Computer Architecture Project

Gautham Bolar : PES1UG20EC044

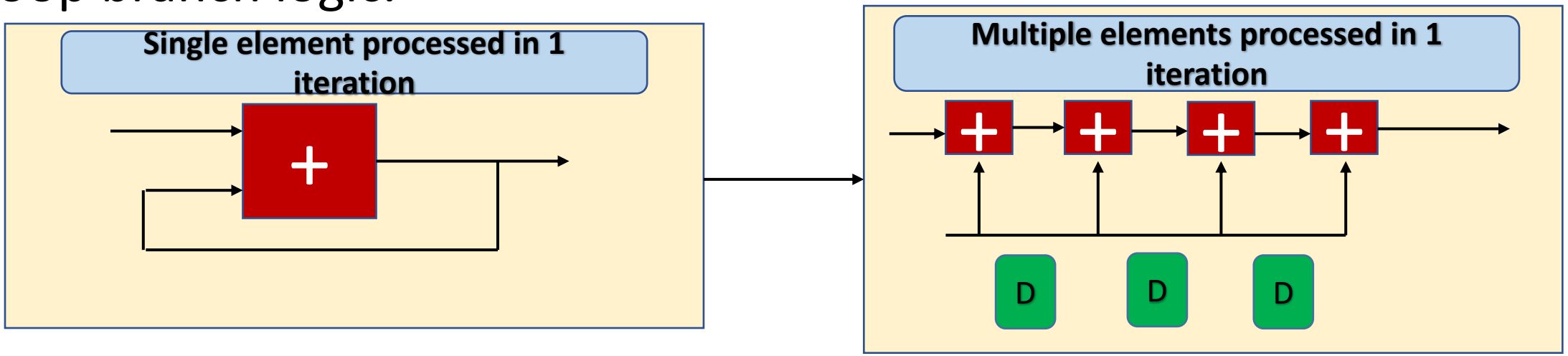
Dheemanth R Joshi: PES1UG20EC059

Aim and Motivation

- In this presentation we will analyze the performance of a pipelined processor. In particular we analyze how the compiler optimization technique called Loop-unrolling can be used to improve the performance of the pipeline at the cost of hardware overhead.

Loop Unrolling

- Loop unrolling is a technique used by the compiler to increase the number of instructions executed between executions of the loop branch logic.



- This reduces the number of times the loop branch logic is executed.

Drawbacks of Pipelined Processor Execution without loop unrolling (sum of n element array)

- **Conventional C code for sum of n element array**

Array and sum initialization

Execution of the sum

```
int a[100];
int c=0;
for (int i=0; i<100;i++)
{
 c=c+a[i];
}
```

Loop unrolling increases the period of flushing by computing more parameters in a single iteration, which increases clock utilization at the cost of more hardware

When the following C code is compiled, without loop unrolling optimization, the processor has to flush the pipeline if the branch fails the execution for every iteration

Assembly Codes for loop unrolled and conventional methods

- Conventional Code:

Here, processor uses fewer registers to compute the sum of the elements

```
.data
    array: .word 1,2,3,4,5,6,7,8,9,10,11,12,
1,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
    array_size: .word 100

.text
    .globl main
    .align 2

main:
    # Load array size
    lw t0, array_size

    # Load array pointer
    la t1, array

    # Initialize sum
    li t2, 0

    # Loop to sum array elements
    loop:
        lw t3, 0(t1)
        addi t1, t1, 4
        add t2, t2, t3
        addi t0, t0, -1
        bnez t0, loop

    # End of program
    nop
```

Only 1 element is loaded in each iteration and is added to the current sum

This code results in branch overhead for each element causing frequent flushing of the pipeline

