

ESE532: System-on-a-Chip Architecture

Day 4: September 15, 2021
Parallelism Overview

From Syllabus:

Preclass

Feedback

Google doc(s) for Builds

(sign up for number now)



Penn ESE532 Fall 2021 -- DeHon

Today

- Compute Models (Part 1)
 - How do we *express* and reason about parallel execution freedom
- Types of Parallelism (Part 2)
 - How can we slice up and think about parallelism?
 - How *exploit* parallelism

Penn ESE532 Fall 2021 -- DeHon

2

Message

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

Penn ESE532 Fall 2021 -- DeHon

3

Parallel Compute Models

Control Flow, Dataflow
Combining
Explicit, Implicit Parallelism

Penn ESE532 Fall 2021 -- DeHon

4

Term: Operation

- **Operation** – logic computation to be performed

Penn ESE532 Fall 2021 -- DeHon

5

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)

Penn ESE532 Fall 2021 -- DeHon

6

Parallelism can be explicit

- State which operations occur on a cycle
- Multiply, add for quadratic equation

cycle	mpy	add
1	B,x	
2	x,x	(Bx)+C
3	A,x ²	
4		Ax ² +(Bx+C)

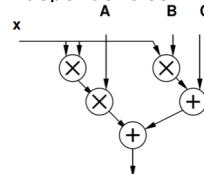
Penn ESE532 Fall 2021 -- DeHon

7

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

Penn ESE532 Fall 2021 -- DeHon

8

Implicit Parallelism

- $d=(x_1-x_2)*(x_1-x_2) + (y_1-y_2)*(y_1-y_2)$
- What parallelism exists here?

Penn ESE532 Fall 2021 -- DeHon

9

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?

Penn ESE532 Fall 2021 -- DeHon

10

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

Penn ESE532 Fall 2021 -- DeHon

11


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

Penn ESE532 Fall 2021 -- DeHon

12

FIFO



- Hardware Block
- Outputs data in order received
 - First-In, First-Out
- Tell it when you are providing data
 - Write
 - May choose not to insert on a cycle
 - Need to signal
- Tell it when you are consuming data
 - Read
- Tells you when it's **empty** and has no data to provide
- Tells you when it's **full** and can hold nothing else

What are data presence indicators here?

Penn ESE532 Fall 2021 -- DeHon 13

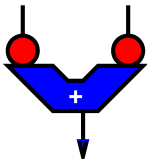
Token Examples?

- How serial link know character present?
- How signal miss in processor data cache and processor needs to wait for data?

