2016

Computer Architecture Lab

LAB 6 :GPU PARALLELISM AND PERFORMANCE

JEFFREY HUANG | RUID:159-00-4687

Code used for Assignment 1

```
// Jeffrey Huang
// RUID: 159-00-4687
// NETID: jh1127
// Assignment 1

.data
    .reg    .s32    r2, r3, r4, r5;         // a, b, c, d = r10, r11, r12, r13
    .global .s32    e[10];                  // initializing space for array e
    .global .s32    f[10];                  // intializing space for array f

main:
    add.s32 r0, 10, 0;                      // initializing r0 (n) to 10
    add.s32 r1, 0, 0;                       // initializing r1 (i) to 0
    add.s32 r2, 0, 0;                       // initializing r2 (a) to 0
    add.s32 r3, 0, 0;                       // initializing r3 (b) to 0
    add.s32 r4, 0, 0;                       // initializing r4 (c) to 0
    add.s32 r5, 0, 0;                       // initializing r5 (d) to 0

loop:
    setp.lt.s32 r6, r1, r0;                 // set r6 to r1 < r0
    @!r6 bra end;                           // branch to end, if r1 (i) < r0 (n = 10)
    mul.s32 r7, r2, r3;                     // r7 = r2(a) * r3(b)
    mul.s32 r8, r4, r5;                     // r8 = r2(c) * r5(d)
    sub.s32 r9, r7, r8;                     // r9 = r7 - r8 = (a * b) - (c * d)
    add.s32 r7, r7, r8;                     // r7 = r7 + r8 = (a * b) + (c * d)
    mov.s32 e[r1], r9;                      // moving r9 to e[i]
    mov.s32 f[r1], r7;                      // moving r7 to f[i]
    add.s32 r2, r2, r3;                     // a = a + b
    add.s32 r4, r4, r5;                     // c = c + d
    add.s32 r1, r1, 1;                      // i++
    @r6 bra loop;                           // branch to loop, if !(i < n)

end:
    exit;                                   // ending the program
```

Code used for Assignment 2

```
# Jeffrey Huang
# RUID: 159-00-4687
# NETID: jh1127
# Assignment 2

.data
        .global .s32    y[10];      // initializing space for array y
        .global .s32    z[10];      // initializing space for array z
                                    // r0 = 64 bit data extending to r1
                                    // r1 = base address for array x
main:
    add.s32 r2, 0, 0;               // initializing r2 (pos) to 0
    add.s32 r3, 0, 0;               // initializing r3 (neg) to 0
    add.s32 r4, 0, 0;               // initializing r4 (zero) to 0
    add.s32 r5, 1, 0;               // initializing r5 (i) to 1
    add.s32 r7, 0, 0;               // initializing r7 (offset) to 0

loop:
    ld.global.s32 r8, [r1+r7];      // load memory value in x[i] into r8
    setp.eq.s32 r6, r8, 0;          // set r6 to r8 == 0
    @r6 bra zero;                   // if r8 == 0, branch to zero
    setp.lt.s32 r6, r8, 0;          // set r6 to r8 < 0
    @r6 bra negative;               // if r8 < 0, branch to negative
    @!r6 bra positive;              // if r8 > 0, branch to positive

zero:
    add.s32 r4, r4, 1;              // incrementing the zero counter (zero)
    @r6 bra increment;              // branch to increment

negative:
    mov.s32 y[r3], r8;              // moving x[i] to y[neg]
    add.s32 r3, r3, 1;              // incrementing the negative counter (neg)
    @r6 bra increment;              // branch to increment

positive:
    mov.s32 z[r2], r2;              // moving x[i] to z[pos]
    add.s32 r2, r2, 1;              // incrementing the positive counter (pos)
    @r6 bra increment;              // branch to increment

increment:
    add.s32 r7, r7, 4;              // increments offset by 4 for word
    add.s32 r5, r5, 1;              // incrementing i by 1
    setp.le.s32 r6, r5, 10;         // set r6 to i <= 10
    @r6 bra loop;                   // branch to loop, if i <= 10
    @!r6 bra end;                   // branch to end, if !(i <= 10)

end:
    exit;                           // ending the program
```

