ESE532: System-on-a-Chip Architecture

Day 5: September 21, 2020

Dataflow Process Model

Today

Dataflow Process Model

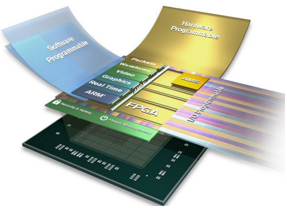
- Terms (part 1)
- Issues
- Abstraction
- Performance Prospects (part 2)
- Basic Approach
- As time permits (part 3)
  - Dataflow variants
  - Motivations/demands for variants

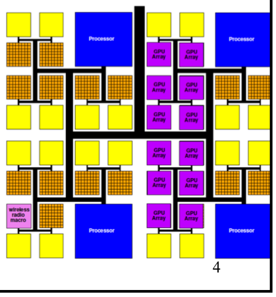
Message

- Parallelism can be natural
- Expression can be agnostic to substrate
  - Abstract out implementation details
  - Tolerate variable delays may arise in implementation
- Divide-and-conquer
  - Start with coarse-grain streaming dataflow
- Basis for performance optimization and parallelism exploitation

Programmable SoC

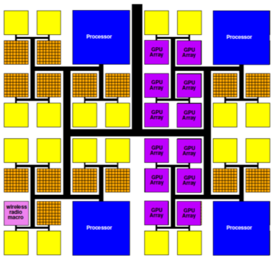
- Implementation Platform for innovation
  - This is what you target (avoid NRE)
  - Implementation vehicle





Reminder

- Goal: exploit parallelism on heterogeneous PSoC to achieve desired performance (energy)



Term: Process

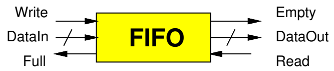
- Abstraction of a processor
- Looks like each process is running on a separate processor
- Has own state, including
  - Program Counter (PC)
  - Memory
  - Input/output
- May not actually run on processor
  - Could be specialized hardware block
  - May share a processor

Thread

- Has a separate locus of control (PC)
- May share memory (contrast process)
  - Run in common address space with other threads
- May not actually run on processor
  - Could be specialized hardware block
  - May share a processor

Day 4

FIFO



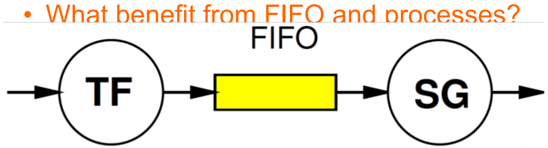
- Hardware Block
- Outputs data in order received
  - First-In, First-Out
- Tell it when you are providing data
  - Write
  - May choose not to insert on a cycle
    - Need to signal
- Tell it when you are consuming data
  - Read
- Tells you when it's **empty** and has no data to provide
- Tells you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

Process

- Processes (threads) allow *expression* of independent control
- Convenient for things that advance independently
- Process (thread) is the easiest way to express some behaviors
  - Easier than trying to describe as a single process
- Can be used for performance optimization to improve resource utilization

Preclass 2

- Average time for TF, SG?
  - 1 cycle 99% of time, 100 cycles 1% of time
- Throughput no FIFO?
  - Hint: what must wait on TF miss? SG miss?
- Throughput with FIFO?
  - How is FIFO changing?
- What benefit from FIFO and processes?



Preclass 2

- Independent probability of miss
  - P_f, P_g
- Concretely
  - 1 cycle in map
  - 100 run function and put in map
- If each runs independently (in isolation)
  - T~= 1*(1-P)+P*100
- If run together in lock step
  - Either can stall: $P=P_f+P_g-P_fP_g$
  - T~= 1*(1-P)+(P)*100

Model (from Day 4) 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

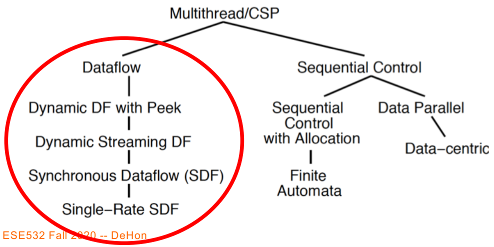
Issues

- **Communication** – how move data between processes?
  - What *latency* does this add?
  - *Throughput* achievable?
- **Synchronization** – how define how processes advance relative to each other?
- **Determinism** – for the same inputs, do we get the same outputs?

