Projeto 3 - Tópicos em Arquitetura e Hardware

Nome: Cleidiana Reis dos Santos

Matrícula: 298254

Introdução

A plataforma Litex é utilizada para a construção de SoCs (System-on-Chip) sobre FPGAs. FPGAs são dispositivos que podem ser reconfigurados para desempenhar diversas funções. O Litex simplifica a integração rápida de componentes essenciais como CPU, memória e periféricos. Este projeto teve como objetivo otimizar um programa no FPGA por meio dos seguintes passos:

- Escolher um programa para executar.
- Executar o programa medindo o desempenho.
- Acelerar o desempenho do programa e medir o desempenho novamente.

Todo o projeto é executado na placa Tang Nano 9k, utilizando como base o desenvolvimento do Projeto 2, portanto não será tratado comandos de desenvolvimento com a placa, compilação do programa, etc.

Passo 1 – Escolher um programa para executar.

Multiplicação de matrizes:

A multiplicação de matrizes é uma operação matemática que combina duas matrizes para resultar uma terceira matriz. Para que a multiplicação seja possível, o número de colunas da primeira matriz deve ser igual ao número de linhas da segunda matriz. O resultante na matriz produto é obtido pela soma dos produtos dos elementos da linha da primeira matriz com os elementos da coluna da segunda matriz.

Seja duas matrizes:

A (m x n) = [2 2]
[3 4]

B (n x p) = [1 1]
[4 5]

Cálculo dos elementos:

C11 = (2 * 1) + (2 * 4) = 10

C12 = (2 * 1) + (2 * 5) = 12

C21 = (3 * 1) + (4 * 4) = 19

C22 = (3 * 1) + (4 * 5) = 23

Resultado:

C (m x p) = [10 12]
[19 23]

O programa em C para realização da multiplicação de matrizes na FPGA ficou dessa maneira:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <irq.h>
#include <libbase/uart.h>
#include <libbase/console.h>
#include <generated/csr.h>

#define CLOCK_FREQUENCY 27000000  // 27 MHz

volatile uint32_t botao_user = 0;
#define SIZE 20

int A[SIZE][SIZE];
int B[SIZE][SIZE];
int C[SIZE][SIZE];

int main(void) {

#ifdef CONFIG_CPU_HAS_INTERRUPT
     irq_setmask(0);
     irq_setie(1);
#endif
```

```
    uart_init();
    uint32_t tempo_inicio, tempo_fim;
    uint64_t ciclos;
    uint32_t mili_segundos;
    leds_out_write(0x00);
    puts("\nAperte o botão para iniciar o teste\n");
    botao_user = gpio_in_read();

    //inicializa as matrizes
    for (int i = 0; i < SIZE; i++){
        for (int j = 0; j < SIZE; j++) {
            A[i][j] = i;
            B[i][j] = j;
            C[i][j] = 0;
        }
    }

    while(botao_user){
        botao_user = gpio_in_read();
    }

    leds_out_write(0x3F);
    printf("Iniciando multiplicação de matrizes...\n");

    timer0_reload_write(0xFFFFFFFF);
    timer0_load_write(0xFFFFFFFF);
    timer0_en_write(1);

    timer0_update_value_write(1);
    tempo_inicio = timer0_value_read();

    //multiplicação matrizes A * B
    for (int i = 0; i < SIZE; i++){
        for (int j = 0; j < SIZE; j++){
            for (int k = 0; k < SIZE; k++){
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    timer0_update_value_write(1);
    tempo_fim = timer0_value_read();

    ciclos = (uint64_t)(tempo_inicio - tempo_fim);
    mili_segundos = (ciclos / (CLOCK_FREQUENCY/1000));

    printf("\nTempo decorrido: %lld ciclos miliseg %ld\n", ciclos,
mili_segundos);
```

```
    leds_out_write(0x00);
    puts("\nFim teste\n");

