Lecture 23: Board Notes: Introduction to Parallel Processing

Part A:

Consider a processor that does register renaming.

- A ROB **<u>IS</u>** part of this processor.
- There **<u>IS NOT</u>** a reservation station bypass.
  Therefore each instruction must spend at least 1 CC in a reservation station
- ALU operations take 1 CC to execute.
  - There are an unlimited number of functional units.
- If an instruction in a RS is waiting for data produced by a previously issued instruction, it will obtain that data during the previously issued instruction's WB stage – and can execute *<u>in the next CC.</u>*
  - i.e. if instruction *j* enters WB in cycle 7, and instruction *j+4* is waiting on data from instruction *j*, instruction *j+4*'s RS will be updated in cycle 7. Instruction *j+4* can execute in cycle 8
- Only 1 instruction is fetched and decoded during each clock cycle.
- Assume RS are unlimited.
- There are unlimited CDB resources. Therefore there are no structural hazard stalls when instructions need to write back.
- 2 instructions may commit in each CC.
- Multiply instructions take 4 CCs to execute, Adds take 1 CC to execute.

Fill in the pipe trace for the instruction sequence shown on the next page.
(F) Fetch, **(D)** Decode, **(RS)** Reservation Station, **(E)** Execute, **(W)** Write Back, **(C)** Commit

	PART A																		
	Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
A	Add r1 ,r1, r1	F	D	R	E	W	C												
B	Add r1, r1, r1		F	D	R	R	E	W	C										
C	Mul r1, r1, r1			F	D	R	R	R	E	E	E	E	W	C					
D	Sub r2, r2, r2				F	D	R	E	W	C	C	C	C	C					
E	Add r1, r2, r2					F	D	R	R	E	W	C	C	C	C				
F	Mul r2, r3, r3						F	D	R	E	E	E	E	W	C				
G	Add r1, r1, r1							F	D	R	R	E	W	C	C	C			

Part B: Example 1:

Assume we want to split up a problem to run on 1024 processors instead of 1. However, only half of the code is parallelizable. What speedup would we see from going from 1 processor to 1024?

$$\boxed{\text{speedup}_{\text{overall}} = \frac{1}{(1-F_{parallel}) + \dfrac{F_{parallel}}{Speedup_{parallel}}} = \frac{1}{(1-0.5) + \dfrac{0.5}{1024}} = 1.998!}$$

If the fraction of code that is parallelizable increases from 0.5 to 0.99, speedup is still only 1024!

Part B: Example 2:

Assume that we have a given workload that involves:

- Sum of 10 scalars
- 10 x 10 matrix sum

Part A:

What is the speedup if we increase the number of processors dedicated to the problem to 10? To 100?

1 Processor:

Time	=	(10 + 100)	x	t_{add}	=	110 x t_{add}

- 10 scalar adds + 100 adds for each element in the matrix

10 Processors:

Time	=	10 x t_{add}	+	(100/10) x t_{add}	=	20 x t_{add}
Speedup	=	110 x t_{add}	/	20 x t_{add}	=	5.5
		(best uniprocessor)			=	55 % of the potential (5.5 / 10)

100 Processors:

Time	=	10 x t_{add}	+	(100/100) x t_{add}	=	11 x t_{add}
Speedup	=	110 x t_{add}	/	11 x t_{add}	=	10
		(best uniprocessor)			=	10 % of the potential (10 / 100)

This assumed that the load can be balanced across processors

Part B:

What is the speedup if the matrix size is now 100 x 100?

1 Processor:

Time	=	(10 + 10000)	x	t_{add}	=	10010 x t_{add}

- 10 scalar adds + 10000 adds for each element in the matrix

10 Processors:

Time	=	10 x t_{add}	+	(10000/10) x t_{add}	=	1010 x t_{add}
Speedup	=	10010 x t_{add}	/	1010 x t_{add}	=	9.9
		(best uniprocessor)			=	99 % of the potential (9.9 / 10)

100 Processors:

Time	=	10 x t_{add}	+	(10000/100) x t_{add}	=	110 x t_{add}
Speedup	=	10010 x t_{add}	/	110 x t_{add}	=	91
		(best uniprocessor)			=	91 % of the potential (91 / 100)

This assumes load balancing is possible; if problem is smaller, scalar parts dominates (not parallel) ust halt to replace???