# Trabalho de Modelagem de Sistemas

Kernel

Tales Bontempo Cunha

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Introdução

O objetivo deste trabalho é criar um módulo kernel para uma aplicação existente afim de permitir aferições de desempenho de máquinas com maior facilidade.

O programa a ser utilizado foi criado pelo aluno e trata-se de uma aplicação que realiza convolução entre imagens PGM.

# Desenvolvimento

O programa é dotado de uma classe **Image** responsável por armazenar a imagem em um buffer, é dotado de um classe *facade* **ImageIO** responsável por prover mecanismos de leitura e escrita da imagem em disco, uma classe *facade* **ImageFacade** responsável por prover funções de transformações em imagens (para este trabalho, somente o algoritmo de convolução foi criado). O programa como um todo é capaz de realizar duas tarefas: gerar imagens aleatórias dado suas dimensões e de realizar convolução entre imagens.

Avaliando o algoritmo criado para convolução foi criado um programa kernel a parte que recebe dimensões das imagens a ser convoluidas e estressa a máquina a fim de realizar medições de tempo.

|  |
| --- |
| **void ImageFacade::convolution(Image& image, const Image& kernel)**  **{**  **assert(kernel.getWidth()%2 == 1);**  **assert(kernel.getWidth() > 1);**  **assert(kernel.getWidth() == kernel.getHeight());**  **unsigned offset = (kernel.getWidth() - 1)/2;**  **Pixel pixel;**  **for (unsigned i = offset; i < image.getHeight() - offset; ++i)**  **{**  **for (unsigned j = offset; j < image.getWidth() - offset; ++j)**  **{**  **pixel = 0;**  **for (unsigned ik = 0; ik < kernel.getHeight(); ++ik)**  **{**  **for (unsigned jk = 0; jk < kernel.getWidth(); ++jk)**  **{**  **Pixel imagePixel = image[i + ik - offset][j + jk - offset];**  **Pixel kernelPixel = kernel.at(kernel.getHeight() - ik - 1, kernel.getWidth() - jk - 1);**  **pixel += imagePixel\*kernelPixel;**  **}**  **}**  **image[i][j] = pixel <= image.getDepth() ? pixel : image.getDepth();**  **}**  **}**  **}** |

**Código 1:** Código da convolução.

|  |
| --- |
| **void runKernel(unsigned long imageWidth,**  **unsigned long imageHeight,**  **unsigned long kernelWidth,**  **unsigned long kernelHeight,**  **unsigned \*out)**  **{**  **unsigned long image = imageWidth\*imageHeight;**  **unsigned long kernel = kernelWidth\*kernelHeight;**  **for (unsigned i = 0; i < image - 1; ++i)**  **{**  **for (unsigned k = 0; k < kernel; ++k)**  **{**  **// Each cicle has a comparation**  **out[i] = out[i] < i ? 100 : 1000;**  **// Each cycle has to read 3 times**  **for (unsigned c = 0; c < 2; ++c)**  **{**  **out[i] = out[i]\*i;**  **out[i] = out[i] - i;**  **out[i] = out[i] < i ? 100 : 1000;**  **out[i] = out[i] > i ? 100 : 1000;**  **}**    **// Each cycle perform the following operations**  **out[i] = out[i]\*i;**  **for (unsigned c = 0; c < 9; ++c)**  **{**  **out[i] = out[i] - i;**  **}**  **}**  **// for each pixel has to write one time**  **out[i] = out[i]\*i;**  **out[i] = out[i] - i;**  **out[i] = out[i] < i ? 100 : 1000;**  **out[i] = out[i] > i ? 100 : 1000;**  **//for each pixel perform the following operations**  **out[i] = out[i] - i;**  **out[i] = out[i] < i ? 100 : 1000;**  **out[i] = out[i] > i ? 100 : 1000;**  **}**  **}** |

**Código 2:** Código do Kernel.

Todo o código do kernel, da amplicação, scripts e Makefiles além deste relatório pode ser encontrado no GitHub do autor: https://github.com/tabocu/modelagem

# Medições

As medições na aplicação foram realizadas levando em conta somente o algoritmo de convolução e, desta forma, o tempo de CPU foi coletado. Realizou-se medições para imagens de 100x100 até 2000x2000 variando em 100 as dimensões. Para facilitar a coleta, um script bash foi criado a fim de gerar as imagens e executar os programas criados com as imagens geradas. A seguir, os resultados obtidos:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | | **N** | **App** | **Kernel** | | 100 | 24047 | 27804 | | 200 | 103155 | 104441 | | 300 | 245998 | 235613 | | 400 | 435290 | 423856 | | 500 | 691843 | 656483 | | 600 | 987278 | 942607 | | 700 | 1357931 | 1286357 | | 800 | 1786625 | 1740482 | | 900 | 2277994 | 2122589 | | 1000 | 2789745 | 2614669 | | 1100 | 3490462 | 3217565 | | 1200 | 4021167 | 3777761 | | 1300 | 4760817 | 4421167 | | 1400 | 5478695 | 5149361 | | 1500 | 6322555 | 5898200 | | 1600 | 7137749 | 6614288 | | 1700 | 8030818 | 7472006 | | 1800 | 9032981 | 8375638 | | 1900 | 10059915 | 9368007 | | 2000 | 11047445 | 10334265 | |

# Conclusão

Os resultados obtidos foram bastante satisfatórios com pouco desvio ao longo de todo espaço testado.