

# ESE5320: System-on-a-Chip Architecture

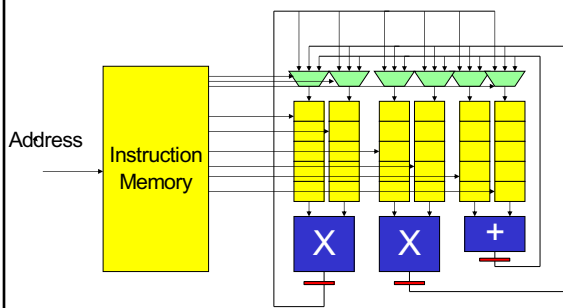
Day 24: November 20, 2024  
Software Pipelining



## Previously: VLIW [Day 24]

- Very Long Instruction Word
- Set of operators
  - Parameterize number, distribution (X, +, sqrt...)
  - More operators → less time, more area
  - Fewer operators → more time, less area
- Memories for intermediate state
  - Also parameterize memories and how connected
- Memory for “long” instructions
- General framework for specializing to problem
- General way to tradeoff area and time

## VLIW



## Today

- Software Pipelining
  - Part 1: Idea, Steady-state
  - Part 2: Prologue, Epilogue
  - Part 3: Acyclic
  - Part 4: Loop Dependencies

## Message

- Pack computations more tightly by scheduling multiple loop instances (loop bodies for multiple indices) together
  - Exploiting **pipelining** of computation
  - $II < \text{latency bound}$

## Problem [Day 24]

- Low utilization of parallel functional units for a single loop body

|   | LD   | LD   | LD   | ST      | * | * | * | +   | i  | i  | i  | sqrt |
|---|------|------|------|---------|---|---|---|-----|----|----|----|------|
| 0 |      |      |      |         |   |   |   | <   | &x | &y | &z |      |
| 1 | X[i] | Y[i] | Z[i] |         |   |   |   |     |    |    |    |      |
| 2 |      |      |      |         | x | y | z |     |    |    |    |      |
| 3 |      |      |      |         |   |   |   | X+y |    |    |    |      |
| 4 |      |      |      |         |   |   |   | +z  |    |    |    |      |
| 5 |      |      |      |         |   |   |   |     |    |    |    | sqrt |
| 6 |      |      |      | Res [i] |   |   |   |     | i  |    |    |      |

## Unroll 4 [Day 24]

|   | LD | LD | LD | ST | *  | *  | *  | +   | +   | i | i  | i  | sqr<br>t |
|---|----|----|----|----|----|----|----|-----|-----|---|----|----|----------|
| 0 |    |    |    |    |    |    |    |     |     | < | x0 | y0 | z0       |
| 1 | x0 | y0 | z0 |    |    |    |    |     |     |   | x1 | y1 | z1       |
| 2 | x1 | y1 | z1 |    | x0 | y0 | z0 |     |     |   | x2 | y2 | z2       |
| 3 | x2 | y2 | z2 |    | x1 | y1 | z1 | xy0 |     |   | x3 | y2 | z3       |
| 4 | x3 | y2 | z3 |    | x2 | y2 | z2 | xy1 | +z0 |   |    |    |          |
| 5 |    |    |    |    | x3 | y2 | z3 | xy2 | +z1 |   |    |    | 0        |
| 6 |    |    |    | 0  |    |    |    | xy3 | +z2 |   |    |    | 1        |
| 7 |    |    |    | 1  |    |    |    |     | +z3 |   |    |    | 2        |
| 8 |    |    |    | 2  |    |    |    |     |     |   |    |    | 3        |
| 9 |    |    |    | 3  |    |    |    |     |     |   | i  |    |          |

Penn ESE5320 Fall 2024 -- DeHon

7

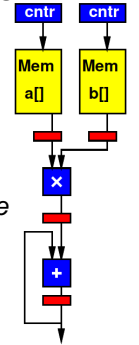
7

## Observation: Pipeline

for (i=0;i<MAX;i++)

c+=a[i]\*b[i];

- When we pipeline, we use all the resources
- ...but, we **don't** operate on a *single* loop body instance at a time
- We cannot hit II=1 for VLIW schedule of a single loop body because of path latency



Penn ESE5320 Fall 2024 -- DeHon

8

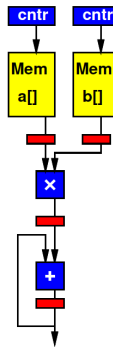
8

## Preclass 1

for (i=0;i<MAX;i++)

c+=a[i]\*b[i];

- Pipeline not operate on a single loop body instance at a time
- Assume add is working on  $a[0]*b[0]$  in same cycle,
  - What is  $a[i]*b[i]$ ?
  - What is lookup  $ld\ a[i],\ b[i]$ ?



