



Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

Sistemas Embarcados - *Mean Filter*

Relatório

SCC0740 - Sistemas Embarcados
Prof. Vanderlei Bonato e Prof. Eduardo do Valle Simões

Marcelo Kiochi Hatanaka nº 10295645
Rodrigo Mendes Andrade nº 10262721
Marcelo Isaias de Moraes Junior nº 10550218

São Carlos, 23 de Novembro de 2020

Algoritmo - Mean Filter

O mean filter é um filtro de imagem que, a partir de uma janela de pixels de tamanho $N \times N$, substitui o pixel central pela média dos valores dos pixels da janela. A janela desliza sobre a imagem até processar todos os pixels. Para os pixels da borda da imagem, estes podem ser apenas desconsiderados ou pode ser acrescentado um padding de zeros ao redor da imagem para o cálculo da média. Escolhemos a primeira abordagem para simplificar o algoritmo.

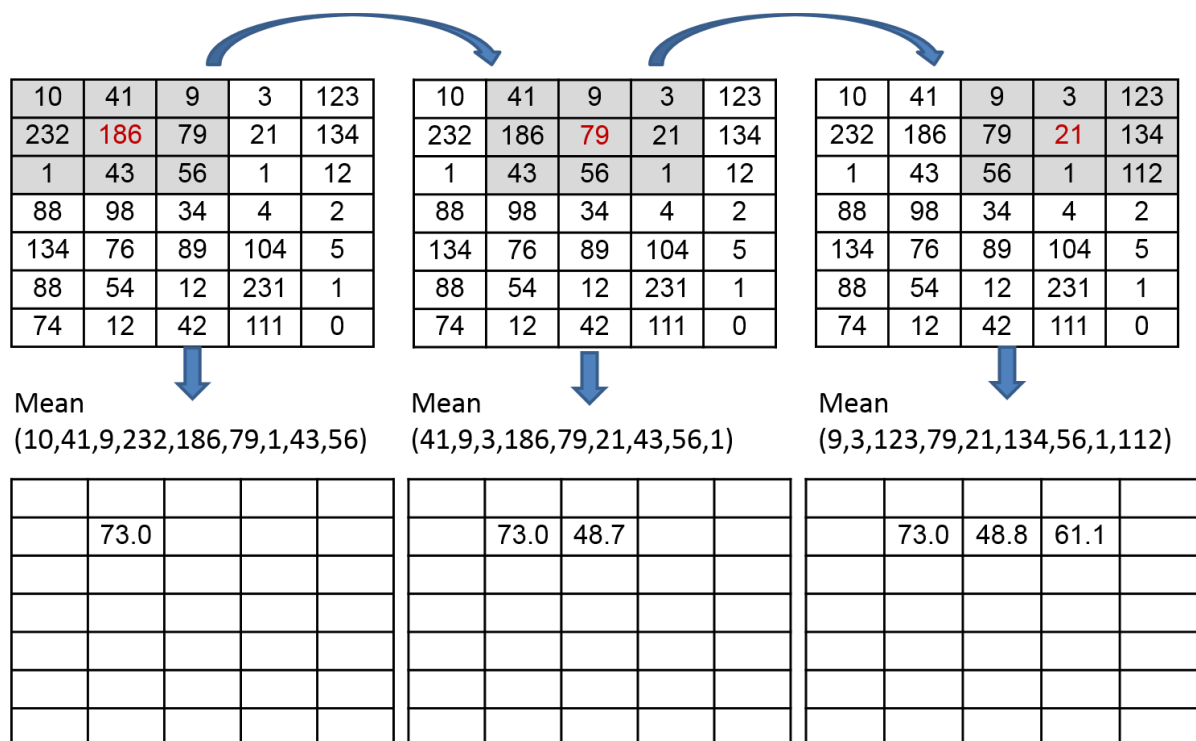


Fig. 01 - Visualização do funcionamento do mean filter com $N=3$. Em cima encontram-se os valores dos pixels da imagem original, enquanto que abaixo encontram-se os valores dos pixels da nova imagem

Implementações

Implementação com minimização de componentes de hardware

A cada ciclo de clock, soma-se o valor de um pixel da janela do filtro. Depois de somar todos os pixels, divide-se a soma total pelo número de pixels da janela e atribui-se o valor ao pixel apropriado na matriz de output. Esse processo é repetido sequencialmente para todos os pixels da imagem.

