Relatório do Desempenho de Aplicações Paralelas

Miguel Nunes

21/09/2022

1 Algoritmos e Descrição do Ambiente

Foram analisados os algoritmos de Mandelbrot e Multiplicação de Matrizes, com apenas leves modificações na entrada de dados e alocação de memória para facilitar a automatização dos testes. Para o algoritmo de Mandelbrot, foram utilizadas os valores:

```
max_row    = 340
max_column = 1000
max_n      = 4600
```

Pois foi observado experimentalmente que, em execuções sequenciais do algoritmo, essas entradas levavam, em média, 64 segundos para serem processadas.

Já para o algoritmo de Multiplicação de Matrizes, a entrada `n = 2340` foi utilizada, pois, análogo ao caso anterior, foi observado que essa entrada levava, em média, 54 segundos para ser processada.

Os testes foram realizados num computador com as seguintes especificações:

CPU `AMD Ryzen 7 5700G with Radeon Graphics (16) @ 4.673GH`

Memória `63588MiB`

OS `Fedora Linux 36 (MATE-Compiz) x86_64`

Kernel `5.18.19-200.fc36.x86_64`

Os testes foram executados automaticamente a partir de um script em bash, com o auxílio de um makefile. As entradas para os testes eram geradas pelo makefile, ao início de uma nova rodada de testes, todos os arquivos gerados no teste anterior são deletados.

2 Coleta dos Dados

Ao fim da execução dos algoritmos, o tempo levado para realizar seu cálculo e a quantidade de *threads* utilizada é é imprimida para `stdout`. Nos testes `stdout` era redirecionado para um arquivo de texto por meio de operações de *pipe* do bash. O nome dos arquivos de saída segue o formato `algoritmo_númeroTeste_númeroThreads`, onde _ é apenas um separador para visualização e não estava de fato no nome dos arquivos. Por exemplo, o arquivo `mandelbrot16` se refere a segunda execução do algoritmo de mandelbrot com 6 *threads*, já o arquivo `matrix58` se refere a sexta execução do algoritmo de multiplicação de matrizes com 8 *threads*.

Foi feito um script em *Python* versão 3.10 para processar esses dados. Os arquivos são lidos "em massa" e seus dados são carregados em estruturas *dict* para serem manipulados. Os dados obtidos são os seguintes:

Threads	Exec 0	Exec 1	Exec 2	Exec 3	Exec 4	Exec 5	Exec 6	Exec 7	Exec 8	Exec 9
1	64.7158	65.21128	64.73953	64.79531	64.67935	64.68465	64.69427	64.66029	64.88569	64.7099
2	33.13408	32.81293	32.88506	32.70995	33.0712	32.92352	33.03085	32.69934	33.02866	32.67746
3	40.49595	40.4585	40.4149	40.46098	40.49784	40.55031	40.53603	40.51463	40.49988	40.4008
4	27.44845	27.21293	27.45597	27.43875	27.26834	27.3451	27.28704	27.28093	27.49284	27.22875
5	27.18402	27.11974	27.0774	27.13122	27.01274	27.07804	27.01961	27.07903	27.11221	27.14181
6	20.75757	20.77389	20.68474	20.81157	20.73202	20.79684	20.76261	20.72167	20.68593	20.75927
7	19.67249	19.67188	19.65971	19.64016	19.71861	19.69955	19.70114	19.65318	19.73501	19.70659
8	16.75083	16.78174	16.77071	16.78651	16.81145	16.74864	16.74882	16.75016	16.79915	16.80264

Table 1: Dados da execução do algoritmo de Mandelbrot

Threads	Exec 0	Exec 1	Exec 2	Exec 3	Exec 4	Exec 5	Exec 6	Exec 7	Exec 8	Exec 9
1	58.19308	58.85921	57.25952	56.50984	57.08937	56.59573	56.55776	56.27447	56.69656	57.30469
2	27.69368	27.36266	27.68316	27.6043	27.63449	27.59407	27.48514	27.69325	27.65521	27.64803
3	18.03399	18.17427	18.11412	18.14828	18.11417	18.18025	17.98689	18.18972	18.00595	18.09148
4	13.51867	13.47603	13.47634	13.47873	13.49899	13.45625	13.61785	13.45564	13.40626	13.47542
5	10.70563	10.71371	10.68173	10.70417	10.70313	10.71331	10.70657	10.74149	10.69732	10.77864
6	8.78966	8.85618	8.8318	8.81042	8.83356	8.88258	8.92194	8.83724	8.88398	8.8047
7	7.48969	7.52113	7.53303	7.58157	7.54399	7.46355	7.46856	7.58963	7.50519	7.47868
8	6.41058	6.43281	6.48943	6.41859	6.42211	6.36786	6.42151	6.43479	6.47839	6.37533

