Final Project: Parallel Processing to Compute VGA Output
Group: Riley Hicks, Nikita Klimov, Kyle Hagy, James Renick

Overview

This project entailed parallelizing computations of pixel data for a VGA frame buffer by running multiple pipelined OTTER MCU's in parallel and interfacing them with a serial VGA driver in order to speed up the availability of data for the VGA display. The interface, consisting of two register files for the addresses to the frame buffer and RGB pixel data, respectively, allows for parallel loading of the MCU outputs that are released in parallel, and allows for serial reading by the VGA driver.

Architecture Description

In the process of creating this system, four OTTER MCUs were connected through their IOBUS_OUT ports to two shared intermediate register files, one holding the addresses to the VGA's frame buffer and another holding pixel RGB data to be placed into the VGA's frame buffer. These intermediate registers allow for parallel loading of values at their four inputs each, and are enabled with an FSM that would enable the "ADDRESS" register file when the first IOBUS_WR-high signal was detected, and would enable the "DATA" register file when the second IOBUS_WR-high signal was detected. At these moments, the MCUs are outputting first the frame buffer address, then the pixel data; and these MCU outputs are written into the respective register files. As a result, data for four pixels is loaded into the register files simultaneously, and the VGA driver can display them. A 2-bit up-counter generates the read address to the register files at the same clock frequency that the VGA driver reads from the frame buffer, and hence the VGA driver receives pixel data from a new MCU every clock cycle.