Code used for Assignment 3

```
# Jeffrey Huang
# RUID: 159-00-4687
# NETID: jh1127
# Assignment 3

## PART (a)
#-------------------------------------------------------------------------------
## MIPS Version
# Registers Used:       $s0 -> value of n
#                       $f0 -> k! -> 1/k!
#                       $f12 -> output register

.data
    input:      .asciiz     "Enter the value of n: "

.text
main:
    li.s $f12, 0.0              # loading intial value of register $12
    li.s $f1, 1.0               # loading constant 1 into register $f1
    li $v0, 4                   # loading syscall service for print_string
    la $a0, input               # loading syscall argument for print_string
    syscall                     # making syscall to print_string
    li $v0, 5                   # loading syscall service for read_int
    syscall                     # making syscall to read_int
    move $s0, $v0               # moving user input into register $s0

loop:
    bltz $s0, printOutput       # if $s0 == 0, branch to printOutput
    move $a0, $s0               # moving $s0 to $a0 for factorial function use.
    jal fact                    # jump and load factorial.
    mtc1 $v0, $f0               # moving k! to floating point regsister
    cvt.s.w $f0, $f0            # converting from double to single.
    div.s $f0, $f1, $f0         # 1 / k!
    add.s $f12, $f12, $f0       # 1/(k-1)! + 1/k!
    addi $s0, $s0, -1           # decrementing the value of n
    j loop                      # jump to loop

printOutput:
    li $v0, 2                   # loading syscall service for print_float
    syscall                     # making syscall to print_float
    li $v0, 10                  # loading syscall service for end_program
    syscall                     # making syscall service to end_program

# FACTORIAL FUNCTION - Provided in lecture 3
fact:
    addi $sp, $sp, -8           #adjust stack pointer
    sw $ra, 4($sp)              #save return address
    sw $a0, 0($sp)              #save argument n
    slti $t0, $a0, 1            #test for n < 1
    beq $t0, $zero, L1          #if n >=1, go to L1
    addi $v0, $zero, 1          #else return 1 in $v0
    addi $sp, $sp, 8            #adjust stack pointer
    jr $ra                      #return to caller

L1:
    addi $a0, $a0, -1           #n >=1, so decrement n
    jal fact                    #call fact with (n-1)
```

```
bk_f:
    lw $a0, 0($sp)              #restore argument n
    lw $ra, 4($sp)              #restore return addr
    addi $sp, $sp, 8            #adjust stack pointer
    mul $v0, $a0, $v0           #$v0 = n * fact(n-1)
    jr $ra                      #return to caller
#-------------------------------------------------------------------------------
// PTX ISA Version

.data

main:
//  add.u32 r1, 0, 10;      // given that n is already stored into r1
    add.f32 r2, 0, 0;       // initialize r2 as 0 -> output register
    add.f32 r3, 1, 0;       // initialize r3 as 1 -> factorial = k!

loop:
    cvt.f32.u32, r4, r1;        // converting counter to floating counter
factLoop:
    mul.f32 r3, r3, r4;     // r4 * r3
    sub.f32 r4, r4, 1.0;        // decrement r4 by 1
    set.lt.f32 r5, r4, 0;   // set r5 to r4 < 0
    @r5 bra summation;      // branch to summation if r4 < 0
    @!r5 bra factLoop;      // branch to factLoop if !(r4 < 0)
summation:
    rcp.f32 r6, r3;         // 1 / k!
    add.f32 r2, r2, r6;     // 1 / (k-1)! + 1 / k!
    sub.f32 r1, r1, 1.0;        // decrementing r1 by 1
    setp.eq.s32 r5, r1, 0;  // set r5 to r1 == 0
    @r5 bra endProgram;     // branch to printOutput if r1 == 0
    @!r5 bra loop;          // branch to loop if r1 != 0
endProgram:
    exit;                   // end the program

#-------------------------------------------------------------------------------
## PART (b)
#-------------------------------------------------------------------------------
# The PTX ISA version is significantly shorter than the MIPS version. The number of
# cycles for the MIPS version for the polynomial of degree 10 would result in around
# about 10000 cycles. As compared to the PTX ISA version, the number of cycles would
# dramatically decrease because there is no stack pointer to adjust or factorial
# function to call upon. It just compares and loops through the factorial and
# reciprocates the value and adds the sum together. Thus the number of cycles for
# the PTX ISA version would come to around a close 5000 cycles, which is about half
# the number of cycles required for the MIPS version.
```

