Sistemas Embarcados - Turma 590 - Projeto Final (Trabalho Prático 2)

O projeto final da disciplina Sistemas Embarcados consiste na implementação paralela e distribuída de um algoritmo de processamento de imagens.

Entrega: O trabalho deve ser realizado em grupos de dois ou três integrantes e entregue via Moodle, juntamente com um relatório explicando a estratégia utilizada para solucionar o problema, o ganho em tempo de processamento, o volume de dados enviado pela rede e as técnicas que foram utilizadas (para resolver problemas de segmentação da imagem, por exemplo). O trabalho será apresentado em aula (pelo Zoom) ou em um video (link fornecido com o relatório) e para isso importante que o grupo se organize para uma apresentação de aproximadamente 10 minutos.

Descrição:

Durante as etapas de processamento, o algoritmo a ser desenvolvido aplica dois filtros sobre cada ponto da imagem, sendo primeiramente aplicado um filtro gaussiano (*blur*) e posteriormente um filtro Sobel (*edge detection*):

(a) (b) (c)

Figure 1: Aplicação de filtros para o processamento de imagens

O ambiente para a elaboração do trabalho consiste no sistema operacional HellfireOS e no simulador MPSoC utilizado em aula. A saída gerada (em um processador usado como mestre / coordenador) deve ser adicionada em um arquivo a ser incluído na ferramenta de geração de imagens *bmp* (*create_bmp.c*) para visualização do resultado do processamento. O resultado final do processamento deve ser idêntico ao obtido pela execução da versão não paralela.

Recomenda-se que os alunos inicialmente familiarizem-se com o material disponibilizado, executando o exemplo e testando as ferramentas de conversão entre *bitmap* (arquivo .bmp) e *hex dump* (saída do processamento obtido no simulador).

São entregues aos alunos arquivos auxiliares para o desenvolvimento e teste da aplicação. Importante lembrar que esses códigos são referência, ficando a cargo do aluno implementar versões mais adequadas a sua solução e também automatizar o processo de compilação e simulação do ambiente.

- HellfireOS (*https://github.com/sjohann81/hellfireos*);
- Simulador MPSoC (disponibilizado juntamente com o HellfireOS, no diretório *hellfireos/usr/sim/mpsoc_sim*);
- Arquivos da aplicação (na distribuição do HellfireOS);

  /hellfireos/platform/noc_3x2/makefile - configuração da plataforma para um MPSoC 3x2 e configuração recomendada do *kernel*;

  /hellfireos/app/img_filter/app.mak - *makefile* para os arquivos da aplicação;

  /hellfireos/app/img_filter/filter.c - fonte do algoritmo monoprocessado com a API do HellfireOS (aplicação embarcada original);

  /hellfireos/app/img_filter/image.h - arquivo contendo a imagem já extraida para um buffer (convertida do arquivo original, utilizada na aplicação embarcada);

- Arquivos auxiliares (*tp2_files.tar.gz*);

  /host/create_image.c - ferramenta para extração de imagem para um buffer para o processamento (utilizada no computador hospedeiro). Essa ferramenta foi usada para criar o arquivo *image.h* a ser utilizado como entrada na aplicacção embarcada;

/host/create_bmp.c - ferramenta para teste da saída resultante do processamento (após os filtros, utilizada no hospedeiro);

/host/filter_image.h - arquivo contendo a imagem processada (após os filtros, saída da aplicação embarcada) em um buffer e pronta para ser remontada;

/host/processing.bmp - exemplo de imagem na qual serão aplicados os filtros;

/host/processing_gauss_sobel.bmp - exemplo de imagem resultante da aplicação dos filtros;

/host/matrix.py - script (em Python) exemplificando como acessar imagens alocadas linearmente na memória como se fossem matrizes.

Executando o exemplo:

Inicialmente deve-se configurar o simulador para a mesma configuração de rede que será usada no HellfireOS (em */usr/sim/mpsoc_sim* digite *make noc_3x2*). O diretório */app/img_filter* contendo aplicação já está presente no diretório de aplicações do HellfireOS. Modifique o *makefile* em */platform/noc_3x2* para compilar essa aplicação. Compile e coloque o(s) binário(s) no simulador (em */usr/sim/mpsoc_ sim/objects* e simule por 20 segundos (esse tempo pode ser reduzido quando o algoritmo distribuído for usado). Observe a saída no arquivo */usr/sim/mpsoc_ sim/reports/out0.txt*.

Compreenda o exemplo e o faça funcionar em um único processador antes de tentar distribuir o algoritmo. A saída será uma imagem (*hex dump*) com os filtros aplicados, pronta para ser usada na ferramenta *create_bmp.c*. Verifique que a versão não paralela executa em aproximadamente 400 milhões ciclos, portanto seu algoritmo deve fazer melhor do que isso! A parte do tempo de simulação gasta para a impressão da imagem resultante do filtro não deve ser contabilizada no tempo do algoritmo (observe no exemplo como realizar a medição de tempo).

O programa exemplo envia para saída da simulação a imagem com o filtro aplicado. Copie os valores da saída para o arquivo *filter_image.h*. O programa *create_bmp.c* deve ser recompilado (no sistema hospedeiro!) e usado para apresentar a saída do filtro (será gerado o arquivo *output.bmp*.

Versão distribuída:

Observe os exemplos de uso envolvendo NoC fornecidos juntamente com os exemplos do HellfireOS. As principais primitivas da API a serem usadas são: *hf_spawn()*, *hf_cpuid()*, *hf_selfid()*, *hf_comm_create()*, *hf_send()*, *hf_probe()* e *hf_recv()*.

Devem ser usadas pelo menos duas configurações de MPSoC com tamanhos diferentes para realização de comparações. É sugerido que sejam usadas configurações entre 9 e 25 processadores (malhas 3x3, 3x4, 4x4, 4x5 e 5x5) e que as unidades de processamento (segmentos) sejam entre 15x15 e 40x40 pixels (devem ser usados pixels adicionais nas bordas para que os filtros possam ser aplicados adequadamente em cada parte). Evite mensagens maiores que 2kB para evitar perdas de dados, uma vez que pode ocorrer falta de memória no destino.

Notas:

O grupo deve utilizar uma outra imagem para a aplicação dos filtros. A imagem deve ser convertida para grayscale, mas salva em 24 bits por pixel (sem informação sobre espaço de cores) e ter a resolução entre 128x128 e 350x350 pixels (preferencialmente, para evitar muito tempo de simulação). Este formato foi escolhido para simplificar a conversão de imagens. Por ter as 3 componentes da imagem (RGB) iguais, apenas a informação de luminância (grayscale) será utilizada, reduzindo a complexidade de processamento.

A imagem utilizada como entrada deve ser dividida em diversas partes durante a execução, para ser processada de forma distribuída. Por exemplo, a partir de uma imagem 256x256 pixels, podem ser obtidas 64 partes com 38x38 pixels (32x32 mais pixels adicionais das bordas) que devem ser distribuídas e processadas de maneira independente em diversos processadores. Um processador pode ser responsável por organizar a distribuição das tarefas (dividir a imagem original em partes e enviar estas para serem processadas), receber o resultado do processamento e remontar uma saída, jogando-a no terminal como na aplicação exemplo.