

UNIT 11B

Concurrency: Pipelining & Sorting Networks

15110 Principles of Computing, Carnegie
Mellon University - CORTINA

1

Concurrency

- Concurrency is the process of performing more than one process at a time.
- Computing has many ways to implement concurrency:
 - pipelining
 - parallel processing
 - multitasking
 - distributed computing

15110 Principles of Computing, Carnegie
Mellon University - CORTINA

2

Pipelining

- Pipelining is similar to an assembly line.
 - Instead of completing one computation before starting another, each computation is split into simpler sub-steps, and computations are started as others are in progress.

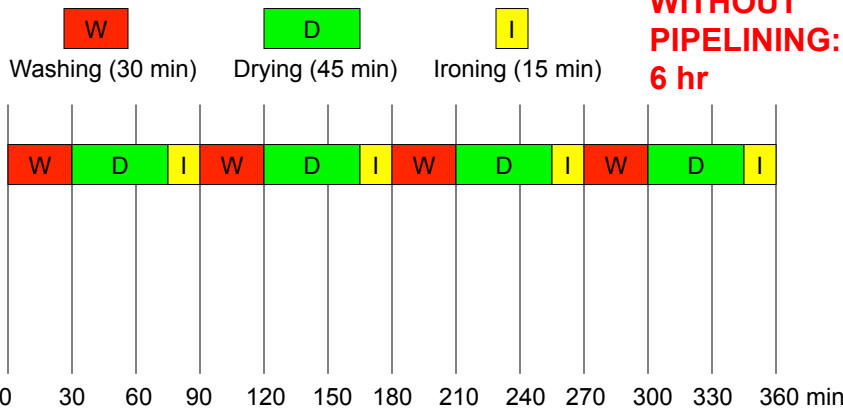


15110 Principles of Computing, Carnegie Mellon University - CORTINA

3

Laundry Without Pipelining

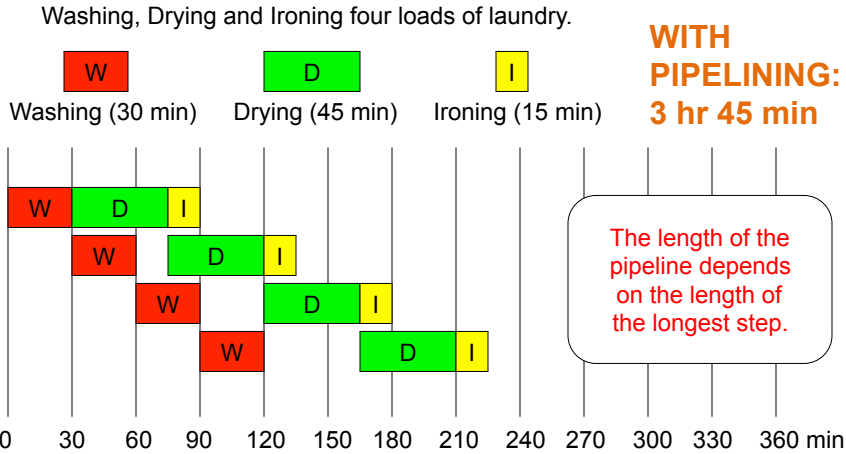
Washing, Drying and Ironing four loads of laundry.



15110 Principles of Computing, Carnegie Mellon University - CORTINA

4

Laundry With Pipelining



15110 Principles of Computing, Carnegie Mellon University - CORTINA

5

Matrix Multiplication

	hw	paper	exam1	exam2	exam3	final		weight		average
student1	95	90	93	91	85	92			student1	
student2	73	80	75	63	79	75	hw	0.15	student2	
student3	85	73	80	85	88	91	paper	0.1	student3	
student4	50	65	50	60	56	47	exam1	0.15	student4	
student5	100	95	98	96	96	90	exam2	0.15	student5	
student6	75	75	75	75	75	75	exam3	0.15	student6	
student7	90	80	80	90	100	100	final	0.3	student7	
student8	88	80	80	70	60	55			student8	

