Comparação de Implementações Sequencial e Paralela no Cálculo do Conjunto de Mandelbrot

Ageu Felipe Nunes Moraes[1], **Érico Meger**[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Paraná (IFPR), Pinhais – PR – Brasil
Departamento de Ciência da Computação - IFPR

Abstract. *The Mandelbrot set is a mathematical fractal known for its complex structure and high computational cost for visualization. This work presents a comparison between sequential and parallel implementations in C++ using OpenMP. Both codes were tested with up to 850,000 iterations on a 1000×1000 image grid. Performance was measured based on execution time and speedup. Results show that parallelization significantly reduced execution time compared to the sequential version, demonstrating the effectiveness of OpenMP in tasks with high computational demand. Future work includes testing different scheduling strategies and GPU-based parallelization.*

Resumo. *O conjunto de Mandelbrot é um fractal matemático conhecido por sua estrutura complexa e pelo alto custo computacional necessário para sua visualização. Este trabalho apresenta uma comparação entre implementações sequencial e paralela em C++ utilizando OpenMP. Ambos os códigos foram testados com até 850.000 iterações em uma grade de 1000×1000 pixels. O desempenho foi medido com base no tempo de execução e no cálculo de speedup. Os resultados mostram que a paralelização reduziu de forma significativa o tempo de execução em relação à versão sequencial, evidenciando a eficiência do OpenMP em tarefas com alta demanda computacional. Como trabalhos futuros, destaca-se a exploração de diferentes estratégias de escalonamento e o uso de paralelização em GPU.*

1. Introdução

Fractais são estruturas matemáticas que apresentam padrões autorreplicáveis em diferentes escalas. O conjunto de Mandelbrot, um dos fractais mais famosos, é definido a partir da iteração da função complexa $z_{n+1} = z_n^2 + c$. Sua visualização exige um grande número de cálculos para cada ponto do plano complexo, o que resulta em alto custo computacional[Vassiliev 2022].

Com o avanço do paralelismo em arquiteturas modernas, técnicas como OpenMP possibilitam distribuir esses cálculos entre múltiplos núcleos de processadores, reduzindo o tempo de execução. Este trabalho tem como objetivo comparar a implementação sequencial e paralela do cálculo do conjunto de Mandelbrot em C++, analisando o impacto da paralelização no desempenho.

A estrutura deste trabalho é a seguinte: a Seção 2 apresenta o referencial teórico, a Seção 3 descreve a metodologia utilizada, a Seção 4 apresenta e discute os resultados obtidos, e a Seção 5 apresenta as conclusões.

2. Referencial Teórico

2.1. Conjunto de Mandelbrot

O conjunto de Mandelbrot é definido pelo comportamento da função $z_{n+1} = z_n^2 + c$, em que z e c são número complexos. Um ponto c pertence ao conjunto se a sequência não diverge para o infinito após infinitas iterações. Na prática, define-se um número máximo de iterações para verificar a divergência.

2.2. Paralelismo com OpenMP

OpenMP é uma API de programação paralela voltada para ambientes multiprocessados, que permite paralelizar loops e trechos de código de forma simples através de diretivas como `#pragma omp parallel for`. Essa técnica é adequada para problemas com alto grau de paralelismo, como o cálculo do Mandelbrot, em que cada pixel pode ser processado independentemente [Vassiliev 2022].

2.3. Métricas de Desempenho

- **Tempo de execução:** mede quanto tempo o programa leva para terminar.
- **Speedup:** razão entre o tempo sequencial e o tempo paralelo ($S = T_{seq}/T_{par}$), representando o ganho de desempenho obtido.

3. Metodologia

3.1. Implementação Sequencial

A versão sequencial percorre cada pixel da imagem 1000×1000 linha por linha e coluna por coluna. Para cada pixel, é feito o mapeamento para o plano complexo, estabelecendo-se a constante c. Em seguida, é iniciada a iteração da função $z_{n+1} = z_n^2 + c$ (com $z_0 = 0$), verificando-se a divergência a cada passo. O processo é interrompido ao atingir o número máximo de 850.000 iterações ou se a magnitude de z ultrapassar o limite de 2.

3.2. Implementação Paralela

A região julgada como altamente paralelizável é o **loop externo** (varredura das linhas da imagem), pois o cálculo de divergência para cada pixel é totalmente independente dos demais, não existindo dependência de dados entre as iterações. Esta característica permite que o trabalho seja distribuído de forma eficiente entre as threads do OpenMP.[Sobral 2018, user41235 and Outofpaper 2017].

3.3. Configuração Experimental

- **Linguagem:** C++
- **Biblioteca paralela:** OpenMP
- **Imagem:** 1000×1000 pixels
- **Máx. iterações:** 850.000
- **Execuções:** 5 vezes por configuração, para média
- **Métrica:** tempo total de execução em milissegundos
- **Código Fonte:** O código completo da implementação sequencial e paralela está disponível no repositório GitHub: `https://github.com/EricoMeger/parallel-mandelbrot`.
- **Hardware:** Processador Intel Core i3-7100; Núcleos/Threads: 2 Cores / 4 Threads; Memória RAM: 8gb DDR4

4. Resultados e Discussão

4.1. Resultados Sequenciais

Tabela 1. Tempos de execução para 850.000 iterações.