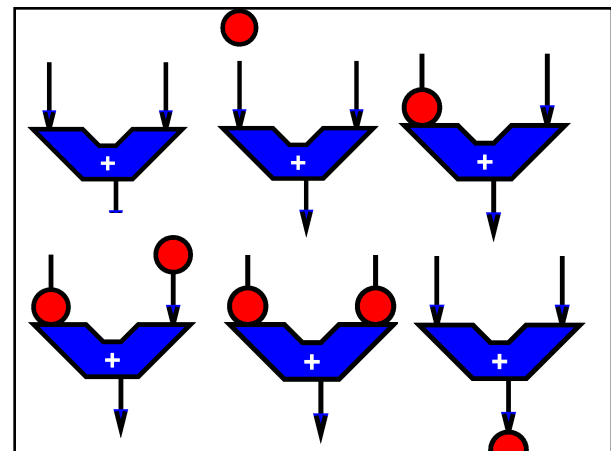
Penn ESE532 Fall 2021 -- DeHon 14

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

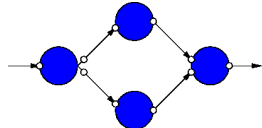


Penn ESE532 Fall 2021 -- DeHon 15

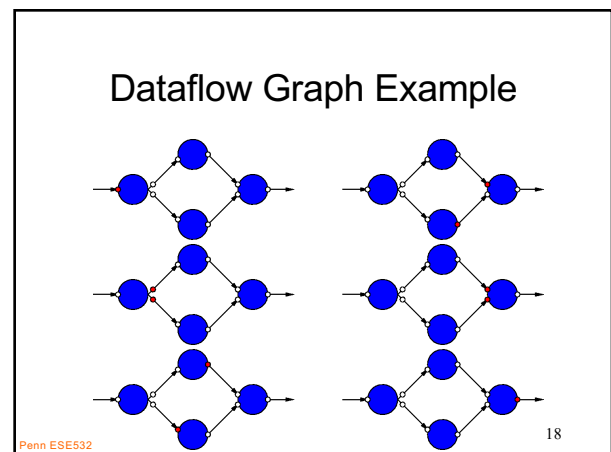


Dataflow Graph

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



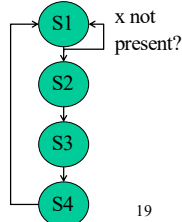
Penn ESE532 Fall 2021 -- DeHon 17



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 ²		S4
S4		Ax ² +(Bx+C)	S1



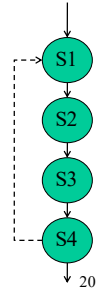
Penn ESE532 Fall 2021 -- DeHon

19

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 ²		S4
S4		Ax ² +(Bx+C)	S1



Penn ESE532 Fall 2021 -- DeHon

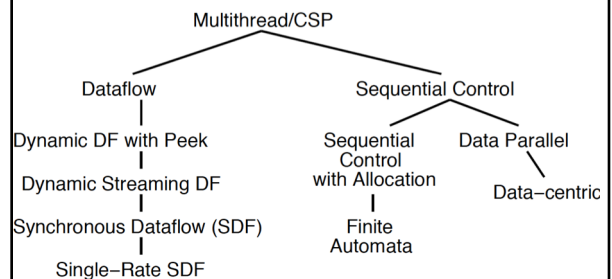
20

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
 - Of control flow and dataflow
- CSP – Communicating Sequential Processes → canonical model example₂₁

Penn ESE532 Fall 2021 -- DeHon

Compute Models



Penn ESE532 Fall 2021 -- DeHon

23

All Used

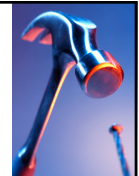
- All of these things get used in modern CPUs and SoCs
 - Sequential control flow
 - Operation parallelism
 - Data presence and data-driven flow
 - Multiple threads
 - Data Parallel

Penn ESE532 Fall 2021 -- DeHon

24

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 2021 -- DeHon

25

Types of Parallelism

Part 2

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.

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

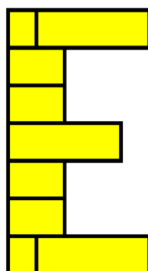
SWAP TO GOOGLE DOC

[Jump to Google Doc return](#)

Build 1

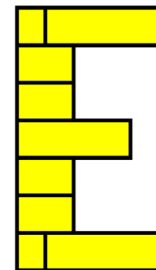
Sequential

- Single person build E
 - Page for each person number
- Latency?
- Throughput?



Data Parallel

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



Data-Level Parallelism

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

Penn ESE532 Fall 2021 -- DeHon

32

Thread Parallel

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

Penn ESE532 Fall 2021 -- DeHon

33

Thread-Level Parallelism

- **Thread or Task Level** – Perform separable (perhaps heterogeneous) tasks independently
- Resource Bound: $T_{tp} = T_{seq}/P$
- $T_{tp} = \max(T_{t1}, T_{t2}, T_{t3}, \dots)$
 - Less speedup than ideal if not balanced
- Can produce a diversity of calculations
 - Useful if have limited need for the **same** calculation

Penn ESE532 Fall 2021 -- DeHon

34

Instruction-Level Parallelism

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

Penn ESE532 Fall 2021 -- DeHon

35

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 2021 -- DeHon

36

Step 0

Penn ESE532 Fall 2021 -- DeHon

37

Step 1

Penn ESE532 Fall 2021 -- DeHon

38

Step 2

Penn ESE532 Fall 2021 -- DeHon

39

Step 3

Penn ESE532 Fall 2021 -- DeHon

40

Instruction-Level Parallelism (ILP)

- Latency?
- Throughput?
- Can reduce **latency** for single letter
- Resource Bound: $T_{\text{latency}} = T_{\text{seq latency}}/P$
 - Remember **critical path bound** applies; dependencies may limit

Penn ESE532 Fall 2021 -- DeHon

41

Instruction-Level Pipeline

Build 4

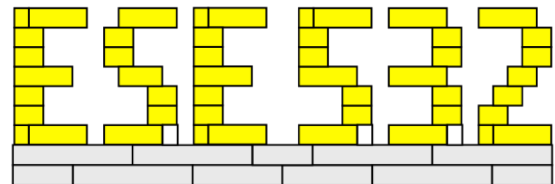
- Each person adds one brick to build
- Resources? (people in pipeline?)
- Run pipeline once alone
- Latency? (brick-adds to build letter)
- Then run pipeline with 5 inputs
- Throughput? (letters/brick-add-time)

Penn ESE532 Fall 2021 -- DeHon

42

Thread Graph

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



Penn ESE532 Fall 2021 -- DeHon

43

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 2021 -- DeHon

44

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 2021 -- DeHon

45

Big Ideas

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

Penn ESE532 Fall 2021 -- DeHon

46

Admin

- Reading Day 5 on web
- HW2 due Friday
- HW3 out

Penn ESE532 Fall 2021 -- DeHon

47