Code used for Assignment 4

```
# Jeffrey Huang
# RUID: 159-00-4687
# NETID: jh1127
# Assignment 4

## MIPS Verison

.data 0x10000800
MatrixA_0:   .word    2,  4,  6,  8, 10
MatrixA_1:   .word   12, 14, 16, 18, 20
MatrixA_2:   .word   22, 24, 26, 28, 30
MatrixA_3:   .word   32, 34, 36, 38, 40
MatrixA_4:   .word   42, 44, 46, 48, 50


.text 0x00400000
.globl main

main:
    la $t0, MatrixA_0               # loading first row of MatrixA to $t0
    la $t1, MatrixA_1               # loading second row of MatrixA to $t1
    la $t2, MatrixA_2               # loading third row of MatrixA to $t2
    la $t3, MatrixA_3               # loading fourth row of MatrixA to $t3
    la $t4, MatrixA_4               # loading fifth row of MatrixA to $t4

# I don't know how to do this. I can't seem to figure out the entire algorithm
# to process the determinant.
```

Code used for Assignment 5

```
# Jeffrey Huang
# RUID: 159-00-4687
# NETID: jh1127
# Assignment 5

## MIPS Version

.data
input:      .asciiz     "Enter value of x where 0 <= x <= π/2 : "
output1:    .asciiz     "\n sinh^-1(x) =  "
output2:    .asciiz     "\n tanh^-1(x) =  "

.text
main:
    li $v0,4                        # loading syscall service to print_string
    la $a0,input                    # loading syscall argument for print_string
    syscall                         # making syscall service to print_string
    li $v0,6                        # loading syscall service to read_float
    syscall                         # making syscall service to read_float
    mov.s $f1, $f0                  # moving user input to $f1
    li.s $f2, 0.3333333             # initializing value of $f2 to 1/3
    li.s $f3,0.20                   # initializing value of $f3 to 1/5
    li.s $f4,0.142857               # initializing value of $f4 to 1/7
    li.s $f5, 0.166667              # initializing value of $f5 to 1/6
    li.s $f6,0.075                  # initializing value of $f6 to 3/40
    li.s $f7, 0.267858              # initializing value of $f7 to 2/7
## sinh^-1(x)
    mul.s $f8, $f1, $f1             # x^2
    mul.s $f8, $f8, $f1             # x^3
    mul.s $f9, $f8, $f5             # x^3/6
    mul.s $f8, $f8, $f1             # x^4
    mul.s $f8, $f8, $f1             # x^5
    mul.s $f10, $f8, $f6            # 3x^5/40
    mul.s $f8, $f8, $f1             # x^6
    mul.s $f8, $f8, $f1             # x^7
    mul.s $f11, $f8, $f7            # 2x^7/7
    sub.s $f12,$f1,$f9              # x - x^3/6
    add.s $f12,$f12,$f10            # x - x^3/6 + 3x^5/40
    sub.s $f12,$f12,$f11            # x - x^3/6 + 3x^5/40 - 2x^7/7
    li $v0,4                        # loading syscall service to print_string
    la $a0,output1                  # loading syscall argument for print_string
    syscall                         # making syscall service to print_string
    li $v0,2                        # loading sysall service to print_float
    syscall                         # making syscall service to print_float
## tanh^-1(x)
    mul.s $f8, $f1, $f1             # x^2
    mul.s $f8, $f8, $f1             # x^3