Today's Stand

- Communication – FIFO-like channels
- Synchronization – dataflow with FIFOs
- Determinism – how to achieve
  - …until you must give it up.

Dataflow Process Model



Multithread/CSP

Dataflow

Dynamic DF with Peek

Dynamic Streaming DF

Synchronous Dataflow (SDF)

Single–Rate SDF

Sequential Control

Sequential Control with Allocation

Finite Automata

Data Parallel

Data–centric

Operation/Operator

- **Operation** – logical computation to be performed
  - A *process* that communicates through dataflow inputs and outputs
- **Operator** – physical block that performs an Operation
  - E.g. processor, hardware block

Day 4

Dataflow / Control Flow

Dataflow

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

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

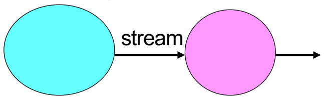
Day 4

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

Stream

- Logical abstraction of a persistent point-to-point communication link between operations (processes)
  - Has a (single) source and sink
  - Carries data presence / flow control
  - Provides in-order (FIFO) delivery of data from source to sink (producer to consumer)

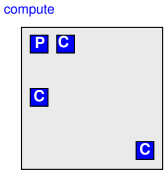


Streams

- Captures communications structure
  - Explicit producer→consumer link up
- Abstract communications
  - Physical resources or implementation
  - Delay from source to sink
- Contrast
  - C: producer->consumer implicit through memory
  - Verilog/VHDL: cycles visible in implementation
  - (can add **on top of** either C or Verilog)

Variable Delay Source to Sink

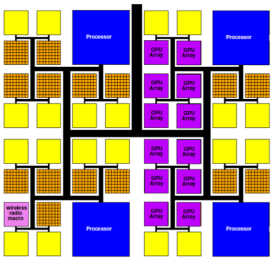
- How would placement of source and sink operator impact delay?



- How could sharing of interconnect between source and sink impact delay?

Communication Latency

- Once map to multiple processors
- Need to move data between processors
- That costs time

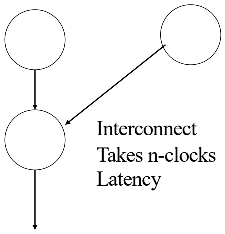


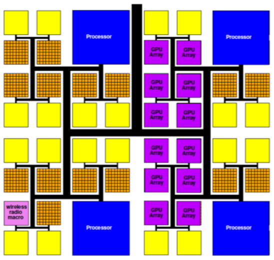
Day 3

On-Chip Delay

- Delay is proportional to distance travelled
- Make a wire twice the length
  - Takes twice the latency to traverse
  - (can pipeline)
- Modern chips
  - Run at 100s of MHz to GHz
  - Take 10s of ns to cross the chip

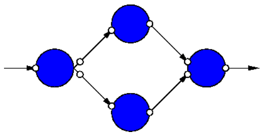
Dataflow gives Clock Independent Semantics





Dataflow Process Network

- Collection of Operations
- Connected by Streams
- Communicating with Data Tokens
- (CSP restricted to stream communication)



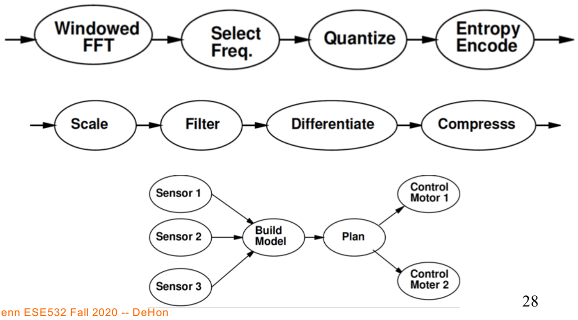
Dataflow Abstracts Timing

- Doesn’t say
  - on which cycle calculation occurs
- Does say
  - What order operations occur in
  - How data interacts
    - i.e. which inputs get mixed together
- Permits
  - Scheduling on different # and types of resources
  - Operators with variable delay
  - Variable delay in interconnect

Dataflow Graphs
Parallel Performance Prospect

Part 2

Some Task Graphs



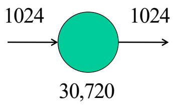
Synchronous Dataflow (SDF) with fixed operators

