Nomes: Pedro Henrique Ferreira Zoz

Análise do Speedup e Eficiência em Diferença dos Quadrados com OpenMP

Introdução:

Aplicações paralelas é um tópico discutido constantemente nos tempos contemporâneos e são de suma relevância para áreas como big data, data mining, inteligência artificial, etc. Elas, no geral, se resumem a utilizar threads concorrentes a fim de garantir uma melhor performance de um determinado algoritmo, aproveitando o desempenho proporcionado pelas arquiteturas multicores.

Neste trabalho é realizado uma análise da performance de um algoritmo de diferença de quadrados de matrizes utilizando a biblioteca open-source de paralelização OpenMP, disponíveis para diferentes linguagens. Neste caso foi utilizado C++17, compilador GCC 13.1.1, em uma máquina com 12 cores, 2 threads cada.

Métodos e Análise:

Foi desenvolvido dois códigos logicamente iguais, porém um utilizando das vantagens da paralelização do OpenMP, e outro não. Esses são parallel.cc e normal.cc respectivamente.
Foi medido o tempo real da execução do normal.cc, assim como o tempo real de diferentes execuções com diferentes números de threads no parallel.cc

Para validar os códigos, foi utilizado o terminal e um exemplo padrão, a partir da entrada do usuário, que pode ser facilmente confirmado, visto na Figura 1. O método foi utilizado para ambos códigos, e pode ser comparado com a Figura 2. Feita no aplicativo Excel.

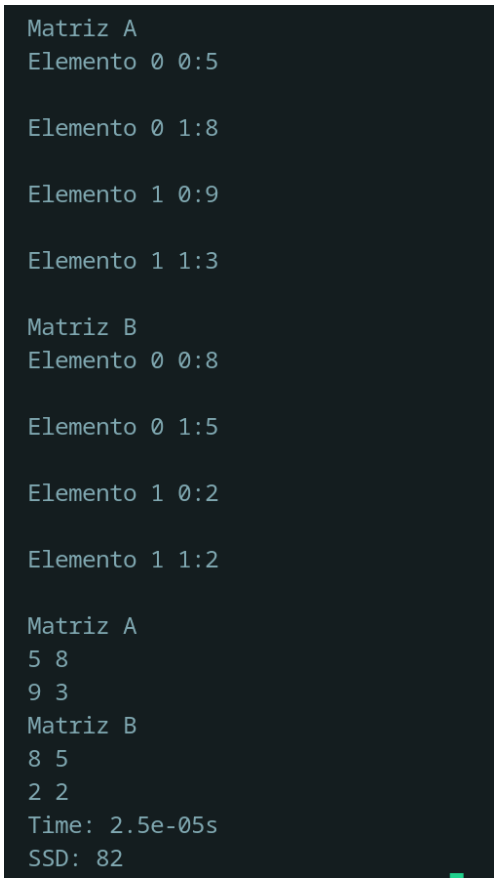

Figura 1: Código de validação em execução.

Figura 2: Validação por Excel.

Os tempos reais coletados podem ser observados na Tabela 1., assim como o cálculo do speedup e eficiência a partir das médias dos tempos.

Sequencial	1	2	4	6	8	10	12	14	16	Threads
11.23	10.386	10.395	10.13	9.961	10.386	10.003	10.107	10.361	10.292	
11.294	10.266	10.341	10.05	9.975	9.957	9.985	9.714	9.753	9.613	
11.203	10.412	10.209	10.1	10.073	10.04	9.563	9.814	9.733	9.711	
11.182	10.581	10.136	10.07	9.997	9.806	9.718	9.692	9.569	9.709	
11.606	10.389	10.152	10.054	10.166	9.825	9.691	9.714	9.792	9.751	
11.327	10.429	10.263	10.188	9.968	9.818	9.659	9.825	9.869	9.591	
11.119	10.368	10.655	10.057	10.047	9.748	9.564	9.84	10.11	9.739	
11.212	10.179	10.619	10.101	10.049	9.617	9.684	9.755	9.78	9.561	
10.78	10.32	10.498	10.098	10.147	9.604	9.699	9.644	9.759	9.621	
11.087	10.29	10.383	10.104	10.34	9.58	9.563	9.741	9.808	9.68	
11.2075	10.377	10.362	10.099	10.048	9.812	9.6875	9.748	9.786	9.6945	Média
1	1.080032765	1.081596217	1.109763343	1.115396099	1.142223808	1.156903226	1.14972302	1.145258533	1.156067874	Speedup
1	1.080032765	0.540798085	0.277440837	0.185899498	0.142777959	0.115690326	0.095810251 68	0.081804180 9	0.072254242 1	Eficiência

Tabela 1: Tabela de Tempos de Execução e cálculos de Speedups e Eficiências. *NOTA: O omp_set_num_threads() como função para definir a quantidade de threads permite '1' thread. Nesse caso não se observa muita diferença entre 1 e 2, imagina-se que talvez o omp realiza 2 threads para a opção de '1' independentemente, resultando nessa possibilidade.*

A seguir o Gráfico 1. mostra o speed up e o Gráfico 2. mostra as eficiências.

