Relatório EP 3

1) Experimentos com MPICH

Foram realizados testes com a versão sequencial do código no arquivo “sequential_julia.c” e a versão paralela “1D_parallel_julia.c”, utilizando o MPICH localmente. Mediu-se o tempo de execução para diferentes tamanhos de imagem (n = 500, 1000, 5000) e diferentes quantidades de processos (p = 2, 4, 8, 16).

Resultados Medidos

	Sequencial	**p=2**	**p=4**	**p=8**	**p=16**
n = 500	**0.137**	**0.346**	**0.339**	**0.395**	**0.502**
n = 1000	**0.506**	**0.340**	**0.332**	**0.356**	**0.542**
n = 5000	**12.432**	**5.393**	**4.603**	**3.445**	**1.870**

Os tempos considerados na tabela foram as médias do resultado de segundos totais de 5 rodagens para cada combinação. Além disso, precisei rodar as medições com a flag -–oversubscribed, dado que meu computador não possui 16 cores, retornando “There are not enough slots available in the system to satisfy the 16 slots that were requested by the application”.

Cálculo de Speedup e Eficiência (n=5000)

Para avaliar a escalabilidade, consideraremos o caso n=5000, pois é o mais significativo em termos de carga de trabalho.

Tempo sequencial: Tempo_Sequencial = 12,432 s

Para 2 processos:

- Tempo paralelo (2 processos) = 5,393 s
- Speedup(2) = Tempo_Sequencial / Tempo paralelo = 12,432 / 5,393 ≈ 2,30
- Eficiência(2): Speedup(2) / 2 = 2,30 / 2 ≈ 1,15 (ou 115%)

Interpretação: O programa rodou aproximadamente 2,3 vezes mais rápido que a versão sequencial, ou seja, um ganho de cerca de 130% sobre o tempo original. A eficiência acima de 100% é anômala, provavelmente devido a otimizações de hardware, como cache.

Para 4 processos:

- Tempo paralelo (4 processos) = 4,603 s
- Speedup(4) = Tempo_Sequencial / Tempo paralelo = 12,432 / 4,603 ≈ 2,70
- Eficiência(4): Speedup(4) / 4 = 2,70 / 4 ≈ 0,675 (ou 67,5%)

Interpretação: isso significa que a execução com 4 processos é cerca de 2,7 vezes mais rápida, resultando em um ganho de aproximadamente 170% sobre a versão sequencial. A eficiência de 67,5% indica que o programa utiliza os recursos de maneira razoável, mas já é afetado pelo overhead de comunicação.

Para 8 processos:

- Tempo paralelo (8 processos) = 3,445 s
- Speedup(8) = Tempo_Sequencial / Tempo paralelo = 12,432 / 3,445 ≈ 3,61
- Eficiência(8): Speedup(8) / 8 = 3,61 / 8 ≈ 0,451 (ou 45,1%)

Interpretação: aqui o ganho é de cerca de 3,61 vezes, ou seja, 261% em relação ao tempo sequencial. A eficiência de 45,1% reflete o impacto crescente do overhead de comunicação e sincronização.

Para 16 processos:

- Tempo paralelo (16 processos) = 1,870 s
- Speedup(16) = Tempo_Sequencial / Tempo paralelo = 12,432 / 1,870 ≈ 6,64
- Eficiência(16): Speedup(16) / 16 = 6,64 / 16 ≈ 0,415 (ou 41,5%)

Interpretação: neste caso, a execução com 16 processos tornou o programa aproximadamente 6,64 vezes mais rápido, um ganho de cerca de 564% em relação ao tempo sequencial. A eficiência de 41,5% sugere que o overhead de comunicação e sincronização se tornou significativo, limitando os ganhos adicionais.

Esses valores de speedup indicam uma melhora significativa no desempenho conforme aumentamos o número de processos, principalmente para um tamanho de problema maior (n=5000), sugerindo boa escalabilidade do programa. Em relação a eficiência, vemos uma certa convergência, entretanto, ao aumentar mais o número de processos obtive apenas pioras no tempo, ou seja, o ganho de eficiência atingiu seu limite.

Análise dos Resultados

- Para n=500 e n=1000, o ganho não é tão marcante e, em alguns casos, a execução paralela não diminui o tempo significativamente. Isso é esperado pois, para problemas menores, o overhead de comunicação e sincronização entre processos pode não compensar.
- Para n=5000, observa-se um aumento no speedup conforme o número de processos cresce. À medida que adicionamos mais processos, o tempo total diminui, sugerindo que o problema é mais adequadamente paralelizável

quando seu tamanho é maior. O aumento do speedup com o número de processos indica boa escalabilidade neste caso.