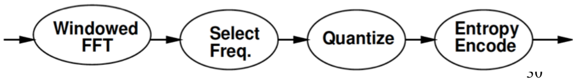
- Particular, restricted form of dataflow
- Each operation
  - Consumes a fixed number of input tokens
  - Produces a fixed number of output tokens
  - Operator performs fixed number of operations (in fixed time)
  - When full set of inputs are available
    - Can produce output
  - Can fire any (all) operations with inputs available at any point in time

SDF Operator

FFT

- 1024 inputs
- 1024 outputs
- 10,240 multiplies
- 20,480 adds
- (or 30,720 primitive operations)





Processor Model

- Simple (for today's lecture)
  - Assume one primitive operation per cycle
- Could embelish
  - Different time per operation type
    - E.g. adds: 1 cycle, multiply: 3 cycles
  - Multiple memories with different timings

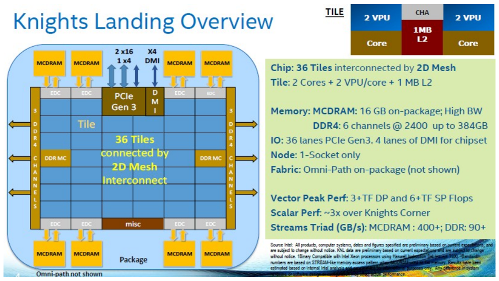
Time for Graph Iteration on Processors

- Single processor $T_{one} = \sum_i Nops_i$
- One processor per Operation (process)
  - ❑ $T_{each} = \max(Nop_1, Nop_2, Nop_3, \ldots)$
- General

$$T_{map} = max\left(\sum_i c(1,i)\times Nops_i, \sum_i c(2,i)\times Nops_i, \sum_i c(3,i)\times Nops_i, \ldots\right)$$

c(x,y) – 1 if Processor x runs task y

Intel Knights Landing



Chip: 36 Tiles interconnected by **2D Mesh**
Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: 16 GB on-package; High BW
DDR4: 6 channels @ 2400 up to 384GB
IO: 36 lanes PCIe Gen3. 4 lanes of DMI for chipset
Node: 1-Socket only
Fabric: Omni-Path on-package (not shown)

Vector Peak Perf: 3+TF DP and 6+TF SP Flops
Scalar Perf: ~3x over Knights Corner
Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+

Source Intel: All products, computer systems, dates and figures specified are preliminary based on current expectations, and are subject to change without notice. KNL data are preliminary based on current expectations and are subject to change without notice. 1Binary Compatible with Intel Xeon processors using Haswell Instruction Set (except TSX). 2Bandwidth numbers are based on STREAM-like memory access pattern when MCDRAM used as flat memory. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

https://www.nextplatform.com/2016/06/20/intel-knights-landing-yields-big-bang-buck-jump/

[Intel, Micro 2016]

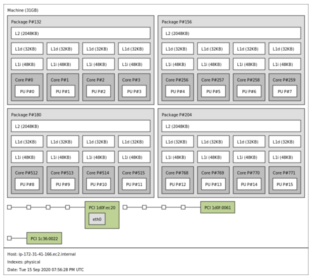
GRVI/Phallanx

- Puts 1680 RISC-V32b Integer cores
- On XCVU9P FPGA
- http://fpga.org/2017/01/12/grvi-phalanx-joins-the-kilocore-club/

Fig 6: A 400 GRVI Phalanx. 10x5 clusters of 8 PEs (KU040)

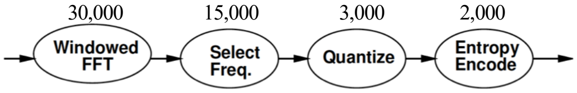
[Gray, FCCM 2016]

AWS a1-metal



Host: ip-172-31-41-166.ec2.internal
Indexes: physical
Date: Tue 15 Sep 2020 07:56:28 PM UTC

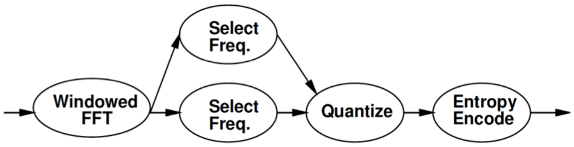
Map to different processors



- Map to (preclass 1)
  - One processor performance?
  - One process per processor performance?
  - Two processors
    - How?
    - Performance?
  - Bottleneck?

