
**INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO**

**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**ANDREAS MUNTE FOERSTER – andreas.foerster@usp.br – #7143997**

# COMPARAÇÃO DE TEMPO DE EXECUÇÃO SEQUENCIAL E CONCORRENTE DE SMOOT EM IMAGENS:

**SÃO CARLOS – SP**

**NOV-15**

## SUMÁRIO



## LISTA DE TABELAS



## LISTA DE FIGURAS

# 1. INTRODUÇÃO

O tratamento de imagens é algo presente no cotidiano das pessoas. Aplicativos de celular e câmeras digitas aplicam filtros às imagens da câmera constantemente, softwares de edição de imagens buscam maneiras eficientes de processar filtros com algoritmos pesados em tempo viável para evitar obrigar o usuário a esperar muito tempo. Uma estratégia para processar imagens mais rapidamente é através da distribuição de trabalho para dois ou mais processadores.

# 2. OBJETIVO

Este trabalho tem como objetivo avaliar o ganho de desempenho ao utilizar bibliotecas de OpenMP e MPI para paralelizar um algoritmo simples de smooth para tratamento de imagens um cluster com vários processadores disponíveis.

# 3. MÉTODOS

## 3.1. Smooth

O algoritmo de smooth utilizado realiza a média aritmética de cada pixel da imagem com os 8 pixels ao seu redor, um algoritmo facilmente paralelizável.

## 3.2. Decomposição

Para a divisão de tarefas entre os diferentes nós do cluster, foi utilizada a decomposição de dados na qual a imagem original é ‘cortada’ vertical e horizontalmente em sub imagens procurando fragmentos com o menor perímetro possível. Cada fragmento recebe, então, uma borda adicional contento uma linha de pixels de cada fragmento próximo a ele.

Desta forma, consegue-se produzir fragmentos com o menor overhead e, portanto, tem-se uma menor sobrecarga da rede na hora de transmitir os fragmentos para seus respectivos nós.

Utilizando esta técnica, espera-se, idealmente, obter um tempo de execução inversamente proporcional ao número de processos utilizados.

Para a divisão de tarefas dentro de um mesmo nó, optou-se por uma alternativa mais simples na qual cada thread é responsável por processar um conjunto de linhas da imagem. Uma alternativa seria atribuir cada canal da imagem a uma thread separada, no entanto, isso causaria um desperdício de recursos uma vez que imagens em preto e branco possuem apenas um canal e imagens coloridas possuem 3 canais, número dificilmente divisível pelo número de núcleos que o processador possuirá.

## 3.3. Mapeamento

Tendo em vista que a técnica de decomposição escolhida produz fragmentos de tamanhos iguais ou muito próximos, é esperado que não haja grandes diferenças no tempo de

execução de cada processo, logo optou-se por um mapeamento estático no qual o número de fragmentos gerados é igual ao número de processos utilizados. O processo mestre encaminha um fragmento para cada processo filho e recebe os fragmentos processados de volta.

### 3.4. Coleta de Dados

Para a coleta de dados foi desenvolvida uma versão sequencial do algoritmos, recebendo como argumentos a imagem a ser processada e o arquivo no qual deve escrever o resultado, e uma versão paralela com argumentos adicionais sendo o número de cortes horizontais e o número de cortes verticais a realizar para produzir os fragmentos.

Foram utilizadas versões coloridas e em escala de cinza e quatro imagens com dimensões definidas por:

- Pequeno (2048x1024)
- Média (4096x2048)
- Grande (8184x4096)
- Muito Grande (16384x8184)

Cada Imagem foi processada usando o algoritmo sequencial e paralelo com 4 e 8 recortes. Cada experimento foi realizado 10 vezes para o cálculo dos média e desvio padrão do tempo de execução.

## 4. RESULTADOS

Para coleta dos dados abaixo foi utilizada uma máquina Linux com as seguintes especificaçõe:

```
Nodes (19 hosts / 72 virtuais) – 18 slaves nodes / 1 master
     node
Intel® Core™2 Quad Processor Q9400 (6M Cache, 2.66 GHz, 1333
     MHz FSB)
     8 GB RAM DDR3 Kingston
     Motherboard Gigabyte G41-MT-S2P
     HD 160GB Seagate Sata II 7200RPM
     Fonte ATX 300W Real
Switches (2)
HP V1910-48G (JE009A) w/ 48 ports 10/100/1000Mbps + 4x mini-
     Gbic SFP (RJ-45 or Fiber)
     Switch 3Com 2920-SFP Plus 16 ports Gigabit Switch –
     3CRBSG209
OS: Linux Ubuntu Server 14.04.1 LTS – 64 Bits (Updated:
     28/09/14)
```

