Relatório Projeto 3 - Aceleração de Código com LiteX

Aluna: Girlana Souza dos Santos – **RA**: 295749

Este projeto tem como objetivo principal a aceleração de um programa na plataforma LiteX.

1. ESCOLHA DO PROGRAMA PARA ACELERAÇÃO

Para este projeto, foi escolhida a operação de multiplicação de matrizes, devido à sua natureza intensiva em cálculos, o que a torna uma candidata ideal para aceleração por hardware.

O código-fonte em C para a multiplicação de matrizes em software é apresentado abaixo.

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <libbase/console.h>
#include <irq.h>
#include <libbase/uart.h>
#include <generated/csr.h>
#define TIMER_LOAD 0xFFFFFFFF
void multiply_matrices_sw(uint32_t *A, uint32_t *B_transposed, uint32_t *C, int N) {
   for (int i = 0; i < N; i++) {
       for (int j = 0; j < N; j++) {
           uint32_t sum = 0;
           for (int k = 0; k < N; k++) {
               sum += A[i * N + k] * B_transposed[j * N + k];
           }
           C[i * N + j] = sum;
       }
   }
}
void medir_tempo_sw(void) {
   int N = 9;
   uint32_t a[N*N];
   uint32_t b[N*N];
   uint32_t b_transposed[N*N];
   uint32_t c[N*N];
   for (int i = 0; i < N; i++) {
       for (int j = 0; j < N; j++) {
           a[i * N + j] = i + j;
           b[i * N + j] = i * j;
       }
   }
   for (int i = 0; i < N; i++) {
       for (int j = 0; j < N; j++) {
           b_transposed[j * N + i] = b[i * N + j];
```

```
        }
    }
    printf("Matriz A:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%6lu", (unsigned long)a[i * N + j]);
        }
        printf("\n");
    }
    printf("Matriz B (original):\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%6lu", (unsigned long)b[i * N + j]);
        }
        printf("\n");
    }
    timer0_en_write(0);
    timer0_load_write(TIMER_LOAD);
    timer0_en_write(1);
    multiply_matrices_sw(a, b_transposed, c, N);
    timer0_update_value_write(1);
    unsigned int restante = timer0_value_read();
    unsigned int ciclos = TIMER_LOAD - restante;
    uint64_t tempo_us = ((uint64_t)ciclos * 1000000) / CONFIG_CLOCK_FREQUENCY;
    printf("Tempo SW: %u ciclos (%u us)\n", ciclos, tempo_us);
    printf("Resultado da multiplicação (Software) para uma matriz (%dx%d):\n", N, N);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%6lu", (unsigned long)c[i * N + j]);
        }
        printf("\n");
    }
}
int main(void) {
    uart_init();
    printf("Iniciando multiplicação de matrizes em software...\n");
    medir_tempo_sw();
    return 0;
}
```

A função `medir_tempo_sw` é responsável por inicializar as matrizes A e B, calcular a transposta de B, e então executar a função `multiply_matrices_sw` medindo o tempo de execução utilizando o `timer0` disponível na plataforma LiteX. O tempo é expresso em ciclos de clock e em microssegundos.

2. EXECUÇÃO E MEDIÇÃO DO DESEMPENHO INICIAL

O ambiente de execução é a plataforma LiteX, com as seguintes especificações do SoC:

```
--================ SoC ==================--
CPU:        VexRiscv @ 1MHz
BUS:        wishbone 32-bit @ 4GiB
CSR:        32-bit data
ROM:        128.0KiB
SRAM:       8.0KiB
MAIN-RAM:   64.0KiB
```

TESTE COM MATRIZ 4X4 (N=4)

Matriz A

0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

Matriz B (original)

0	0	0	0
0	1	2	3
0	2	4	6
0	3	6	9

TESTE COM MATRIZ 8X8 (N=8)

Matriz A

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14

Matriz B (original)

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7
0	2	4	6	8	10	12	14
0	3	6	9	12	15	18	21
0	4	8	12	16	20	24	28
0	5	10	15	20	25	30	35
0	6	12	18	24	30	36	42
0	7	14	21	28	35	42	49

TESTE COM MATRIZ 9X9 (N=9)

Matriz A

0	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	10
3	4	5	6	7	8	9	10	11
4	5	6	7	8	9	10	11	12
5	6	7	8	9	10	11	12	13
6	7	8	9	10	11	12	13	14
7	8	9	10	11	12	13	14	15
8	9	10	11	12	13	14	15	16

Matriz B (original)

0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8
0	2	4	6	8	10	12	14	16
0	3	6	9	12	15	18	21	24
0	4	8	12	16	20	24	28	32
0	5	10	15	20	25	30	35	40
0	6	12	18	24	30	36	42	48
0	7	14	21	28	35	42	49	56
0	8	16	24	32	40	48	56	64

TESTE COM MATRIZ 16X16 (N=16)

Matriz A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Matriz B (original)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
0	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45
0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75
0	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90
0	7	14	21	28	35	42	49	56	63	70	77	84	91	98	105
0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120
0	9	18	27	36	45	54	63	72	81	90	99	108	117	126	135
0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
0	11	22	33	44	55	66	77	88	99	110	121	132	143	154	165
0	12	24	36	48	60	72	84	96	108	120	132	144	156	168	180
0	13	26	39	52	65	78	91	104	117	130	143	156	169	182	195
0	14	28	42	56	70	84	98	112	126	140	154	168	182	196	210
0	15	30	45	60	75	90	105	120	135	150	165	180	195	210	225

RESULTADOS DOS TESTES

Observa-se que ao aumentar o tamanho das matrizes o tempo também aumenta, o que é esperado.

Tamanho da Matriz	Tempo SW (ciclos)	Tempo SW (s)
4x4	1208	0,001208
8x8	7559	0,007559
9x9	10520	0,01052
16x16	54419	0,054419
20x20	109319	0,109325

Os resultados reforçam a ideia de que a multiplicação de matrizes em software puro no LiteX SoC tem limitações de escalabilidade. Para aplicações que requerem processamento de matrizes maiores ou em alta frequência, o desempenho em software se torna um gargalo, justificando a busca por soluções de aceleração de hardware.

3. ACELERAÇÃO DO DESEMPENHO DO PROGRAMA

Para acelerar a operação de multiplicação de matrizes, foi criado um acelerador de hardware para a operação **MAC (Multiply-Accumulate)**.

O módulo Verilog `matrix_multiplier` implementa a lógica da unidade MAC. Ele recebe dois valores de 32 bits (`a_val_in` e `b_val_in`) , um sinal de controle `add_to_accum` para iniciar a operação de multiplicação e acumulação , e `reset_accum` para resetar o acumulador. O resultado da acumulação é disponibilizado em `accum_out` , e um `done_flag` sinaliza a conclusão de uma operação.

```
module matrix_multiplier (
   input clk,
   input reset,
   input [31:0] a_val_in,
   input [31:0] b_val_in,
   input add_to_accum,
   input reset_accum,
   output reg [31:0] accum_out,
   output reg done_flag
);
reg [31:0] accum;
reg prev_add_to_accum;
always @(posedge clk) begin
   if (reset) begin
       accum <= 0;
       accum_out <= 0;
       done_flag <= 0;
       prev_add_to_accum <= 0;
   end else begin
       done_flag <= 0;
       prev_add_to_accum <= add_to_accum;
       if (reset_accum) begin
           accum <= 0;
       end
       else if (add_to_accum && !prev_add_to_accum) begin
           accum <= accum + (a_val_in * b_val_in);
           done_flag <= 1;
       end
       accum_out <= accum;
   end
end
endmodule
```

A interface Python (Wrapper), através da classe `MatrixMultiplier` , integra o módulo Verilog ao SoC LiteX . Esta classe utiliza os **Common Standard Registers (CSRs)** para permitir a comunicação entre o software (CPU) e o hardware (acelerador MAC).

Componente	**Função**
`self.a_val_in = CSRStorage(32)`	Registrador para enviar o valor A ao hardware
`self.b_val_in = CSRStorage(32)`	Registrador para enviar o valor B ao hardware
`self.add_to_accum = CSR(1)`	Registrador para sinalizar a operação de acumulação
`self.reset_accum = CSR(1)`	Registrador para resetar o acumulador no hardware

`self.accum_out = CSRStatus(32)`	Registrador de status para ler o valor acumulado do hardware
`self.done_flag = CSRStatus(1)`	Registrador de status para verificar a conclusão da operação