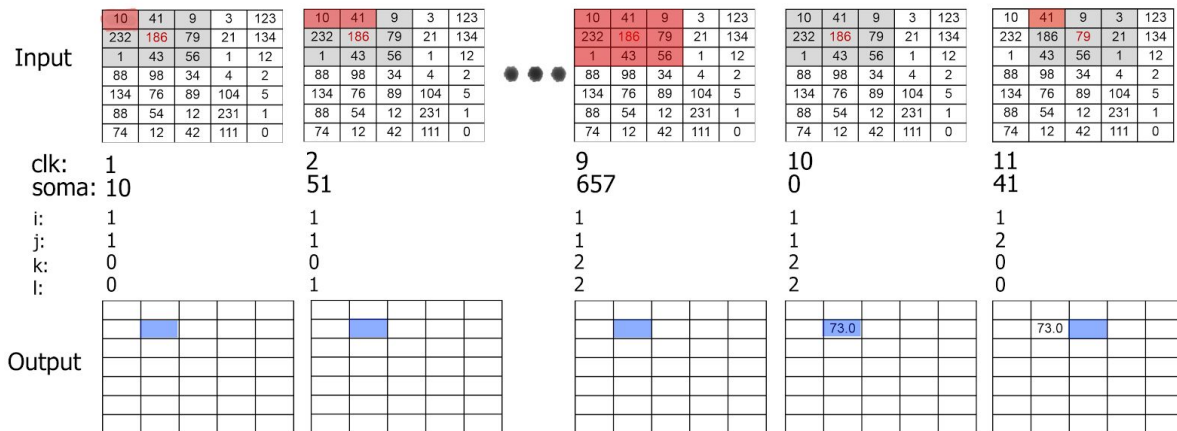
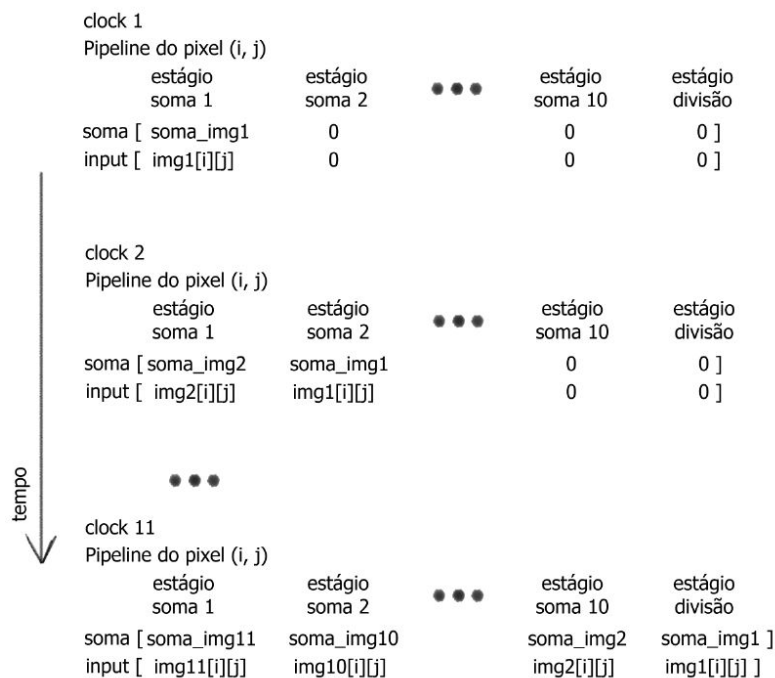
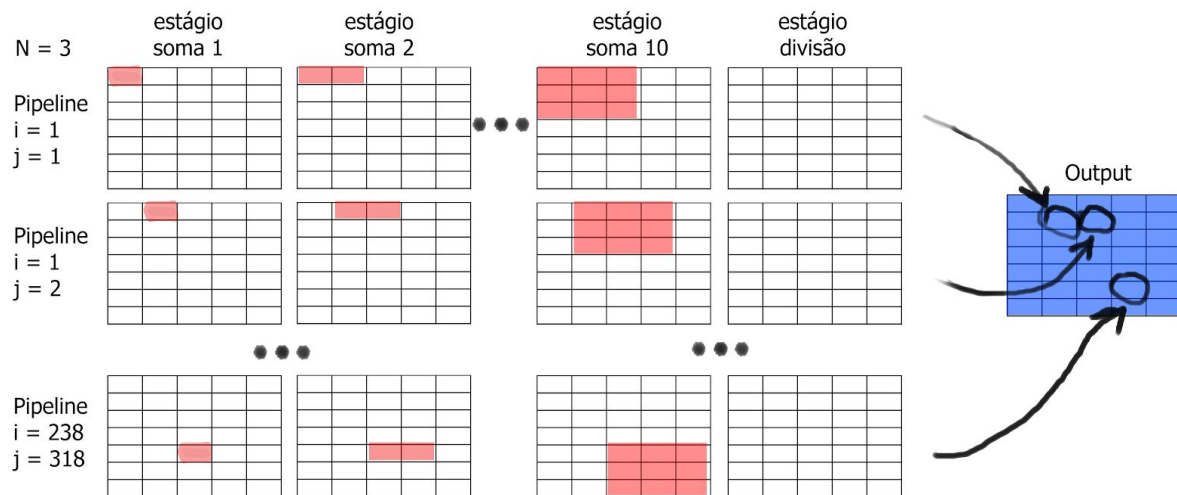


