EP2 - Filtro de Realce Seletivo

UFABC - MCZA020-13 - Programação Paralela

Emilio Francesquini - e.francesquini@ufabc.edu.br

2025.Q2

Prazo: 31/08/2025

1 O projeto

Neste projeto vamos aplicar conceitos de paralelismo em memória compartilhada (Pthreads e OpenMP) e em GPU (CUDA), comparando desempenho, correção e escalabilidade usando o mesmo problema — um pipeline de processamento de imagem não trivial. Você irá implementar um filtro de realce seletivo em imagens PPM P3. Imagens PPM P3 são um formato de imagem simples, baseado em arquivos de texto puro que podem ser facilmente geradas e visualizadas utilizando aplicações como o Gimp.

O filtro de realce que trabalharemos consiste em 3 etapas que são aplicadas a cada um dos pixels da imagem de entrada:

1. Conversão para tons de cinza (*grayscale*) [1] para calcular um valor de referência local.
2. Desfoque de raio variável (*variable-radius blur*) — para cada pixel, calcule um raio `r_pixel` dependente dos valores originais (R,G,B) do pixel usando uma fórmula baseada em módulo; aplique um blur (média) usando esse raio (inclui vizinhança 2D).
3. Ajuste seletivo (*sharpen*) — aplique um reforço (sharpen) nos pixels que satisfaçam um critério (por exemplo, `R_original > limiar`) combinando o valor original e o valor borrado.

Você deve entregar quatro versões deste filtro:

1. Sequencial: uma versão sequencial do filtro, que será usada como
2. Pthreads: uma versão paralelizada usando Pthreads, com divisão manual do trabalho;
3. OpenMP: uma versão paralelizada usando OpenMP;
4. CUDA: uma para GPU usando CUDA, idealmente otimizada que não utilize simplesmente a memória global para implementar exatamente a mesma ideia usada em Pthreads ou Openmp.

[1] A tradução para *grayscale* engloba tons de cinza, escala de cinza, nível de cinza, nível de cinzento, … Não vou entrar nessa flame war! :D

2 Formato de entrada e saída

A entrada do seu programa será uma imagem PPM P3, e a saída será uma imagem PPM P3 com o filtro aplicado. O formato PPM P3 é um formato de imagem simples, baseado em texto, que pode ser facilmente lido e escrito.

A especificação completa do formato PPM P3 pode ser encontrada em Wikipedia.

Os seus programas devem receber os seguintes parâmetros:

- `input.ppm` — arquivo de entrada (PPM P3).
- `output.ppm` — arquivo de saída (PPM P3).
- `M` - inteiro usado na fórmula do raio variável (detalhes abaixo).
- `limiar` — inteiro `[0..255]` para a etapa de sharpen seletivo.
- `sharpen_factor` — float que controla intensidade do reforço (por exemplo, 0.0 a 2.0).

Exemplo de uma possível invocação do seu programa sequencial:

```
./seq input.ppm output.ppm 7 180 1.25
```

Aqui `M = 7`, `threshold = 180`, `sharpen_factor=1.25`.

Você também pode, é claro, adicionar parâmetros extras a depender de cada implementação e da sua necessidade. Por exemplo:

- Pthreads/OpenMP: número de threads;
- CUDA: número de blocos, tamanho do bloco, etc.

3 O filtro

O filtro proposto é um pipeline com três etapas, as quais descrevemos abaixo.

3.1 Fórmula do raio variável (`r_pixel`)

Seja $S(x, y)$ a soma dos canais originais no pixel:

$$S(x, y) = R(x, y) + G(x, y) + B(x, y),$$

e M um inteiro fornecido pelo usuário. O raio variável para o pixel (x, y) é dado por:

$$r(x, y) = (\, S(x, y) \bmod M \,) + 1$$

Ou seja, o $r(x, y) \in 1, 2, \ldots, M$.

3.2 Blur (média na vizinhança)

Para cada pixel (x,y) raio `r = r_pixel`, canal $C \in R, G, B$, calcule:

$$\text{blur}_{\text{C}}(x, y) = \frac{1}{N} \sum_{i=-r}^{r} \sum_{j=-r}^{r} C(x + i,\, y + j)$$

Onde N representa o número de pixels efetivamente somados (respeitando as bordas). Para o tratamento das bordas da imagem fique a vontade para escolher a sua estratégia. Você pode, por exemplo, escolher entre usar *clamp*, ou seja, replicar a borda (quando $x + i$ for fora da imagem o usar índice válido mais próximo); ou usar um halo preenchido com zeros. Faça sua escolha e explique no relatório.

3.3 Sharpen seletivo

Dado o limiar T (*threshold*) e o fator de reforço α (`sharpen_factor`), o pixel final para o canal C é:

$$\text{out}_C(x,y) = \begin{cases} \text{clamp}(C(x,y) + \alpha(C(x,y) - \text{blur}_C(x,y))), & \text{se } R(x,y) > T \\ \text{round}(\text{blur}_C(x,y)), & \text{caso contrário} \end{cases}$$

Onde clamp(x) é uma função que limita o valor de x ao intervalo $[0, 255]$.

3.4 Conversão para tons de cinza

Após a aplicação do Blur e do sharpen seletivo, você deve finalmente converter a sua imagem em tons de cinza.