Schematic

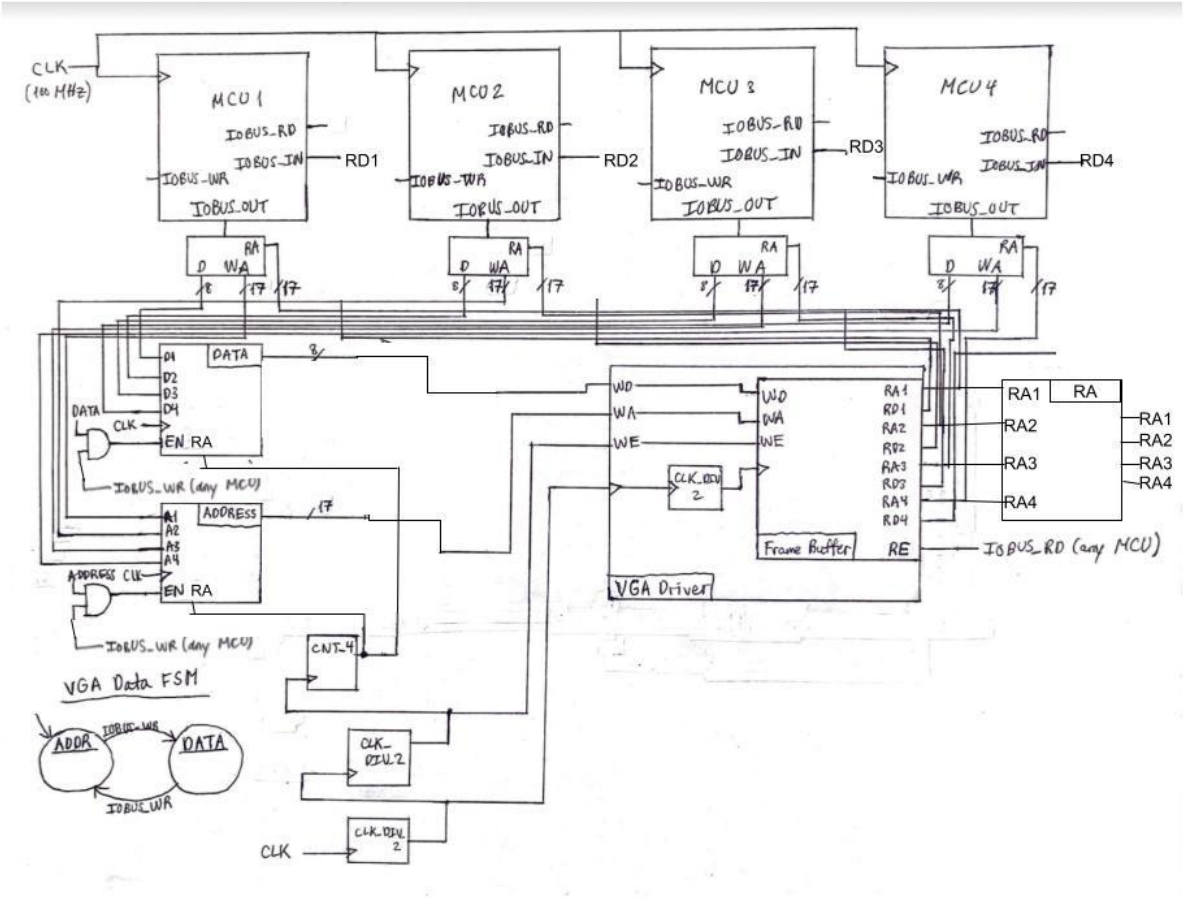

Figure 1: Schematic diagram of our implementation

The schematic illustrated above displays how the 4 OTTER's were implemented with the VGA driver. The data and address intermediate registers on the left are used for pixels being written to the VGA driver, and the address register on the right is for MCU requests to read pixels. The current computed pixel value sent from each register is driven by an FSM which regulates which register is outputting. The base clock is divided by four to alternate which MCU's data is written to the VGA driver.

Demo

For our project we used different colors for each OTTER MCU to print out. This means that we had to recompile our C code 4 different times with different color outputs and copy the machine code of each program into each respective OTTER MCU's memory file. We used the colors 0xE0, 0xFF, 0x07, 0x3F respectively to display the screen in segments.

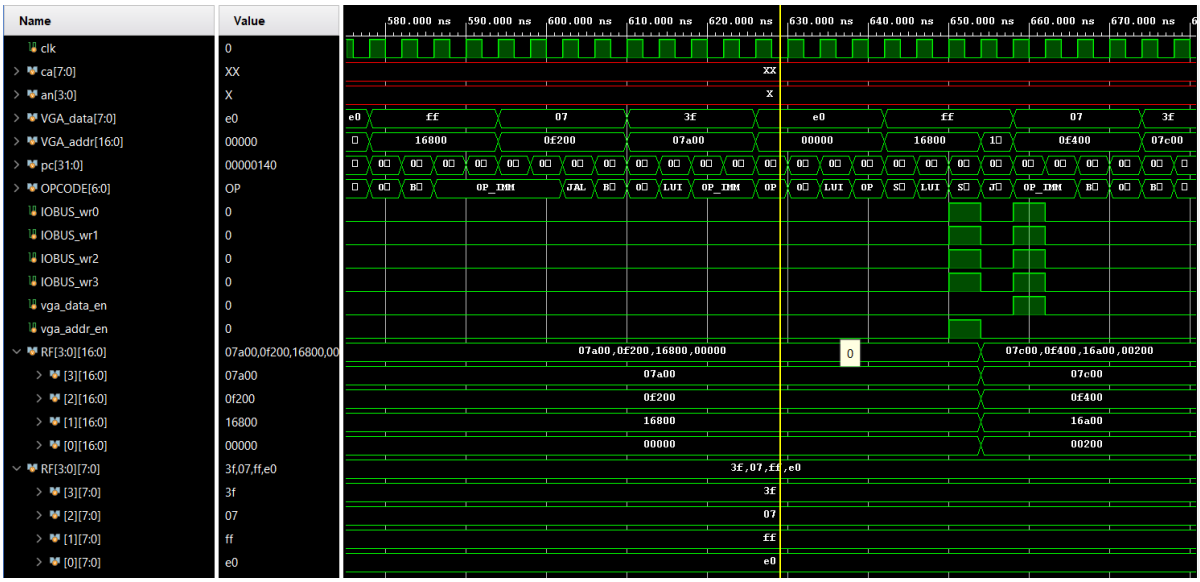

Figure 2: Simulation results for OTTER Multi-Core. Notice how "RF" contains the colors for each segment, and VGA_Data changes.