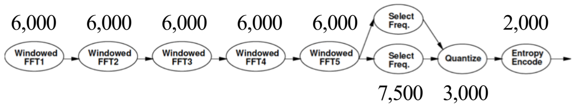
Refine Data Parallel

- If component is data parallel, can split out parallel tasks



Refine Pipeline

- If operation internally pipelineable, break out pipeline into separate tasks

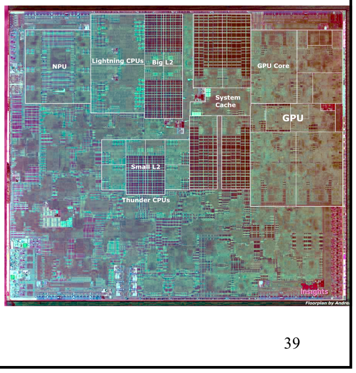


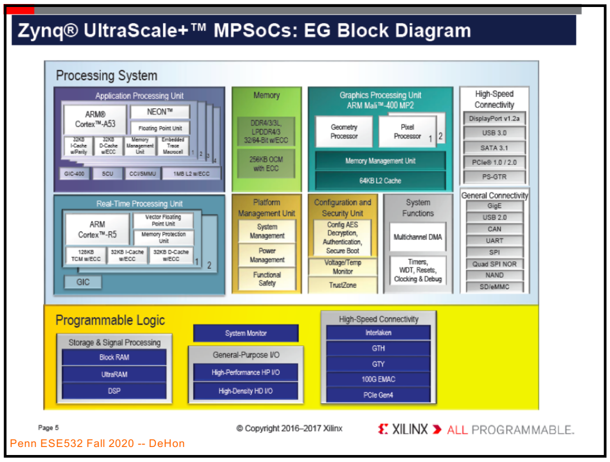
Performance with one processor per operation?

Achieve same performance with how many processors?

Apple A13 Bionic

- $98mm^2$, 7nm
- 8.5 Billion Tr.
- iPhone 11 +
- 6 ARM cores
  - 2 fast (2.6GHz)
  - 4 low energy
- 4 custom GPUs
- Neural Engine
  - 5 Trillion ops/s?





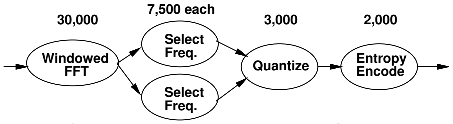
Zynq® UltraScale+™ MPSoCs: EG Block Diagram

Page 5

© Copyright 2016–2017 Xilinx

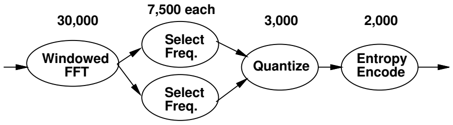
XILINX ALL PROGRAMMABLE.

Heterogeneous Processor

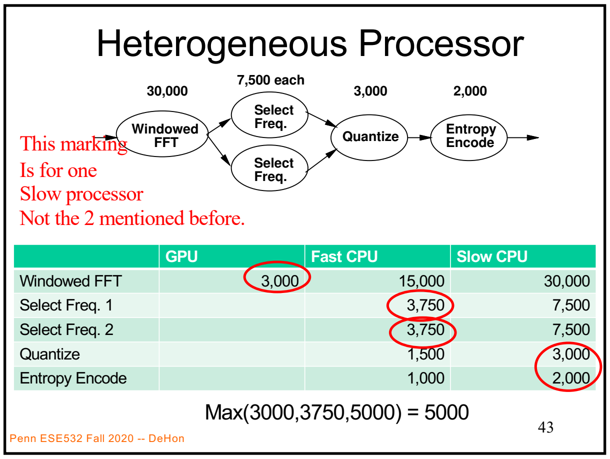


- GPU perform 10 primitive FFT Ops per cycle
- Fast CPU can perform 2 ops/cycle
- Slow CPU 1 op/cycle
- Map: FFT to GPU, Select to 2 Fast CPUs, quantize and Entropy each to own Slow CPU
- Cycles/graph iteration?

