ESE532: System-on-a-Chip Architecture

Day 4: September 11, 2019

Parallelism Overview

Pickup:
1 Preclass
1 Lego instructions
1 feecback
1 bag of legos

Penn

Today

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

Message

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

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)

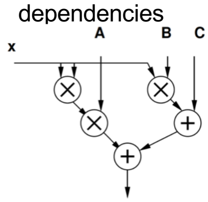
Parallelism can be explicit

- 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)

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

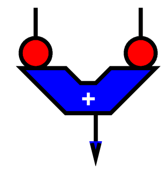
Token

- 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

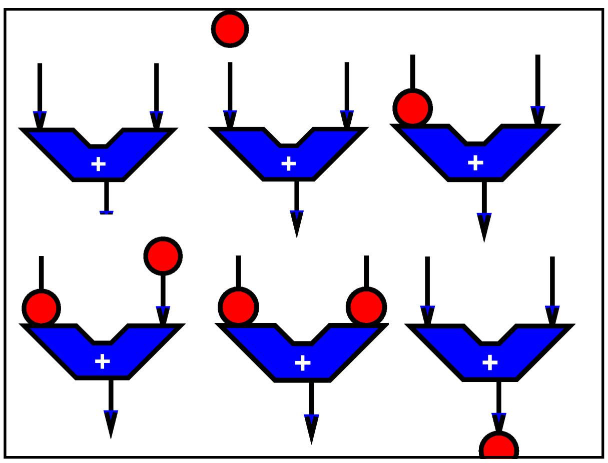
Token Examples?

- How ethernet know when a packet shows up?
  - Versus when no packets are arriving?
- How serial link know character present?
- How signal miss in processor data cache and processor needs to wait for data?

Operation

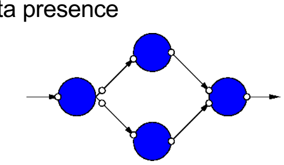


- Takes in one or more inputs
- Computes on the inputs
- Produces results
- Logically **self-timed**
  - "Fires" only when input set present
  - Signals availability of output

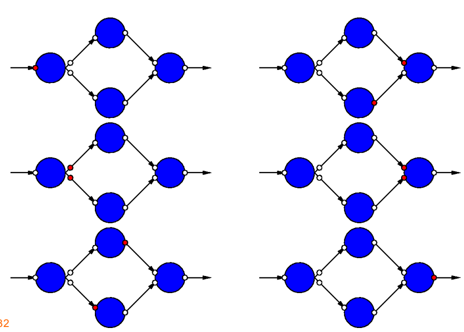


Dataflow Graph

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



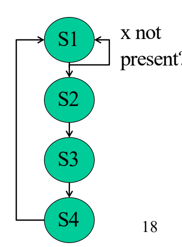
Dataflow Graph Example



Sequential / FSM

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

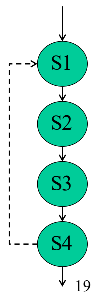
cycle	mpy	add	next
S1	B,x		x-->S2, else S1
S2	x,x	(Bx)+C	S3
S3	A,x^2		S4
S4		$Ax^2+(Bx+C)$	S1



Sequential / FSM

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

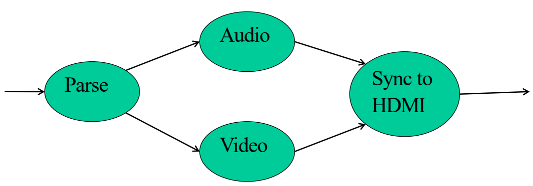
cycle	mpy	add	next
S1	B,x		x-->S2, else S1
S2	x,x	(Bx)+C	S3
S3	A,x^2		S4
S4		$Ax^2+(Bx+C)$	S1



Communicating Threads

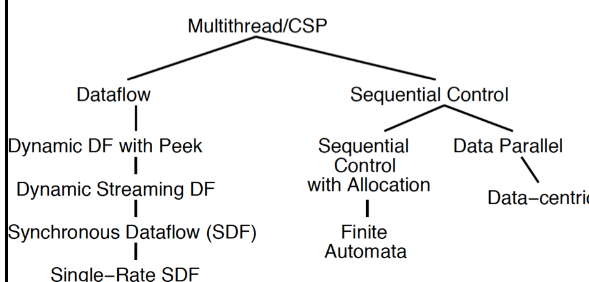
- Computation is a collection of sequential/control-flow "threads"
- Threads may communicate
  - Through dataflow I/O
  - (Through shared variables)
- View as hybrid or generalization
- CSP – Communicating Sequential Processes → canonical model example

Video Decode

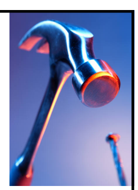


- Why might need to synchronize to send to HDMI?

Compute Models



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

Types of Parallelism

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.

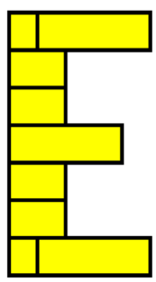
Pipeline Parallelism

- 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

Build 1

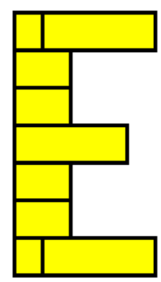
Sequential

- Single person build E
- 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
- Ideal: $T_{dp} = T_{seq}/P$
- (with enough independent problems, match our resource bound computation)

Build 2

Thread Parallel

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

Thread-Level Parallelism

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

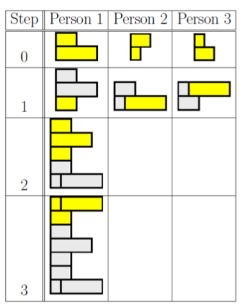
Build 3

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

Group Communication

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



Step 0

Step 1

Step 2

Step 3

Instruction-Level Parallelism (ILP)

- Latency?
- Throughput?

- Can reduce **latency** for single letter
- Ideal: $T_{latency} = T_{seqlatency}/P$
  - …but **critical path bound** applies, dependencies may limit

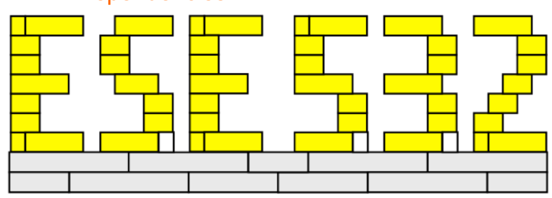
Build 4

Bonus (time permit): Instruction-Level Pipeline

- 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)

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.

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

Big Ideas

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

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