**Tabela 1 - Resultados para Experimentos Sequencial e Paralelos**

| Versão | Tempo de Execução (s) | Speedup |
|---|---|---|
| Imagem Pequena (2048x1024) em Preto e Branco | | |
| Seq. | 1.344 +/- 0.029 | - |
| P4 | 2.776 +/- 0.070 | 0.484 +/- 0.016 |
| P8 | 3.313 +/- 0.054 | 0.838 +/- 0.023 |
| Imagem Pequena (2048x1024) em Colorida | | |
| Seq. | 1.630 +/- 0.029 | - |
| P4 | 3.109 +/- 0.022 | 0.524 +/- 0.010 |
| P8 | 3.682 +/- 0.029 | 0.844 +/- 0.016 |
| Imagem Média (4096x2048) em Preto e Branco | | |
| Seq. | 5.163 +/- 0.027 | - |
| P4 | 5.744 +/- 0.020 | 0.899 +/- 0.006 |
| P8 | 5.600 +/- 0.113 | 1.026 +/- 0.021 |
| Imagem Média (4096x2048) em Colorida | | |
| Seq. | 6.576 +/- 0.760 | - |
| P4 | 7.093 +/- 0.034 | 0.927 +/- 0.107 |
| P8 | 7.110 +/- 0.156 | 0.998 +/- 0.117 |
| Imagem Grange (8192x4096) em Preto e Branco | | |
| Seq. | 20.409 +/- 0.179 | - |
| P4 | 17.679 +/- 0.080 | 1.154 +/- 0.011 |
| P8 | 14.923 +/- 0.549 | 1.185 +/- 0.045 |
| Imagem Grange (8192x4096) em Colorida | | |
| Seq. | 26.329 +/- 3.429 | - |
| P4 | 23.133 +/- 0.105 | 1.138 +/- 0.148 |
| P8 | 20.456 +/- 0.378 | 1.131 +/- 0.149 |
| Imagem Muito Grange (16384x8192) em Preto e Branco | | |
| Seq. | 81.590 +/- 1.136 | - |
| P4 | 65.207 +/- 0.410 | 1.251 +/- 0.019 |
| P8 | 50.096 +/- 0.271 | 1.302 +/- 0.019 |
| Imagem Muito Grange (16384x8192) em Preto e Branco | | |
| Seq. | 104.748 +/- 11.491 | - |
| P4 | 85.863 +/- 1.076 | 1.220 +/- 0.135 |
| P8 | 72.154 +/- 1.042 | 1.190 +/- 0.132 |

## 5. DISCUSSÕES

Para melhor avaliação dos resultados, estes serão novamente apresentados em forma de gráficos:

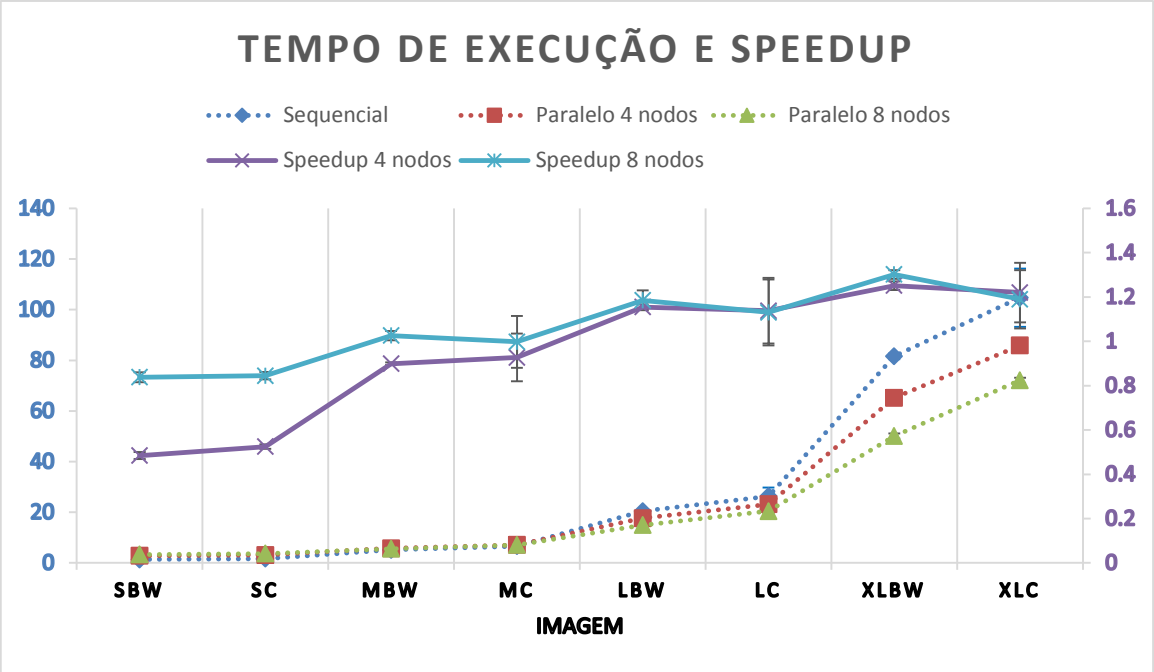


**Figura 1 - Gráfico do tempo de execução e speedup para cada Imagem**

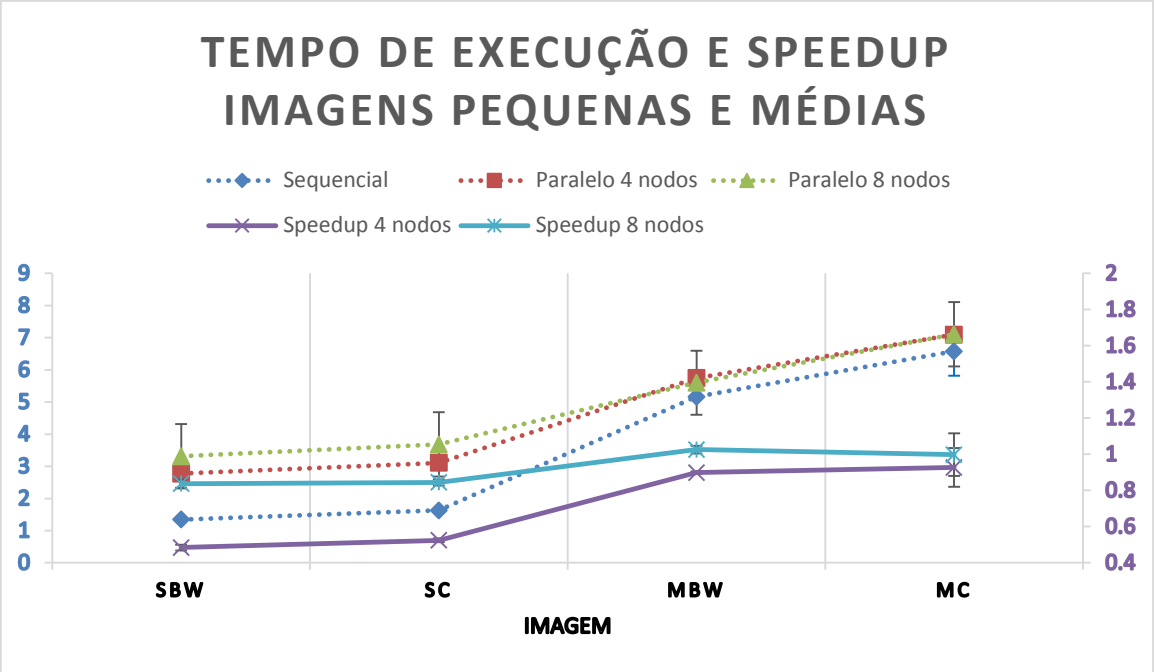


**Figura 2 - Gráfico do tempo de execução e speedup para cada imagens Pequenas e Médias**

Nota-se que para imagens pequenas e médias não há melhorias no tempo de execução devido ao tempo gasto na distribuição de tarefas, envio de dados e remontagem da imagem o ganho de desempenho. Este resultado é esperado, uma vez que as imagens utilizadas não são muito grandes.

