Fernando Cardoso Celho GRR20082084
Luan Haddad Ricardo dos Santos GRR20083152
Jonatas Teixeira GRR20082754

PROGRAMAÇÃO PARALELA

Professor: Fabiano Silva

Problema proposto

Foi proposta uma alternativa para redução de ruídos em arquivos de audio, optamos por manipular arquivos wave, para evitar compactação de dados, que é existente na maioria dos formatos de arquivos de audio.

Soluções sequenciais não foram fáceis de serem encontradas, todas as possíveis soluções utilizavam transformada de Fourrier para obtenção do espectro de frequência do audio e redes neurais detecção de padrões nesse espectro.

Tal solução seria válida, se houvesse alguma já feita sequencialmente, porém não encontramos nenhuma implementação ou estudo sobre redução de ruídos focadas para sinais monodimensionais, e sim sempre focados para bi ou tri dimensionais, como é o caso das imagens. E começar um estudo nesse sentido desfocaria o objetivo do trabalho, que é a paralelização de uma solução sequencial.

Como alternativa às redes neurais simplificamos a solução, sacrificando um pouco o resultado final do método. Implementamos alguns filtros conhecidos, usados no processamento de imagens.

Base da Solução

Decidimos implementar uma versão simplificada de biblioteca para ler e escrever arquivos wave, apesar de existirem bibliotecas de manipulação de som, muitas delas são muito mais complexas do que necessário para o nosso objetivo, ou não funcionam em qualquer plataforma. Além disso, implementando nossa própria biblioteca temos controle de como ela funciona exatamente, e podemos paralelizar qualquer parte da interação com os arquivos wave.

Inicialmente precisamos entender como é essencialmente o arquivo, essa foi a parte fácil, já que o formato é muito simples e bem documentado. E por fim entender exatamente o que significa cada pedaço dos arquivos.

Básicamente um wave é:

[Cabeçalho]

[Samples]

No cabeçalho estão informaçoes como qualidade, rate, tamanho do som e outras pertinentes a leitura do arquivo.

Onde samples é um vetor de bytes ou shorts, dependendo da qualidade do arquivo, separados da seguinte forma para stereo, no caso de mono é apenas um vetor:

[esquerda] - [direita] - [esquerda] - [direita] - …

Os valores dentro desse vetor representam a amplitude da onda naquele instante, que é dado pelo sample rate (em hertz, do cabeçalho). Ou seja, cada segundo tem N valores no vetor, onde N é o sample rate. Estes valores variam de acordo com a qualidade, se for 8 bits, vão de -128 até 127, se for 16 bits de -32.768 até 32.767.

Os lados do stereo são independentes, cada um tem sua própria onda com suas próprias amplitudes e frequências, a única semelhança é a qualidade, sample rate e informações definidas pelo cabeçalho.

Com tais informações foi possível tratar a onda sonora como necessário, passar os filtros e paralelizá-los.

Filtros

1. Filtro gaussiano ou desfoque gaussiano

Trata-se da soma dos seus vizinhos multiplicados pelo seu correspondente no vetor de pesos gaussiano normalizado. Podemos dividir o processo em três passos.

I - Obtenção de um vetor gaussiano definida pela seguinte equação:

$$h[k] = \frac{1}{\alpha * \sqrt{2 * \pi}} * e^{-(\frac{k2}{2 * \alpha 2})}$$

aonde h é o vetor gaussiano.

Uma boa regra empírica é que α deve ser cerca 70% do tamanho da vizinhança do filtro, vizinhança é a quantidade de elementos que teremos a direita e a esquerda do elemento central no vetor.

II - Normalização do vetor gaussino para evitar perca na informação após aplicação do filtro na trilha de audio e pode ser feita da seguinte forma:

$$h[k] = \frac{h[k]}{\Sigma h}$$

aonde Σh é a soma de todos os valores do vetor gaussiano.