Figure 2: Simulation results for OTTER Single-Core. This data is shown in the WD signal. (sorry for all the green)

Figure 3: VGA results to screen

C Code

Single Core

```
#include "draw.h"
#include "otter.h"
#include <math.h>
#define Width 320
#define Height 240

void main()
{
    while (1)
    {
        int x,y;

        for(x = 0; x < Width; x++)
        {
            for(y = 0; y < Height / 4; y++)
            {
                draw_dot(x, y, 0b11100000);
            }

            for(y = Height / 4 + 1; y < Height / 2; y++)
            {
```

```
                draw_dot(x, y, 0b11111111);
            }

            for(y = Height / 2 + 1; y < 3 * Height / 4; y++)
            {
                draw_dot(x, y, 0b00000111);
            }
            for(y = 3 * Height / 4; y < Height; y++)
            {
                draw_dot(x, y, 0b00111111);
            }
        }
    }
    while (1);  //j 0
}
```

Multi-Core

Compiled code for 1st OTTER CORE

```
#include "draw.h"
#include "otter.h"
#include <math.h>
#define Width 320
#define Height 240

void main()
{
    while (1)
    {
        int x,y;

        for(x = 0; x < Width; x++)
        {
            for(y = 0; y < Height / 4; y++)
            {
                draw_dot(x, y, 0b11100000);
            }

        }
    }
    while (1);  //j 0
}
```

Compiled code for 2nd OTTER CORE

```
...
        for(y = Height / 4 + 1; y < Height / 2; y++)
        {
            draw_dot(x, y, 0b11111111);
        }
...
```

Compiled code for 3rd OTTER CORE

```
...
```

```
            for(y = Height / 2 + 1; y < 3 * Height / 4; y++)
            {
                draw_dot(x, y, 0b00000111);
            }
...
```

Compiled code for 4th OTTER CORE

```
...
            for(y = 3 * Height / 4; y < Height; y++)
            {
                draw_dot(x, y, 0b00111111);
            }
...
```

