

# ESE532: System-on-a-Chip Architecture

Day 4: September 12, 2018  
Parallelism Overview



Penn ESE532 Fall 2018 -- DeHon

## Today

- Types of Parallelism
  - How can we slice up and think about parallelism?
- Compute Models
  - How do we express and reason about parallel execution freedom

Penn ESE532 Fall 2018 -- DeHon

2

## Message

- Many useful models for parallelism
  - Help conceptualize
- One-size does not fill all
  - Match to problem

Penn ESE532 Fall 2018 -- DeHon

3

## Types of Parallelism

Penn ESE532 Fall 2018 -- DeHon

4

## Types of Parallelism

- **Data Level** – Perform same computation on different data items
- **Thread or Task Level** – Perform separable (perhaps heterogeneous) tasks independently
- **Instruction Level** – Within a single sequential thread, perform multiple operations on each cycle.

Penn ESE532 Fall 2018 -- DeHon

5

## Pipeline Parallelism

- Pipeline – organize computation as a spatial sequence of concurrent operations
  - Can introduce new inputs before finishing
  - Instruction- or thread-level
  - Use for data-level parallelism
  - Can be directed graph

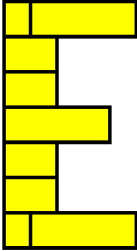
Penn ESE532 Fall 2018 -- DeHon

6

**Build 1**

## Sequential

- Single person build E
- Latency?
- Throughput?



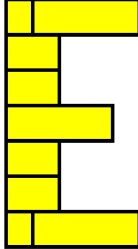
7

Penn ESE532 Fall 2018 -- DeHon

**Build 2**

## Data Parallel

- Everyone in class build own E
- Latency?
- Throughput?
- Ideal speedup?
- Resource Bound?
  - 100 Es, 12 people
- When useful?



8

Penn ESE532 Fall 2018 -- DeHon

## Data-Level Parallelism

- **Data Level** – Perform same computation on different data items
- Ideal:  $T_{dp} = T_{seq}/P$
- (independent problems, match our resource bound computation)

9

Penn ESE532 Fall 2018 -- DeHon

**Build 3**

## Thread Parallel

- Each person build indicated letter
- Latency?
- Throughput?
- Speedup over sequential build of 6 letters?

10

Penn ESE532 Fall 2018 -- DeHon

## Thread-Level Parallelism

- **Thread or Task Level** – Perform separable (perhaps heterogeneous) tasks independently
- Ideal:  $T_{tp} = T_{seq}/P$
- $T_{tp} = \max(T_{t1}, T_{t2}, T_{t4}, \dots)$ 
  - Less speedup than ideal if not balanced
- Can produce a diversity of calculations
  - Useful if have limited need for the **same** calculation

11

Penn ESE532 Fall 2018 -- DeHon

**Build 4**

## Instruction-Level Parallelism

- Build single letter in lock step
- Groups of 3
- Resource Bound for 3 people building 9-brick letter?
- Announce steps from slide
  - Stay in step with slides

12

Penn ESE532 Fall 2018 -- DeHon

## Group Communication

- Groups of 3
- Note who was person 1 task
- 2, 3 will need to pass completed substructures

| Step | Person 1 | Person 2 | Person 3 |
|------|----------|----------|----------|
| 0    |          |          |          |
| 1    |          |          |          |
| 2    |          |          |          |
| 3    |          |          |          |

Penn ESE532 Fall 2018 -- DeHon

13

Step 0

Penn ESE532 Fall 2018 -- DeHon

14

Step 1

Penn ESE532 Fall 2018 -- DeHon

15

Step 2

Penn ESE532 Fall 2018 -- DeHon

16

Step 3

Penn ESE532 Fall 2018 -- DeHon

17

## Instruction-Level Parallelism

- Latency?
- Throughput?
- Can reduce latency for single letter
- Ideal:  $T_{\text{latency}} = T_{\text{seqlatency}}/P$ 
  - ...but critical path bound applies, dependencies may limit

Penn ESE532 Fall 2018 -- DeHon

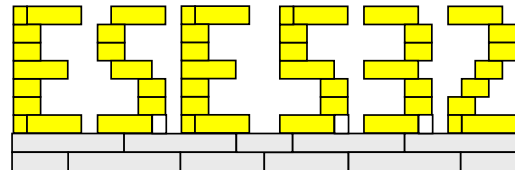
18

## Instruction-Level Pipeline

- Each person adds one brick to build
- Run pipeline once alone
- Latency?
- Then run pipeline with 5 inputs
- Throughput?