```
from migen import *
from litex.soc.interconnect.csr import CSRStorage, CSRStatus, CSR
from litex.gen import *
class MatrixMultiplier(LiteXModule):
   def __init__(self):
       self.a_val_in = CSRStorage(32)
       self.b_val_in = CSRStorage(32)
       self.add_to_accum = CSR(1)
       self.reset_accum  = CSR(1)
       self.accum_out    = CSRStatus(32)
       self.done_flag    = CSRStatus(1)
       self.specials += Instance("matrix_multiplier",
           i_clk=ClockSignal(),
           i_reset=ResetSignal(),
           i_a_val_in = self.a_val_in.storage,
           i_b_val_in = self.b_val_in.storage,
           i_add_to_accum = self.add_to_accum.re,
           i_reset_accum  = self.reset_accum.re,
           o_accum_out = self.accum_out.status,
           o_done_flag = self.done_flag.status
       )
       self.comb += [
           self.add_to_accum.re.eq(0),
           self.reset_accum.re.eq(0)
       ]
```

A linha `self.specials += Instance(“matrix_multiplier”, ...)` instancia o módulo Verilog `matrix_multiplier` no design do LiteX, conectando as portas de entrada e saída do módulo Verilog aos CSRs criados na classe Python. Isso permite que o software interaja diretamente com o acelerador de hardware, escrevendo valores de entrada nos registradores CSR de armazenamento e lendo os resultados dos registradores CSR de status.

➢ **Código utilizado para medição**

Para utilizar o acelerador, o código C em software foi modificado. O código em C abaixo realiza a multiplicação de matrizes utilizando um acelerador de hardware. Para isso, ele:

I. Inicializa as matrizes A e B e gera a transposta de B.

II. Para cada elemento `C[i][j]` da matriz resultado:

   A. Reinicia o acumulador no acelerador via CSR `reset_accum`

B. Itera sobre o índice k, enviando os pares de elementos `A[i][k]` e `B[k][j]` para os registradores `a_val_in` e `b_val_in`

C. Após cada par ser enviado, aciona o sinal `add_to_accum` para acumular o produto no hardware

D. Ao fim da acumulação, lê o valor final do acumulador via `accum_out` e armazena em `C[i][j]`

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <libbase/console.h>
#include <irq.h>
#include <libbase/uart.h>
#include <generated/csr.h>
#define TIMER_LOAD 0xFFFFFFFF
void medir_tempo_hw(void);
void multiply_matrices(uint32_t *A, uint32_t *B_transposed, uint32_t *C, int N);
void multiply_matrices(uint32_t *A, uint32_t *B_transposed, uint32_t *C, int N) {
   for (int i = 0; i < N; i++) {
       for (int j = 0; j < N; j++) {
           matrix_multiplier_reset_accum_write(1);
           for (int k = 0; k < N; k++) {
               uint32_t a_val = A[i * N + k];
               uint32_t b_val = B_transposed[j * N + k];
               matrix_multiplier_a_val_in_write(a_val);
               matrix_multiplier_b_val_in_write(b_val);
               matrix_multiplier_add_to_accum_write(1);
           }
           C[i * N + j] = matrix_multiplier_accum_out_read();
       }
   }
}
void medir_tempo_hw(void) {
   int N = 9;
   uint32_t a[N*N];
   uint32_t b[N*N];
   uint32_t b_transposed[N*N];
   uint32_t c[N*N];
   for (int i = 0; i < N; i++) {
       for (int j = 0; j < N; j++) {
           a[i * N + j] = i + j;
           b[i * N + j] = i * j;
       }
   }
```

```
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            b_transposed[j * N + i] = b[i * N + j];
        }
    }
    printf("Matriz A:\n");
    for (int i = 0; i < N * N; i++) {
        printf("%6lu", (unsigned long)a[i]);
        if ((i + 1) % N == 0) printf("\n");
    }
    printf("Matriz B (original):\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            printf("%6lu", (unsigned long)b[i * N + j]);
        }
        printf("\n");
    }
    for (int i = 0; i < N * N; i++) c[i] = 0;
    timer0_en_write(0);
    timer0_load_write(TIMER_LOAD);
    timer0_en_write(1);
    multiply_matrices(a, b_transposed, c, N);
    timer0_update_value_write(1);
    unsigned int restante = timer0_value_read();
    unsigned int ciclos = TIMER_LOAD - restante;
    unsigned int tempo_us = (ciclos * 1000000) / CONFIG_CLOCK_FREQUENCY;
    printf("Tempo HW: %u ciclos (%u us)\n", ciclos, tempo_us);
    printf("Resultado da multiplicação (Hardware) para uma matriz (%dx%d):\n", N, N);
    for (int i = 0; i < N * N; i++) {
        printf("%6lu", (unsigned long)c[i]);
        if ((i + 1) % N == 0) printf("\n");
    }
}
int main(void) {
    uart_init();
    printf("Iniciando multiplicação de matrizes via acelerador...\n");
    medir_tempo_hw();
    return 0;
}
```

