Bonus4 va, IS2202

Linghan Zhao
020717-5563
linghanz@kth.se

March 1, 2024

FP ADD and FP MUL Execution pipeline length $PL = 5 + d = 5 + 1 = 6$ (stages)
Number of Memory banks $\#B = 2^{(3+M)} = 2^3 = 8$
Bank busy time $Tbusy = 4 + D = 4 + 1 = 5$(ccs)
Memory Bus access latency $Taccess = 2 + y = 2 + 2 = 4$(ccs)
$Tbusy + Taccess = 9$(ccs)

Problem 1. Matrix Multiplication part I

a)
Because the number of banks is 8, which is larger than the bank busy time 5 ccs,
For stride = 1, the load will take $5 + 4 + 64 = 73$ ccs.
For stride = 2, assume bank0 to bank7, time is determined by bank6. The first time bank6 access is 13rd ccs, then wait until 15th ccs for bank0 to load. So total time should be $13 + 5 \times \frac{64-4}{4} = 88$ ccs.
For stride = 4, time is determined by bank4. The first load for bank6 is 11th ccs, then bank0 needs to wait until 15th ccs, so second for bank4 is 16th ccs. $11 + 5 \times \frac{64-2}{2} = 166$ ccs.
For stride = 8, $10 + (64 - 1) \times 5 = 325$ ccs.
For stride = 16, $10 + 5 \times 63 = 325$ ccs.
For stride = 32, $10 + 5 \times 63 = 325$ ccs.
For stride = 64, $10 + 5 \times 63 = 325$ ccs.
b)
For A[i,j], A is accessed like A[0,0], then A[0,1], so the stride is 1.
For B[i,k], B is accessed like B[0,0], then B[0,1], so the stride is 1.
For C[k,j], C is accessed like C[0,0], then C[1,0], so the stride is 64.
c)

```
    LIU R5,#512 ; Row Stride = 64*8
    LIU Ri,#64 ; Nr of iterations
Loop_i:
    LIU Rj,#64 ; Nr of iterations
    LV.D V1,Rb ; Load B[i][k], row i i â€\ Convoy 1
Loop_j:
    LVWS.D V2,(Rc,R5) ; Load c[k][j], column j j â€\ Convoy 1
    MULVS.D V3,V1,V2 ; B[i][k]*c[k][j]
    LIU Rk,#64 ; Nr of elements
    L.D F0,#0 ; s=A[i][j]=0.0
Loop_k:
    MOVVI.D F1,(V3,Rk) ; bd = B[i][k] * c[k][j];
    ADD.D F0,F0,F1 ; s=s + B[i][k] * c[k][j];
    SUBIU Rk,Rk,#1 ; compute bound (k)
    BNEZ Rk,Loop_k ; Check if finished

    S.D Ra,F0 ; Store result
    ADDIU Ra,#8 ; Start of next value in A[i][j]
    ADDIU Rc,Rc,#512 ; Start of next column in C
    SUBIU Rj,Rj,#1 ; compute bound (j)
    BNEZ Rj,Loop_j ; Check if finished

    SUBIU Rc,Rc,#32768 ; Start of C
    ADDIU Rb,Rb,#512 ; Start of next row in B
    SUBIU Ri,Ri,#1 ; compute bound (i)
    BNEZ Ri,Loop_i ; Check if finished
```

d)
Convoys:
Loop_k: 0
$T_k = 0ccs$
Loop_j: 1. LVWS MULVS
$T_j = 1 * 64 + 64 * T_k$
Loop_i: 1. LV
$T_i = 1 * 64 + 64 * T_j$
Execution time $= 64 * T_i = 64 * 64 * 65 = 266240$ ccs
e)
Overheads is included,
For loop k, $T'_{loop_k} = 4 * 64 = 256\ ccs$.
For loop j, there are 9 scalar instructions. The memory access latency is $T_{access} = 4\ ccs$ and the multiplication operation start-up time corresponds to the pipeline stages, which is $T_{MUL} = PL = 6\ ccs$. So $T'_{loop_j} = 64 * (9 + 4 + 64 * 5 + 6 + T'_{loop_k}) = 38080\ ccs$.
For loop i, there are 6 scalar instructions. The start-up overhead for memory access is $T_{memory} = 4(accesslatency) + 4(pipelineemptyorfirstbusytime)\ ccs$. So $T'_{loop_i} = 64 * (6 + 4 + 4 + 64 + T'_{loop_j}) = 2442112\ ccs$.
There are 2 more scalar instructions outside the loop. So the $T_{total} = T'_{loop_i} + 2 = 2442114\ ccs$.
f)
If matrix C is of the size [64][65], the stride should be 65.
The loop of j should be 65 other than 64.
The execution time including overheads should be :
For loop k, $T'_{loop_k} = 4 * 64 = 256\ ccs$.
For loop j, there are 9 scalar instructions. The memory access latency is $T_{access} = 4\ ccs$ and the multiplication operation start-up time corresponds to the pipeline stages, which is $T_{MUL} = PL = 6\ ccs$. So $T'_{loop_j} = 65 * (9 + 4 + 65 * 5 + 6 + T'_{loop_k}) = 39000\ ccs$.
For loop i, there are 6 scalar instructions. The start-up overhead for memory access