15110 Principles of Computing, Carnegie Mellon University - CORTINA

6

Matrix Multiplication

$$0 + 95 \cdot 0.15 + 90 \cdot 0.1 + 93 \cdot 0.15 + 91 \cdot 0.15 + 85 \cdot 0.15 + 92 \cdot 0.3 = 91.2$$

	hw	paper	exam1	exam2	exam3	final		weight		average
student1	95	90	93	91	85	92			student1	91.2
student2	73	80	75	63	79	75	hw	0.15	student2	
student3	85	73	80	85	88	91	paper	0.1	student3	
student4	50	65	50	60	56	47	exam1	0.15	student4	
student5	100	95	98	96	96	90	exam2	0.15	student5	
student6	75	75	75	75	75	75	exam3	0.15	student6	
student7	90	80	80	90	100	100	final	0.3	student7	
student8	88	80	80	70	60	55			student8	

Matrix Multiplication

$$0 + 73 \cdot 0.15 + 80 \cdot 0.1 + 75 \cdot 0.15 + 63 \cdot 0.15 + 79 \cdot 0.15 + 75 \cdot 0.3 = 74.0$$

	hw	paper	exam1	exam2	exam3	final		weight		average
student1	95	90	93	91	85	92			student1	91.2
student2	73	80	75	63	79	75	hw	0.15	student2	74.0