4. MEDIÇÃO DO DESEMPENHO ACELERADO E ANÁLISE

TESTE COM MATRIZ 4X4 (N=4)

Matriz A **Matriz B (original)**

0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

0	0	0	0
0	1	2	3
0	2	4	6
0	3	6	9

TESTE COM MATRIZ 8X8 (N=8)

Matriz A

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14

Matriz B (original)

0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7
0	2	4	6	8	10	12	14
0	3	6	9	12	15	18	21
0	4	8	12	16	20	24	28
0	5	10	15	20	25	30	35
0	6	12	18	24	30	36	42
0	7	14	21	28	35	42	49

TESTE COM MATRIZ 9X9 (N=9)

Matriz A

0	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	9
2	3	4	5	6	7	8	9	10
3	4	5	6	7	8	9	10	11
4	5	6	7	8	9	10	11	12
5	6	7	8	9	10	11	12	13
6	7	8	9	10	11	12	13	14
7	8	9	10	11	12	13	14	15
8	9	10	11	12	13	14	15	16

Matriz B (original)

0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8
0	2	4	6	8	10	12	14	16
0	3	6	9	12	15	18	21	24
0	4	8	12	16	20	24	28	32
0	5	10	15	20	25	30	35	40
0	6	12	18	24	30	36	42	48
0	7	14	21	28	35	42	49	56
0	8	16	24	32	40	48	56	64

TESTE COM MATRIZ 16X16 (N=16)

Matriz A

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Matriz B (original)

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
0	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45
0	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75
0	6	12	18	24	30	36	42	48	54	60	66	72	78	84	90
0	7	14	21	28	35	42	49	56	63	70	77	84	91	98	105
0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120
0	9	18	27	36	45	54	63	72	81	90	99	108	117	126	135
0	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
0	11	22	33	44	55	66	77	88	99	110	121	132	143	154	165
0	12	24	36	48	60	72	84	96	108	120	132	144	156	168	180
0	13	26	39	52	65	78	91	104	117	130	143	156	169	182	195
0	14	28	42	56	70	84	98	112	126	140	154	168	182	196	210
0	15	30	45	60	75	90	105	120	135	150	165	180	195	210	225

RESULTADOS DOS TESTES

Ao comparar os resultados, percebe-se que, para as matrizes menores (4x4, 8x8 e 9x9), o tempo de execução em hardware foi ligeiramente maior do que em software. Isso sugere que a sobrecarga da comunicação via CSRs para cada operação MAC individual pode ter impactado negativamente o desempenho para essas dimensões.