is $T_{memory} = 4(accesslatency) + 4(pipelineemptyorfirstbusytime)\ ccs$. So $T'_{loop_i} = 64*(6+4+4+64+T'_{loop_j}) = 2500992\ ccs$.
There are 2 more scalar instructions outside the loop. So the $T_{total} = T'_{loop_i} + 2 = 2500994\ ccs$.

Problem 2. Matrix Multiplication part II

a)
For A[i,j], A is accessed like A[0,0], then A[1,0], so the stride is 64.
For B[i,k], B is accessed like B[0,0], then B[1,0], so the stride is 64.
For C[k,j], C is accessed like C[0,0], then C[1,0], so the stride is 64, but it needs 64 times to change a stride.
b)

```
    LIU R4,#64 ; Nr of iterations
    LIU R5,#512 ; Row Stride = 64*8
Loop_j:
    FILL.D V1,#0 ; V1=A jth column=0.0 i - Convoy 1
    ADDIU R6,Rc,#32768 ; Last address to load
Loop_k:
    L.D F0,Rc ; Rc pointing at D[k][j]
    LVWS.D V2,(Rb,R5) ; Rb pointing at B kth column 2nd i - Convoy
    MULVS.D V3,V2,F0 ; B[i][k]*c[k][j]
    ADDVV.D V1,V1,V3 ; A[i][j] + B[i][k] * c[k][j];
    ADDU Rc,Rc,R5 ; Increment c column address
    SUBU R20,R6,Rc ; compute bound (k)
    BNEZ R20,Loop_k ; check if done

    SVWS.D (Ra,R5), V1 ; Ra pointing at A jth column j - Convoy 1
    SUBIU Rc,Rc,#32760 ; Start of next column in c
    ADDIU Rb,Rb,#8 ; Start of next column in B
    ADDIU Ra,Ra,#8 ; Start of next column in A
    SUBIU R4,R4,#1 ; compute bound (j)
    BNEZ R4,Loop_j ; Check if finished
```

c)
Convoys (per inner loop):
Loop_k: 1
$T_k = 64ccs$
Loop_j: 1
$T_j = 1*64 + 64*T_k$
Loop_i: 1
$T_i = 1*64 + 64*T_j$
So total Convoys should be $64*(1+64) = 4160$ Execution time $= 64*T_i = 64*(64 + 64*64) = 266240$ ccs
d)
Overheads is included,
For loop k, $T'_{loop_k} = 7*64 = 448\ ccs$.

For loop j, there are 8 scalar instructions. The memory access latency is $T_{access} = 4\ ccs$ and the multiplication operation start-up time corresponds to the pipeline stages, which is $T_{MUL} = PL = 6\ ccs$. So $T'_{loop_j} = 64 * (8 + 4 + 64 * 5 + 6 + T'_{loop_k}) = 50304\ ccs$.
For loop i, there are 2 scalar instructions. The start-up overhead for memory access is $T_{memory} = 4(access latency) + 4(pipeline empty or first busy time)\ ccs$. So $T'_{loop_i} = 64 * (2 + 4 + 4 + 64 + T'_{loop_j}) = 3224192\ ccs$.
There are 2 more scalar instructions outside the loop. So the $T_{total} = T'_{loop_i} + 2 = 3224194\ ccs$.
e)
I think Malvinas' is better. Because the time consuming which includes overhead is less, more efficient. And the reduction of loop, conciser code and easier to understand.

References

Computer Architecture: A quantitative approach, 6th ed. by John Hennessy and David Patterson Morgan Kaufmann, 2017 ISBN: 978-0-12-811905-1.