Table 2: Dados da execução do algoritmo de Multiplicação de Matrizes

3 Análise do Dados

Sobre os dados apresentados acima, podemos obter as seguintes informações:

	Média do Tempo de Execução	Desvio Padrão
1 Thread	64.777607	0.16630814071409003
2 Threads	32.897305	0.16732219181035715
3 Threads	40.482982	0.04887138853948963
4 Threads	27.345909	0.10432392396335134
5 Threads	27.095582	0.05337008271390313
6 Threads	20.748611	0.04259400204254115
7 Threads	19.685831	0.03082819193170025
8 Threads	16.775065	0.02461598543404037

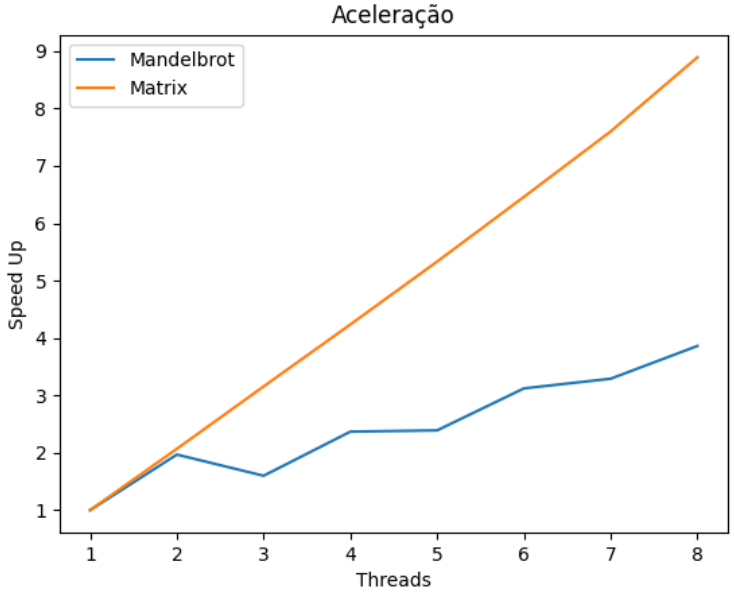
Table 3: Médias e desvio padrão do algoritmo de Mandelbrot

	Média do Tempo de Execução	Desvio Padrão
1 Thread	57.134023	0.8223537073013178
2 Threads	27.605399	0.10552352749242376
3 Threads	18.103912	0.07352478279086958
4 Threads	13.486018	0.054917365964187136
5 Threads	10.71457	0.027061583677070934
6 Threads	8.8452060	0.0411281758517063
7 Threads	7.517502	0.04472313639369324
8 Threads	6.4251400	0.03823902369743929

Table 4: Médias e desvio padrão do algoritmo de Multiplicação de Matrizes

É evidente que o aumento de threads diminui significativamente o tempo de execução dos algoritmos sem causar aumento significativo no desvio padrão.

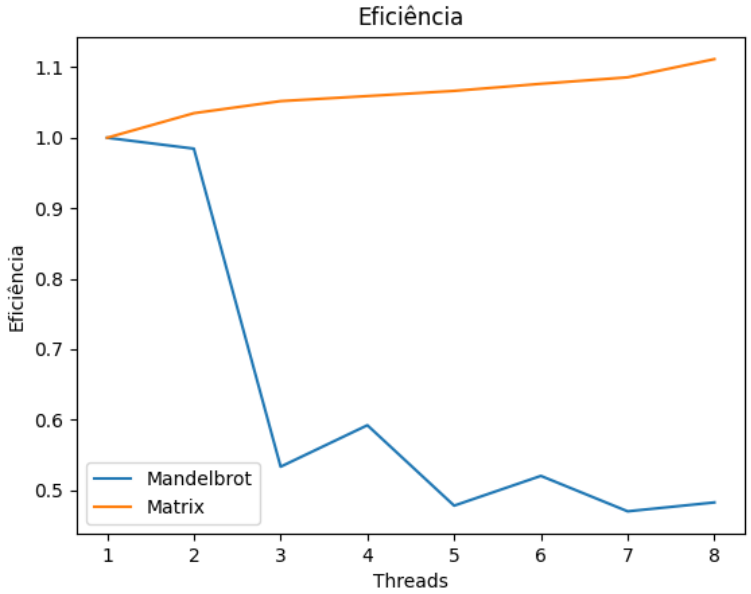
Ademais, os seguinte gráficos de aceleração e eficiência foram elaborados:



É possível observar que o algoritmo de Mandelbrot não teve bom desempenho ao ser paralelizado, apesar de ter um aumento de performance significativo em uma execução com 8 threads se comparada com a execução sequencial, esse aumento não é tão grande se comparado com uma execução com apenas

6 threads. Mais ainda, o desempenho não é consistente, isto é, houveram situações onde execuções com menos threads tiveram desempenho melhor que execuções com mais threads.