- Mesmo assim, é importante observar que os resultados podem variar dependendo das condições de execução, do hardware disponível e da afinidade dos processos. Em situações reais, raramente se obtém escalabilidade tão linear, e outros fatores, como comunicação entre nós, cache compartilhado ou I/O, podem afetar o desempenho.

Conclusão

A análise dos resultados mostra que, para um tamanho de problema maior (n=5000), o programa obtém ganhos significativos de desempenho com o aumento do número de processos, apresentando speedups consideráveis. Isso sugere que a paralelização é benéfica, especialmente para problemas de maior escala, levando a um melhor aproveitamento dos recursos de hardware disponíveis. Além disso, p=16 parece ser um número ideal de processos para executar o programa, excluindo os casos em que n é muito pequeno.

2) Experimentos com SimGrid

Impacto da heterogeneidade dos nós

Inicialmente realizei o experimento para os nós heterogêneos, rodando com n=500 e p=2 e p=16.

Cluster Heterogêneo

n=500, p=16

- Rank 0: 0.004445 s
- Rank 15: 0.004584 s
- Rank 1: 0.005794 s
- Rank 14: 0.005732 s
- Rank 13: 0.005973 s
- Rank 2: 0.007163 s
- Rank 12: 0.009141 s
- Rank 3: 0.009675 s
- Rank 11: 0.013599 s
- Rank 4: 0.013986 s
- Rank 10: 0.017339 s
- Rank 5: 0.017890 s
- Rank 9: 0.018731 s
- Rank 8: 0.019154 s

- Rank 6: 0.019467 s
- Rank 7: 0.019480 s

n=500, p=2

- Rank 0: 0.111419 s
- Rank 1: 0.113820 s

Para o cluster homogêneo, copiei o cluster em um novo xml e padronizei os speeds para serem speed=”200.0Gf”.

Cluster Homogêneo

n=500, p=16

- Rank 15: 0.004582 s
- Rank 0: 0.004669 s
- Rank 1: 0.005789 s
- Rank 14: 0.005627 s
- Rank 2: 0.006204 s
- Rank 13: 0.006064 s
- Rank 12: 0.009108 s
- Rank 3: 0.009502 s
- Rank 11: 0.013197 s
- Rank 4: 0.013898 s
- Rank 10: 0.017308 s
- Rank 5: 0.017717 s
- Rank 9: 0.018760 s
- Rank 8: 0.019065 s
- Rank 6: 0.019379 s
- Rank 7: 0.019498 s

n=500, p=2

- Rank 0: 0.097156 s
- Rank 1: 0.098242 s

Conclusões Iniciais

1. **Tempos no Heterogêneo**: No cluster heterogêneo, observamos uma maior variação entre os tempos de cálculo dos ranks. Alguns processos terminam bem mais rápido (próximos de 0,004 s), enquanto outros chegam a quase

0,02 s. Isso mostra que há nós mais rápidos e outros mais lentos, o que causa um desempenho global menor, pois o tempo total é ditado pelos processos mais lentos.

2. **Tempos no Homogêneo**: No cluster homogêneo, os tempos são semelhantes aos do heterogêneo em termos de ordem de grandeza, mas a tendência é que a diferença de tempo entre os processos seja um pouco menor, já que todos os nós têm capacidades similares. Porém, como o exemplo é o mesmo problema e configuração (n=500), as diferenças não desapareceram completamente, mas tendem a ser menores ou similares. Provavelmente em cenários maiores a vantagem do homogêneo se tornaria mais evidente.

3. **Efeito do Número de Processos (p)**: Com p=2, os tempos individuais por processo aumentam significativamente. Isso acontece porque cada processo computa uma parcela maior do problema. No cluster heterogêneo, esse aumento é ainda mais sentido por conta do nó mais lento. No homogêneo, a variação entre os dois processos é menor.

Em suma, a heterogeneidade tende a acentuar as diferenças de desempenho entre processos, tornando o tempo total do programa maior devido a presença de nós mais lentos. Já em um cluster homogêneo, mesmo com o mesmo número de processos, espera-se menor desvio entre os tempos individuais.

Observação: Os dados de rank acima foram rodados apenas uma vez

Impacto da Latência

Foram testados dois cenários quanto ao número de processos:

- p=16 processos
- p=2 processos

Em cada cenário, foi medida a duração total do cálculo (tempo de finalização - tempo de início) do processo mais lento (aquele que termina por último), pois este determina o tempo total para conclusão de todo o programa.

