PDP - Atividade Prática 2 (OpenMP)

Arthur Adolfo Gabriel Seibel

8 de Junho de 2019

Introdução

Este documento descreve os resultados obtidos de um breve estudo aplicado sobre OpenMP. Utilizando a linguagem C, foram testados dois programas simples:

- Aproximaçao do número Pi
- Multiplicação de matrizes

Mais especificamente, foi medido o tempo de execução de cada programa sob diferentes configurações e, após, foram calculadas as suas acelerações (speedups) e a eficiências.

Metodologia

Abaixo consta o código utilizado durante o estudo, separado em duas categorias: computações e medições.

Computações

Para obter-se informações sobre o comportamento dos programas de aproximáção de pi e de multiplicação de matrizes, é necessário implementá-los. A atividade proposta já disponibilizava um código base para tanto, que foi modificado principalmente para utilizar corretamente funcionalidades da biblioteca OpenMP.

Aproximação de Pi

Para aproximar Pi, é utilizada uma redução sobre a variável `sum`, definida com uma diretiva ao OpenMP.

```
double pi_omp() {
    int i;
    double pi, x, sum = 0.0, step = 1.0 / (double) steps;

    #pragma omp parallel for reduction(+:sum) private(x)
    for (i = 0; i < steps; i++) {
        x = (i + 0.5) * step;
        sum += 4.0 / (1.0 + x * x);
    }
    pi = step * sum;
    return pi;
}
```

Multiplicação de Matrizes

Na implementação de multiplicação de matrizes, foi utilizada uma diretiva de laço paralelo, no `for` mais externo.

```
void mm_omp(const double *A, const double *B, double *C, int n) {
    #pragma omp parallel for
    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            double dot = 0;
            for (int k = 0; k < n; k++) {
                dot += A[i * n + k] * B[k * n + j];
            }

            C[i * n + j] = dot;
        }
    }
}
```

Medições

A fim de medir os tempos de execução dos programas, com diferentes graus de paralelismo (número `t` de threads usadas pelo OpenMP nas computações), foi criada uma rotina que, para cada `t`, executa e cronometra a computação algumas vezes e, após, calcula a média de tempo de execução. Abaixo, é mostrada a rotina para medição da multiplicação de matrizes, mas a rotina é praticamente idêntica no caso da aproximação de pi - difere pela chamada da computação.

```
for (t = 1; t <= max_threads; t++) {

    double mean_time = 0;
    for (i = 0; i < execs; ++i) {
        omp_set_num_threads(t);

        start = omp_get_wtime();
        mm_omp(A, B, C, n);
        delta = omp_get_wtime() - start;

        mean_time += delta;
    }
    mean_time /= execs;

    printf("Running on %d threads: ", t);
    printf("MM computed in %.4g seconds (on average of %d executions) \n", mean_time, execs);
}
```

Resultados

Os resultados aqui relatados foram gerados utilizando uma máquina com as seguintes configurações de CPU: Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz - 4 cores (8 threads).

Utilizando um script que executa os programas passando parâmetros de número máximo de threads (sempre 8, dada a máquina utilizada) e número de execuções usadas para tirar médias, foram obtidos os tempos de execução médios. A partir destes, foram calculadas a aceleração (speedup) e a eficiência em cada configuração. Os resultados são mostrados a abaixo, em Tables e gráficos.

Aproximação de Pi

Table 1: Aproximação de Pi

Threads	Tempo	Speedup	Eficiência
1	11	1.0000	1.0000
2	5.551	1.9816	0.9908
3	3.873	2.8402	0.9467
4	3.051	3.6054	0.9013
5	2.564	4.2902	0.8580
6	2.127	5.1716	0.8619
7	1.911	5.7561	0.8223
8	1.749	6.2893	0.7862