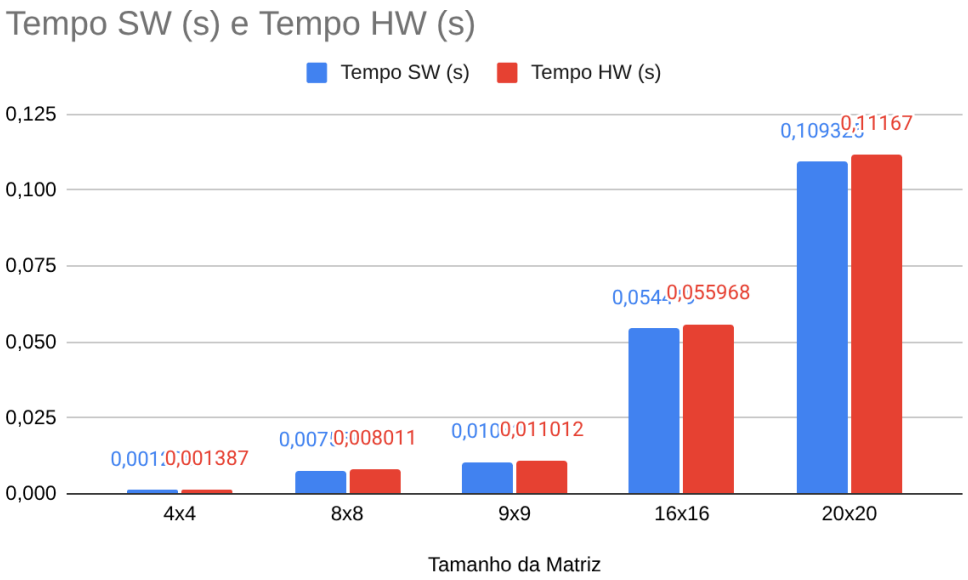
No entanto, à medida que o tamanho da matriz aumenta (16x16 e 20x20), a diferença de tempo entre software e hardware se torna menos significativa, e em alguns casos o hardware ainda não apresenta uma aceleração clara em termos de tempo total, mesmo com a operação MAC sendo executada em hardware. Isso se dá porque embora a operação MAC esteja acelerada por hardware, a CPU ainda é responsável por gerenciar os loops externos da multiplicação, o que limita o ganho de desempenho geral e contribui para o overhead da comunicação.

Tamanho da Matriz	Tempo HW (ciclos)	Tempo HW (s)
4x4	1387	0,001387
8x8	8011	0,008011
9x9	11012	0,011012
16x16	55968	0,055968
20x20	111670	**0,11167**

5. COMPARAÇÃO DO DESEMPENHO: SOFTWARE VS HARDWARE

A tabela a seguir resume os resultados de desempenho em software (SW) e hardware (HW) para as diferentes dimensões de matrizes.

Tamanho da Matriz	Tempo SW (ciclos)	Tempo SW (us)	Tempo HW (ciclos)	Tempo HW (us)
4x4	1208	0,001208	1387	0,001387
8x8	7559	0,007559	8011	0,008011
9x9	10520	0,01052	11012	0,011012
16x16	54419	0,054419	55968	0,055968
20x20	109319	0,109325	111670	0,11167



➢ **Análise Comparativa e Discussão**

O objetivo principal deste projeto foi acelerar a operação de multiplicação de matrizes, que foi identificada como uma tarefa computacionalmente intensiva em sua execução puramente via software. O acelerador de hardware projetado para a operação MAC visava aliviar o processador principal dessa carga.

Para as matrizes, o tempo de execução em software foi ligeiramente superior ao tempo de execução em hardware. Essa diferença, é indicativa da sobrecarga (overhead) associada à comunicação entre a CPU e o acelerador de hardware via Common Standard Registers (CSRs). Cada operação MAC no hardware requer interações com a CPU para enviar os operandos e ler o resultado, e para matrizes pequenas, o tempo gasto nessas comunicações pode superar o ganho de velocidade da execução em hardware da operação MAC individual.

À medida que o tamanho das matrizes aumenta, como para as dimensões 16x16 e 20x20, a diferença entre os tempos de execução em software e hardware torna-se menos pronunciada. Isso ocorre porque a arquitetura atual do acelerador, embora execute a operação MAC em hardware, ainda exige que a CPU gerencie os loops externos da multiplicação de matrizes. A CPU continua sendo responsável por orquestrar grande parte do processo, o que limita o potencial de aceleração e mantém uma parcela considerável do overhead de comunicação.

Em resumo, os resultados demonstram que, embora o conceito de aceleração por hardware seja válido para operações computacionalmente intensivas, a implementação atual, focada apenas na operação MAC e dependente da CPU para o gerenciamento dos loops, não resultou em ganhos de desempenho consistentes em todas as dimensões de matrizes. Para alcançar uma aceleração mais significativa em todas as dimensões, futuras melhorias poderiam focar na delegação de blocos maiores da computação de matrizes para o hardware, minimizando as interações e o overhead de comunicação com a CPU