Penn ESE5320 Fall 2024 -- DeHon

9

9

## Observation: Unroll

- Works like a pipeline
- Only works because overlap data among loop instances

|   | LD | LD | LD | ST | *  | *  | *  | +   | +   | i | i  | i  | sqr<br>t |
|---|----|----|----|----|----|----|----|-----|-----|---|----|----|----------|
| 0 |    |    |    |    |    |    |    |     |     | < | x0 | y0 | z0       |
| 1 | x0 | y0 | z0 |    |    |    |    |     |     |   | x1 | y1 | z1       |
| 2 | x1 | y1 | z1 |    | x0 | y0 | z0 |     |     |   | x2 | y2 | z2       |
| 3 | x2 | y2 | z2 |    | x1 | y1 | z1 | xy0 |     |   | x3 | y3 | z3       |
| 4 | x3 | y3 | z3 |    | x2 | y2 | z2 | xy1 | +z0 |   |    |    |          |
| 5 |    |    |    |    | x3 | y3 | z3 | xy2 | +z1 |   |    |    | 0        |
| 6 |    |    |    | 0  |    |    |    | xy3 | +z2 |   |    |    | 1        |
| 7 |    |    |    | 1  |    |    |    |     | +z3 |   |    |    | 2        |
| 8 |    |    |    | 2  |    |    |    |     |     |   |    |    | 3        |
| 9 |    |    |    | 3  |    |    |    |     |     |   | i  |    |          |

Penn ESE5320 Fall 2024 -- DeHon

10

10

## Observation: Unroll

- Works like a pipeline
- If keep going, fill like pipeline...

|   | LD | LD | LD | ST | *  | *  | *  | +   | +   | i | i  | i  | sqr<br>t |
|---|----|----|----|----|----|----|----|-----|-----|---|----|----|----------|
| 0 |    |    |    |    |    |    |    |     |     | < | x0 | y0 | z0       |
| 1 | x0 | y0 | z0 |    |    |    |    |     |     |   | x1 | y1 | z1       |
| 2 | x1 | y1 | z1 |    | x0 | y0 | z0 |     |     |   | x2 | y2 | z2       |
| 3 | x2 | y2 | z2 |    | x1 | y1 | z1 | xy0 |     |   | x3 | y3 | z3       |
| 4 | x3 | y3 | z3 |    | x2 | y2 | z2 | xy1 | +z0 |   | x4 | y4 | z4       |
| 5 | x4 | y4 | z4 |    | x3 | y3 | z3 | xy2 | +z1 |   | x5 | y5 | z5       |
| 6 | x5 | y5 | z5 | 0  | x4 | y4 | z4 | xy3 | +z2 |   | x6 | y6 | z6       |
| 7 | x6 | y6 | z6 | 1  | x5 | y5 | z5 | xy4 | +z3 |   | x7 | y7 | z7       |
| 8 | x7 | y7 | z7 | 2  | x6 | y6 | z6 |     | +z4 |   | x8 | y8 | z8       |
| 9 | x8 | y8 | z8 | 3  | x7 | y7 | z7 |     | +z5 |   | x9 | y9 | z9       |

Penn ESE5320 Fall 2024 -- DeHon

11

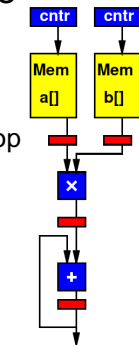
11

## Observation: Pipeline

for (i=0;i<MAX;i++)

c+=a[i]\*b[i];

- Pipeline not operate on a single loop body instance at a time
- II of pipeline is 1
- What is latency bound for  $c+=a[i]*b[i]$  ?