Verilog Code

```
        module Data_registerFile #(parameter data_width = 8) (WD1, WD2, WD3, WD4, ReadAddr, RegWrite, ReadData,
clock);
            input [1:0] ReadAddr; //the register number to read
            input [data_width-1:0] WD1, WD2, WD3, WD4; //data to write
            input RegWrite, //the write control
                clock;  // the clock to trigger write
            output logic [data_width-1:0] ReadData; // the register values read
            logic [data_width-1:0] RF[3:0];

            always_comb
                ReadData = RF[ReadAddr];

            always@(posedge clock) begin // write the register with the new value if Regwrite is high
                if(RegWrite) begin
                    RF[0] <= WD1;
                    RF[1] <= WD2;
                    RF[2] <= WD3;
                    RF[3] <= WD4;
                end
           end
         endmodule


        module Multicore_GPU_Wrapper(
            input CLK,
            input BTNL,
            input BTNC,
            input [15:0] SWITCHES,
            output [15:0] LEDS,
            output [7:0] CATHODES,
            output [3:0] ANODES,
            output [7:0] VGA_RGB,
            output VGA_HS,
            output VGA_VS
            );

            // INPUT PORT IDS ////////////////////////////////////////////////////////
            localparam SWITCHES_AD = 32'h11000000;
            localparam CLKCNTLO_AD = 32'h11400000;
            localparam CLKCNTHI_AD = 32'h11400004;
            localparam VGA_READ_AD  = 32'h11040000;

            // OUTPUT PORT IDS ///////////////////////////////////////////////////////
            localparam LEDS_AD     = 32'h11080000;
```

```
localparam SSEG_AD     = 32'h110C0000;
localparam VGA_ADDR_AD  = 32'h11100000;
localparam VGA_COLOR_AD = 32'h11140000;

// Signals for connecting OTTER_MCU to OTTER_wrapper ////////////////////////
logic s_interrupt, s_reset;
logic sclk;
logic CLK_50MHz = 0;
logic CLK_25MHz = 0;

// Signals for connecting VGA Framebuffer Driver ///////////////////////////
logic r_vga_we;             // write enable
logic [16:0] r_vga_wa;      // address of framebuffer to read and write
logic [7:0] r_vga_wd;       // pixel color data to write to framebuffer
logic [7:0] r_vga_rd;       // pixel color data read from framebuffer

// Registers for IOBUS /////////////////////////////////////////////////////
logic [15:0] r_SSEG = '0;
logic [15:0] r_LEDS = '0;
logic [63:0] r_CLKCNT = '0;

// Signals for IOBUS ///////////////////////////////////////////////////////
logic [31:0] IOBUS_out0, IOBUS_in0, IOBUS_addr0;
logic [31:0] IOBUS_out1, IOBUS_in1, IOBUS_addr1;
logic [31:0] IOBUS_out2, IOBUS_in2, IOBUS_addr2;
logic [31:0] IOBUS_out3, IOBUS_in3, IOBUS_addr3;

logic IOBUS_wr0;//, IOBUS_rd0;
logic IOBUS_wr1;//, IOBUS_rd1;
logic IOBUS_wr2;//, IOBUS_rd2;
logic IOBUS_wr3;//, IOBUS_rd3;

logic [7:0] VGA_data;
logic [16:0] VGA_addr;

/*logic [16:0] RA1, RA2, RA3, RA4;*/

clk_div CLK_DIV(.clk(CLK), .sclk(sclk));

// Declare OTTER_CPU's /////////////////////////////////////////////////////
OTTER_MCU0 MCU0(.RESET(s_reset), .INTR(s_interrupt), .CLK(CLK),
              .IOBUS_OUT(IOBUS_out0), .IOBUS_IN(IOBUS_in0),
              .IOBUS_ADDR(IOBUS_addr0), .IOBUS_WR(IOBUS_wr0)/*,
              .IOBUS_RD(IOBUS_rd0)*/);

OTTER_MCU1 MCU1(.RESET(s_reset), .INTR(s_interrupt), .CLK(CLK),
              .IOBUS_OUT(IOBUS_out1), .IOBUS_IN(IOBUS_in1),
              .IOBUS_ADDR(IOBUS_addr1), .IOBUS_WR(IOBUS_wr1)
              /* ,.IOBUS_RD(IOBUS_rd1)*/);


OTTER_MCU2 MCU2(.RESET(s_reset), .INTR(s_interrupt), .CLK(CLK),
              .IOBUS_OUT(IOBUS_out2), .IOBUS_IN(IOBUS_in2),
              .IOBUS_ADDR(IOBUS_addr2), .IOBUS_WR(IOBUS_wr2)
              /* ,.IOBUS_RD(IOBUS_rd1)*/);


OTTER_MCU3 MCU3(.RESET(s_reset), .INTR(s_interrupt), .CLK(CLK),
              .IOBUS_OUT(IOBUS_out3), .IOBUS_IN(IOBUS_in3),
              .IOBUS_ADDR(IOBUS_addr3), .IOBUS_WR(IOBUS_wr3)
              /* ,.IOBUS_RD(IOBUS_rd1)*/);