III - Aplicação do filtro, percorrer toda a trilha atribuindo a cada elemento da trilha o valor calculado pela soma de todos os seu vizinho multiplicados pelo seu correspondente no vetor de pesos gaussiano normalizado.

2. Filtro mediana

Percorrer toda a trilha de audio atribuindo a cada elemento da trilha o valor da mediana entre os seus vizinhos. Para o calculo da mediana é necessário ordenar os seus vizinho e pegar o valor central.

3. Filtro media

Percorrer toda a trilha de audio atribuindo a cada elemento da trilha o valor da medio calculado entre os seus vizinhos.

4. Filtro passa alta

Um filtro passa-altas é um filtro que permite a passagem das frequências altas com facilidade, porém atenua (ou reduz) a amplitude das frequências abaixo de frequência de corte. A quantidade de atenuação para cada frequência varia de filtro para filtro. O filtro passa-altas possui um princípio de funcionamento oposto ao do filtro passa-baixas.

Percorrer toda a trilha de audio atribuindo a cada elemento da trilha o valor de alfa multiplicado pelo produto do elemento anterior ao atual e a

diferença entre o elemento atual sem alterações pelo anterior sem alterações.

$$y[k] = \alpha * (y[k-1] + x[k] - x[k-1])$$

Solução

Sequencial

Para implementar a solução sequencial apenas escrevemos os algorítimos dos filtros em nossa biblioteca na linguagem C.