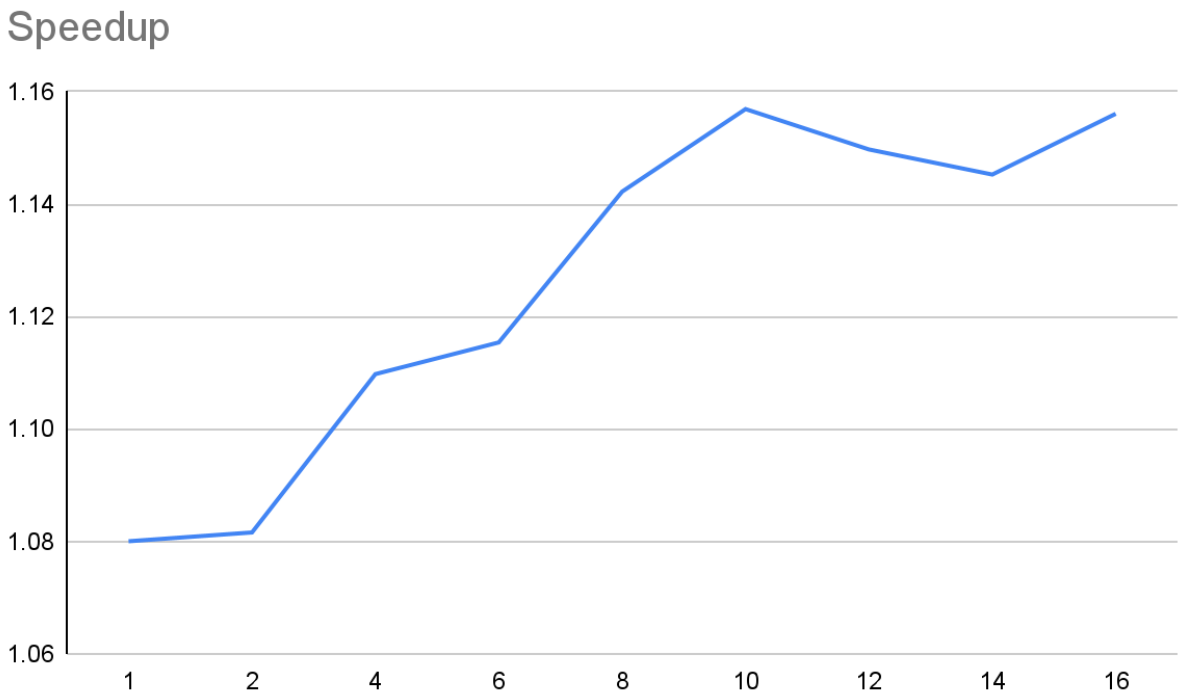

Gráfico 1. Speedups.

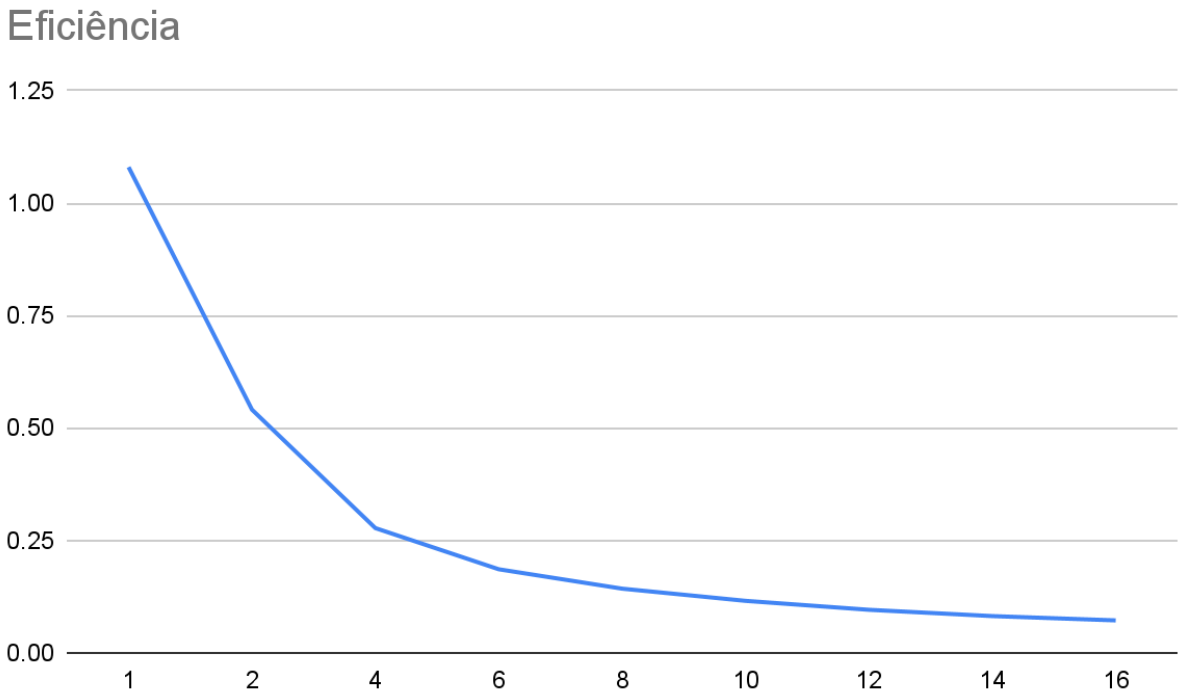

Grafico 2. Eficiências

Conclusões:

Percebe-se que o ápice da paralelização encontra-se em 10 threads, e que o ganho se estabiliza a partir deste ponto. No geral a performance não melhorou drasticamente, porém obteve-se um bom sucesso imediatamente a partir do uso de 4 threads e mais.
Notou-se o declive ocorrido em 12 e 14 threads, indicando desperdício de performance, provavelmente devido a necessidade de sincronizações, ou futilidade geral a aumentar a quantidade de threads. Esses resultados são mais facilmente compreensíveis no gráfico 2, ao entender o declive da eficiência geral no aumento das threads.