Penn ESE5320 Fall 2024 -- DeHon

12

12

## Observation

- To have  $II < \text{Latency Bound}$ 
  - (for a loop body)
- ...must spread one loop body calculation over multiple loop iterations

Penn ESE5320 Fall 2024 -- DeHon

13

13

## Idea: Software Pipelining

- Schedule VLIW operators across multiple loop iterations
- Treat execution on operators as pipeline

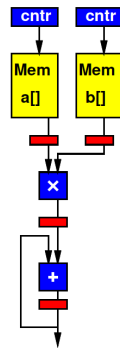
Penn ESE5320 Fall 2024 -- DeHon

14

14

## Pipeline

- Fully pipelined
  - Computation in a pipeline stage in a cycle, depends on output of a different stage on previous cycle
- What we compute in each cycle
  - is a set of pipeline stages
  - each operating on a different set of input data items



Penn ESE5320 Fall 2024 -- DeHon

15

15

## Pipeline Rewrite

```
for (i=0; i<MAX; i++)
```

```
  c+=a[i]*b[i];
```

```
for (i=0; i<MAX; i++) {
```

```
  aptr++; bptr++;
```

```
  la=*aptr; lb=*bptr;
```

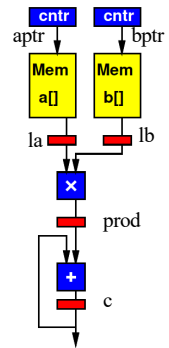
```
  prod=la*lb;
```

```
  c=c+prod;
```

```
}
```

Goal: each register compute based on value set previous cycle.

Rewrite body to match cycle of pipeline



Penn ESE5320 Fall 2024 -- DeHon

16

16

## Pipeline Rewrite

```
for (i=0; i<MAX; i++)
```

```
  c+=a[i]*b[i];
```

```
for (i=0; i<MAX; i++) {
```

```
  aptr++; bptr++;
```

```
  la=*aptr; lb=*bptr;
```

```
  prod=la*lb;
```

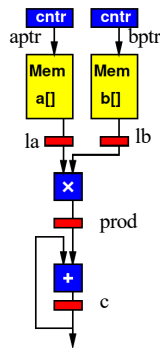
```
  c=c+prod;
```

```
}
```

```
{ c=c+prod;
  ??
}
```

Goal: each register compute based on value set previous cycle.

Rewrite body to match cycle of pipeline



Penn ESE5320 Fall 2024 -- DeHon

17

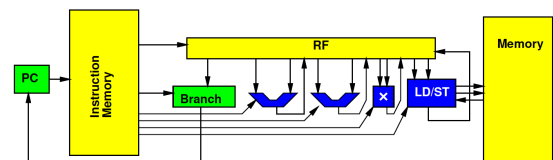
17

## Software Pipelined Version

```
for (i=0; i<MAX; i++)
```

```
  { c=c+prod; prod=la*lb; la=a[i]; lb=b[i]; }
```

- Use this to compact schedule



Penn ESE5320 Fall 2024 -- DeHon

18

18

## Schedule Software Pipelined (Preclass 3)

How many cycles for preclass 3 schedule?

| Cycle | Branch | ALU | ALU | Multiply | Ld/St |
|-------|--------|-----|-----|----------|-------|
| 0     |        |     |     |          |       |
| 1     |        |     |     |          |       |
| 2     |        |     |     |          |       |
| 3     |        |     |     |          |       |
| 4     |        |     |     |          |       |

Penn ESE5320 Fall 2024 -- DeHon

19

19

## Schedule Software Pipelined (Preclass 3)

| Cycle | Branch | ALU    | ALU    | Multiply | Ld/St |
|-------|--------|--------|--------|----------|-------|
| 0     | Bzneq  | Add r8 | Add r1 | Mul r6   | Ld r4 |
| 1     |        | Add r4 | Add r5 |          | Ld r5 |
| 2     | Br top | Sub r2 |        |          |       |