## Thread Graph

- How would we build with task level parallelism?
  - Tasks?
  - Dependencies?



## Types of Parallelism

- **Data Level** – Perform same computation on different data items
- **Thread or Task Level** – Perform separable (perhaps heterogeneous) tasks independently
- **Instruction Level** – Within a single sequential thread, perform multiple operations on each cycle.

## Parallel Compute Models

## Sequential Control Flow

### Control flow

- Program is a sequence of operations
- Operation reads inputs and writes outputs into common store (memory)
- One operation runs at a time
  - defines successor

Model of correctness is sequential execution

### Examples

C (Java, ...)  
Finite-State Machine (FSM) / Finite Automata (FA)

## Parallelism can be explicit

- ILP Build example
- Multiply, add for quadratic equation

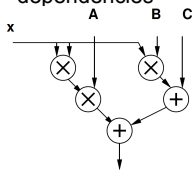
- Coordinate data parallel operations

| cycle | mpy              | add                     |
|-------|------------------|-------------------------|
| 1     | B,x              |                         |
| 2     | x,x              | (Bx)+C                  |
| 3     | A,x <sup>2</sup> |                         |
| 4     |                  | Ax <sup>2</sup> +(Bx+C) |

- Coordinate ILP

## Parallelism can be implicit

- Sequential expression
- Infer data dependencies



$T1=x*x$   
 $T2=A*T1$   
 $T3=B*x$   
 $T4=T2+T3$   
 $Y=C+T4$

• Or  
 $Y=A*x*x+B*x+C$

## Implicit Parallelism

- $d=(x1-x2)*(x1-x2) + (y1-y2)*(y1-y2)$
- What parallelism exists here?

## Parallelism can be implicit

- Sequential expression
- Infer data dependencies

for (i=0;i<100;i++)  
 $y[i]=A*x[i]*x[i]+B*x[i]+C$

Why can these operations be performed in parallel?

## Term: Operation

- Operation** – logic computation to be performed

## Dataflow / Control Flow

### Dataflow

- Program is a graph of operations
- Operation consumes **tokens** and produces tokens
- All operations run concurrently

### Control flow (e.g. C)

- Program is a sequence of operations
- Operation reads inputs and writes outputs into common store
- One operation runs at a time
  - defines successor

## Token

- Data value with presence indication
  - May be conceptual
    - Only exist in high-level model
    - Not kept around at runtime
  - Or may be physically represented
    - One bit represents presence/absence of data

## Token Examples?

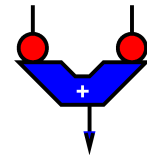
- What are familiar cases where data may come with presence tokens?
  - Network packets
  - Memory references from processor
    - Variable latency depending on cache presence
  - Start bit on serial communication

Penn ESE532 Fall 2018 -- DeHon

31

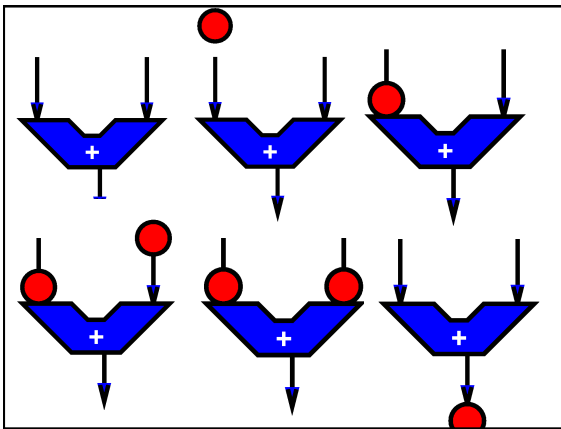
## Operation

- Takes in one or more inputs
- Computes on the inputs
- Produces results
- Logically **self-timed**
  - “Fires” only when input set present
  - Signals availability of output