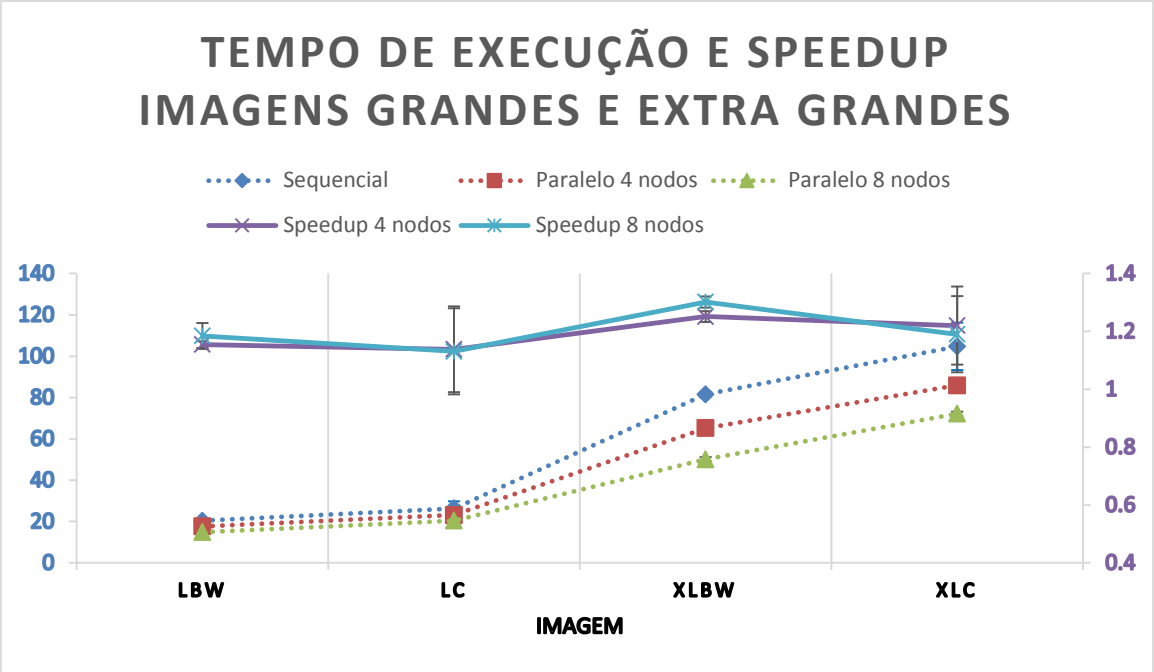


**Figura 3 - Gráfico do tempo de execução e speedup para cada imagens Grandes e Extra Grandes**

Na figura 3 nota-se para imagens grandes e extra grandes há um speedup de cerca de 20%, valor muito abaixo do esperado para estes experimentos. Isto deve-se parcialmente devido ao tempo necessário para transmitir a grande carga de dados para cada processo, mas principalmente devido à estrutura utilizada para o código testado que reparte a imagem completamente antes de enviar os fragmentos.

Outro ponto relevante é o uso do paradigma orientado a objeto que gerou a necessidade de se replicar a imagem a ser enviada duas vezes na memória causando uma perda significativa em imagens grandes.

Outra sugestão de melhoria seria o uso de threads, no processo principal, para produzir fragmentos ao mesmo tempo que envia os que já terminou de recortar da imagem principal, economizando parte do tempo que o processo principal perde bloqueado.

Também vale mencionar que a imagem poderia ser recortada em mais pedaços menores que seriam enviados para os processos filhos cada vez que estes terminassem de processar um fragmento. Esta abordagem, apesar de não necessária para a distribuição de carga, poderia servir para evitar que os processos filhos percam muito tempo esperando pelo seu primeiro fragmento, uma vez que o último filho precisa esperar que todos os outros tenham recebido fragmentos antes de receber seu próprio. No entanto, esta possibilidade precisaria ser testada e não foi abordada neste trabalho.

## 6. CONCLUSÃO

Apesar da distribuição de trabalho para vários processadores utilizando a biblioteca MPI para processar smooth de imagens ter grande potencial, este trabalho não conseguiu reproduzir os resultados desejados devido à implementação do algoritmo não ter considerado alguns

pontos relevantes para o experimento. Mesmo assim, nota-se que há ganho de tempo ao repartilhar o processamento de imagens grandes.

O uso dessa técnica para imagens pequenas deve ser evitado, uma vez que o processamento gasto com a distribuição do trabalho supera o ganho de tempo do processamento em paralelo.

## 7. REFERÊNCIAS

Grama, A., Gupta, A., Karypis, G., & Kumar, V. (2003). *Introduction to Parallel Computing* (2nd ed.). Pearson.

Tanenbaum, A. S. (2006). *Structured Computer Organization* (5ft ed.). Upper Saddle River, New Jersey: Pearson Education.

## 8. APÊNDICE – A: SOBRE O PROGRAMA

O código escrito para este projeto pode ser obtido através de um repositório online do GitHub, assim como as matrizes utilizadas como exemplo e instruções de compilação e uso.

Projeto GitHub:

https://github.com/maglethong/Trab02-Grupo20-A.git

Para compilar os programas execute:

```
mkdir bin
g++ -o ./bin/sequencial ./src/Image.cc ./src/Pixel.cc
     ./src/sequencial.cc
mpiCC -o ./bin/parallel ./src/Image.cc ./src/Pixel.cc
     ./src/parallel.cc -fopenmp
```

Para executar o programa sequencial execute:

```
./bin/sequencial arquivo_de_entrada arquivo_de_saida
```

Para executar o programa paralelo execute:

```
mpirun --host lista_de_nodos -np numero_de_processos ./bin/parallel
     arquivo_de_entrada arquivo_de_saida numero_de_cortes_verticais
     numero_de_cortes_horizontais
```

**Importante**:

```
(V)  numero_de_cortes_verticais
(H)  numero_de_cortes_horizontais
(NP) numero_de_processos
NP = V*H
```

**NP deve ser igual a V * H**. Caso contrário o programa não executará corretamente.

Exemplo:

```
mpirun --host node03,node04,node07,node11 -np 4 ./bin/parallel
     arquivo_de_entrada arquivo_de_saida 2 2
```