Heterogeneous Processor



	GPU	Fast CPU	Slow CPU
Windowed FFT	3,000	15,000	30,000
Select Freq. 1		3,750	7,500
Select Freq. 2		3,750	7,500
Quantize		1,500	3,000
Entropy Encode		1,000	2,000



Heterogeneous Processor

30,000 | 7,500 each | 3,000 | 2,000

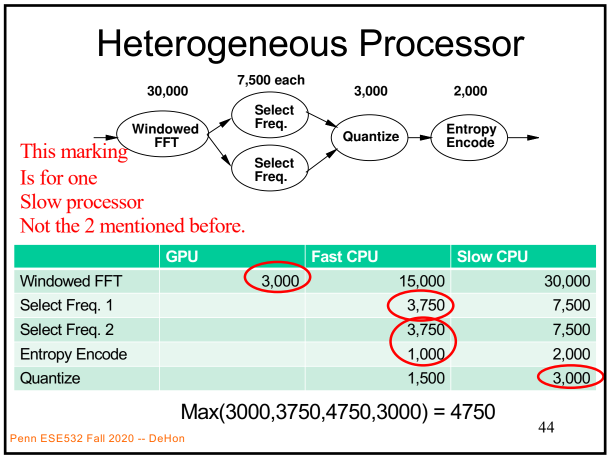
Windowed FFT → Select Freq. / Select Freq. → Quantize → Entropy Encode

This marking
Is for one
Slow processor
Not the 2 mentioned before.

	GPU	Fast CPU	Slow CPU
Windowed FFT	3,000	15,000	30,000
Select Freq. 1		3,750	7,500
Select Freq. 2		3,750	7,500
Quantize		1,500	3,000
Entropy Encode		1,000	2,000

Max(3000,3750,5000) = 5000

Penn ESE532 Fall 2020 -- DeHon 43



Heterogeneous Processor

30,000 | 7,500 each | 3,000 | 2,000

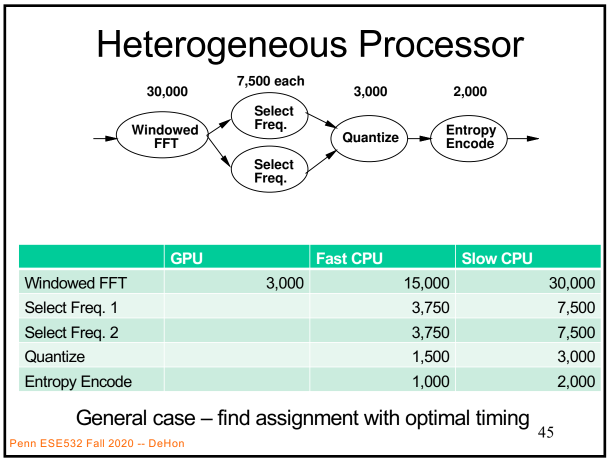
Windowed FFT → Select Freq. / Select Freq. → Quantize → Entropy Encode

This marking
Is for one
Slow processor
Not the 2 mentioned before.

	GPU	Fast CPU	Slow CPU
Windowed FFT	3,000	15,000	30,000
Select Freq. 1		3,750	7,500
Select Freq. 2		3,750	7,500
Entropy Encode		1,000	2,000
Quantize		1,500	3,000

Max(3000,3750,4750,3000) = 4750

Penn ESE532 Fall 2020 -- DeHon 44



Heterogeneous Processor

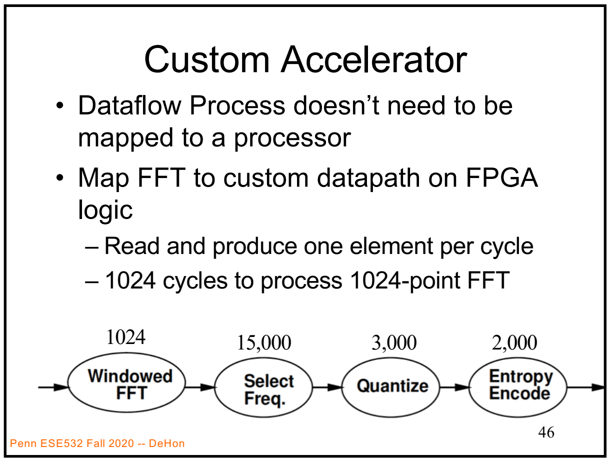
30,000 | 7,500 each | 3,000 | 2,000

Windowed FFT → Select Freq. / Select Freq. → Quantize → Entropy Encode

	GPU	Fast CPU	Slow CPU
Windowed FFT	3,000	15,000	30,000
Select Freq. 1		3,750	7,500
Select Freq. 2		3,750	7,500
Quantize		1,500	3,000
Entropy Encode		1,000	2,000

