Prova 2 - Relatório de Soluções Paralelas com OpenMP

Paulo Herique Almeida de Andrade[1]

[1]Universidade Federal Rural do Semi-Árido (UFERSA)

paulo.andrade@alunos.ufersa.edu.br

1. Equipamento Utilizado

Para os teste citados neste relatório o equipamento foi composto de um notebook Acer Nitro V15, equipado com uma CPU Intel(R) Core(TM) i5-13420H, e 16 GB RAM. O processador em questão possui como frequência base 2,10 GHz, chegando até 4,60 GHz em modo turbo. Além disso, o modelo integra 8 núcleos físicos, divididos entre 4 núcleos de desempenho (P-cores) e 4 núcleos de eficiência (E-cores), e 12 processadores lógicos ao todo. Quanto ao sistema operacional, utlilizou-se o Arch Linux, simulado em Windows Subsystem for Linux (WSL), tendo o Windows 11 como sistema hospedeiro.

2. Método do Trapézio

Para a implementação do método do trapézio paralelo, foi conduzida uma análise de desempenho utilizando uma carga de trabalho com $N = 10^8$ (100 milhões) iterações. Esta alta contagem de iterações define um cenário limitado pelo processamento (*compute-bound*), onde o tempo de cálculo é significativamente maior que o custo de gerenciamento das *threads*. Os resultados estão apresentados na tabela 2 e serão discutidos a seguir.

N Threads	Método	Tempo (s)	Speedup	Eficiência
1	Sequencial	0.196378	-	-
1	Reduction	0.193153	1.02x	1.02
1	Manual	0.193213	1.02x	1.02
2	Reduction	0.098189	2.00x	1.00
2	Manual	0.093695	2.10x	1.05
4	Reduction	0.056278	3.49x	0.87
4	Manual	0.056194	3.50x	0.87
8	Reduction	0.046776	4.20x	0.53
8	Manual	0.043914	4.48x	0.56
10	Reduction	0.035864	5.48x	0.55
10	Manual	0.040170	4.89x	0.49
12	Reduction	0.035780	5.49x	0.46
12	Manual	0.037794	5.20x	0.43

Table 1. *Benchmark* do Método do Trapézio em Paralelo

2.1. *Speedup*

O *Speedup* é uma importante métrica de algoritmos paralelos que mede o ganho de velocidade obtido em relação ao tempo sequencial.

Para $P = 2$, observou-se um *speedup* ideal, atingindo 2.10x. Isso indica que, ao dobrar o número de processadores, o tempo de execução foi reduzido pela metade. Para

$P = 4$, o desempenho continuou a escalar bem, alcançando um speedup de 3.50x. Já para $P = 8$ ocorreu uma queda notável no ganho. Embora o número de threads tenha dobrado (de 4 para 8), o speedup aumentou apenas de 3.50x para aproximadamente 4.50x. Isso demonstra rendimentos decrescentes.

Em $P = 10$ e $P = 12$, o sistema atingiu a saturação. O speedup estagnou perto de 5.5x. No método *Reduction* o tempo de execução para 10 e 12 threads foi quase idêntico (0.0358s vs 0.0357s), indicando que adicionar mais processadores não trouxe benefícios.

2.2. Eficiência

A variação de *Speedup* pode ser analisada quando colocada em proporção com a quantidade de *threads*. Esta métrica se chama eficiência ($E = S/P$), que avalia o quanto cada processador está sendo efetivamente utilizado para trabalho útil.

Para $P = 1$ e $P = 2$, a eficiência foi ideal, permanecendo em 100% ou mais. Vale lembrar que valores acima de 100% são geralmente anomalias de medição ou efeitos de cache. Em $P = 4$, a eficiência manteve-se alta, em 87%, mostrando bom aproveitamento dos recursos.

Já para $P = 8$, a eficiência caiu drasticamente, para perto de 55%. Isso indica que quase metade do tempo de CPU estava sendo gasta em overhead (sincronização, troca de contexto) e não no cálculo da integral. Por fim, em $P = 12$, a eficiência foi ainda mais baixa, caindo para aproximadamente 45%, confirmando a saturação do sistema.

