

Atividade 3 - Org. Arg. 1

Profa. Dra. Cíntia Borges Margi
(cintia@usp.br)

Guilherme S. Salustiano
(salustiano@usp.br)

Parte 1 - Emulação de ponto flutuante

Contexto

Podem ser encontrado no mercado diversos processadores sem unidade de ponto flutuante, seja por uma questão de custo, área ou limite energético. Entretanto o 'float' e 'double' são definidos no C, assim sendo os códigos contendo esse tipo precisam rodar em todas as arquiteturas. Como isso é possível?

Para investigar vamos tentar compilar o seguinte código para uma arquitetura sem ponto flutuante, no caso do RISC-V isso é representado pela ausência da letra f e d da ISA, por se tratar de uma ISA modular ela tem a base de inteiros (como rv32i ou rv64i), e cada letra representa um novo conjunto de instruções [1].

Para isso vamos usar o [godbolt](#), com o compilador RISC-V (64-bits) gcc com as flags -O2 -march=rv64i -mabi=lp64, em -march definimos a arquitetura alvo como um processador RISC-V sem nenhuma extensão e em -mabi definimos a interface binária para também não usar ponto flutuante.

Agora digitando o seguinte código:

```
float sum(float x, float y) {  
    return x + y;  
}
```

Observamos a seguinte saída:

```
sum(float, float):  
    mv     a5,a1  
    mv     a1,a0  
    addi   sp,sp,-16  
    mv     a0,a5  
    sd     ra,8(sp)  
    call   __addsf3  
    ld     ra,8(sp)  
    addi   sp,sp,16  
    jr     ra
```

Conforme esperado o processador não realiza a soma, e delegou a função __addsf3 que recebe y e x nos registradores a0 e a1 (observe que ele precisa alterar os parâmetros, uma vez que a soma de pontos flutuantes não é cumulativa).

A função __addsf3 faz parte da biblioteca de runtime de C, que permite a portabilidade da linguagem em diversos sistemas. As outras rotinas podem ser encontradas [aqui](#).

Tarefa

Implemente as seguintes funções de uma biblioteca de ponto flutuante:

- mfloat floatsisf (mint i) - converte um inteiro para a representação ponto flutuante
- mint fixsfsi (mfloat a) - converte um ponto flutuante para a representação inteira
- mfloat negsf2 (mfloat a) - retorna o negado de a (Dica: é apenas um bit flip)
- mfloat addsf3 (mfloat a, mfloat b) - retorna a soma entre a e b
- mfloat subsf3 (mfloat a, mfloat b) - retorna a subtração entre a e b (Dica: pode ser definido a partir da combinação de outras duas funções)

Sendo mfloat e mint tipos definidos a partir da inttypes.h para garantir compatibilidade.

```
#include <inttypes.h>
```

```
typedef mint int32_t  
typedef mfloat uint32_t
```

O código não pode conter a palavra reservada float ou double e será checado estáticamente. Palavras derivadas (acrecidas de caracteres antes ou após) como float_valor, valor_float, são permitidas.

Seu código não precisa tratar casos os casos de NaN, Infinity ou underflow.

Você pode encontrar um template com alguns casos de teste [no repositório do experimento](#). Você também pode contribuir com mais casos de teste publicos via o github.

Parte 2 - Benchmark SIMD

Aplicações

A detecção de bordas é uma técnica basica de processamento de imagens que permite extrair informações de uma imagem, a técnica consiste em aplicar um filtro de convolução na imagem.

A convolução discreta em duas dimensões é definida como a soma do produto de uma matriz de pesos (kernel) elemento a elemento da imagem original, conforme a figura abaixo.

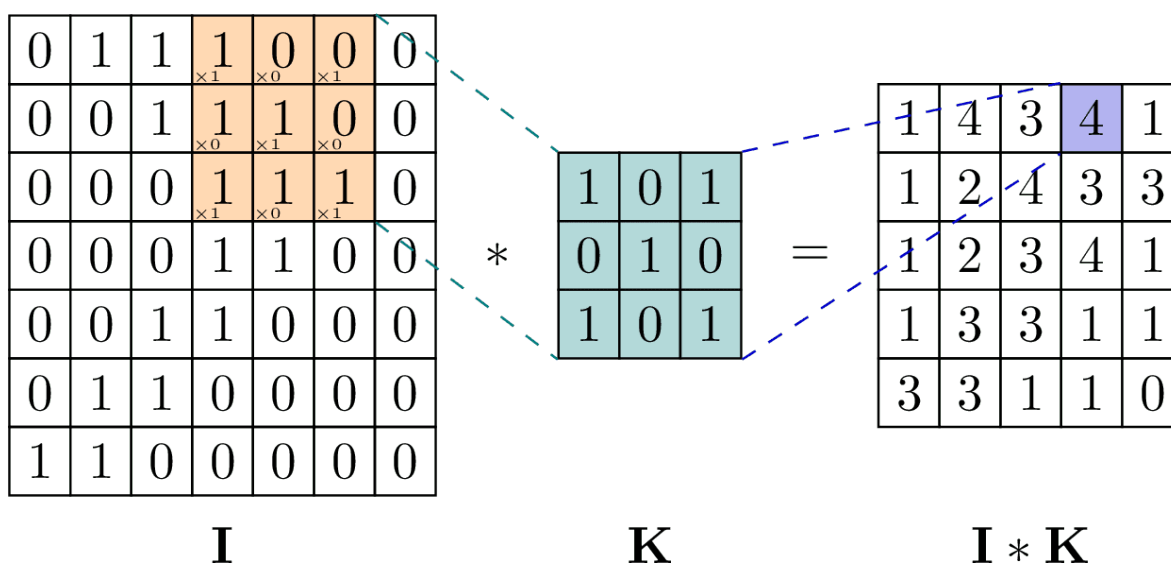


Figura 1: Convolução discreta em duas dimensões. Fonte: <https://tikz.net/janosh/conv2d.png>

Um exemplo animado de uma matriz gaussiana sendo aplicado em uma imagem pode ser visto [neste link](#), gerando assim uma imagem desfocada.