    printf("Resultado Matriz C:\n");
    int PRINT_C = 0;
    if(PRINT_C){
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                printf("%3d ", C[i][j]);
            }
            printf("\n");
        }
    }
    return 0;
}
```

Passo 2 – Executar o programa medindo o desempenho.

Para cálculo do tempo de execução em ciclos e milissegundos do programa escolhido será utilizado o módulo de timer incorporado no Projeto 2 da disciplina. Para iniciar a execução do programa basta apertar o botão de usuário disponível na placa, também já desenvolvido no Projeto 2.

Após a gravação do código apresentado no Passo 1, o tempo resultante para multiplicação de duas matrizes de dimensão 20x20 foi de 99206 ciclos e 3ms. A Figura 1 mostra a saída resultante.

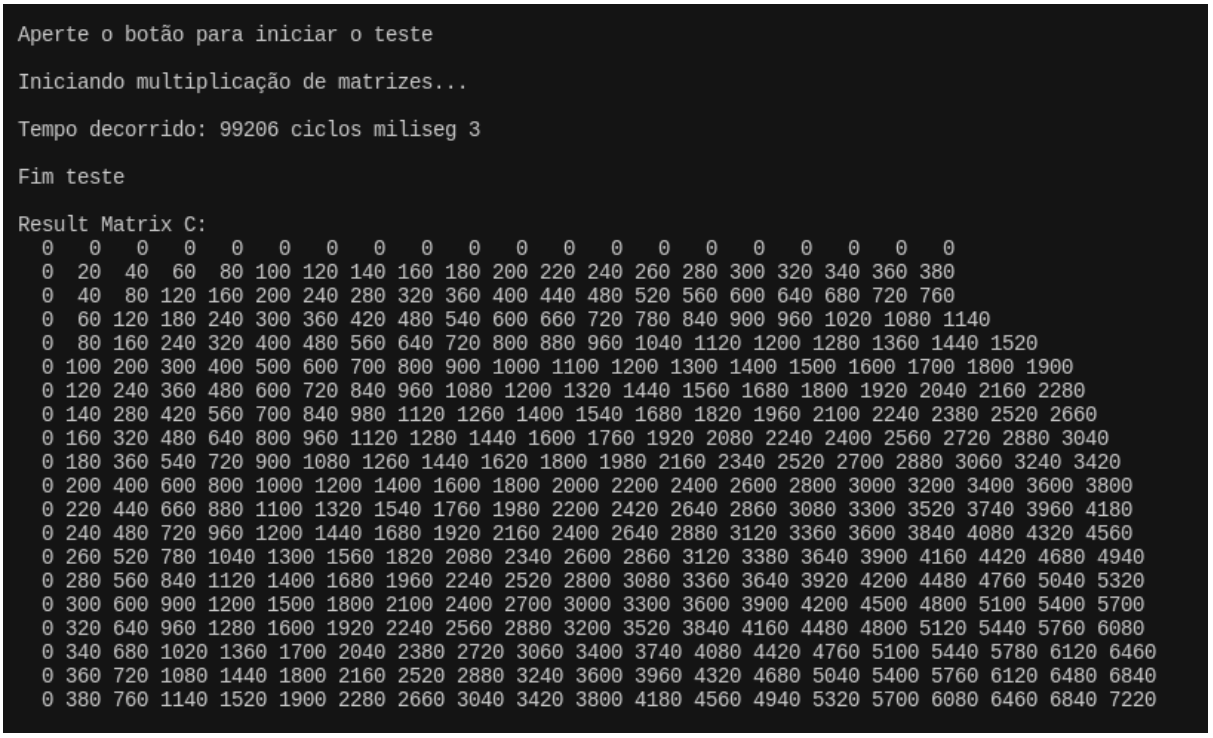

Figura 1: Tempo inicial da execução do programa.

Passo 3 – Acelerar o desempenho do programa e medir o desempenho novamente

A fim de reduzir o número de ciclos é proposto a divisão da matriz em matrizes menores (submatrizes).

Seja duas matrizes:

A (m x n) = [2 2 1 1]
[3 4 4 4]
[2 2 1 1]
[3 4 4 4]

B (n x p) = [3 3 2 2]
[2 5 5 5]
[3 3 2 2]
[2 3 3 3]

As matrizes A e B pode ser divida em 4 matrizes de 2x2, ficando da seguinte forma:

A00 = [2 2] A01 = [1 1] A10 = [2 2] A11 = [1 1]
[3 4] [4 4] [3 4] [4 4]

B00 = [3 3] B01 = [2 2] B10 = [3 3] B11 = [2 2]
[2 5] [5 5] [2 3] [3 3]

O cálculo da matriz resultante C, pode ser feito por blocos, utilizando os submatrizes menores das matrizes A e B:

- C00 = A00 × B00 + A01 × B10:

A00 × B00 = [2×3 + 2×2 2×3 + 2×5] = [10 16]

[3×3 + 4×2 3×3 + 4×5] = [17 29]

A01 × B10 = [1×3 + 1×2 1×3 + 1×3] = [5 6]

[4×3 + 4×2 4×3 + 4×3] = [20 24]

C00 = A00 × B00 + A01 × B10 = [15 22]

[37 53]

- C01 = A00 × B01 + A01 × B11 = [19 19]

[46 46]

- C10 = A10 × B00 + A11 × B10 = [15 22]

[37 53]

- C11 = A10 × B01 + A11 × B11 = [19 19]

[46 46]

Resultado:

C = [15 22 19 19]
[37 53 46 46]
[15 22 19 19]
[37 53 46 46]

O programa em C com essa otimização para realização da multiplicação de matrizes na FPGA ficou dessa maneira:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <irq.h>
#include <libbase/uart.h>
#include <libbase/console.h>
#include <generated/csr.h>

#define CLOCK_FREQUENCY 27000000  // 27 MHz

volatile uint32_t botao_user = 0;
#define SIZE 20
#define SUB_SIZE 2


int A[SIZE][SIZE];
int B[SIZE][SIZE];
int C[SIZE][SIZE];

int main(void) {

#ifdef CONFIG_CPU_HAS_INTERRUPT
     irq_setmask(0);
     irq_setie(1);
#endif
     uart_init();
    uint32_t tempo_inicio, tempo_fim;
    uint64_t ciclos;
    uint32_t mili_segundos;
    leds_out_write(0x00);
    puts("\nAperte o botão para iniciar o teste\n");