student3	85	73	80	85	88	91	paper	0.1	student3	
student4	50	65	50	60	56	47	exam1	0.15	student4	
student5	100	95	98	96	96	90	exam2	0.15	student5	
student6	75	75	75	75	75	75	exam3	0.15	student6	
student7	90	80	80	90	100	100	final	0.3	student7	
student8	88	80	80	70	60	55			student8	

Matrix Multiplication

$$0 + 85 \cdot 0.15 + 73 \cdot 0.1 + 80 \cdot 0.15 + 85 \cdot 0.15 + 88 \cdot 0.15 + 91 \cdot 0.3 = 85.3$$

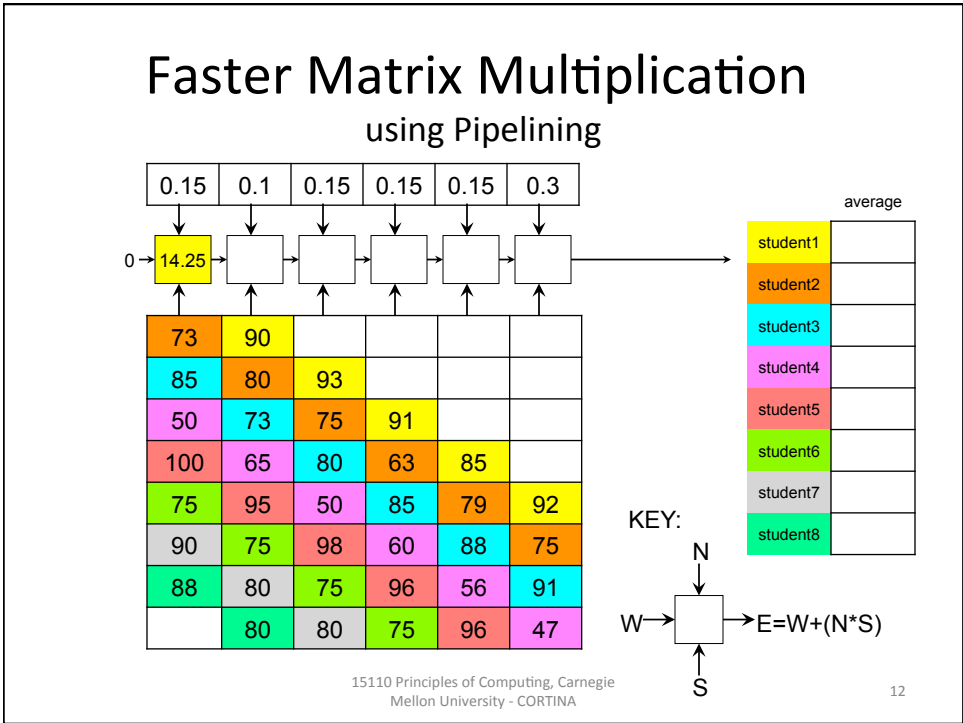
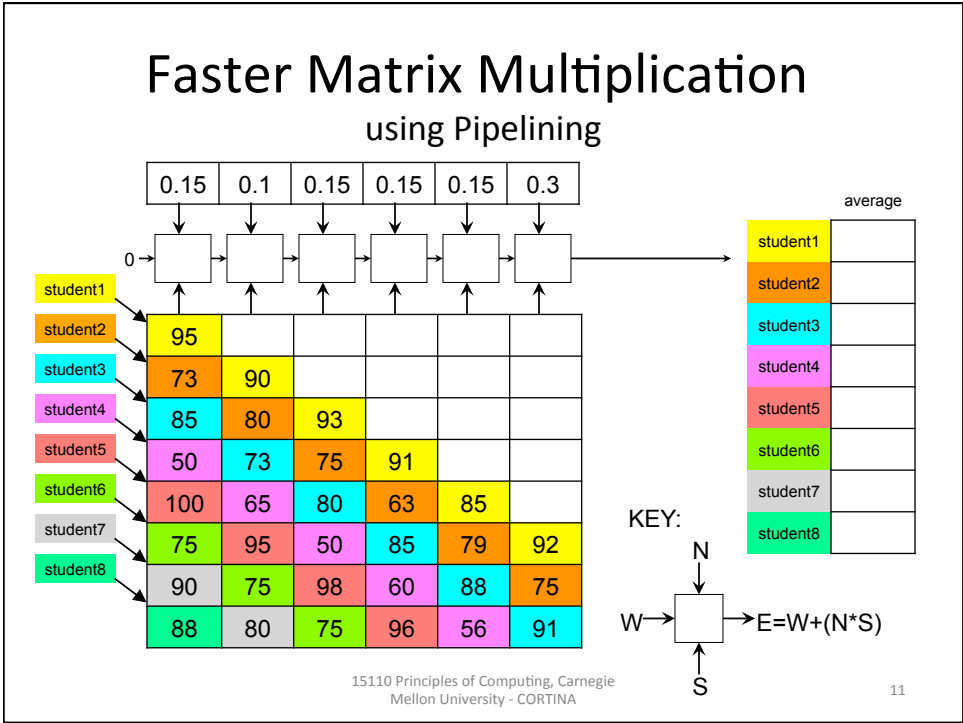
	hw	paper	exam1	exam2	exam3	final		weight		average
student1	95	90	93	91	85	92			student1	91.2
student2	73	80	75	63	79	75	hw	0.15	student2	74.0
student3	85	73	80	85	88	91	paper	0.1	student3	85.3
student4	50	65	50	60	56	47	exam1	0.15	student4	
student5	100	95	98	96	96	90	exam2	0.15	student5	
student6	75	75	75	75	75	75	exam3	0.15	student6	
student7	90	80	80	90	100	100	final	0.3	student7	
student8	88	80	80	70	60	55			student8	