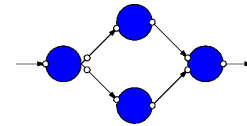
Penn ESE532 Fall 2018 -- DeHon

32



## Dataflow Graph

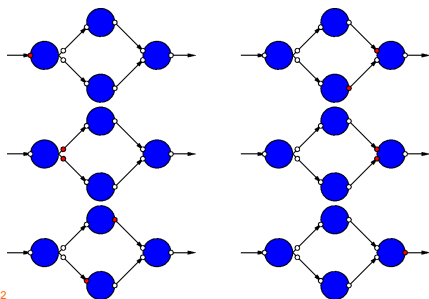
- Represents
  - computation sub-blocks
  - linkage
- Abstractly
  - controlled by data presence



Penn ESE532 Fall 2018 -- DeHon

34

## Dataflow Graph Example



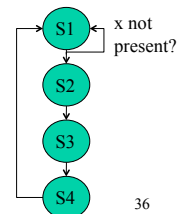
Penn ESE532

35

## Sequential / FSM

- FSM is degenerate dataflow graph where there is exactly one token

| cycle | mpy              | add                     | next               |
|-------|------------------|-------------------------|--------------------|
| S1    | B,x              |                         | x-->S2,<br>else S1 |
| S2    | x,x              | (Bx)+C                  | S3                 |
| S3    | A,x <sup>2</sup> |                         | S4                 |
| S4    |                  | Ax <sup>2</sup> +(Bx+C) | S1                 |



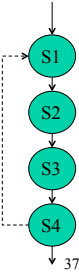
Penn ESE532 Fall 2018 -- DeHon

36

## Sequential / FSM

- FSM is degenerate dataflow graph where there is exactly one token

| cycle | mpy              | add                     | next           |
|-------|------------------|-------------------------|----------------|
| S1    | B,x              |                         | x->S2, else S1 |
| S2    | x,x              | (Bx)+C                  | S3             |
| S3    | A,x <sup>2</sup> |                         | S4             |
| S4    |                  | Ax <sup>2</sup> +(Bx+C) | S1             |



Penn ESE532 Fall 2018 -- DeHon

37

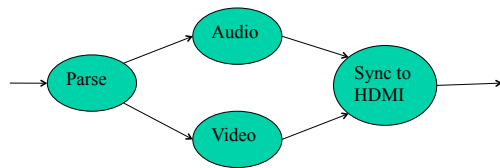
## Communicating Threads

- Computation is a collection of sequential/control-flow "threads"
- Threads may communicate
  - Through dataflow I/O
  - (Through shared variables)
- View as hybrid or generalization
- CSP – Communicating Sequential Processes → canonical model example

Penn ESE532 Fall 2018 -- DeHon

38

## Video Decode

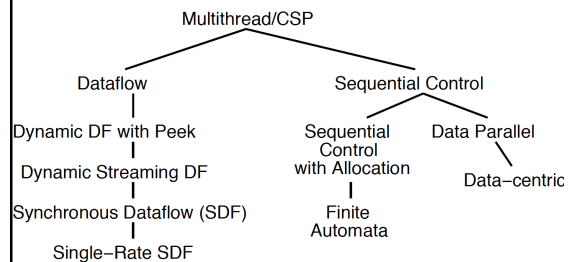


- Why might need to synchronize to send to HDMI?

Penn ESE532 Fall 2018 -- DeHon

39

## Compute Models



Penn ESE532 Fall 2018 -- DeHon

40

## Value of Multiple Models

- When you have a big enough hammer, everything looks like a nail.
- Many stuck on single model
  - Try to make all problems look like their nail
- Value to diversity / heterogeneity
  - One size does not fit all



Penn ESE532 Fall 2018 -- DeHon

41

## Big Ideas

- Many parallel compute models
  - Sequential, Dataflow, CSP
- Find natural parallelism in problem
- Mix-and-match

Penn ESE532 Fall 2018 -- DeHon

42

## Admin

- Reading Day 5 on web
- HW2 due Friday
- HW3 out
  
- Return Legos ☺
- Recitation in here at noon
  - Will take questions after class in hall