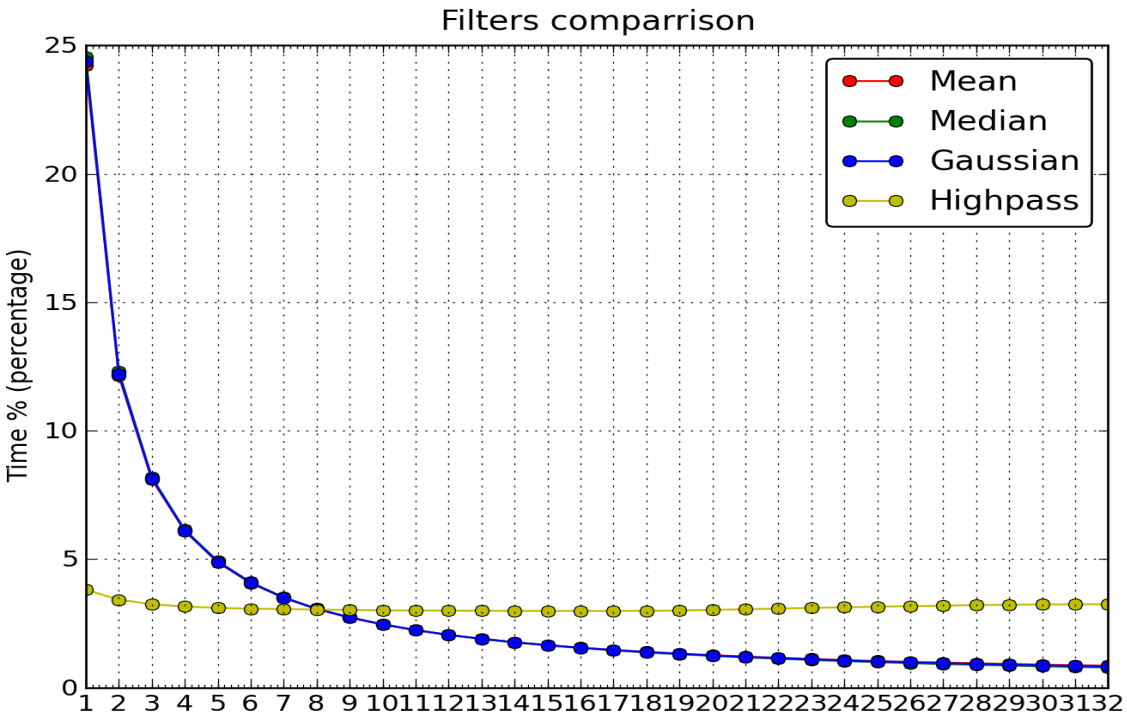
Paralela

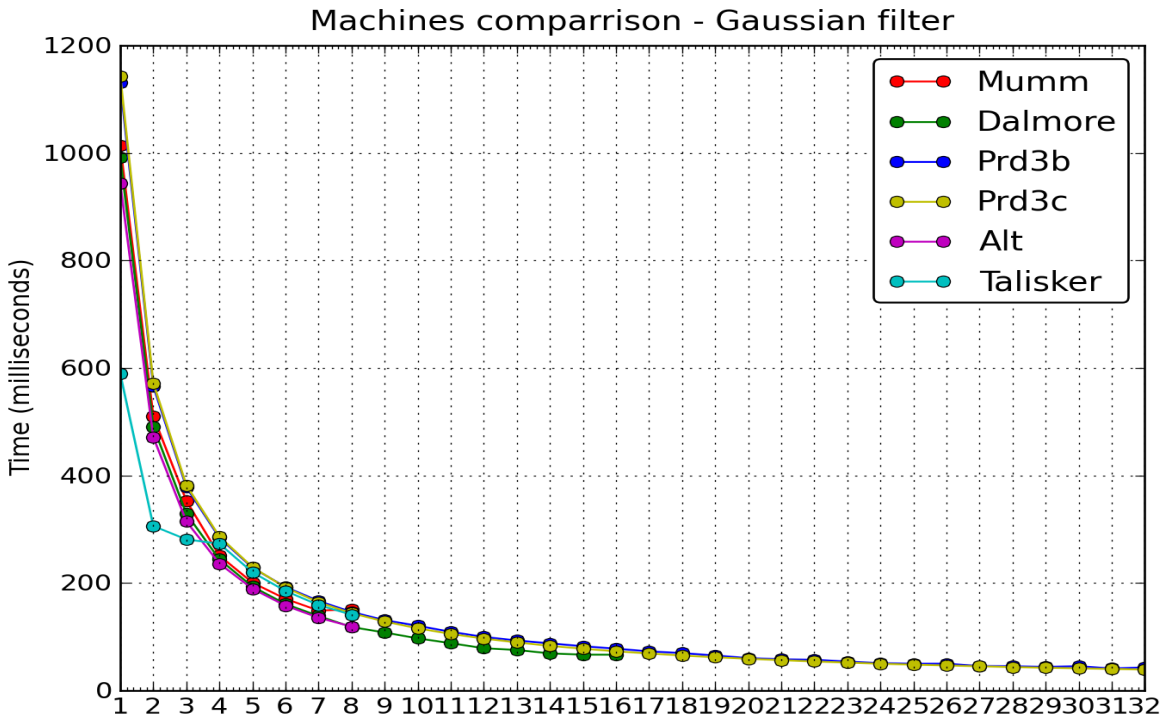
A solução paralela utilizou-se da baixa dependência de dados dos filtros para distribuir os processadores sobre os dados, os resultados foram satisfatórios em OpenMP. Já em CUDA o overhead de cópia de/para o dispositivo foi grande demais, e como os filtros são operações simples sobre os dados, o resultado não foi tão interessante como no OpenMP.