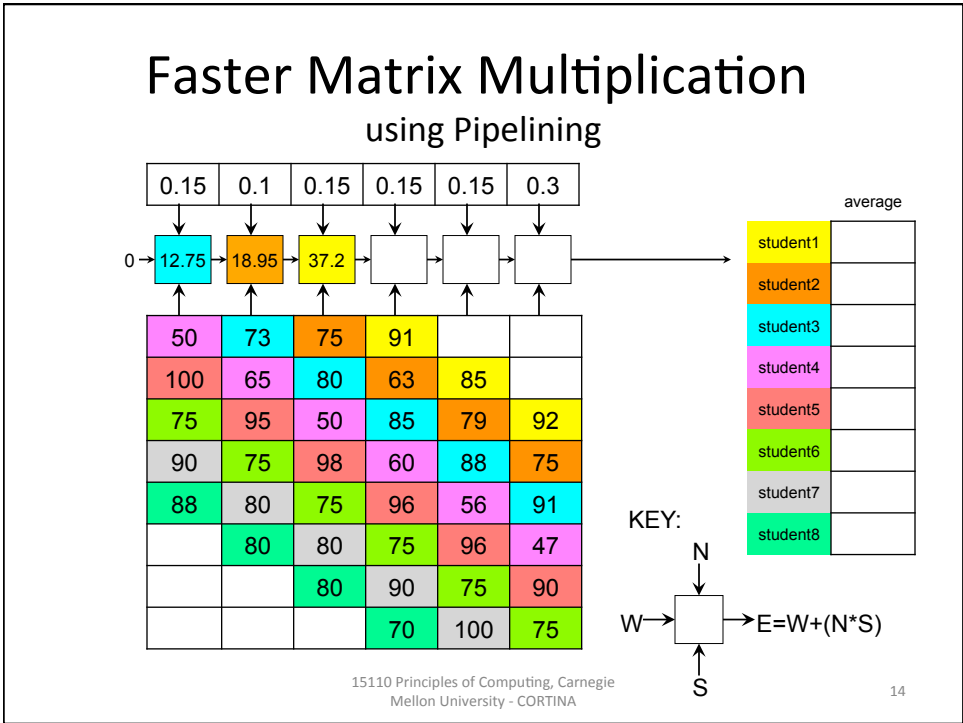
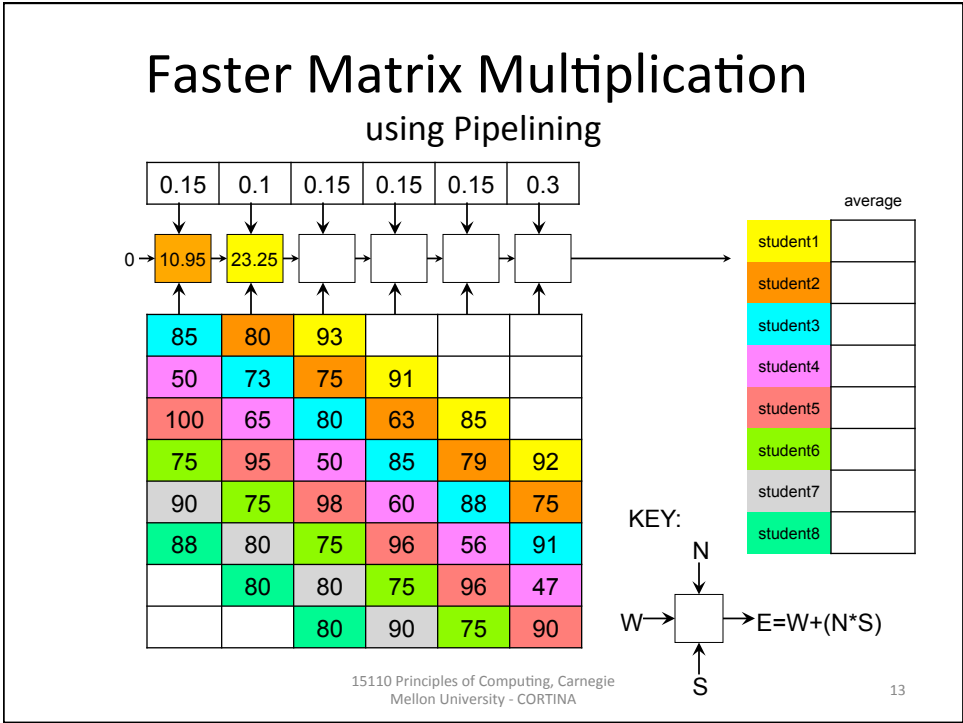
....and so on...

