# Assignment 2 – Text part.

**Tycho Canter Cremers**

**All code can be found in the weekly 2 folder.**

## Task I – CUDA programming

**I.1**

**I.2**

**I.3**

I timed the futhark sequential version and flat parallel version to compare them. On the standard suggested data set given in the comments the sequential version was just slightly faster than the parallel version. When I increased the dataset 10 fold the results below showed that in that case the parallel version is about twice as fast.

[mns267@a00332 SpMatVecMul-Futhark]$ futhark-dataset --i32-bounds=0:9999 -g [10000000]i32 --f32-bounds=-7.0:7.0 -g [10000000]f32 --i32-bounds=1000:1000 -g [10000]i32 --f32-bounds=-10.0:10.0 -g [10000]f32 | ./spMVmult-flat -r 10 -t /dev/stderr

7269

7232

7394

7293

7297

7276

7265

7346

7258

7307

seq

[mns267@a00332 SpMatVecMul-Futhark]$ futhark-dataset --i32-bounds=0:9999 -g [10000000]i32 --f32-bounds=-7.0:7.0 -g [10000000]f32 --i32-bounds=1000:1000 -g [10000]i32 --f32-bounds=-10.0:10.0 -g [10000]f32 | ./spMVmult-seq -r 10 -t /dev/stderr

15098

15053

15061

15049

15099

15094

15100

15033

15037

15097

## Task II – Hardware

**II.1**

**A.**

I hope this is correct. I honestly can’t find a good resource on where to actually put those NOOPS. I’m assuming you need on after every dependency hazard.

SEARCH:

LW R5, 0(R3) /I1 Load item

SUB R6, R5, R2 /I2 compare with key

NOOP

BNEZ R6, NOMATCH /I3 check for match

NOOP

ADDI R1, R1, #1 /I4 count matches

NOMATCH:

ADDI R3, R3, #4 /I5 next item

BNE R4, R3, SEARCH /I6 continue until all items

**B.**

Again, I really hoping I’m doing this right.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MATCH |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c10 | c11 | c12 | c13 | c14 |
| SEARCH | L1 | LW R5,0(R3) | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |
|  | L2 | SUB R6, R5, R2 |  | F | D | D | D | EX | ME | WB |  |  |  |  |  |  |
|  | L3 | BNEZ, R6, NOMATCH |  |  | F | F | F | D | D | D | EX | ME | WB |  |  |  |
|  | L4 | ADDI R1, R1, #1 |  |  |  |  | F | F | F | F | D | EX | ME | WB |  |  |
| NOMATCH | L5 | ADDI R3, R3, #4 |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |  |
|  | L6 | BNE R4, R3, SEARCH |  |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| NO MATCH |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c10 | c11 | c12 | c13 | c14 |
| SEARCH | L1 | LW R5,0(R3) | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |
|  | L2 | SUB R6, R5, R2 |  | F | D | D | D | EX | ME | WB |  |  |  |  |  |  |
|  | L3 | BNEZ, R6, NOMATCH |  |  | F | F | F | D | D | D | EX | ME | WB |  |  |  |
|  | L4 | ADDI R1, R1, #1 |  |  |  |  | F | F | F | F | NOP |  |  |  |  |  |
| NOMATCH | L5 | ADDI R3, R3, #4 |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |  |
|  | L6 | BNE R4, R3, SEARCH |  |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |

L2 has to wait until the c5 WB to be able to access R5. Following, L3 has to wait for L2’s writeback until he can access R6. This creates a huge delay.

So in both cases (MATCH and NO MATCH) it takes 14 clocks because you don’t gain any time from stopping L4.

**C.**

Due to register forwarding L2 now only has to wait until c4 to access R5. Additionally, L3 has to wait one less cycle too, to access R6 at c7. This saves 2 clocks. So both cases can now do with 12 clocks.

**D.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| MATCH |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c10 | c11 | c12 | c13 | c14 |
| SEARCH | L1 | LW R5,0(R3) | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |
|  | L2 | SUB R6, R5, R2 |  | F | D | D | EX | ME | WB |  |  |  |  |  |  |  |
|  | L3 | BNEZ, R6, NOMATCH |  |  | F | F | D | EX | ME | WB |  |  |  |  |  |  |
|  | L4 | ADDI R1, R1, #1 |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |  |
| NOMATCH | L5 | ADDI R3, R3, #4 |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |
|  | L6 | BNE R4, R3, SEARCH |  |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| D |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| NO MATCH |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c10 | c11 | c12 | c13 | c14 |
| SEARCH | L1 | LW R5,0(R3) | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |
|  | L2 | SUB R6, R5, R2 |  | F | D | D | EX | ME | WB |  |  |  |  |  |  |  |
|  | L3 | BNEZ, R6, NOMATCH |  |  | F | F | D | EX | ME | WB |  |  |  |  |  |  |
|  | L4 | ADDI R1, R1, #1 |  |  |  |  | F | mop | |  |  |  |  |  |  |  |
| NOMATCH | L5 | ADDI R3, R3, #4 |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |
|  | L6 | BNE R4, R3, SEARCH |  |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |

Due to full forwarding L3 gets R6 from the start of the EX phase in L2, saving another clock cycle. Now both cases can do it in 11 clocks.