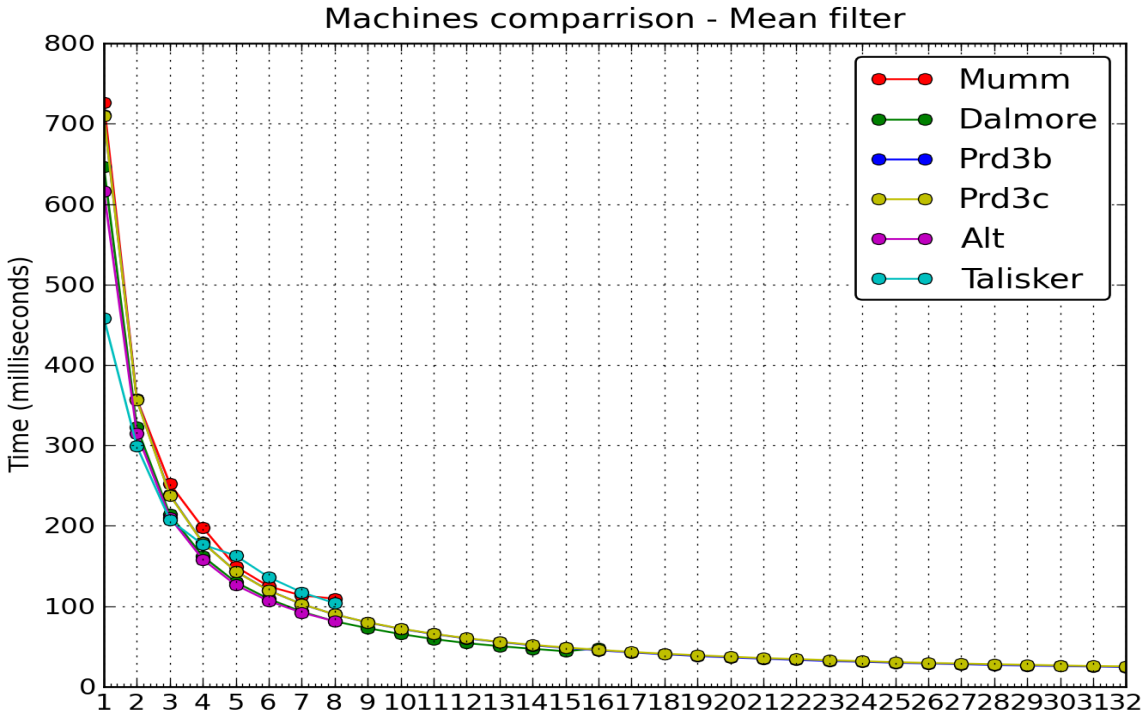
Comparativo Paralelo (OpenMP)