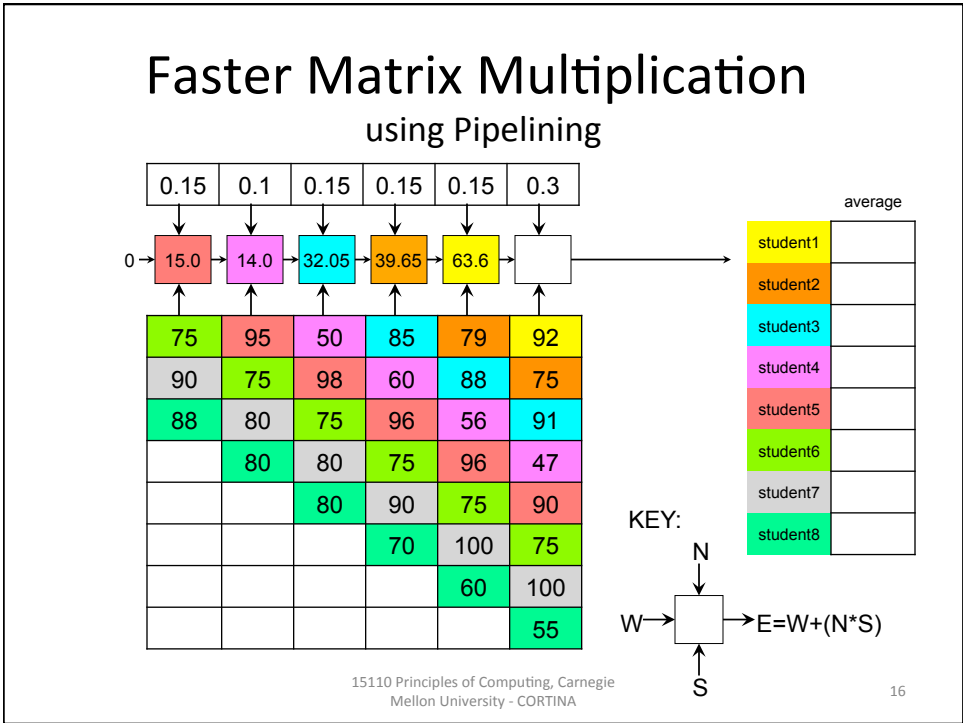
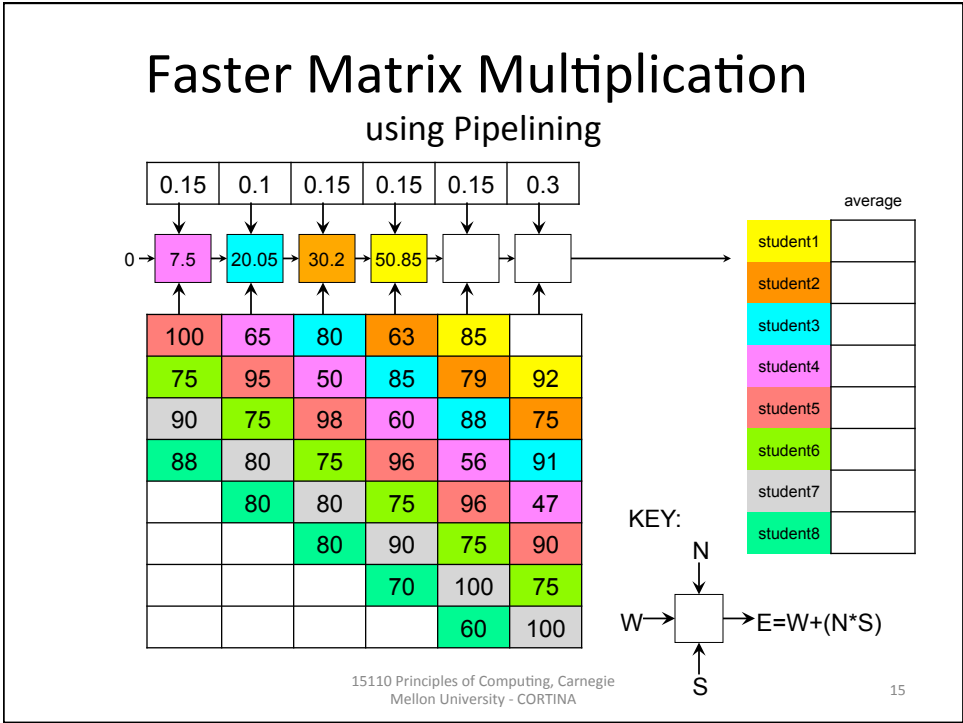
Matrix Multiplication