    botao_user = gpio_in_read();

    //inicializa as matrizes
    for (int i = 0; i < SIZE; i++){
        for (int j = 0; j < SIZE; j++) {
            A[i][j] = i;
            B[i][j] = j;
            C[i][j] = 0;
        }
    }

    while(botao_user){
        botao_user = gpio_in_read();
    }

    leds_out_write(0x3F);
```

```
    printf("Iniciando multiplicação de matrizes...\n");

    timer0_reload_write(0xFFFFFFFF);
    timer0_load_write(0xFFFFFFFF);
    timer0_en_write(1);

    timer0_update_value_write(1);
    tempo_inicio = timer0_value_read();


    //percorre todos os blocos SUB_SIZExSUB_SIZE da matriz C
    for (int bi = 0; bi < SIZE; bi += SUB_SIZE) {
        for (int bj = 0; bj < SIZE; bj += SUB_SIZE) {
            int Cblk[SUB_SIZE][SUB_SIZE] = {0};

            //soma dos produtos Ablk * Bblk
            for (int bk = 0; bk < SIZE; bk += SUB_SIZE) {
                int Ablk[SUB_SIZE][SUB_SIZE];
                int Bblk[SUB_SIZE][SUB_SIZE];

                //montagem da submatriz A
                for (int i = 0; i < SUB_SIZE; i++) {
                    for (int j = 0; j < SUB_SIZE; j++) {
                        Ablk[i][j] = A[bi + i][bk + j];
                    }
                }

                //montagem da submatriz B
                for (int i = 0; i < SUB_SIZE; i++) {
                    for (int j = 0; j < SUB_SIZE; j++) {
                        Bblk[i][j] = B[bk + i][bj + j];
                    }
                }

                //multiplicação submatrizes Ablk * Bblk
                for (int i = 0; i < SUB_SIZE; i++) {
                    for (int j = 0; j < SUB_SIZE; j++) {
                        for (int k = 0; k < SUB_SIZE; k++) {
                            Cblk[i][j] += Ablk[i][k] * Bblk[k][j];
                        }
                    }
                }
            }

            //copia o resultado Cblk para C
            for (int i = 0; i < SUB_SIZE; i++) {
                for (int j = 0; j < SUB_SIZE; j++) {
                    C[bi + i][bj + j] = Cblk[i][j];
```

```
                }
            }
        }
    }

    timer0_update_value_write(1);
    tempo_fim = timer0_value_read();

    ciclos = (uint64_t)(tempo_inicio - tempo_fim);
    mili_segundos = (ciclos / (CLOCK_FREQUENCY/1000));

    printf("\nTempo decorrido: %lld ciclos miliseg %ld\n", ciclos,
mili_segundos);

    leds_out_write(0x00);
    puts("\nFim teste\n");

    printf("Resultado Matriz C:\n");
    int PRINT_C = 1;
    if(PRINT_C){
        for (int i = 0; i < SIZE; i++) {
            for (int j = 0; j < SIZE; j++) {
                printf("%3d ", C[i][j]);
            }
            printf("\n");
        }
    }
    return 0;
}
```