Já no caso do algoritmo de Multiplicação de Matrizes, temos um caso ideal, onde o desempenho do programa cresce linearmente com o aumento do número de threads utilizadas.



O algoritmo de Mandelbrot novamente teve um desempenho ruim. O gráfico torna evidente que este algoritmo, na sua implementação testada, não lida bem com paralelização além de duas threads. A baixa eficiência do algoritmo nos casos com mais de duas threads pode ser interpretada como uma indicação que a implementação de paralelismo testada não é a ideal e pode ser melhorada.

Já o algoritmo de Multiplicação de Matrizes teve um desempenho peculiar. Ao contrário do algoritmo de Mandelbrot, sua eficiência continuamente aumentou com a adição de novas threads, não só isso, mas ela esteve acima de 100% em todo os momentos, o que pode ser um indicador de que a implementação, no computador onde foi testada, tem comportamento superlinear.

Após obtidos os dados descritos acima, uma nova rodada de testes foi realizada devido ao comportamento peculiar do algoritmo de Mandelbrot. Nos testes originais, memória era alocada sem ser liberada ao fim da execução e o tempo era medido antes e depois da função `pthread_join`, da seguinte forma:

```
// Codigo foi omitido
start_time = wtime();
for (int tid = 0; tid < thread_amount; tid++){
        // esperando as threads terminarem
        pthread_join(threads[tid], NULL);
        }
end_time = wtime();
```

Esse trecho de código foi movido para uma função diferente, que além de lidar com a finalização de threads, também lida com sua criação e liberação de sua memória após o término da sua execução, da seguinte forma:

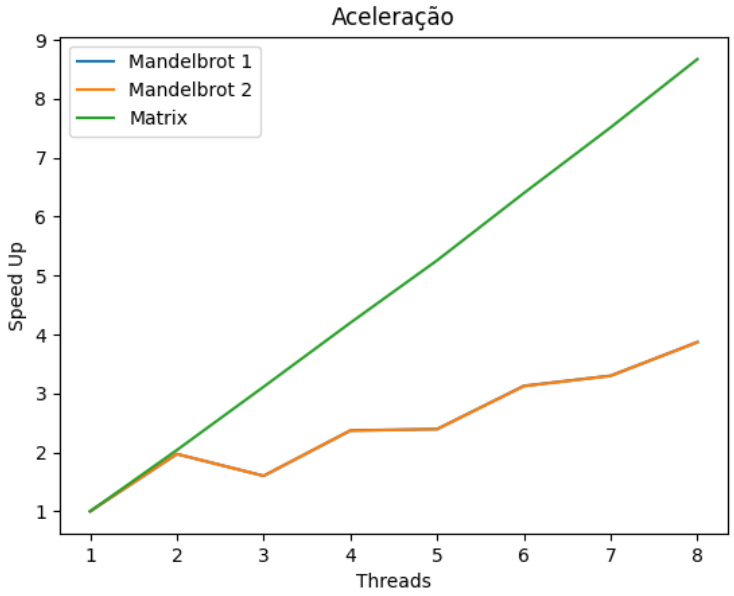
```
// Codigo foi omitido
void handle_threads(pthread_t *threads, thread_param *param, int thread_amount){
        for(int tid = 0; tid < thread_amount; tid++){
                // criando as threads
                pthread_create(&threads[tid], NULL, mandelbrot, (void*) &param[tid]);
        }
        for (int tid = 0; tid < thread_amount; tid++){
                // esperando as threads terminarem
                pthread_join(threads[tid], NULL);
```

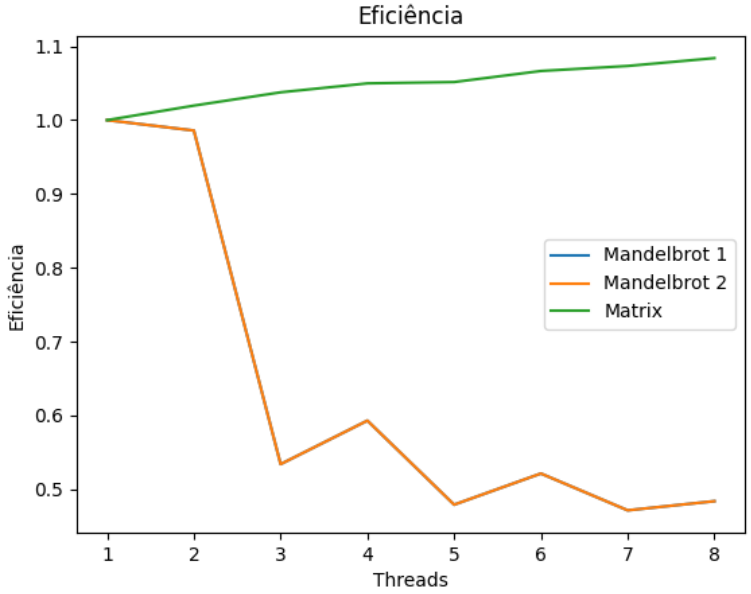
```
        }
        free(threads);
        free(param);
}


        // ...
        // Codigo foi omitido

        start_time = wtime();
        handle_threads(threads, param, thread_amount);
        end_time = wtime();
```