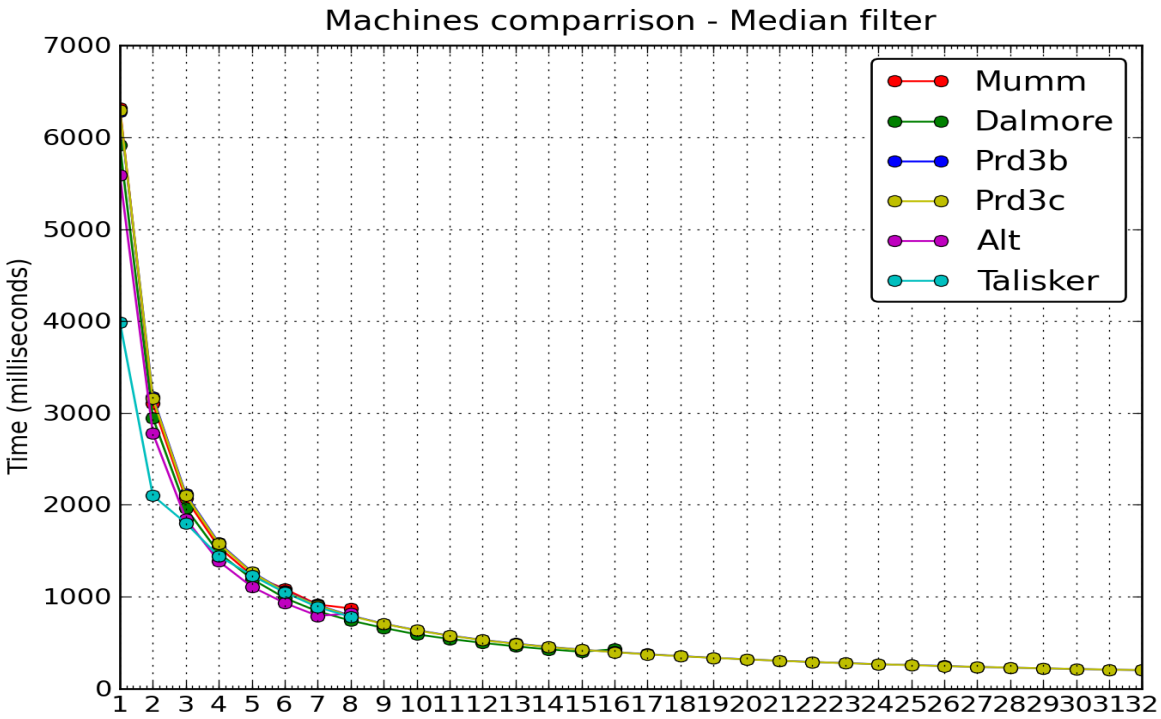
Tamanho de arquivo utilizado: 1.67 Mb

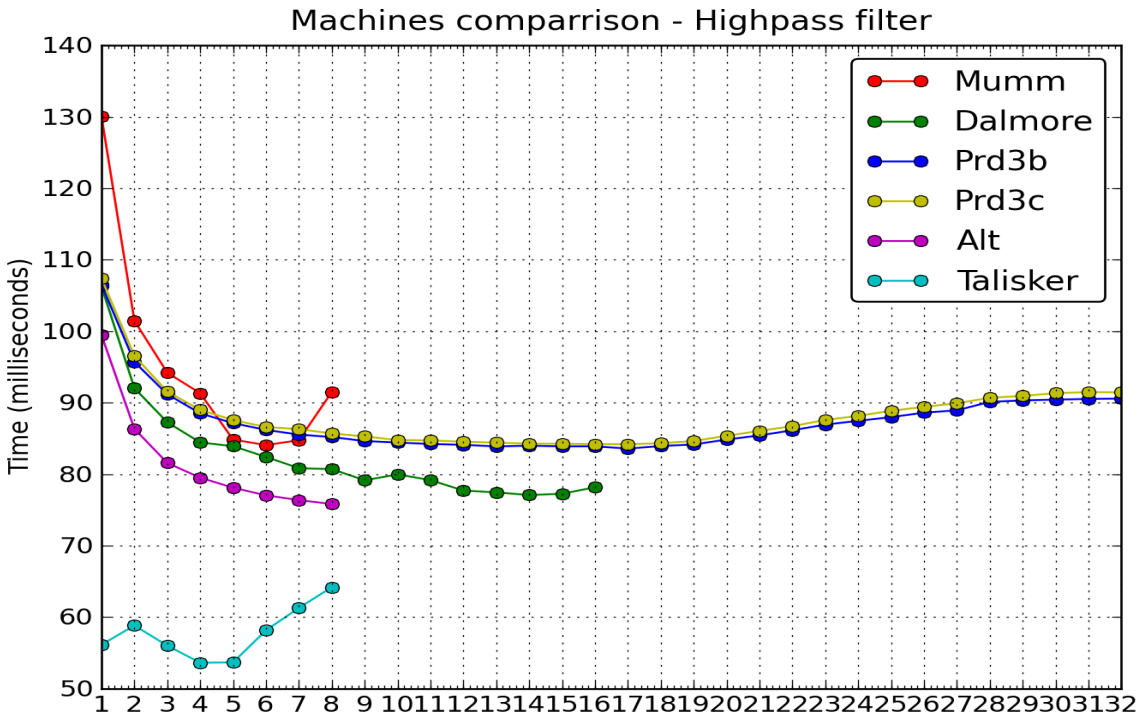
Os gráfico abaixo demonstram a relação de tempo (ordenadas) pela quantidade de processadores (abcissas) utilizada em cada tipo de hardware.