**E.**

Total number of clocks of 2 iterations = 11 \* 2 = 22 clocks.

If I did it correctly, unrolling the loop twice combined with register forwarding should remove all the dependency hazards for matches and not matches. So the total amount of clocks for a loop unrolled twice is 16:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| MATCH |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c10 | c11 | c12 | c13 | c14 | c15 | c16 |
| SEARCH | L1 | LW R5,0(R3) | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |  |  |
|  | L2 | LW R52,0(R32) |  | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |  |
|  | L3 | SUB R6, R5, R2 |  |  | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |
|  | L4 | SUB R62, R52, R22 |  |  |  | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |
| NOMATCH | L5 | BNEZ, R6, NOMATCH |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |  |  |  |
|  | L6 | ADDI R1, R1, #2 |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |  |  |
|  | L7 | ADDI R3, R3, #8 |  |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |  |
| NOMATCH | L8 | BNEZ, R62, NOMATCH | |  |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |
|  | L9 | ADDI R12, R12, #2 |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |
|  | L10 | ADDI R32, R32, #8 |  |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |  |  |
|  | L11 | BNE R4, R3, SEARCH |  |  |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |  |
|  | L12 | BNE R42, R32, SEARCH | |  |  |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| NO MATCH |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c10 | c11 | c12 | c13 | c14 | c15 | c16 |
| SEARCH | L1 | LW R5,0(R3) | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |  |  |
|  | L2 | LW R52,0(R32) |  | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |  |
|  | L3 | SUB R6, R5, R2 |  |  | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |  |
|  | L4 | SUB R62, R52, R22 |  |  |  | F | D | EX | ME | WB |  |  |  |  |  |  |  |  |
|  | L5 | BNEZ, R6, NOMATCH |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |  |  |  |
|  | L6 | ADDI R1, R1, #2 |  |  |  |  |  | F | nop |  |  |  |  |  |  |  |  |  |
| NOMATCH | L7 | ADDI R3, R3, #8 |  |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |  |
|  | L8 | BNEZ, R62, NOMATCH | |  |  |  |  |  |  | F | D | EX | ME | WB |  |  |  |  |
|  | L9 | ADDI R12, R12, #2 |  |  |  |  |  |  |  |  | F | nop |  |  |  |  |  |  |
| NOMATCH | L10 | ADDI R32, R32, #8 |  |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |  |  |
|  | L11 | BNE R4, R3, SEARCH |  |  |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |  |
|  | L12 | BNE R42, R32, SEARCH | |  |  |  |  |  |  |  |  |  |  | F | D | EX | ME | WB |

Which means that unrolling the loop would be usefull.

I don’t think delayed branches would help because they didn’t cause any delays in the first place.

**Taske II.2**

**A.**

ADDI R6,R0,#1

L.V V1,0(R1),R6 //Load slice of vector x

L.V V2,0(R2),R6 //Load slive of vector y

MUL.V V3,V1,V2 //Multiply x and y

ADD.V V4,V4,V3 // P += x\*y

SUBBI R1,R1,#512 //jump to next slices,

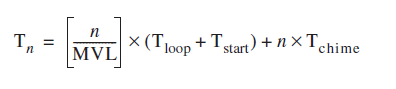
SUBBI R2,R2,#512 //i.e., 8\*64

SUBBI R3,R3,#64

BNEZ R3,LOOP

**B.**

From https://www.cs.indiana.edu/~achauhan/Teaching/B649/2010-Spring/StudentPresns/Vector\_processors.pdf

Execution time =

I can’t find anything about loop overhead or Tloop but in the examples they use 15. So I’ll go with that.

N = 1024

MVL = 64

Tloop = 15

Tstart = 2\*30+10+5 = 75

Tchime = 3 (I think)

So Tn = 16 \* (15 + 75) + 1024 \* 3 = 4512

**C.**

(1024\*1024) = 1048576 dot products to be calculated.

So it’s 1048576 \* answer to b.

1048576 \* 4512 = 4 731 174 912