Loop Unrolled Code

```
.data
    array: .word 1,2,3,4,5,6,7,8,9,10,11,12,13,14
1,62,63,64,65,66,67,68,69,70,71,72,73,74,75,76,77
    array_size: .word 100

.text
    .globl main
    .align 2

main:
    # Load array size
    lw t0, array_size

    # Load array pointer
    la t1, array

    # Initialize sum
    li t2, 0
    li a0,4
    # Loop unrolled to sum array elements
    loop_unrolled:
        lw t3, 0(t1)
        lw t4, 4(t1)
        lw t5, 8(t1)
        lw t6, 12(t1)
        add t2, t2, t3
        add t2, t2, t4
        add t2, t2, t5
        add t2, t2, t6
        addi t1, t1, 16
        addi t0, t0, -4
        ble t0,a0,loop
        j loop_unrolled
    # Loop to sum remaining array elements
    loop:
        lw t3, 0(t1)
        add t2, t2, t3
        addi t1, t1, 4
        addi t0, t0, -1
        bnez t0, loop

    # End of program
    nop
```

Processor utilizes more registers, but the clock cycles are significantly reduced as frequency of flushing the pipeline is significantly reduced

In this code, there is a branch overhead for every 4 elements, which results in better clock utilization

Here, Multiple elements are loaded in multiple registers in a single iteration

Results

- Pipeline Diagram of Conventional code

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
auipc x5 0x10000	IF	ID	EX	MEM	WB																
lw x5 400 x5		IF	ID	EX	MEM	WB															
auipc x6 0x10000			IF	ID	EX	MEM	WB														
addi x6 x6 -8				IF	ID	EX	MEM	WB													
addi x7 x0 0					IF	ID	EX	MEM	WB												
lw x28 0 x6						IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB			IF	ID
addi x6 x6 4							IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB			IF
add x7 x7 x28								IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB		
addi x5 x5 -1									IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB	
bne x5 x0 -16 <loop>										IF	ID	EX	MEM	WB			IF	ID	EX	MEM	WB
addi x0 x0 0											IF	ID						IF	ID		