Machines comparrison - Gaussian filter
Time (milliseconds)
Mumm
Dalmore
Prd3b
Prd3c
Alt
Talisker


Machines comparrison - Mean filter
Time (milliseconds)
Mumm
Dalmore
Prd3b
Prd3c
Alt
Talisker


Machines comparrison - Median filter
Time (milliseconds)
Mumm
Dalmore
Prd3b
Prd3c
Alt
Talisker


Machines comparrison - Highpass filter
Time (milliseconds)
Mumm
Dalmore
Prd3b
Prd3c
Alt
Talisker

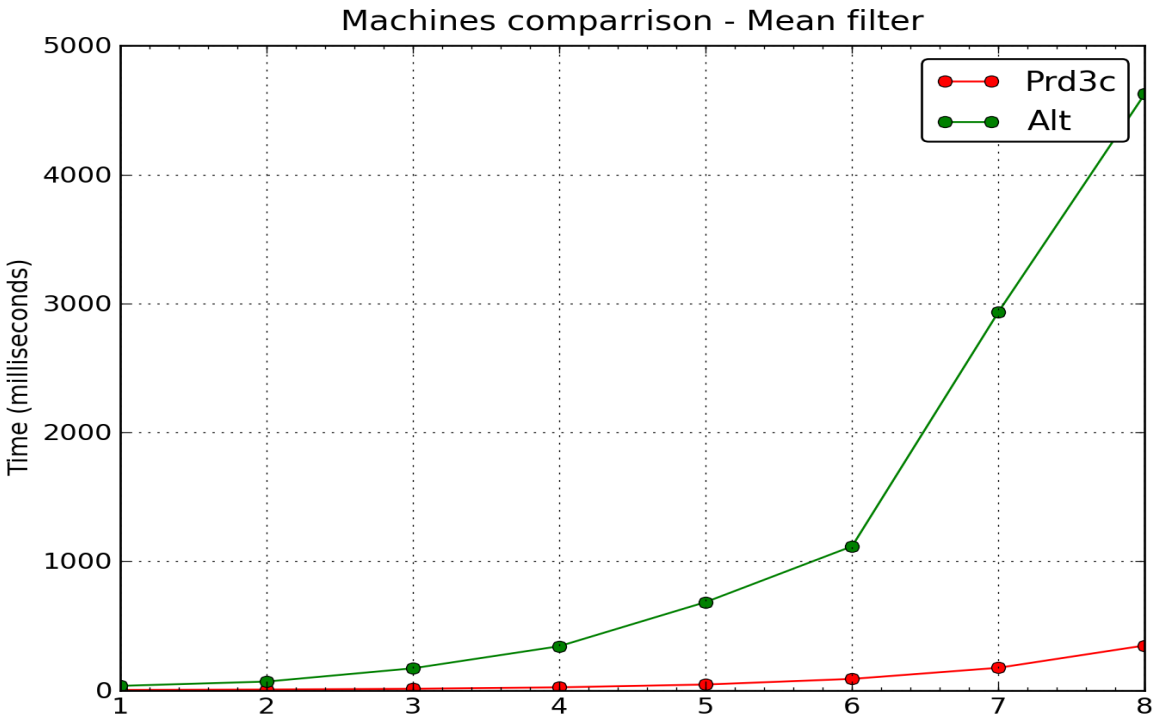
Solução Massivamente Paralela (CUDA)