Penn ESE5320 Fall 2024 -- DeHon

20

20

## Prologue, Epilogue

Part 2

Penn ESE5320 Fall 2024 -- DeHon

21

21

## Prime Pipeline (as in ``Priming the Pump’’)

- For this body to work, will need to setup the steady-state condition for the pipeline

```
for (i=2;i<MAX;i++)
  { c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}
```

Penn ESE5320 Fall 2024 -- DeHon

22

22

## Prologue

- What need to do to define the loop variables used in the pipeline?
  - prod?
  - la, lb?
- Hint: what prod need to be entering i=2 loop body?

```
for (i=2;i<MAX;i++)
  { c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}
```

Penn ESE5320 Fall 2024 -- DeHon

23

23

## With Prologue

```
prod=a[0]*b[0];
la=a[1]; lb=b[1];
for (i=2;i<MAX;i++)
  { c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}
```

Penn ESE5320 Fall 2024 -- DeHon

24

24

## Flush Pipeline

- For this body to work, will need to finish the pipeline

```
for (i=2;i<MAX;i++)
  { c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}
```

- What need to do after loop?  
– Hint: what does c, prod, la, lb hold at loop exit?

Penn ESE5320 Fall 2024 -- DeHon

25

25

## With Epilogue

```
prod=a[0]*b[0];
la=a[1]; lb=b[1];
for (i=2;i<MAX;i++)
  { c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}
c=c+prod;
c=c+la*lb;
```

Penn ESE5320 Fall 2024 -- DeHon

26

26

## Full Software Pipelined Loop

```
prod=a[0]*b[0];
la=a[1]; lb=b[1];
for (i=2;i<MAX;i++)
  { c=c+prod; prod=la*lb; la=a[i]; lb=b[i];}
```

```
c=c+prod;
c=c+la*lb;
```

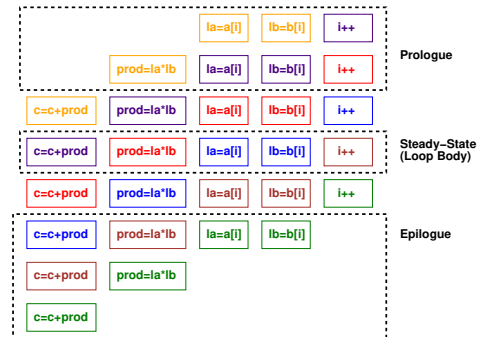
- Software pipelined loop requires a loop prologue and loop epilogue

Penn ESE5320 Fall 2024 -- DeHon

27

27

## Software Pipeline



Penn ESE5320 Fall 2024 -- DeHon

28

28

## Acyclic (Data Parallel)

Part 3

Penn ESE5320 Fall 2024 -- DeHon

29

29

## Preclass 4

### Revisit Day 24, Preclass 2

```
for (xptr=&x;xptr<XMAX;xptr++)
  res[i]=sqrt(x[i]*x[i]+y[i]*y[i]+z[i]*z[i]);
  <XMAX;xptr++; yptr++; zptr++; ld x; ldy; ldz;
  x[i]²; y[i]²; z[i]²; x[i]²+y[i]²; +z[i]²; sqrt; res[i]
```

- What resources would it take to achieve each ll by resource bound?

| ll | Ld | St | * | + | inc | sqrt |
|----|----|----|---|---|-----|------|
| 3  |    |    |   |   |     |      |
| 2  |    |    |   |   |     |      |
| 1  |    |    |   |   |     |      |

Penn ESE5320 Fall 2024 -- DeHon

30

30

## Revisit Day 26, Preclass 2

- Schedule for  $II=3$  (work back from res store)
- $\langle XMAX; xptr++; yptr++; zptr++; ld\ x; ldy; ldz; x[i]^2; y[i]^2; z[i]^2; x[i]^2+y[i]^2; +z[i]^2; sqrt; res[i]$

| Cycle | Br | ALU | Mpy | RP | WP    | Incr | Sqrt |
|-------|----|-----|-----|----|-------|------|------|
| 0     |    |     |     |    |       |      |      |
| 1     |    |     |     |    |       |      |      |
| 2     |    |     |     |    | res 0 |      |      |