Outcome: The processor uses more clock cycles because of frequent flushing

Pipeline getting flushed for every element in each iteration

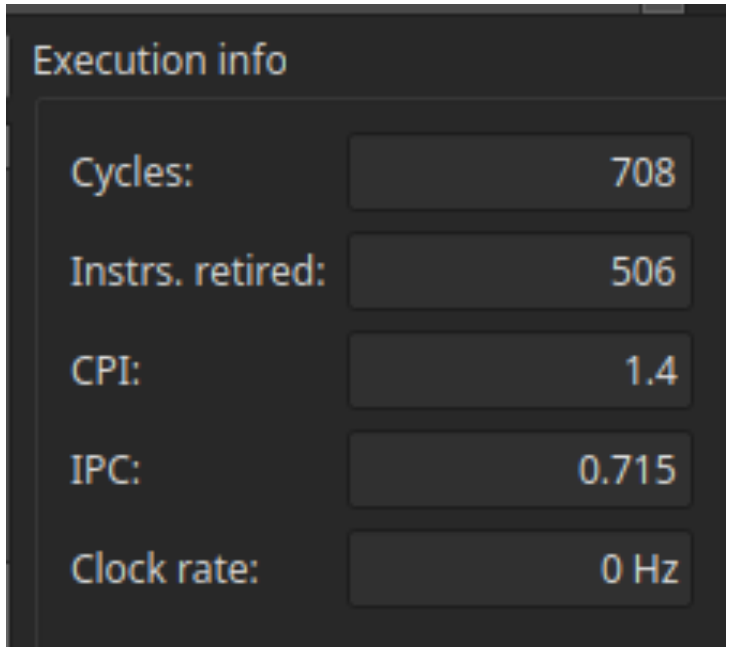
Pipeline diagram of loop unrolled code

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
auipc x5 0x10000	IF	ID	EX	MEM	WB																					
lw x5 400 x5		IF	ID	EX	MEM	WB																				
auipc x6 0x10000			IF	ID	EX	MEM	WB																			
addi x6 x6 -8				IF	ID	EX	MEM	WB																		
addi x7 x0 0					IF	ID	EX	MEM	WB																	
addi x10 x0 4						IF	ID	EX	MEM	WB																
lw x28 0 x6							IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB	
lw x29 4 x6								IF	ID	EX	MEM	WB										IF	ID	EX	MEM	WB
lw x30 8 x6									IF	ID	EX	MEM	WB										IF	ID	EX	MEM
lw x31 12 x6										IF	ID	EX	MEM	WB										IF	ID	EX
add x7 x7 x28											IF	ID	EX	MEM	WB										IF	ID
add x7 x7 x29												IF	ID	EX	MEM	WB										IF
add x7 x7 x30													IF	ID	EX	MEM	WB									
add x7 x7 x31														IF	ID	EX	MEM	WB								
addi x6 x6 16															IF	ID	EX	MEM	WB							
addi x5 x5 -4																IF	ID	EX	MEM	WB						
bge x10 x5 8 <loop>																	IF	ID	EX	MEM	WB					
jal x0 -44 <loop_unrolled>																		IF	ID	EX	MEM	WB				
lw x28 0 x6																			IF	ID						
add x7 x7 x28																				IF						
addi x6 x6 4																										
addi x5 x5 -1																										
bne x5 x0 -16 <loop>																										
addi x0 x0 0																										

The pipeline is flushed after 4 elements were loaded

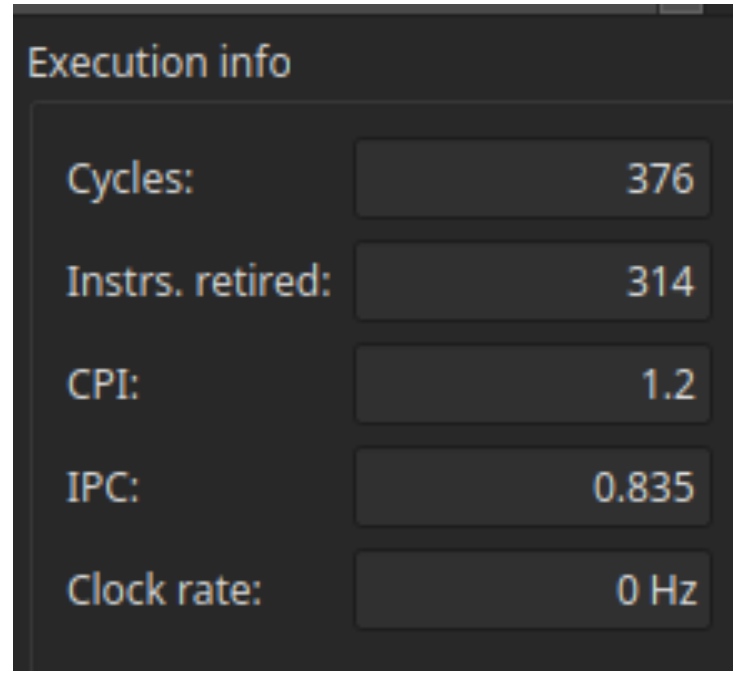
Outcome: The processor uses lesser number of clock cycles because frequency of flushing is less

Clock Utilization Comparison



Conventional Pipeline V/S Loop Unrolled Pipeline with 3 additional registers

Loop Unrolled Pipeline takes 332 clock cycles less than the standard approach with a 3 register hardware overhead



Simulation was run on 5 stage pipelined RISC V processor providing 3 additional load registers for the loop unrolled assembly code.

Conclusion

- C code of sum of n array elements was written in assembly and executed on a 5 stage pipelined RISC V processor with and without loop unrolling methods
- We observed a significant drop of clock cycle usage in the case loop unrolled instructions.
- However, the conventional instructions had lesser hardware overhead than the loop unrolled instructions.
- Based on the application, Loop unrolling is a good compiler optimization technique which can be used to optimize the processor utilities.

QNA

Thanks