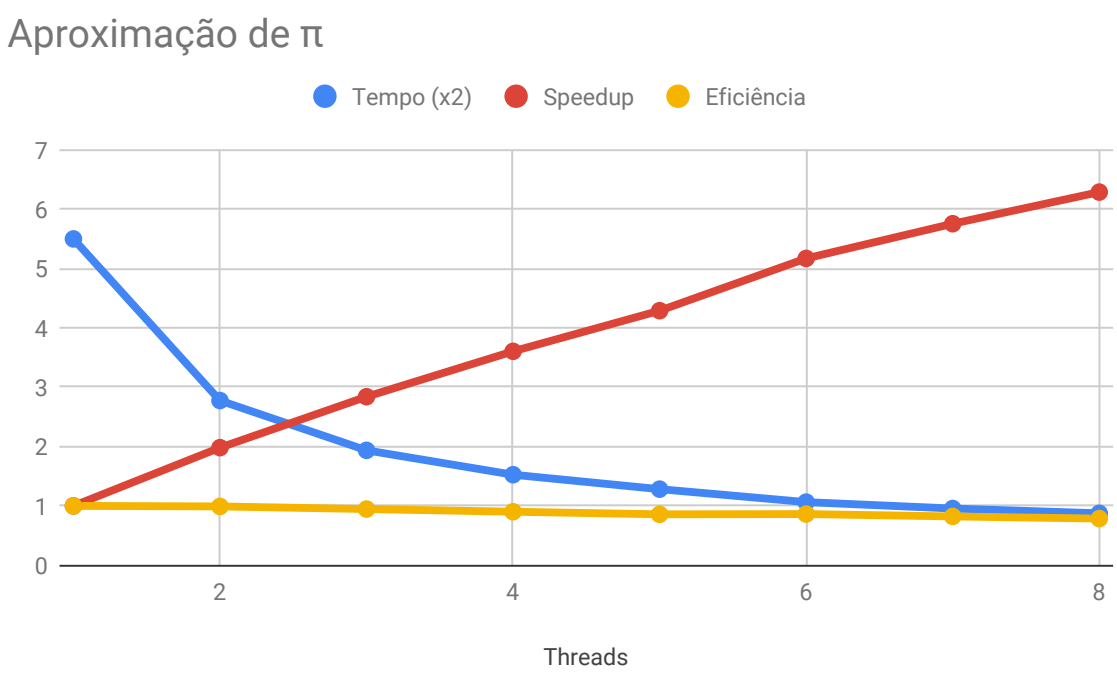

Figure 1: Aproximação de Pi

Multiplicação de Matrizes 500x500

Table 2: Multiplicação de Matrizes 500x500

Threads	Tempo	Speedup	Eficiência
1	0.3487	1.0000	1.0000
2	0.1770	1.9701	0.9850
3	0.1326	2.6297	0.8766
4	0.1175	2.9677	0.7419
5	0.1362	2.5602	0.5120
6	0.1229	2.8373	0.4729
7	0.1196	2.9156	0.4165
8	0.1163	2.9983	0.3748

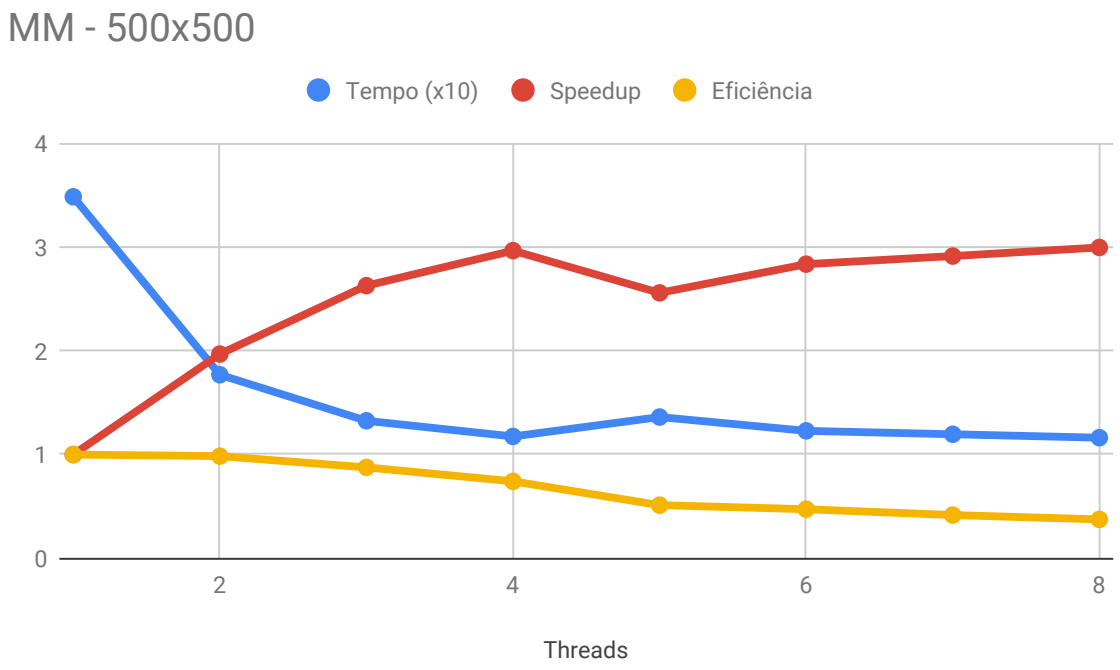

Figure 2: Multiplicação de Matrizes 500x500

Multiplicação de Matrizes 750x750

Table 3: Multiplicação de Matrizes 750x750

Threads	Tempo	Speedup	Eficiência
1	1.2600	1.0000	1.0000
2	0.6927	1.8190	0.9095
3	0.5051	2.4946	0.8315
4	0.4169	3.0223	0.7556
5	0.5126	2.4581	0.4916
6	0.4669	2.6987	0.4498
7	0.5342	2.3587	0.3370
8	0.6993	1.8018	0.2252