Penn ESE5320 Fall 2024 -- DeHon

31

31

## $II=3$

| Cycle | Br | ALU   | Mpy | RP     | WP   | Incr   | Sqrt  |
|-------|----|-------|-----|--------|------|--------|-------|
| 0     | <  | +z0   | x1  | x[i] 2 |      | xptr 3 |       |
| 1     |    |       | y1  | y[i] 2 |      | yptr 3 | sqrt0 |
| 2     |    | x1+y1 | z1  | z[i] 2 | res0 | zptr 3 |       |

| Operations      | Instance |
|-----------------|----------|
| Incr            | +3       |
| ld              | +2       |
| $x^2, y^2, z^2$ | +1       |
| $x^2+y^2$       | +1       |
| $+z^2$          | +0       |
| sqrt            | +0       |
| st              | +0       |

Penn ESE5320 Fall 2024 -- DeHon

32

32

## $II=2$

| Cyc | Br | ALU  | Mpy | Mpy | RP | RP | WP   | i  | i  | sqrt  |
|-----|----|------|-----|-----|----|----|------|----|----|-------|
| 0   | <  | x+y1 | x2  | y2  | x3 | y3 |      | x4 | y4 | sqrt0 |
| 1   |    | +z1  | z2  |     | z3 |    | res0 | z4 |    |       |

| Operations | Instance |
|------------|----------|
| Incr       | +4       |
| ld         | +3       |
| $x^2$      | +2       |
| $x^2+y^2$  | +1       |
| $+z^2$     | +1       |
| sqrt       | +0       |
| st         | +0       |

Penn ESE5320 Fall 2024 -- DeHon

33

33

## $II=1$

| Br | A  | A  | M  | M  | M  | RP | RP | RP | WP  | i  | i  | i  | sqrt |
|----|----|----|----|----|----|----|----|----|-----|----|----|----|------|
| <  | +y | +z | x4 | y4 | z4 | x5 | y5 | z5 | Res | x6 | y6 | z6 | xyz1 |
|    | 3  | 2  |    |    |    |    |    |    | 0   |    |    |    |      |

| Operations | Instance |
|------------|----------|
| Incr       | +6       |
| ld         | +5       |
| $x^2$      | +4       |
| $x^2+y^2$  | +3       |
| $+z^2$     | +2       |
| sqrt       | +1       |
| st         | +0       |

Penn ESE5320 Fall 2024 -- DeHon

34

34

## Software Pipelining

- Observe
  - For cases without loop dependencies,
  - if willing to mix any number of loop instances,  $\rightarrow$  can achieve resource bound
- May require more registers to hold state

Penn ESE5320 Fall 2024 -- DeHon

35

35

## More Registers Example

- May require more registers to hold state
  - Implement:  $Y=a*(b+c*(d+e))$
  - Sequence:  $r1=d+e; r1=c*r1; r1=b+r1; r1=a*r1;$
  - soft pipe:  $r1=a*r2; r2=b+r3; r3=c*r4; r4=d+e$

Penn ESE5320 Fall 2024 -- DeHon

36

36

## Compare to Day 24 (no software pipelining)

- [for preclass 4]
- One of each operator: II=8
  - Software pipelined 3
- Latency lower bound (roughly II=2 hardware here)
  - II=7, software pipelined 2
- One operator each
  - II=2.5 for unroll 4 iterations (10 cycles)

Penn ESE5320 Fall 2024 -- DeHon

Software pipelined 1

37

37

## Cyclic: Loop Dependencies

Part 4

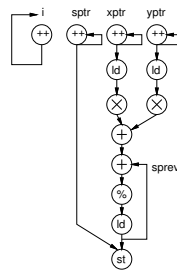
Penn ESE5320 Fall 2024 -- DeHon

38

38

## Example w/ Loop Dependency

- for (i=0; i<MAX; i++)  
s[i]=t[(s[i-1]+x[i]\*x[i]+y[i]\*y[i])%p];
- Assume +, \*, ld, st, % single cycle
- latency bound for loop body?
- cycle bound for loop?