logic [1:0] CLK_CNT_25MHz = 2'b0;
logic vga_data_en, vga_addr_en;

Data_registerFile #(.data_width(8)) DATA_FILE (.WD1(IOBUS_out0[7:0]), .WD2(IOBUS_out1[7:0]),
.WD3(IOBUS_out2[7:0]), .WD4(IOBUS_out3[7:0]), .ReadAddr(CLK_CNT_25MHz),
.RegWrite(vga_data_en), .ReadData(VGA_data), .clock(CLK));

Data_registerFile #(.data_width(17)) WADDR_FILE (.WD1(IOBUS_out0[16:0]), .WD2(IOBUS_out1[16:0]),
.WD3(IOBUS_out2[16:0]), .WD4(IOBUS_out3[16:0]), .ReadAddr(CLK_CNT_25MHz),
.RegWrite(vga_addr_en), .ReadData(VGA_addr), .clock(CLK));

/*ReadAddr_Buffer RADDR_FILE (.InA1(IOBUS_out0[16:0]), .InA2(IOBUS_out0[16:0]),
.InA3(IOBUS_out0[16:0]), .InA4(IOBUS_out0[16:0]), .RegWrite(IOBUS_rd0), .OutA1(RA1),
.OutA2(RA2), .OutA3(RA3), .OutA4(RA4), .clock(CLK));*/

// Declare Seven Segment Display ///////////////////////////////////////////
```

```
    SevSegDisp SSG_DISP(.DATA_IN(r_SSEG), .CLK(CLK), .MODE(1'b0),
                        .CATHODES(CATHODES), .ANODES(ANODES));

    // VGA Buffer Driver ////////////////////////////////////////////////////
    vga_fb_driver VGA_Driver(.CLK(CLK_50MHz), .WA(VGA_addr), .WD(VGA_data),
                             .WE(CLK_50MHz), .RD(IOBUS_in1[7:0]), .ROUT(VGA_RGB[7:5]),
                             .GOUT(VGA_RGB[4:2]), .BOUT(VGA_RGB[1:0]),
                             .HS(VGA_HS), .VS(VGA_VS));

    // Connect LEDS register to port ////////////////////////////////////////
    assign LEDS = r_LEDS;

    // Debounce/one-shot the reset and interrupt buttons ////////////////////
    debounce_one_shot DB_I(.CLK(CLK_50MHz), .BTN(BTNL), .DB_BTN(s_interrupt));
    debounce_one_shot DB_R(.CLK(CLK_50MHz), .BTN(BTNC), .DB_BTN(s_reset));

    // Performance counter (MCU clock cycles) ///////////////////////////////
    always_ff @(posedge CLK_50MHz) begin
        r_CLKCNT = r_CLKCNT + 1;
    end

    // Clock Divider to create 50 MHz Clock /////////////////////////////////
    always_ff @(posedge CLK) begin
        CLK_50MHz <= ~CLK_50MHz;
    end

    // Clock Divider to create 25 MHz Clock
    always_ff @(posedge CLK_50MHz) begin
         CLK_25MHz <= ~CLK_25MHz;
    end

    //Clock Counter to handle register files from MCU to VGA
    always_ff @(posedge CLK_25MHz) begin
        CLK_CNT_25MHz += 1;
    end

    // Connect board peripherals (Memory Mapped IO devices) to IOBUS ////////
    // Inputs
    always_comb begin
        IOBUS_in0 = 32'b0;
        case (IOBUS_addr0)
            SWITCHES_AD: IOBUS_in0[15:0] = SWITCHES;
            CLKCNTLO_AD: IOBUS_in0 = r_CLKCNT[31:0];
            CLKCNTHI_AD: IOBUS_in0 = r_CLKCNT[63:32];
            VGA_READ_AD: IOBUS_in0[7:0] = r_vga_rd;
        endcase
    end
    // Outputs
    always_ff @(posedge CLK) begin
        r_vga_we <= 0;
        if (IOBUS_wr0) begin
            case (IOBUS_addr0)
                LEDS_AD: r_LEDS <= IOBUS_out0[15:0];
                SSEG_AD: r_SSEG <= IOBUS_out0[15:0];
            endcase
        end
    end


//FSM for VGA
typedef enum {addr, data} vga_state_type;
vga_state_type vga_state, vga_nstate;

always_ff @(posedge CLK) begin
    vga_state <= vga_nstate;
end

always_comb begin
    case(vga_state)
        addr: begin
            if(IOBUS_wr0) begin
                vga_nstate = data;
                vga_addr_en = 1;
                vga_data_en = 0;
            end
            else begin
                vga_nstate = addr;
                vga_addr_en = 0;
                vga_data_en = 0;
            end
        end
```

```
        data: begin
            if(IOBUS_wr0) begin
                vga_nstate = addr;
                vga_addr_en = 0;
                vga_data_en = 1;
            end
            else begin
                vga_nstate = data;
                vga_addr_en = 0;
                vga_data_en = 0;
            end
        end
        default: begin
            vga_nstate = addr;
            vga_addr_en = 0;
              vga_data_en = 0;
        end
    endcase
end


endmodule
```