2.3. Escalabilidade

Observa-se que a qualidade da escalabilidade parece se limitar a medida que a quantidade de *threads* se aproxima da quantidade de núcleos físicos da máquina. A partir de 8 threads, o sistema passou a usar os núcleos de eficiência junto aos de desempenho, o que explica por que ainda houve algum ganho de velocidade, mas com uma penalidade severa na eficiência. Acima disso, o sistema depende do *Hyper-Threading*, agregando um custo de gerenciamento de *threads* que não foi compensado pelos benefícios e estagnou o desempenho.

3. Filtro de Suavização (Blur)

Seguindo a estratégia de buscar um critério que traga um cenário limitado pelo processamento, foi definido a dimensionalidade da matriz de referência como 2000x2000. Apesar disso, notou-se que o problema é conceitualmente *memory-bound*, ou seja, o maior gargalo certamente será o acesso de memória da matriz. Na tabela 3 estão os resultados que serão discutidos.

3.1. *Speedup*

O *speedup* para a implementação paralela (método Parallel) do filtro de *blur* demonstrou um comportamento sub-linear desde o início. Em $P = 1$, o custo de gerenciamento da *thread* paralela (0.152s) superou o da versão puramente sequencial (0.139s), resultando em um *speedup* de 0.91x. O ganho máximo foi atingido em $P = 12$, com apenas 3.30x.

Em contrapartida, a abordagem de *Cache Blocking* demonstrou um desempenho drasticamente superior. Já em $P = 1$, obteve um *speedup* de 1.95x sobre o sequencial, in-

N Threads	Método	Tempo (s)	Speedup	Eficiência
1	Sequencial	0.139734	-	-
1	Parallel	0.152783	0.91x	0.91
1	Cache Blocking	0.071445	1.95x	1.95
2	Parallel	0.102639	1.36x	0.68
2	Cache Blocking	0.032409	4.31x	2.15
4	Parallel	0.066796	2.09x	0.52
4	Cache Blocking	0.026686	5.24x	1.31
8	Parallel	0.049735	2.81x	0.35
8	Cache Blocking	0.018240	7.66x	0.95
10	Parallel	0.045855	3.04x	0.30
10	Cache Blocking	0.017785	7.86x	0,79
12	Parallel	0.042343	3.30x	0.27
12	Cache Blocking	0.013886	10.06x	0,84

Table 2. *Benchmark* do Algoritmo de *Blur*

dicando que a otimização de localidade de dados por si só já supera a implementação original. O ganho escalou, atingindo 4.31x para $P = 2$ e 7.66x para $P = 8$. O método continuou a escalar bem mesmo com 12 *threads*, atingindo um *speedup* máximo de 10.06x.

3.2. Eficiência

A eficiência ($E = S/P$) do algoritmo de *blur* reflete diretamente o *speedup* observado. A implementação *Parallel* demonstrou baixa eficiência, iniciando em 68% para $P = 2$ e decaindo acentuadamente até atingir seu ponto mais baixo, 27%, em $P = 12$.

A estratégia de *Cache Blocking* reverteu este cenário. Para $P = 1$, $P = 2$ e $P = 4$, a eficiência foi super-linear (1.95, 2.15 e 1.31, respectivamente). A métrica manteve-se alta com mais processadores, atingindo 95% em $P = 8$ (um aproveitamento quase ideal dos núcleos) e permanecendo em 84% mesmo com $P = 12$.

3.3. Escalabilidade

A escalabilidade geral do algoritmo *Parallel* foi consideravelmente inferior à do método do trapézio, tratando-se de uma tarefa intrinsecamente *memory-bound*. A operação de *blur* requer a leitura de vizinhos, gerando alta contenção no barramento de memória. Isso explica por que a eficiência foi baixa, pois o gargalo rapidamente se deslocou do processamento para a largura de banda da memória.

A implementação de *Cache Blocking* resolveu este gargalo. Ao processar a matriz em blocos (tiles) e usar a diretriz collapse(2), a localidade dos dados foi maximizada. Isso garantiu que os dados lidos da RAM fossem reutilizados intensivamente pela cache do processador, minimizando a contenção no barramento. As eficiências super-lineares observadas provavelmente ocorreram pois a otimização de cache é tão superior que supera, mesmo com uma única *thread*, a versão sequencial original.