ATIVIDADE 2 COMPUTAÇÃO CONCORRENTE

Aluno: Thierry Pierre Dutoit
DRE: 119040432

Junto com este PDF temos 2 arquivos em C, utilizados para obter os dados que serão analisados. Um programa fará uso de computação concorrente para a multiplicação de matrizes e o outro fará de forma sequencial.

Para a análise do programa sequencial foram feitas 5 execuções com matrizes quadradas de dimensão 500, 1000 e 2000, e os resultados foram:

Seq	Inicialização	Multiplicação	Finalização	Tempo Total	Tempo Mínimo
500	0,003398	0,716937	0,000132	0,720467	0,699036
	0,003951	0,701794	0,000144	0,705889	
	0,003877	0,695018	0,000141	0,699036	
	0,003589	0,696814	0,000141	0,700544	
	0,003414	0,701814	0,000141	0,705369	
1000	0,010637	8,726325	0,000611	8,737573	8,4636266
	0,011942	8,529584	0,000552	8,542078	
	0,011334	8,727073	0,000621	8,739028	
	0,0011056	8,461958	0,000563	8,4636266	
	0,011366	8,894747	0,000597	8,90671	
2000	0,034729	74,292617	0,002091	74,329437	72,170885
	0,036254	72,245201	0,001875	72,28333	
	0,034102	72,134739	0,002044	72,170885	
	0,034911	75,285468	0,00213	75,322509	
	0,035368	74,66101	0,002187	74,698565	

Vemos claramente que, ao aumentar a dimensão, aumentamos o tempo de execução do programa. Com dimensão igual a 2000, o programa demora mais de 1 minuto, e 99,9% do tempo gasto é na multiplicação de matrizes. Portanto, o uso de concorrência se mostra fundamental para obter melhores resultados, como:

Primeiro Exemplo -> [dimensão = 500] [threads = 1, 2, 4]

Concorrente	Inicialização	Multiplicação	Finalização	Tempo Total	Tempo Mínimo
500 1	0,003631	0,727021	0,000172	0,730824	0,720157
	0,003435	0,73181	0,00019	0,735435	
	0,003502	0,716483	0,000172	0,720157	
	0,003333	0,721351	0,000241	0,724925	
	0,003048	0,72002	0,000241	0,723309	
500 2	0,003162	0,369056	0,000262	0,37248	0,364912
	0,003577	0,388402	0,000303	0,392282	
	0,003347	0,37638	0,000197	0,379924	
	0,002059	0,362568	0,000285	0,364912	
	0,003262	0,372712	0,000294	0,376268	
500 4	0,002089	0,318213	0,000196	0,320498	0,224611
	0,003033	0,31771	0,000261	0,321004	
	0,003296	0,304375	0,000286	0,307957	
	0,003048	0,221306	0,000257	0,224611	
	0,003241	0,322168	0,000288	0,325697	

Segundo Exemplo -> [dimensão = 1000] [threads = 1, 2, 4]

Concorrente	Inicialização	Multiplicação	Finalização	Tempo Total	Tempo Mínimo
1000 1	0,010991	9,201398	0,000767	9,213156	9,213156
	0,011033	9,535099	0,000634	9,546766	
	0,011024	9,371397	0,000849	9,38327	
	0,009875	9,374034	0,000661	9,38457	
	0,011469	9,431009	0,000789	9,443267	
1000 2	0,009932	4,861604	0,000814	4,87235	4,573335
	0,010828	4,790761	0,000814	4,802403	
	0,011036	4,561516	0,000783	4,573335	
	0,010646	4,640961	0,000709	4,652316	
	0,010911	4,736761	0,000752	4,748424	
1000 4	0,011108	2,641605	0,001061	2,653774	2,462417
	0,011108	3,104227	0,000849	3,116184	
	0,009174	2,513961	0,001014	2,524149	
	0,012421	3,039972	0,001042	3,053435	
	0,010157	2,451251	0,001009	2,462417	

Terceiro Exemplo -> [dimensão = 2000] [threads = 1, 2, 4]

Concorrente	Inicialização	Multiplicação	Finalização	Tempo Total	Tempo Mínimo
2000 1	0,036047	78,117259	0,003497	78,156803	78,156803
	0,033436	79,303687	0,002354	79,339477	
	0,035011	78,436385	0,002585	78,473981	
	0,034671	78,580032	0,003826	78,618529	
	0,036119	78,52889	0,003674	78,568683	
2000 2	0,032765	38,980925	0,002453	39,016143	38,674792
	0,032978	39,074606	0,00238	39,109964	
	0,036199	39,941936	0,00275	39,980885	
	0,035638	38,858495	0,003946	38,898079	
	0,034277	38,637406	0,003109	38,674792	
2000 4	0,036127	24,86956	0,002481	24,908168	22,931533
	0,036137	26,996272	0,003562	27,035971	
	0,035691	25,298401	0,003828	25,33792	
	0,035388	22,893757	0,002388	22,931533	
	0,035852	24,300389	0,003433	24,339674	

Agora que temos os resultados usando threads podemos comparar os dois programas: com e sem threads. Fazemos essa análise calculando (T_sequencial/T_concorrente) e obtemos os seguintes ganhos de desempenho:

	Dim 500	**Dim 1000**	**Dim 2000**
1 Thread	sequencial: 0,699036 concorrente: 0,720157 Ganho = **0,97**	sequencial: 8,4636266 concorrente: 9,213156 Ganho = **0,92**	sequencial: 72,170885 concorrente: 78,156803 Ganho = **0,92**
2 Threads	sequencial: 0,699036 concorrente: 0,364912 Ganho = **1,91**	sequencial: 8,4636266 concorrente: 4,573335 Ganho = **1,85**	sequencial: 72,170885 concorrente: 38,674792 Ganho = **1,86**
4 Threads	sequencial: 0,699036 concorrente: 0,224611 Ganho = **3,11**	sequencial: 8,4636266 concorrente: 2,462417 Ganho = **3,43**	sequencial: 72,170885 concorrente: 22,931533 Ganho = **3,14**

Por mais que o programa concorrente executando com 1 thread tenha apenas um fluxo de execução (assim como o sequencial) ele é mais lento que o sequencial. Isso se deve ao overhead da criação da thread, sendo mais vantajoso o programa sequencial.

No entanto, ao usar mais de uma thread, observamos um ganho significativo de desempenho do programa. O cpu utilizado nos testes possui limite de 4 threads, então o esperado era um ganho de 2x de tempo ao usar 2 threads, e um ganho de 4x ao usar 4 threads. Isso ocorreria numa situação utópica, onde o processador poderia se dedicar completamente ao programa. Como não é o caso, ao usar 4 threads vemos um ganho mais próximo à 3x do que 4x, indicando que o processador não vai focar apenas no programa. Além disso, como o programa sequencial não possui o overhead da criação de threads, perde-se ganho de desempenho ao comparar o programa concorrente com o sequencial.

No mais, o ganho de desempenho valida o trabalho da implementação de concorrência na multiplicação das matrizes quando estas são de tamanhos grandes.