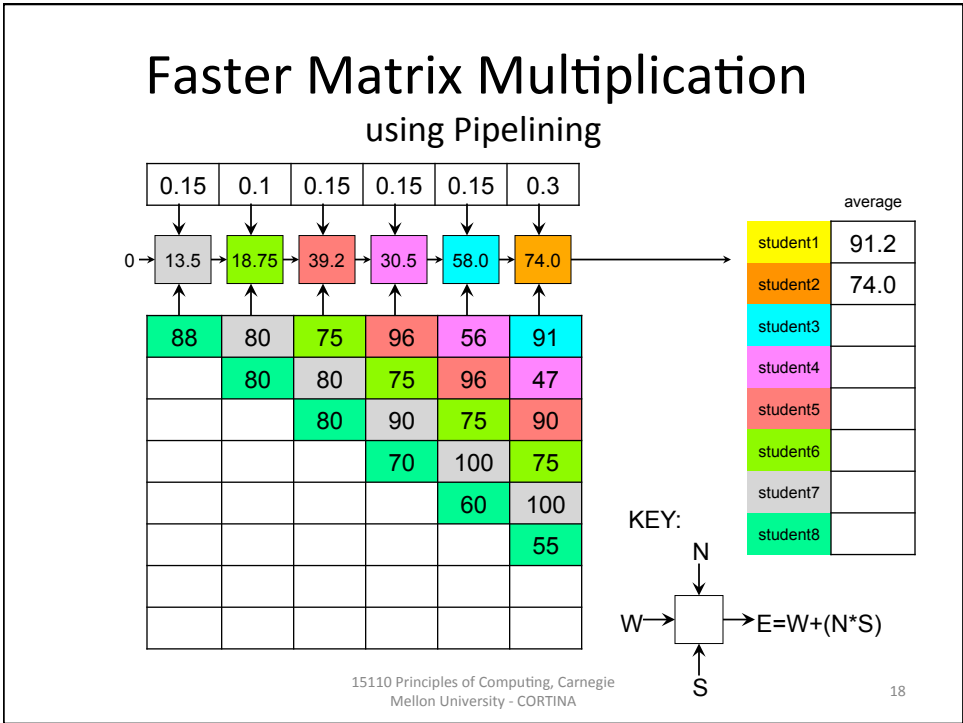
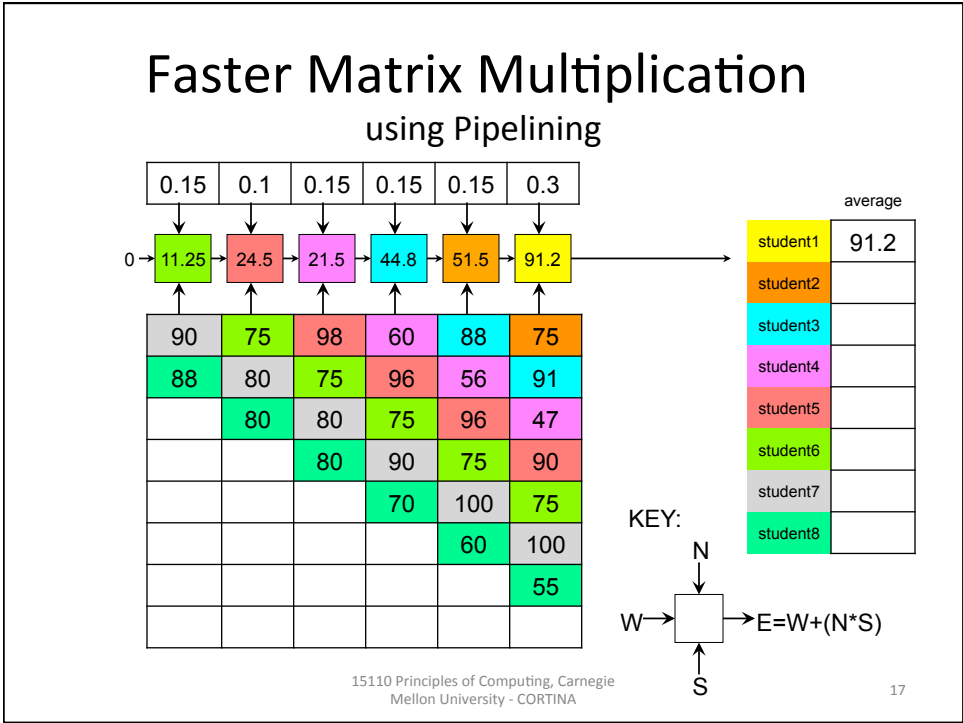
If each multiply/add takes 1 time unit,
this non-pipelined matrix multiplication takes 48 time units.