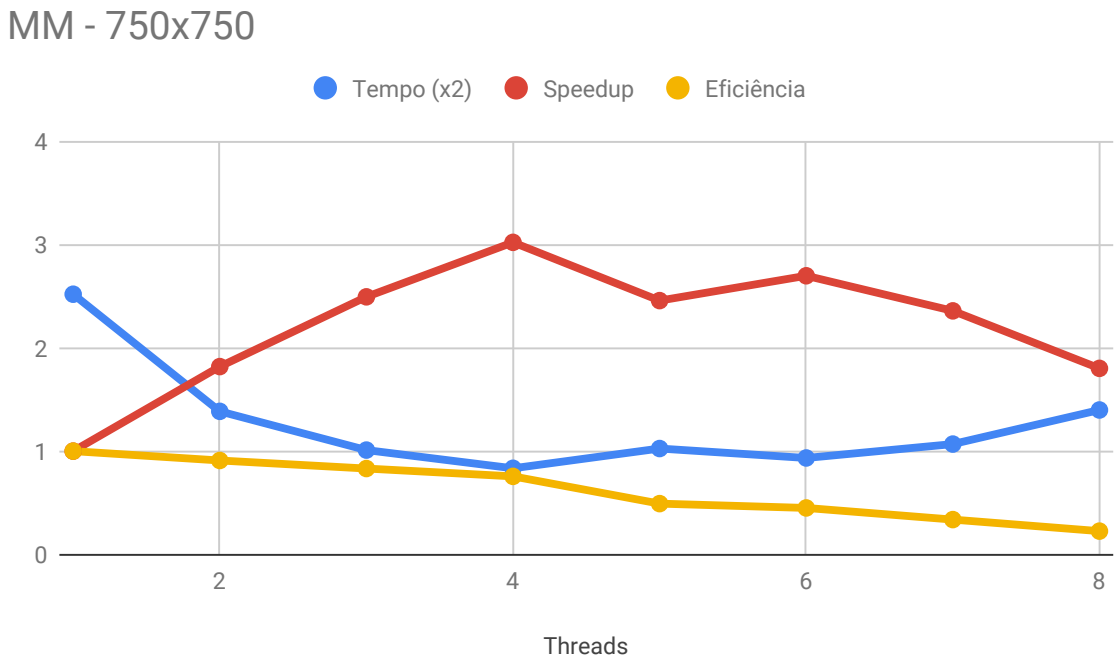

Figure 3: Multiplicação de Matrizes 750x750

Multiplicação de Matrizes 1000x1000

Table 4: Multiplicação de Matrizes 1000x1000

Threads	Tempo	Speedup	Eficiência
1	4.1660	1.0000	1.0000
2	1.9700	2.1147	1.0574
3	1.3980	2.9800	0.9933
4	1.2120	3.4373	0.8593
5	1.4790	2.8168	0.5634
6	1.3090	3.1826	0.5304
7	1.4670	2.8398	0.4057
8	1.7640	2.3617	0.2952