Para cada pixel com canais (R, G, B) (valores 0..255 ou normalizados para este intervalo baseando-se nos metadados do arquivo PPM), calcular `Y = round(0.299*R + 0.587*G + 0.114*B)` (ou usar média simples, ou qualquer outro método que preferir[2]).

3.5 Requisitos gerais para sua implementação

1. O seu programa deve ser capaz de ler e escrever arquivos PPM P3.
2. O seu programa deve ser capaz de lidar com imagens de tamanho arbitrário (diferentes larguras e alturas).
3. O seu programa deve ser capaz de lidar com imagens que não possuem múltiplos de 4 pixels (ou seja, larguras e alturas ímpares).
4. O seu programa deve ser capaz de lidar com imagens que possuem canais fora do intervalo [0, 255] (por exemplo, se o arquivo PPM P3 tiver valores normalizados).
5. Não use bilbliotecas externas para ler, escrever ou processar as imagens PPM P3. Você deve implementar a leitura e escrita do formato PPM P3 manualmente.
6. Validar argumentos de entrada e reportar ajuda quando parâmetros faltarem.
7. Implementar verificação de correção: você deve ter uma maneira praa que a saída de cada versão (PThreads/OpenMP/CUDA) possa ser comparada ao seu baseline (sua implementação sequencial). Essa saída não necessáriamente deverá ser exatamente igual (já que a implementação usará números de ponto flutuante), mas deve-se verificar que cada pixel não destoa do outro até um limite.
8. Código bem comentado e modular (funções para leitura, escrita, grayscale, blur, sharpen, etc).

4 A Entrega

A entrega do código deverá ser feita exclusivamente via GitHub Classroom através do link divulgado no site da disciplina. Será considerado como entrega o último *commit* (não esqueça de dar *push*) no repositório até a **data limite de 31/08/2025**.

Para **dúvidas específicas sobre o seu projeto**, seu código ou que contenham informações sensíveis (que você não quer que os outros grupos tenham acesso), crie um issue no seu próprio repositório (e não esqueça de marcar o professor para que ele seja notificado). Você também pode conversra diretamente comigo após as aulas, por e-mail ou durante o horário de atendimento.

[2] A conversão de imagens para tons de cinza não é tão trivial quanto pode parecer. Existem diversos métodos cada um com suas vantagens e desvantagens. Veja esta página da Wikipedia para um resumão.

4.1 Código

1. O seu repositorio deve conter todo o código fonte do seu programa juntamente com instruções claras sobre sua compilação e execução. Veja alguns ótimos exemplos em alguns projetos abertos no GitHub.
2. Programas com **erros de compilação receberão nota 0**.
3. O ambiente de testes será Linux com GCC e NVCC.
4. Programas que não cumprirem os requisitos (resultados incorretos, erros durante a execução, ...) descritos na seção anterior receberão nota 0.
5. Programas sem boa documentação no código terão a sua nota reduzida.
6. Programas desorganizados (nomes de variáveis/funções ruins, falta de identação, ...) terão a sua nota reduzida.

4.2 Relatório

Juntamente com o seu código você deverá entregar um relatório (máximo de 4 páginas) que contenha:

1. Uma explicação detalhada de como o particionamento das tarefas foi feito entre os processos na sua implementação.
2. Tabelas e gráficos do *speedup* e da eficiência para diversos números de processadores e entradas além da comparação entre os métodos de paralelismo (Pthreads, OpenMP e CUDA).
3. Interpretação dos valores mostrados na tabela anterior. Descreva e interprete a variação do *speedup* e da eficiência em relação à ao tamanho do problema e P. Explique os resultados obtidos levando em consideração as características da máquina utilizada.
4. Análise da escalabilidade da sua implementação (cada uma delas). Ela é escalável? Apresenta escalabilidade forte ou fraca?
5. Análise do balanceamento de carga. Todos os processos recebem a mesma quantidade de trabalho? Todos acabam a execução aproximadamente ao mesmo tempo?

Obs.: Certifique-se de executar o seu programa pelo menos 3 vezes e relate o tempo da **execução mais rápida** (*wall-clock time*). Veja a motivação para fazê-lo nas notas de aula e/ou no livro [PP]-Cap. 3.

4.3 Desempenho

1. É esperado que você consiga *speedup > 1* na versão paralela.
2. O teste de desempenho será efetuado em uma máquina Intel Xeon E5 com 16 cores (32 threads) com 64 GB de RAM e uma GPU NVIDIA RTX A5500.
3. Os desempenhos tanto da versão sequencial quanto da versão paralela do seu algoritmo serão avaliados.

4.4 Bônus (Até 2 pontos)

Caso o seu programa seja o mais rápido da turma, ele receberá 2 pontos adicionais de bônus. Caso o seu programa seja o segundo mais rápido da turma, ele receberá 1 ponto adicional de bônus.

O desempenho será comparado pela média aritmética simples entre as 3 paralelizações apresentadas.

4.5 Política de atrasos

Projetos entregues com atraso sofrerão uma redução da nota conforme a tabela abaixo:

Dias em Atraso	Nota Máxima
0	10
1	7
2	6
3	5
>3	0

ATENÇÃO: Para entregas em atraso, um novo repositório será criado cujo link será disponibilizadona página da disciplina.