Fig. 02 - Visualização do algoritmo do mean filter para o hardware com minimização de recursos. A variável soma acumula a soma parcial dos pixels da janela atual. As variáveis i e j são índices da imagem. As variáveis k e l são índices da janela do filtro.

Implementação com maximização do Throughput

A ideia da maximização se baseia na implementação de um pipeline para cada pixel da imagem (**320 * 240** pipelines no total). O tamanho do pipeline depende do tamanho da janela do filtro. Considerando uma janela de tamanho **N**, o pipeline possui **N²** estágios de soma (1 para cada pixel da janela), e mais um estágio final para realização da divisão, obtendo-se a média e portanto o valor do pixel sobre o qual o pipeline atua. Em melhores palavras, o pipeline vai acumulando o valor dos pixels da janela e, ao final, obtém a média. Portanto, cada um dos **320 * 240** pipelines possuem **N² + 1** estágios. Considerando apenas uma imagem no pipeline (quase vazio) levará **N² + 1** ciclos de clock para obter a imagem resultante do processo. Agora, considerando quando um pipeline estiver cheio, isto é, (**N² + 1**) imagens, a cada ciclo de clock haverá **1** imagem resultante.



Simulação

Input

Devido a limitações do ambiente de simulação, foi necessário fazer um pré-processamento das imagens de entrada, tornando-as monocromáticas, reescalando-as para as dimensões **320 x 240 pixels** e convertendo-as em uma matriz de pixels de **8 bits**, cujos valores foram divididos entre 4 arquivos de extensão “.txt”. Tais arquivos são utilizados como entrada para o ambiente de simulação. O algoritmo de pré-processamento pode ser executado com o script python em “convert_img.ipynb”

Simulação

Utilizamos o [EDAPlayground](#) para desenvolver e simular o hardware para o projeto, selecionando o simulador Cadence Xcelium 20.09.