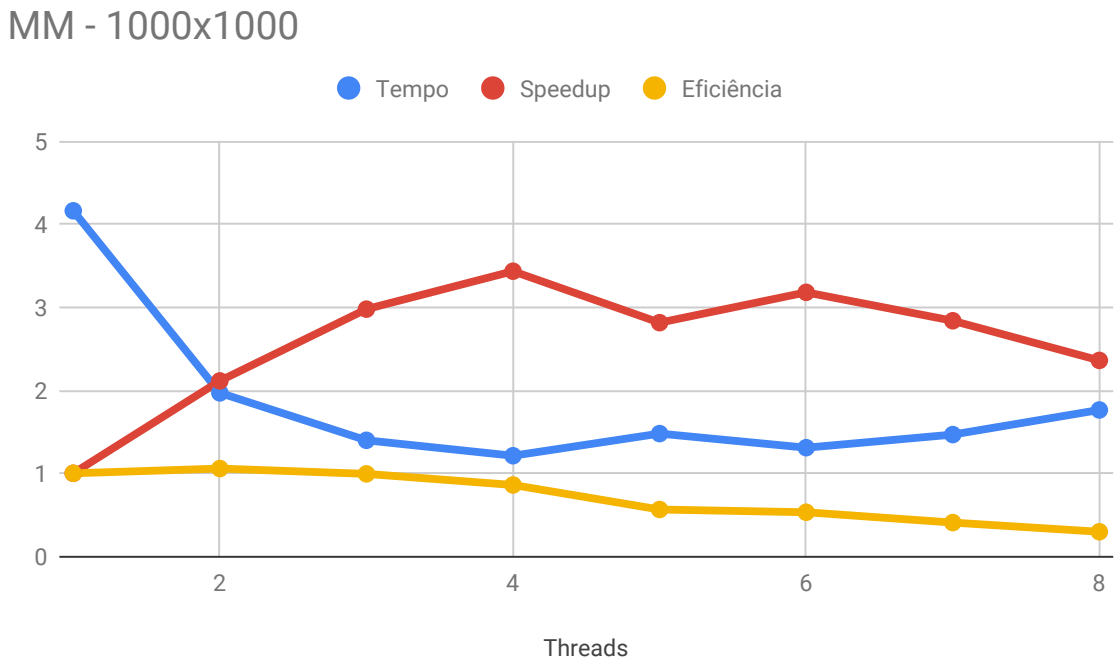

Figure 4: Multiplicação de Matrizes 1000x1000

Multiplicação de Matrizes 2000x2000

Table 5: Multiplicação de Matrizes 2000x2000

Threads	Tempo	Speedup	Eficiência
1	61.1800	1.0000	1.0000
2	30.4600	2.0085	1.0043
3	21.6200	2.8298	0.9433
4	18.1100	3.3782	0.8446
5	20.1200	3.0408	0.6082
6	17.4500	3.5060	0.5843
7	17.0500	3.5883	0.5126
8	16.9300	3.6137	0.4517

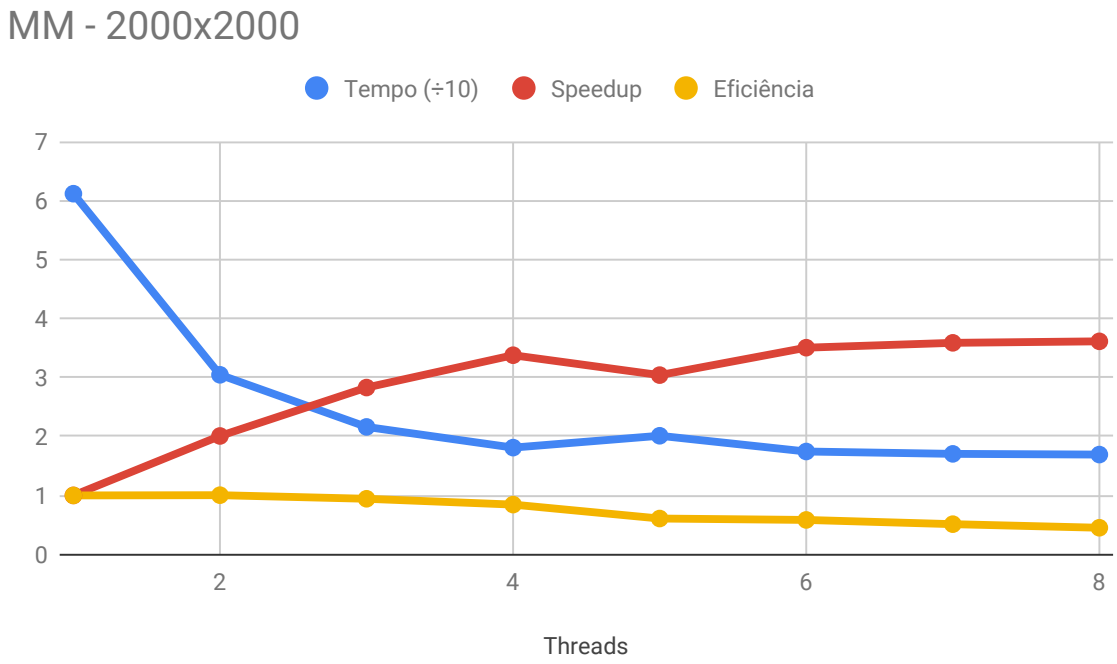

Figure 5: Multiplicação de Matrizes 2000x2000

Conclusões

É possível observar comportamentos interessantes nos resultados obtidos. De imediato, percebemos que o comportamento de aceleração e de eficiênciencia da aproximação de pi é muito próximo do ideal - o speedup é aproximadamente igual ao número de threads, e a eficiência se mantém próxima de 1, decrescendo pouco.

Já a multiplicação de matrizes mostra perdas de speedup e eficiência depois de se chegar a um pico de menor tempo de execução possível. Esse pico não parece depender do tamanho da entrada do programa (tamanho das matrizes), pois é sempre atingido nas execuções com 4 threads. Imaginamos que isso seja uma limitação causada pela implementação utilizada, não pela máquina em si, visto que a aproximação de pi não tem o mesmo problema, apresentando reduções de tempo de execução com mais threads que 4.