Other Notes

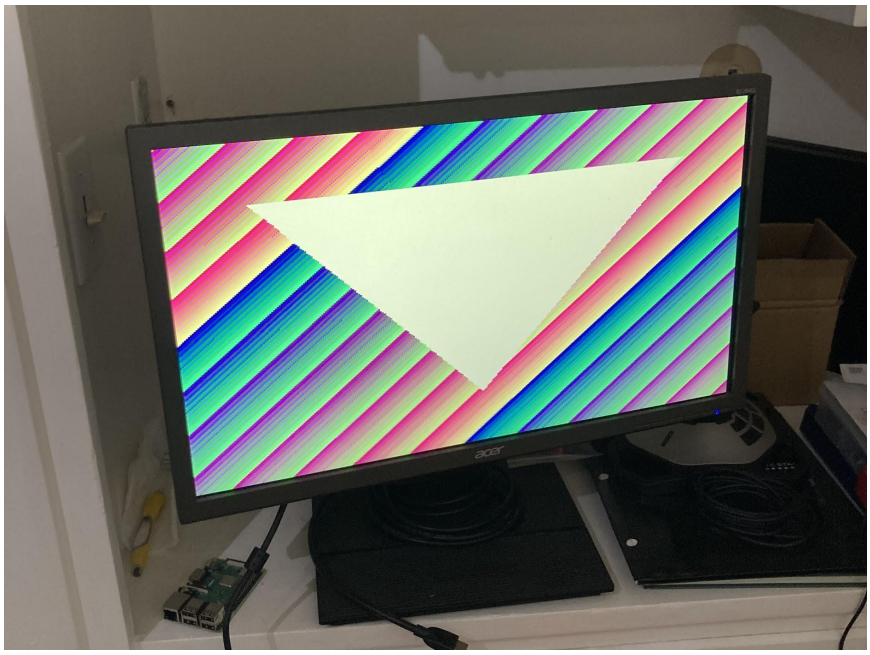

Figure 4: Had a little fun and displayed a rasterized triangle to the screen.

C code for above

```
##include "draw.h"
#include "otter.h"
#define Width 320
#define Height 240

typedef enum
{
  false, true
}

bool;

typedef struct
{
  int x;
  int y;
}

point;


bool edgeFunction(const point a, const point b, const point c)
{
  return ((c.x - a.x) *(b.y - a.y) - (c.y - a.y) *(b.x - a.x) >= 0);
}
```

```
        void main()
        {
          while (1)
          {
            int x, y;
            point V0;
            V0.x = 160;
            V0.y = 200;

            point V1;
            V1.x = 40;
            V1.y = 40;

            point V2;
            V2.x = 280;
            V2.y = 40;

            int add;
            for (add = 0; add < 10000; add++)
            {
              for (x = 0; x < Width; x++)
              {
                for (y = 0; y < Height; y++)
                {
                  point p;
                  p.x = x;
                  p.y = y;

                  if ( edgeFunction(V0, V1, p) || edgeFunction(V1, V2, p) ||
edgeFunction(V2, V0, p) )
                  {
                    //white
                    draw_dot(x, y, 0b11111111);
                  }
                  else
                  {

                   // point p is inside triangle defined by vertices v0, v1, v2
                    draw_dot(x, y, (unsigned char)(x + y + add));
                  }
                }
              }
            }
          }

          while (1);    //j 0
        }
```