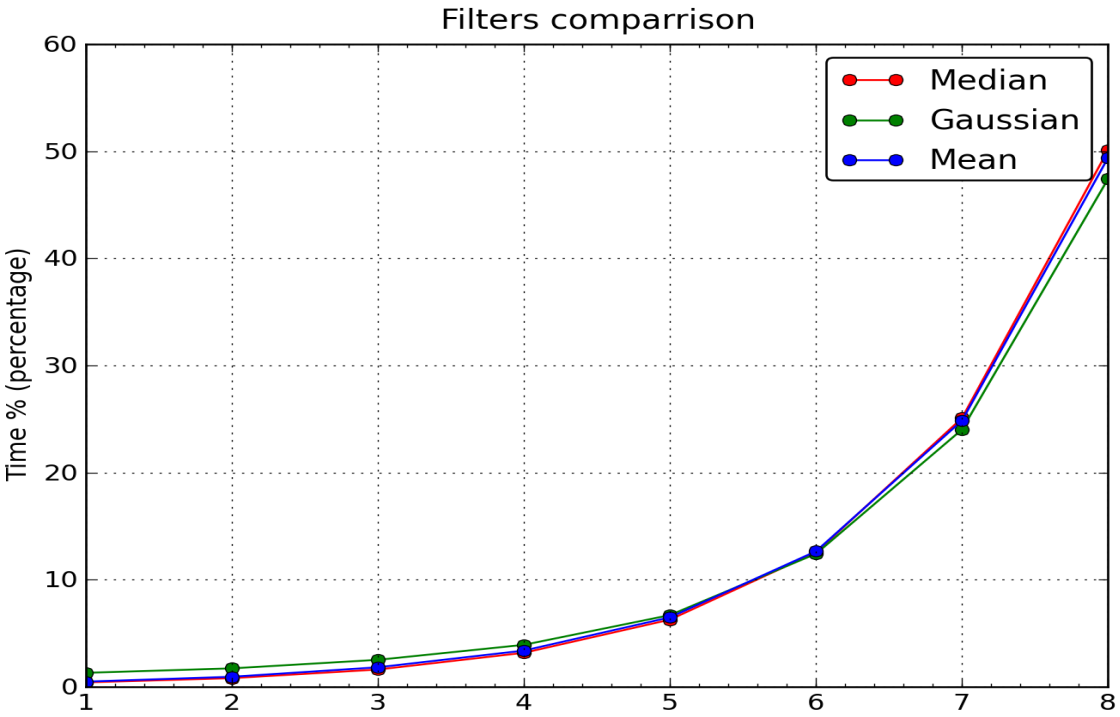
Tamanho inicial do arquivo utilizado: 1.67 Mb

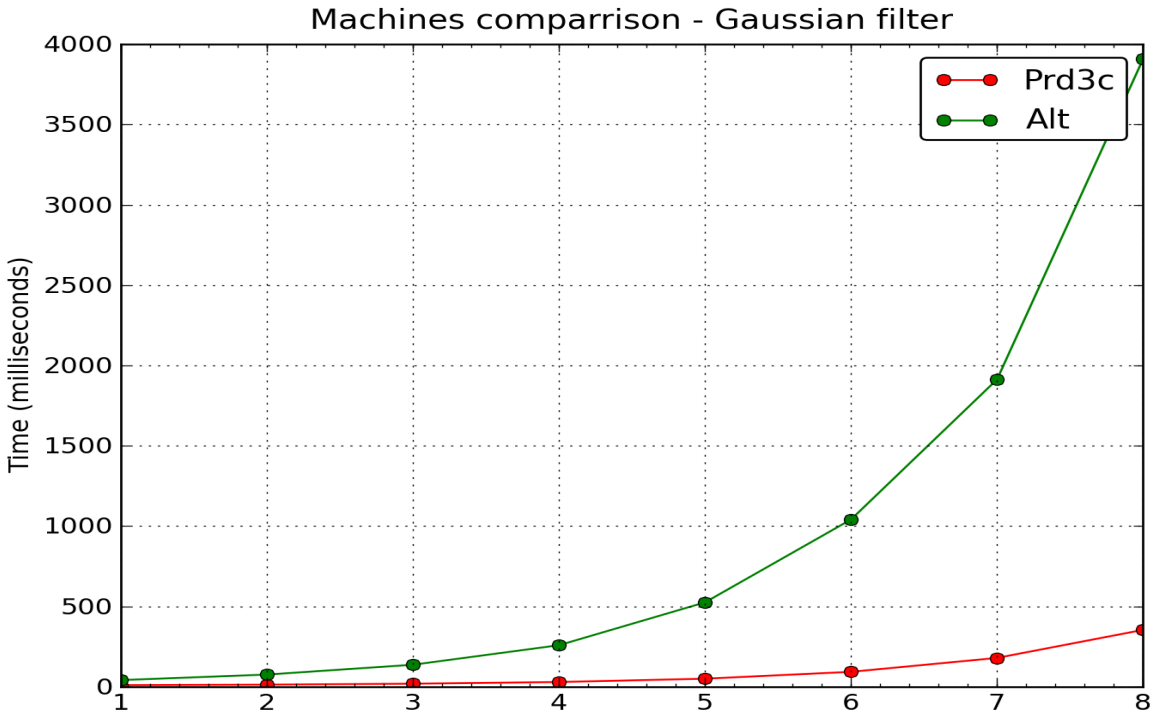
Os gráficos abaixo representam a relação de tempo (ordenadas) pelo tamanho do arquivo (abcissas) que é duplicado a cada iteração.