Segue Figura 2 resultado da separação em blocos 2x2.

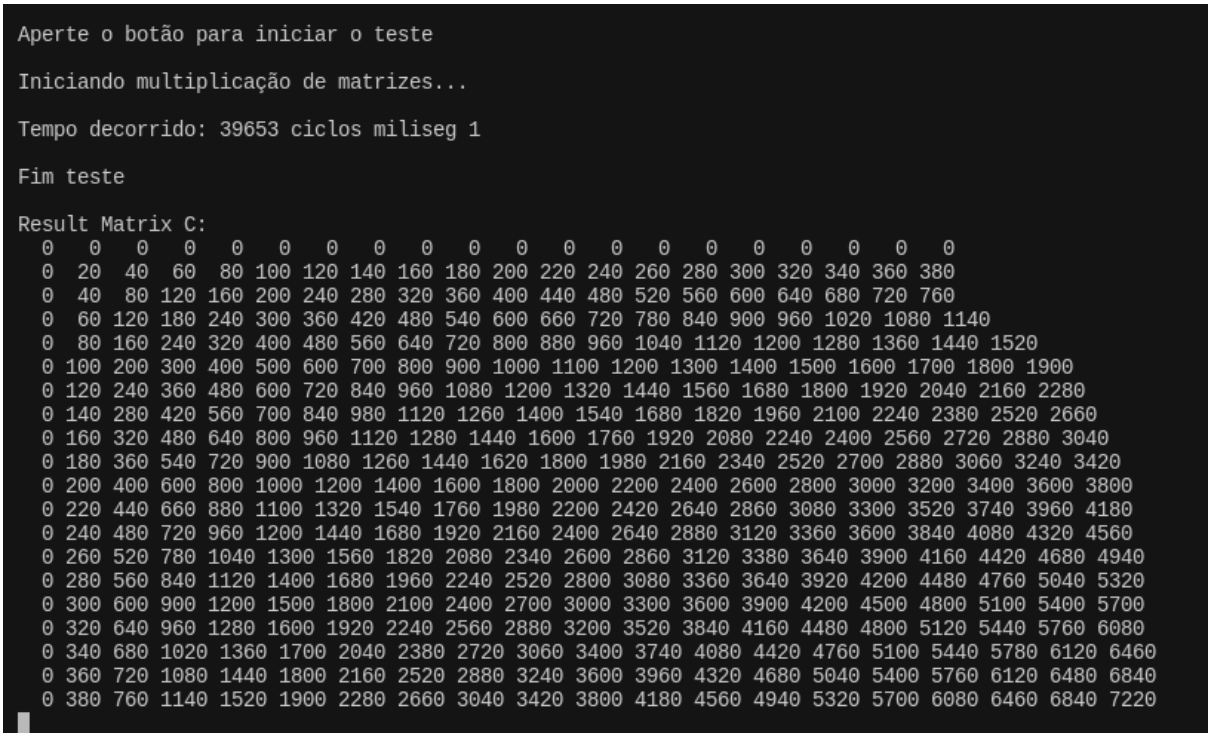

Figura 2: Tempo de execução após aceleração

O tempo resultante para multiplicação de duas matrizes de dimensão 20x20 foi de 39653 ciclos e 1ms, para divisão de blocos (submatrizes) de tamanho 2x2. Foram avaliadas diferentes configurações de tamanho de bloco (SUB_SIZE) trocando apenas o `define` no código apresentado. A Tabela 1 mostra o resultado de acordo com a variação de divisão de blocos. Apesar da expectativa inicial de que blocos maiores resultam em maior desempenho devido a redução no número total de blocos processados, foi observado que a configuração com SUB_SIZE = 2 apresentou o melhor desempenho prático.

Dimensão (SUB_SIZE)	Quant. de ciclos	Tempo de execução em milissegundos
2	39653	1
4	182844	6
5	157220	5

Tabela 1: Comparação de Tempo de Execução (Simulador x Tang Nano 9K)

Conclusão final:

O Projeto apresentou a comparação entre os testes de multiplicação de matrizes utilizando a multiplicação tradicional e divisão por blocos. Após a otimização, a divisão em blocos 2x2, o tempo de execução foi reduzido para aproximadamente 1/3 do tempo com o algoritmo inicial, foi de 99206 ciclos para 39653 ciclos de clock. Blocos com outros tamanhos não obtiveram resultado satisfatório.

Essa melhoria de desempenho se deve, principalmente, à redução dos tamanho das matrizes na multiplicação.

Além disso, pode destacar a flexibilidade para modificar o código e realizar experimentos com diferentes abordagens, bem como a facilidade de medição precisa do tempo de execução, viabilizada pelo módulo Timer desenvolvido anteriormente no Projeto 2.

O código desse projeto está disponível em um repositório no GitHub: https://github.com/ic-unicamp/mo801-2025s1-p3-cleidiana.