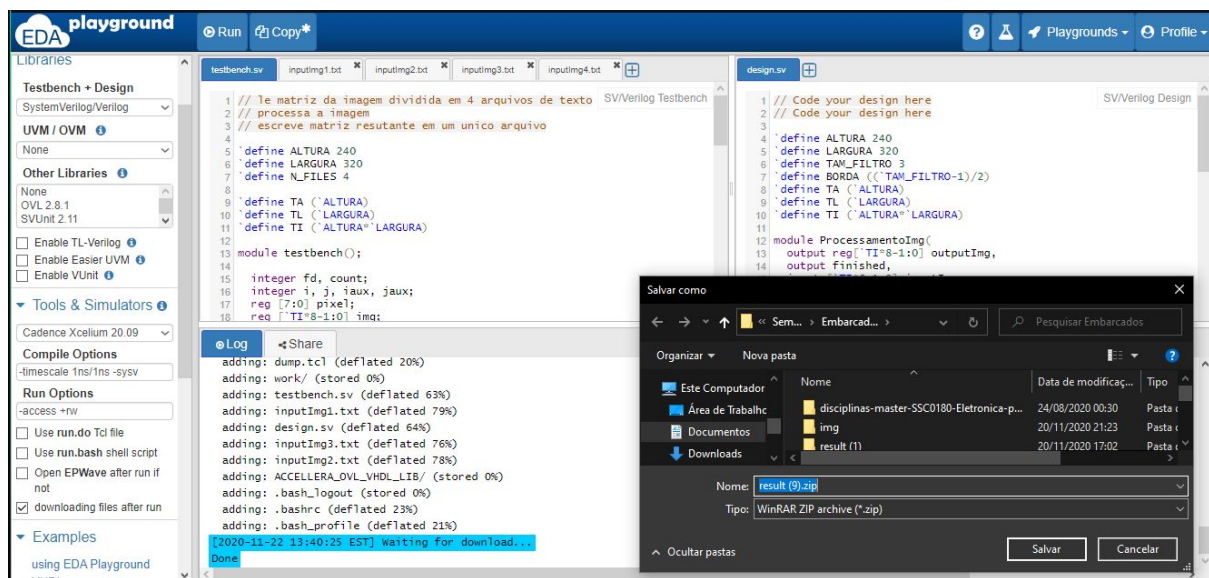



Fig. 01 - Simulação do hardware voltado para minimização de recursos

Output

A imagem gerada pelo hardware é armazenada na forma de um vetor de valores de pixels no arquivo “output.txt” gerado pelo EDAPlayground. A validação do resultado foi feita comparando-se a imagem gerada pelo hardware com a imagem gerada pelo algoritmo mean filter implementado em python.

```
[ ] 1 img3 = filtro_medio(img1, 3)
2
3 plt.figure(figsize=(20, 20))
4 plt.subplot(131); plt.imshow(img1, cmap='gray', vmin=0, vmax=255); plt.axis('off')
5 plt.subplot(132); plt.imshow(img2, cmap='gray', vmin=0, vmax=255); plt.axis('off')
6 plt.subplot(133); plt.imshow(img3, cmap='gray', vmin=0, vmax=255); plt.axis('off')

(-0.5, 319.5, 239.5, -0.5)
```



```
[ ] 1 def root_squared_error(image_ref, image_mod):
2     temp_power = np.power(image_mod.astype(np.float) - image_ref.astype(np.float), 2)
3
4     temp_sum = np.sum(temp_power)
5
6     RSE = np.sqrt(temp_sum)
7     return RSE

[ ] 1 print(root_squared_error(img2, img3))

0.0
```

Fig. 02 - Jupyter Notebook comparando a imagem original (à esquerda) com a imagem gerada pelo hardware (meio) e a imagem gerada pelo algoritmo em python (à direita).

Link para a implementação sequencial (EDA):

<https://www.edaplayground.com/x/h9DG>

Link para a implementação em paralelo (EDA):

<https://www.edaplayground.com/x/agC5>

Análise de Desempenho

Timing

O timing limita a frequência de clock, determinado pelo caminho crítico do circuito.

Devido a problemas encontrados com os sintetizadores da ferramenta, não foi possível determinarmos a medida de timing dos circuitos. Além disso, os diagramas dos circuitos gerados pelo sintetizador Yosys 0.9.0 impossibilitaram que encontrássemos seus caminhos críticos.

Latência

A latência determina a quantidade de ciclos de clock que o hardware demora para executar sua operação.

A implementação que visa a maximização de throughput possui uma estrutura em pipeline cujo número de estágios depende do tamanho **N** do kernel escolhido para o filtro (neste caso, **N=5**). Além disso, a estrutura é paralelizada em relação aos pixels da imagem. considerando **N=5**, tal implementação possui uma latência de **$N * N + 1 = 26$ ciclos de clock**.

Já a implementação que minimiza o uso de recursos demora **$N * N + 1 = 26$ ciclos de clock** para processar 1 pixel da imagem. Considerando uma imagem de **320 x 240 pixels**, tal implementação possui uma latência de **1.996.800 ciclos de clock**.

Throughput

O throughput determina a quantidade de imagens geradas pelo hardware por uma unidade de tempo.

A implementação que visa a maximização de throughput, após o período de preenchimento do pipeline e considerando um clock de período **5 ns**, possui um throughput de 1 imagem filtrada a cada **5 ns** ou **200.000.000 de imagens filtradas por segundo**.

Já a implementação que minimiza o uso de recursos, considerando um clock de período **5 ns** e a latência de **1.996.800 ciclos de clock**, possui um throughput de **100 imagens filtradas por segundo**.