Refazendo os testes com as duas versões do algoritmo de Mandelbrot teve os seguintes resultados, onde "Mandelbrot 1" se refere a versão do algoritmo de Mandelbrot que foi testada na primeira rodada e "Mandelbrot 2" se refere a nova versão:





Threads	Exec 0	Exec 1	Exec 2	Exec 3	Exec 4	Exec 5	Exec 6	Exec 7	Exec 8	Exec 9
1	64.62736	64.67839	64.63666	64.61962	64.6403	64.65312	64.65453	64.63046	64.72165	64.70816
2	32.83472	32.69233	32.79807	32.73635	32.83295	32.82157	32.83873	32.60417	32.87441	32.87749
3	40.2882	40.34767	40.39883	40.27371	40.35982	40.36073	40.38627	40.41132	40.40303	40.34727
4	27.2599	27.19623	27.28722	27.11876	27.28726	27.16293	27.29407	27.29752	27.36482	27.31697
5	26.93793	26.9792	26.91464	26.98675	26.98278	26.94668	27.02919	27.03469	27.01199	27.05143
6	20.63402	20.64316	20.71388	20.6398	20.71072	20.70096	20.65364	20.65875	20.66586	20.67334
7	19.56035	19.57419	19.60095	19.54903	19.59881	19.58438	19.59719	19.62478	19.61138	19.59587
8	16.68858	16.63405	16.70305	16.71212	16.7054	16.70179	16.67909	16.74911	16.75054	16.69679

Table 5: Dados da execução do algoritmo de Mandelbrot

Threads	Exec 0	Exec 1	Exec 2	Exec 3	Exec 4	Exec 5	Exec 6	Exec 7	Exec 8	Exec 9
1	64.78292	64.75616	64.74639	64.73518	64.74628	64.73306	64.77348	64.72006	64.81891	64.79415
2	32.80633	32.85529	32.89766	32.69866	32.88976	32.88579	32.89068	32.70192	32.94923	32.73196
3	40.43616	40.45387	40.46876	40.43574	40.46572	40.43019	40.47768	40.45154	40.45898	40.45161
4	27.26963	27.2999	27.43431	27.28458	27.30729	27.30768	27.34535	27.2183	27.50286	27.40373
5	27.09982	27.06084	27.05968	27.04448	27.03954	26.98255	27.06923	27.06201	27.13602	27.06951
6	20.67464	20.73882	20.74125	20.76735	20.7366	20.66983	20.74995	20.78269	20.81839	20.67273
7	19.64159	19.65127	19.64159	19.66322	19.58801	19.59799	19.65135	19.66773	19.62519	19.67771
8	16.72959	16.69829	16.72722	16.76021	16.75628	16.75078	16.70956	16.72717	16.71292	16.79798

Table 6: Dados da execução do algoritmo de Mandelbrot Alterado

Threads	Exec 0	Exec 1	Exec 2	Exec 3	Exec 4	Exec 5	Exec 6	Exec 7	Exec 8	Exec 9
1	56.15589	54.12894	56.12302	55.5218	55.31708	55.64865	54.40723	55.40056	56.47196	56.75455
2	27.59129	27.07759	26.88887	27.01806	27.06087	27.35283	27.44551	27.57996	27.4063	27.19097
3	17.83495	17.83843	17.55545	17.75712	17.90629	18.00604	17.85106	17.88062	17.99675	17.96056
4	13.24683	13.08665	13.12122	13.27832	13.21705	13.20396	13.3324	13.36276	13.20081	13.3456
5	10.41503	10.71177	10.64331	10.47601	10.55589	10.60537	10.47509	10.58808	10.6196	10.6443
6	8.5535	8.80323	8.78748	8.67657	8.53783	8.69524	8.67108	8.69343	8.72068	8.73561
7	7.38204	7.37056	7.40603	7.46451	7.39406	7.42783	7.39396	7.38062	7.38261	7.39095
8	6.36447	6.56002	6.37577	6.35428	6.41281	6.40217	6.36861	6.35494	6.48218	6.43428

Table 7: Dados da execução do algoritmo de Multiplicação de Matrizes

É evidente que as alterações no algoritmo não tiveram impacto significativo no seu desempenho.