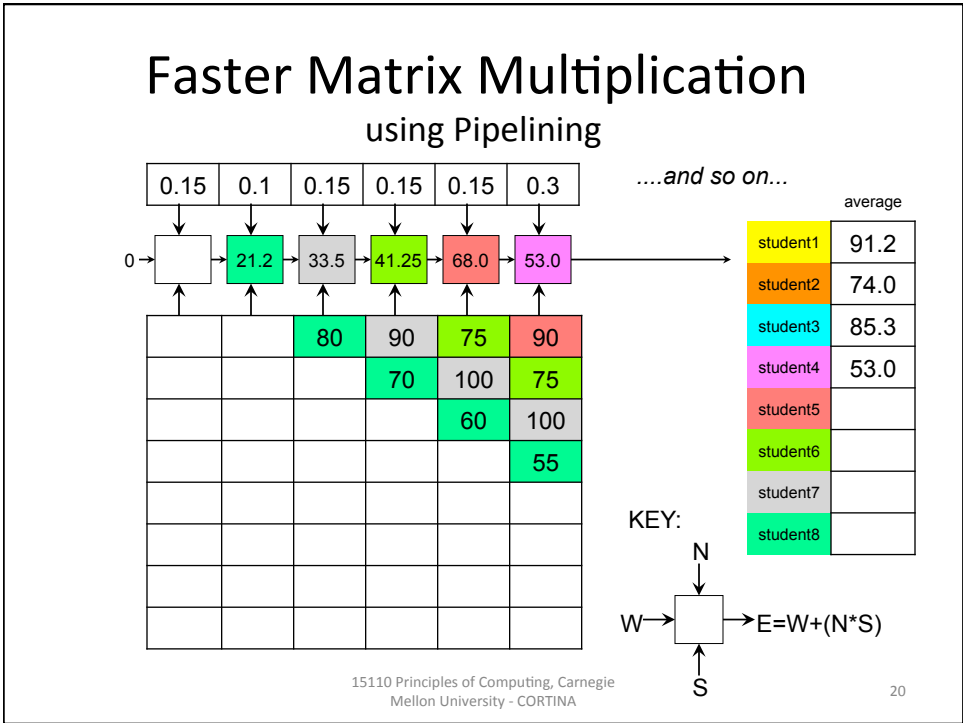
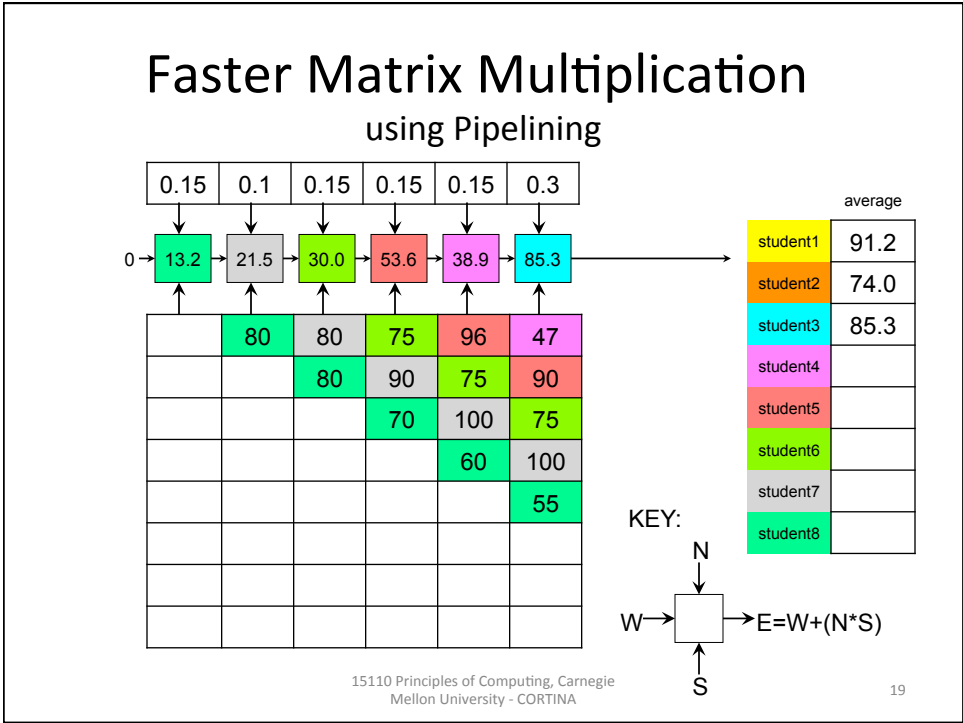
	hw	paper	exam1	exam2	exam3	final		weight		average
student1	95	90	93	91	85	92			student1	91.2
student2	73	80	75	63	79	75	hw	0.15	student2	74.0
student3	85	73	80	85	88	91	paper	0.1	student3	85.3
student4	50	65	50	60	56	47	exam1	0.15	student4	53.0
student5	100	95	98	96	96	90	exam2	0.15	student5	95.0
student6	75	75	75	75	75	75	exam3	0.15	student6	75.0
student7	90	80	80	90	100	100	final	0.3	student7	92.0
student8	88	80	80	70	60	55			student8	69.2