Nesse exemplo vamos usar o [kernel de sobel](#), que permite identificar bordas verticais e horizontais, conforme a figura abaixo.

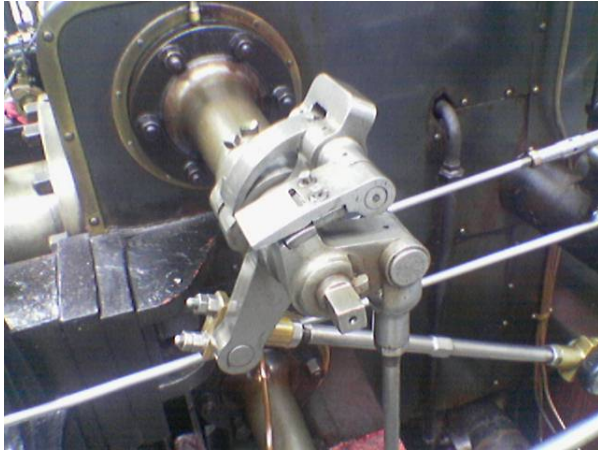


Figura 2: Uma imagem colorida de uma maquina.
Fonte: Wikipedia

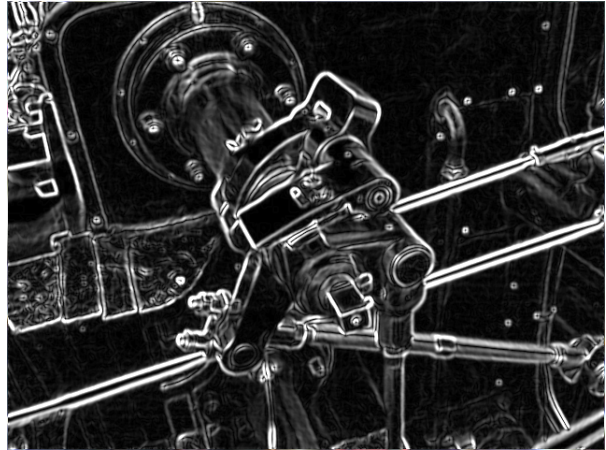


Figura 3: *Seahorse Valley* O sobel aplicado a imagem original. Fonte: Wikipedia

A implementação em C da convolução é a seguinte:

```
typedef struct {
    int width;
    int height;
    int32_t *data;
} Matrix;

Matrix conv2d(Matrix img, Matrix kernel) {
    Matrix out = {
        img.width - kernel.width + 1,
        img.height - kernel.height + 1,
        malloc(img.width * img.height * sizeof(int32_t))
    };

    #pragma omp parallel for
    for (int i = 0; i < out.height; i++) {
        for (int j = 0; j < out.width; j++) {
            int32_t sum = 0;
            for (int k = 0; k < kernel.height; k++) {
                for (int l = 0; l < kernel.width; l++) {
                    int x = j + l;
                    int y = i + k;

                    int32_t weight = kernel.data[k*kernel.width + l];
                    int32_t pixel = img.data[y*img.width + x];

                    sum += weight * pixel;
                }
            }
            sum /= FIXED_POINT;
            out.data[i * out.width + j] = sum;
        }
    }

    return out;
}
```

Os valores da imagem só podem variar entre 0 e 255, sendo utilizado o tipo `uint16_t` para facilitar o tratamento de overflow. Esse é o típico exemplo onde o uso de SIMD pode ser muito útil, pois podemos realizar a soma de 4 pixels ao mesmo tempo em um registrador de 128 bits.

Tarefa

Nos iremos realizar uma comparativo de desempenho entre a implementação básica e com SIMD, para isso baixe o código do experimento disponível na pasta `simd_benchmarking`.

Compare os códigos `edge_float.c`, `edge_int.c` e `edge_simd.c` e identifique as diferenças (Dica: use o comando `diff` para comparar os arquivos).

Entre no diretório e compile o código com o comando:

```
$ make
```

E então vamos executar o benchmark com o `perf` assim como na atividade anterior:

```
$ perf stat -x ';' -r 10 -e cycles,instructions,duration_time ./edge_float 2>&1 | tee edge_float.csv
$ perf stat -x ';' -r 10 -e cycles,instructions,duration_time ./edge_int 2>&1 | tee edge_int.csv
$ perf stat -x ';' -r 10 -e cycles,instructions,duration_time ./edge_simd 2>&1 | tee edge_simd.csv
```

Para facilitar a comparação dos resultados vamos usar o script `src/plot.py` que gera um gráfico com os resultados disponíveis na pasta `plots/`.

```
$ python plot.py
```

Entrega final

Ao final, gere um zip `atv3.zip` com os arquivos.

```
atv3.zip
├─ float_lib.c
├─ edge_float.c
├─ edge_int.c
├─ edge_simd.csv
```

Bibliografia

- [1] Wikipedia, “RISC-V --- Wikipedia, the free encyclopedia,” 2023. (<http://en.wikipedia.org/w/index.php?title=RISC-V&oldid=1179194505>)