Abaixo, sintetizamos os dados a partir dos resultados fornecidos:

p=16

- Latência 10 µs: O tempo máximo de cálculo foi de aproximadamente 2,19 s

- Latência 100 µs: O tempo máximo de cálculo foi de aproximadamente 2,11 s
- Latência 500 µs: O tempo máximo de cálculo foi de aproximadamente 2,08 s

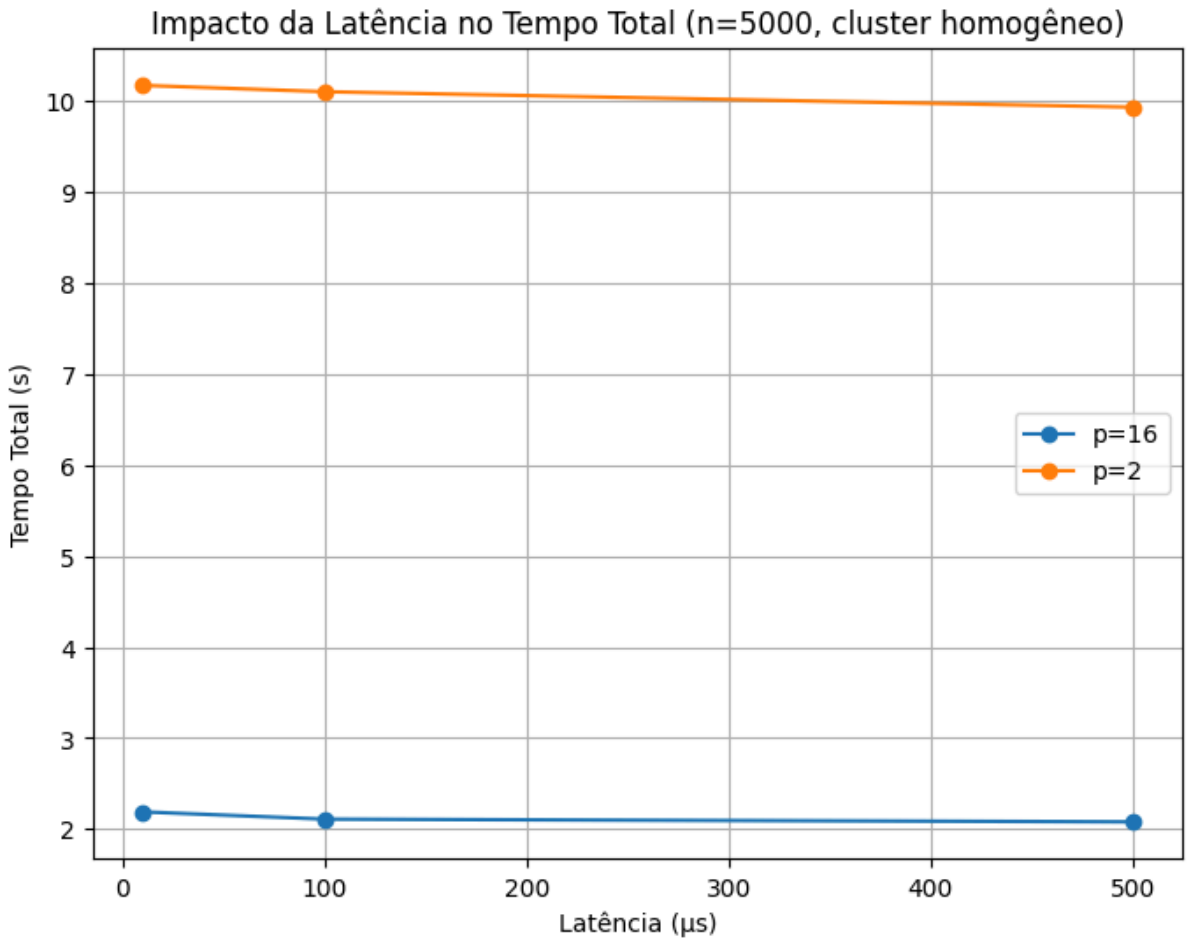
p=2

- Latência 10 µs: Tempo máximo de cálculo ≈ 10,17 s
- Latência 100 µs: Tempo máximo de cálculo ≈ 10,10 s
- Latência 500 µs: Tempo máximo de cálculo ≈ 9,93 s

Análise e Conclusões

Observando os dados, percebe-se que, neste caso, o aumento da latência não resultou em um aumento significativo do tempo total. Na verdade, os tempos medidos variaram de forma não monotônica. Isso pode ocorrer porque o custo computacional é maior que o overhead de comunicação, tornando a variação da latência pouco impactante. Em outras palavras, a computação domina o tempo total.

Para cenários onde a comunicação entre processos fosse mais frequente ou crítica, seria esperado que uma latência maior resultasse em um tempo total maior. Porém, pelo menos considerando os tempos máximos de cálculo, isso não ocorreu expressivamente.



Vemos que o tempo total não diminui com o aumento das latências. Talvez se analisarmos a média ao invés do maior tempo o resultado seria diferente.

Observação: os dados acima são as médias dos maiores valores para 3 rodagens.