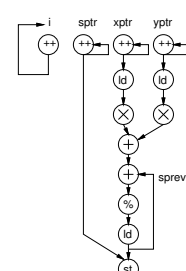
Penn ESE5320 Fall 2024 -- DeHon

39

39

## Computational Graph

```
for (i=0; i<MAX; i++)
    s[i]=t[(s[i-1]+x[i]*x[i]+y[i]*y[i])%p];
```



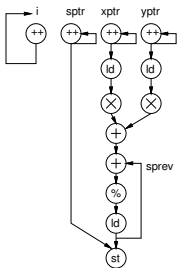
Penn ESE5320 Fall 2024 -- DeHon

40

40

## Resources?

- Resources to support cycle bound (set resource bound as low as cycle bound):
  - Adders (for increment and add)?
  - Load units?

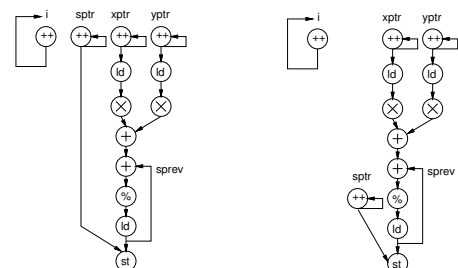


Penn ESE5320 Fall 2024 -- DeHon

41

41

## Computational Graph



Penn ESE5320 Fall 2024 -- DeHon

42

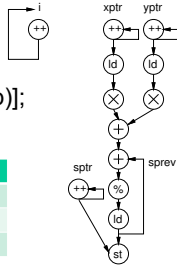
42

## Computational Graph

- **Schedule**

- for (i=0;i<MAX;i++)  
s[i]=t[(s[i-1]+x[i]\*x[i]+y[i]\*y[i])%p];

|   | add | add | ld | st  | mul | mod |
|---|-----|-----|----|-----|-----|-----|
| 0 |     |     |    | S-1 |     |     |
| 1 |     |     |    |     |     |     |
| 2 |     |     |    |     |     |     |



Penn ESE5320 Fall 2024 -- DeHon

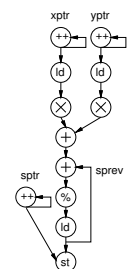
43

43

## Computational Graph

Note: loop completed in one iteration of code  
(one pipeline stage)

|   | add     | add                               | ld   | st    | mul               | mod |
|---|---------|-----------------------------------|------|-------|-------------------|-----|
| 0 | Xptr +2 | sprev0                            | Y +1 | S[-1] | X <sup>2</sup> +1 |     |
| 1 | Yptr+2  | sptr 0                            | X +2 |       | Y <sup>2</sup> +1 | 0   |
| 2 | I +2    | X <sup>2</sup> -Y <sup>2</sup> +1 | t[0] |       |                   |     |



Penn ESE5320 Fall 2024 -- DeHon

44

44

## Loops with Dependencies

- Loops with dependencies
  - Limited by cycle bound
  - Cycles=max(I<sub>cycle\_bound</sub>,RB)
    - Probably not this ideal in all cases, more like max(scheduled-loop,RB)
- (another example: Fall 2018 final, Question 2)

Penn ESE5320 Fall 2024 -- DeHon

45

45

## Lessons

- VLIW provides rich area-time tradeoffs
- Pipelining not just for hardware
  - Already seen for coarse operation pipelining, even with processors
    - Process- or thread-level parallelism
  - Saw for Compute/Communicate overlap
    - Vitis clEnqueueMigrateObjects; dataflow
  - Now see for ILP (instruction-level parallelism)
  - Necessary to achieve II < Latency Bound

Penn ESE5320 Fall 2024 -- DeHon

46

46

## Automation

- Good compiler should do this for you
- Worth understanding to reason about II should achieve
- If compiler not achieving, hint may need to check if there's a dependency the compiler thinks exists
- Gives you an idea of how to disambiguate for the compiler

Penn ESE5320 Fall 2024 -- DeHon

47

47

## Big Ideas:

- Pipelining of data processing useful for software scheduled on processors
  - VLIW (and pipelined, superscalar)
  - Not just hardware pipelines

Penn ESE5320 Fall 2024 -- DeHon

48

48



## Admin

- Feedback
- P4 due Friday
- Next week
  - Real-Time lecture on Monday
  - RF schedule on TW, so no Wed. lecture
  - Thanksgiving