    mul.s $f9,$f8,$f2               # x^3/3
    mul.s $f8,$f8,$f1               # x^4
    mul.s $f8,$f8,$f1               # x^5
    mul.s $f10,$f8,$f3              # x^5/5
    mul.s $f8,$f8,$f1               # x^6
    mul.s $f8,$f8,$f1               # x^7
    mul.s $f11, $f8,$f4             # x^7/7
    add.s $f12,$f1,$f9              # x + x^3/3
    add.s $f12,$f12,$f10            # x + x^3/3 + x^5/5
    add.s $f12,$f12,$f11            # x + x^3/3 + x^5/5 + x^7/7
    la $a0,output2                  # loading syscall argument to print_string
    li $v0,4                        # loading syscall service to print_string
    syscall                         # making syscall service to print_string
    li $v0, 2                       # loading syscall argument to print_float
    syscall                         # making syscall service to print_float
    li $v0, 10                      # loading syscall service to end_program
    syscall                         # making syscall service to end_program
```

```
# Jeffrey Huang
# RUID: 159-00-4687
# NETID: jh1127
# Assignment 5

## PTX ISA Version

.data
// Assuming that r1 contains the value of x where x exists in the [0,π/2]
main:
    div.f32 r2, 1, 3                    // initializing value of $r2 to 1/3
    div.f32 r2, 1, 5                    // initializing value of $r2 to 1/5
    div.f32 r2, 1, 7                    // initializing value of $r2 to 1/7
    div.f32 r2, 1, 6                    // initializing value of $r2 to 1/6
    div.f32 r2, 3, 40                   // initializing value of $r2 to 3/40
    div.f32 r2, 2, 7                    // initializing value of $r2 to 2/7

// sinh^-1(x)
mul.f32 $r8, $r1, $r1;                  // x^2
    mul.f32 $r8, $r8, $r1;              // x^3
    mul.f32 $r9, $r8, $r5;              // x^3/6
    mul.f32 $r8, $r8, $r1;              // x^4
    mul.f32 $r8, $r8, $r1;              // x^5
    mul.f32 $r10, $r8, $r6;             // 3x^5/40
    mul.f32 $r8, $r8, $r1;              // x^6
    mul.f32 $r8, $r8, $r1;              // x^7
    mul.f32 $r11, $r8, $r7;             // 2x^7/7
    sub.f32 $r63,$r1,$r9;               // x - x^3/6
    add.f32 $r63,$r63,$r10;             // x - x^3/6 + 3x^5/40
    sub.f32 $r63,$r63,$r11;             // x - x^3/6 + 3x^5/40 - 2x^7/7

// tanh^-1(x)
    mul.f32 $r8, $r1, $r1;              // x^2
    mul.f32 $r8, $r8, $r1;              // x^3
    mul.f32 $r9,$r8,$r2;                // x^3/3
    mul.f32 $r8,$r8,$r1;                // x^4
    mul.f32 $r8,$r8,$r1;                // x^5
    mul.f32 $r10,$r8,$r3;               // x^5/5
    mul.f32 $r8,$r8,$r1;                // x^6
    mul.f32 $r8,$r8,$r1;                // x^7
    mul.f32 $r11, $r8,$r4;              // x^7/7
    add.f32 $r64,$r1,$r9;               // x + x^3/3
    add.f32 $r64,$r64,$r10;             // x + x^3/3 + x^5/5
    add.f32 $r64,$r64,$r11;             // x + x^3/3 + x^5/5 + x^7/7
    exit;

// The PTX ISA Version of the MIPS code that accomplishes the same thing still takes
// about the same amount of time since there is no need to print or take input from.
```