Execução	Tempo (ms)
1	626.226
2	629.329
3	626.475
4	626.596
5	625.738
Média: $\sim$626.472 ms ($\approx$ 10min45s).	

4.2. Resultados Paralelos

A implementação paralela foi testada com 2, 4 e 8 threads, utilizando agendamentos dinâmico e guiado. A Tabela 2 demonstra os dados brutos e a Tabela 3 resume os tempos médios de execução obtidos para cada configuração.

Tabela 2. Tempos de Execução Paralela (em milissegundos) para Diferentes Configurações OpenMP.

Threads	Schedule	Chunk	T1	T2	T3	T4	T5
Agendamento Dynamic							
2	dynamic	2	312550	312464	312040	312317	312069
4	dynamic	2	181596	192887	181660	181619	181658
8	dynamic	2	187129	187202	187164	187160	187226
Agendamento Guided							
2	guided	2	311889	311912	311915	311919	311912
4	guided	2	335748	322774	338335	335707	319162
8	guided	2	206783	203418	206389	215196	214031

Tabela 3. Tempos Médio de Execução Paralela para Diferentes Configurações OpenMP.

Threads	Schedule	Chunk	Média (ms)	Média ($\approx$)	Speedup
Agendamento Dynamic					
2	dynamic	2	312.288	5min20s	2.01
4	dynamic	2	**183.884**	**3min06s**	**3.41**
8	dynamic	2	187.176	3min11s	3.35
Agendamento Guided					
2	guided	2	311.909	5min19s	2.01
4	guided	2	330.345	5min50s	1.90
8	guided	2	209.163	3min48s	3.00

4.3. Discussão de Desempenho

A análise teórica do problema reforça a alta escalabilidade da implementação. O Speedup Máximo Teórico para o cálculo do Conjunto de Mandelbrot é regido pela Lei de Amdahl, que estabelece um limite de ganho baseado na fração do código que deve ser executada sequencialmente.

No nosso código, a região de tempo medida (≈626 segundos) consiste quase que integralmente no loop de cálculo iterativo, que é a parte paralelizável. As operações sequenciais (como a configuração inicial das threads e o cálculo final do tempo) são microscópicas em comparação com o volume de trabalho (1000×1000 pixels com até 850.000 iterações). Como a Fração Sequencial (o tempo não paralelizável) tende a zero ($F_{seq} \to 0$):

$$\text{Speedup Máximo Teórico} = \frac{1}{\text{Fração Sequencial}} = \frac{1}{\approx 0} \to \infty$$

Isso significa que o problema de Mandelbrot é classificado como **Embaraçosamente Paralelo** (*Embarrassingly Parallel*), não possuindo um limite teórico imposto pela estrutura do algoritmo. No entanto, o ganho real é limitado pelo overhead prático do gerenciamento de threads e pelo número físico de núcleos disponíveis, o que explica por que o Speedup máximo observado foi de 3.41 (com 4 threads) e não 4.0.

1. **Comparação entre agendamentos (`dynamic` vs `guided`)**
    - Para 2 threads, *dynamic* e *guided* tiveram tempos quase idênticos (∼5min20s), mostrando que com poucas threads a diferença é mínima.
    - Para 4 threads, *dynamic* apresentou melhor média (3min06s) que *guided* (5min50s). Isso sugere que o balanceamento dinâmico por *chunks* pequenos foi mais eficiente neste caso.
    - Para 8 threads, *guided* teve tempo médio (3min48s) ligeiramente pior que *dynamic* (3min11s), mostrando *overhead* maior no escalonamento guiado com muitas threads.
2. **Speedup relativo ao sequencial (∼10min45s)**
    - 2 threads: 10min45s / 5min20s ≈ 2.01 (perfeito, quase linear)
    - 4 threads: 10min45s / 3min06s ≈ 3.41 (não linear, mas bom)
    - 8 threads: 10min45s / 3min11s ≈ 3.35 (diminuição do ganho devido a overhead)

5. Conclusão

O problema do Mandelbrot apresenta alta escalabilidade, e a solução com OpenMP mostrou-se eficiente ao alcançar um **speedup de 3,41** com apenas 4 threads, superando os desafios de balanceamento de carga irregular. Este trabalho implementou e comparou versões sequencial e paralela do cálculo do conjunto de Mandelbrot. Os resultados confirmam que o uso de OpenMP é eficaz para reduzir o tempo de execução em aplicações fortemente paralelizáveis. Apesar disso, o ganho real de desempenho depende de fatores como o número de threads em relação aos núcleos físicos disponíveis, a política de escalonamento adotada e o overhead de sincronização.

Referências

[Sobral 2018] Sobral, B. d. L. (2018). Openmp dynamic scheduling.

[user41235 and Outofpaper 2017] user41235 and Outofpaper (2017). Openmp dynamic vs. guided scheduling.

[Vassiliev 2022] Vassiliev, S. (2022). Parallel mandelbrot with openmp.