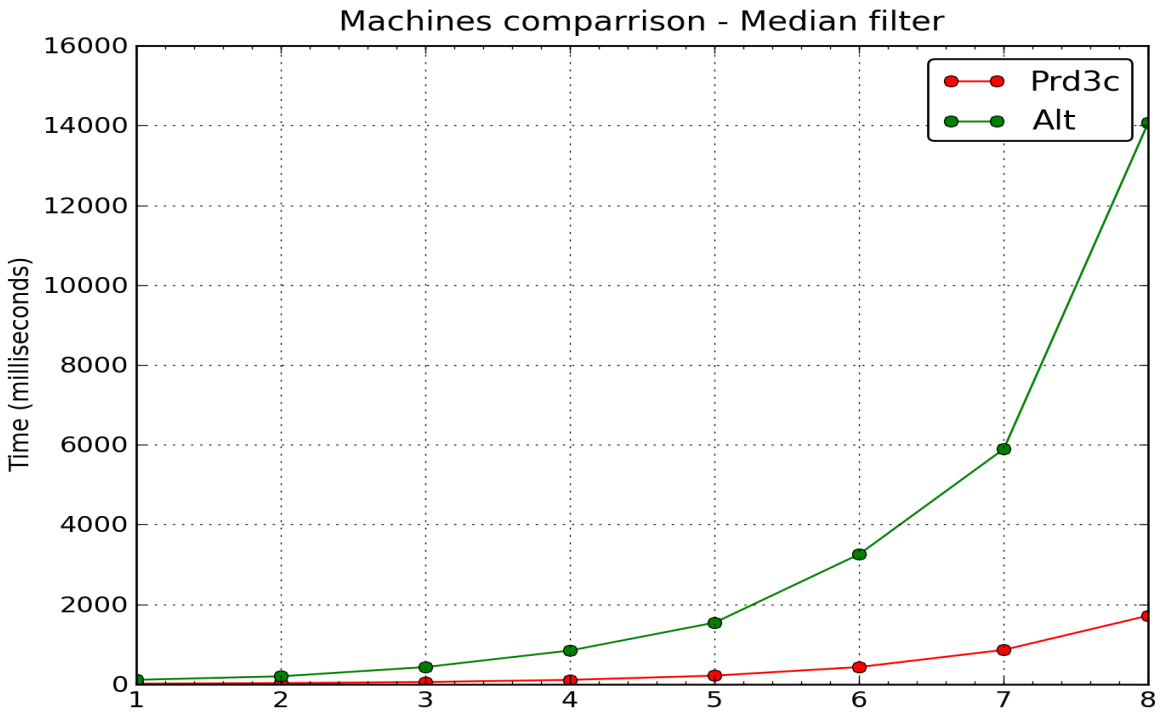
O tamanho do arquivo em um dado instante X é dado por:

$$Tamanho = 1{,}67 * 2^{X}$$




Filters comparrison
Median
Gaussian
Mean
Time % (percentage)
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8


Machines comparrison - Gaussian filter
Prd3c
Alt
Time (milliseconds)
4000
3500
3000
2500
2000
1500
1000
500
0
1
2
3
4
5
6
7
8


Machines comparrison - Median filter
Time (milliseconds)
16000
14000
12000
10000
8000
6000
4000
2000
0
1
2
3
4
5
6
7
8
Prd3c
Alt