 26.35 34.81 01.79 02.58 16.81 02.21
55.81 06.73 33.68 29.09 51.98 84.37 58.02 92.30
39.49 46.66 96.38 25.62 37.06 43.71 30.64 11.47
30.14 03.15 18.54 38.51 77.74 11.46 43.59 77.36
94.13 75.96 32.54 15.72 29.80 50.51 68.89 10.42
07.24 26.37 09.34 86.87 51.28 13.41 02.09 41.48
78.32 77.30 82.55 07.50 15.98 99.84 47.04 10.94
```

Figura 1: Matriz de entrada A.

```
53.38 49.13 12.99 75.91 51.64 73.20 12.52 14.88
18.77 32.11 42.48 32.74 92.06 82.02 02.43 94.08
20.69 53.17 36.81 11.17 45.50 13.89 57.19 14.55
90.37 29.36 44.99 61.69 28.21 17.39 45.93 86.46
81.73 63.82 32.24 34.04 22.88 99.28 03.76 83.08
24.02 54.03 01.59 33.12 59.44 23.96 25.51 99.03
13.58 89.30 64.88 50.61 98.33 94.87 90.77 59.74
80.96 10.86 93.02 75.87 58.01 48.39 10.26 54.28
```

Figura 2: Matriz de entrada B.

```
> mat <- scan('tests/matA.txt')
Read 64 items
> matA <- matrix(mat, ncol=8, byrow=TRUE)
> mat <- scan('tests/matB.txt')
Read 64 items
> matB <- matrix(mat, ncol=8, byrow=TRUE)
> matA %*% matB
[,1]      [,2]      [,3]     [,4]     [,5]     [,6]      [,7]     [,8]
[1,] 19085.60 25052.627 15739.183 18835.35 25870.69 27321.73 15405.991 26215.50
[2,] 11112.08 11920.428  9268.277 13877.77 17987.71 17764.61  6129.639 15531.88
[3,] 20966.57 19662.331 17719.448 21130.57 23118.34 22763.81 12538.554 25618.59
[4,] 12716.70 16902.660 11514.659 12312.51 18567.07 16690.43 11449.760 18455.39
[5,] 19015.72 14011.617 15489.058 16075.05 15010.09 19263.26  8549.099 18740.86
[6,] 13972.23 20151.302 12834.484 17929.95 24840.34 25054.49 11706.685 22538.45
[7,] 16824.95  8883.894 11134.989 12318.85 10259.21 11951.11  5829.013 18200.02
[8,] 13246.08 21673.026 12420.705 16922.20 26688.47 22320.94 13222.982 24906.10
```

Figura 3: Matriz de saída C - obtida no R.

```
19085.60 25052.62 15739.18 18835.35 25870.69 27321.73 15405.99 26215.50
11112.08 11920.43 09268.28 13877.76 17987.71 17764.61 06129.64 15531.88
20966.57 19662.33 17719.45 21130.57 23118.35 22763.81 12538.55 25618.59
12716.70 16902.66 11514.66 12312.51 18567.07 16690.43 11449.76 18455.39
19015.71 14011.62 15489.06 16075.05 15010.09 19263.26 08549.10 18740.86
13972.23 20151.30 12834.48 17929.95 24840.34 25054.49 11706.68 22538.45
16824.95 08883.89 11134.99 12318.85 10259.21 11951.11 05829.01 18200.02
13246.08 21673.03 12420.70 16922.20 26688.47 22320.95 13222.98 24906.10
```

Figura 4: Matriz de saída C - obtida em todos os testes.

B. Comparando speedup e eficiência

Na Tabela **??** estão disponíveis os resultados dos testes.

III. Experimento II - Odd-Even Sort

Nesse experimento foi feita a paralelização do algoritmo Odd-Even Sort, realizando a ordenação de valores pseudo aleatórios com 1, 2 e 4 threads obteve-se os mesmos resultados, como visto na Figura 5.

```
Entrada: 6 8 5 3 6 9 8 1 9 4
1 Thread :
2 Threads:
4 Threads:
```

Figura 5: Resultados dos testes.

IV. Experimento III - Contagem de números

Nesse experimento foi implementada a paralelização de um algoritmo para contagem da ocorrência de números em um vetor, para 1, 2 e 4 threads foi possível identificar que o valor total da soma de ocorrências de cada número é igual a 10^8, que é igual ao tamanho do vetor de entrada, o que confirma a validade da solução utilizada para paralelização.

A comparação de tempo de execução, speedup e eficiência para 1, 2 e 4 threads está disponível na Tabela **??**.

V. Experimento IV - Conjectura de Goldbach

Nesse experimento foi feita a comparação entre diferentes modos de escalonamento para a paralelização do algoritmo da conjectura de goldbach.

VI. Experimento V - Jogo da Vida

Nesse experimento foi implementada a paralelização de um algoritmo do jogo da vida. Para 1, 2 e 4 threads foi possível identificar que a solução permanece a mesma, em um tabuleiro de tamanho 1000x1000.