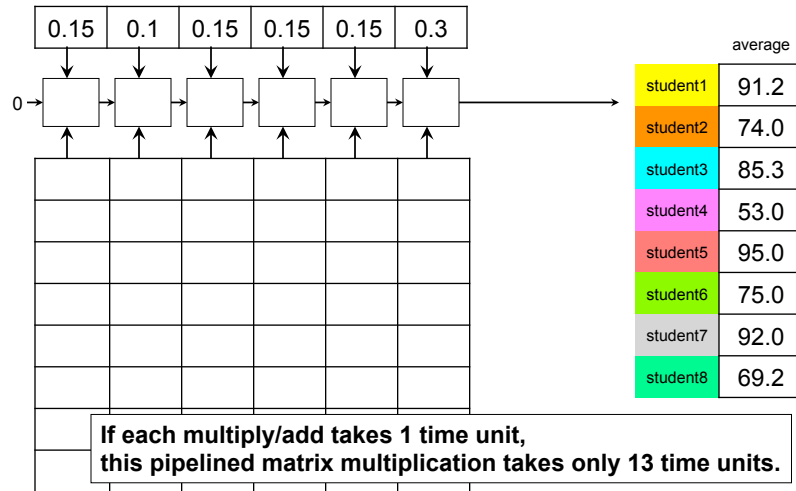








Faster Matrix Multiplication using Pipelining

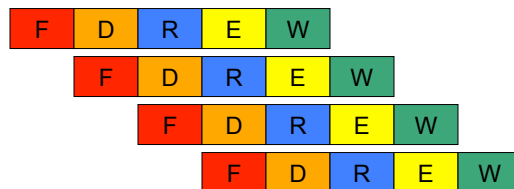


15110 Principles of Computing, Carnegie Mellon University - CORTINA

21

Pipelining in Computing

- Fetch instruction from memory
- Decode the instruction
- Read data from registers
- Execute the instruction
- Write the result into a register



15110 Principles of Computing, Carnegie Mellon University - CORTINA

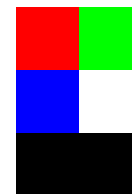
22

Problems?

- What if the result of the current instruction is needed as data for the next instruction?
 - reorder instructions so the next instruction comes later, when the current instruction's results are ready, if possible
- What if the next instruction depends on the results of a comparison (i.e. an if statement)?
 - set up two pipelines, one with each of the possible "next" instructions until we know which one to use

Bitmap Images

- screen consists of individual pixels
 - pixel = picture elements
- arranged into rows and columns
 - projector 1024x768
 - 720p = 1280x720
 - 1080p = 1920x1080
- Bitmap as a 3-D Python Lists

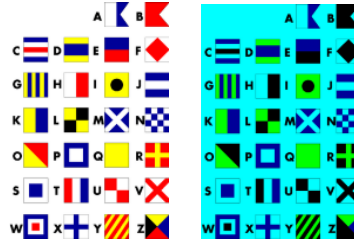


2 X 3 pixel image

```
bitmap = [ [ [255,0,0], [0,255,0] ],
           [ [0,0,255], [255,255,255] ],
           [ [0,0,0] , [0,0,0] ] ]
```

Example: Image Processing

```
def remove_red(image):
    num_rows = len(image)
    num_columns = len(image[0])
    for row in range(0,num_rows):
        for column in range(0,num_columns):
            image[row][column][0] = 0
    return None
```



15110 Principles of Computing, Carnegie Mellon University - CORTINA

25

Example: Image Processing

- What order are the pixels processed?
 - row by row, one pixel at a time
- Does this matter?
 - not really: all pixel computations are independent of one another
 - if we have multiple processors (cores), we could have each work on part of the image independently → faster results
 - Graphical Processing Units (GPU)

15110 Principles of Computing, Carnegie Mellon University - CORTINA

26

Example: Sorting Networks

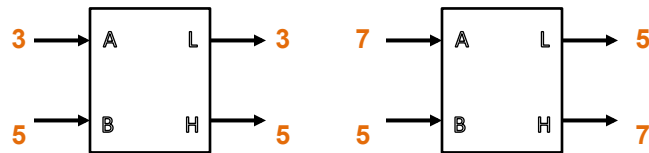
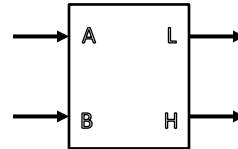
- Comparator

- Input: A and B

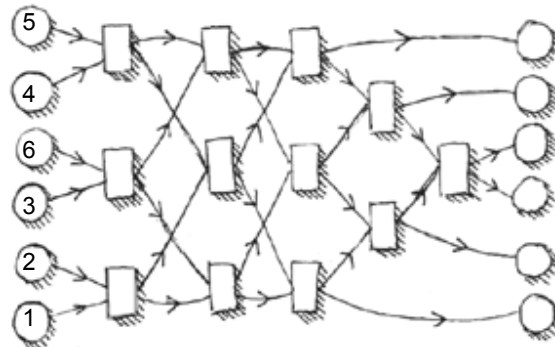
- Output:

- If $A \leq B$, then $L = A$ and $H = B$

- If $A > B$, then $L = B$ and $H = A$



Activity: Sorting Network Simulation



Input: [5, 4, 6, 3, 2, 1]

How many steps does this take . . . sequentially? concurrently?