Some useful notes from our project just in case someone decides to look over this.

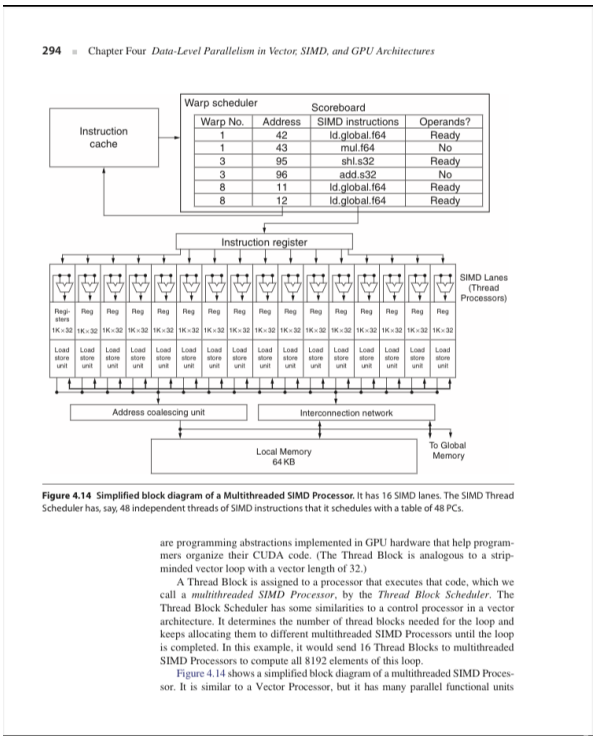

Figure 5: Taken from "Computer Architecture: A Quantitative Approach"

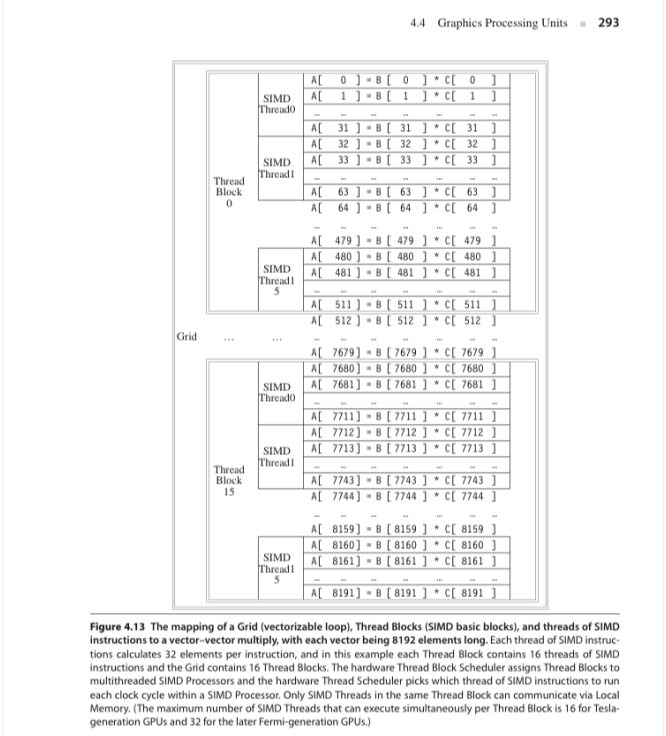

Figure 6: Taken from "Computer Architecture: A Quantitative Approach"