General case – find assignment with optimal timing

Penn ESE532 Fall 2020 -- DeHon 45



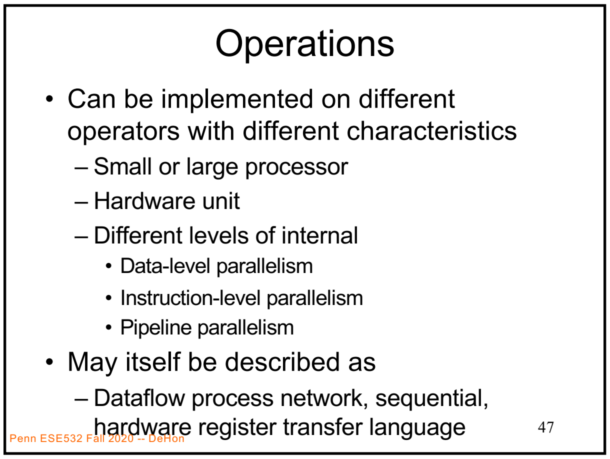
Custom Accelerator

- Dataflow Process doesn't need to be mapped to a processor
- Map FFT to custom datapath on FPGA logic
  - Read and produce one element per cycle
  - 1024 cycles to process 1024-point FFT

1024 | 15,000 | 3,000 | 2,000

Windowed FFT → Select Freq. → Quantize → Entropy Encode

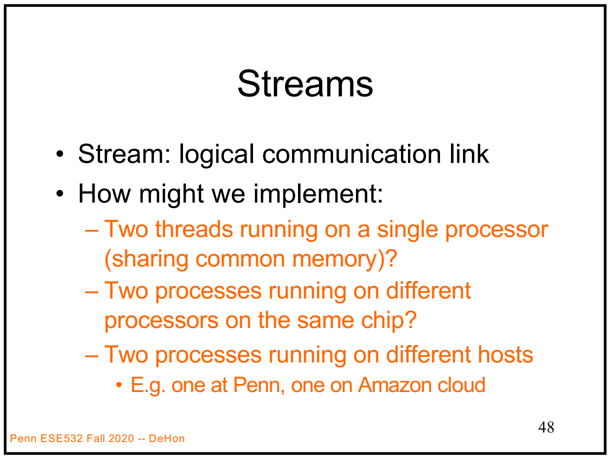
Penn ESE532 Fall 2020 -- DeHon 46



Operations

- Can be implemented on different operators with different characteristics
  - Small or large processor
  - Hardware unit
  - Different levels of internal
    - Data-level parallelism
    - Instruction-level parallelism
    - Pipeline parallelism
- May itself be described as
  - Dataflow process network, sequential, hardware register transfer language

Penn ESE532 Fall 2020 -- DeHon 47



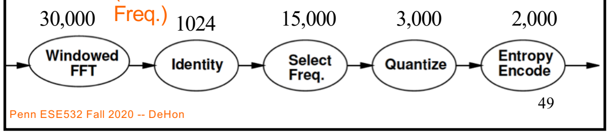
Streams

- Stream: logical communication link
- How might we implement:
  - Two threads running on a single processor (sharing common memory)?
  - Two processes running on different processors on the same chip?
  - Two processes running on different hosts
    - E.g. one at Penn, one on Amazon cloud

Penn ESE532 Fall 2020 -- DeHon 48

Add Delay

- What does it do to computation if add an operation that copies inputs to outputs with some latency?
  - Impact on function?
  - What is throughput impact when Identity operation has
    - Latency 10, throughput 1 value per cycle?
    - (reminder 1024 values between FFT and Select Freq.)



Semantics (meaning)

- Need to implement semantics
  - *i.e.* get same result as if computed as indicated
- But can implement any way we want
  - That preserves the semantics
  - Exploit freedom of implementation

Basic Approach

Approach (1)

- Identify natural parallelism
- Convert to streaming flow
  - Initially leave operations in software
  - Focus on correctness
- Identify flow rates, computation per operator, parallelism needed
- Refine operations
  - Decompose further parallelism?
  - E.g. data parallel split, ILP implementations
  - model potential hardware

Approach (2)

- Refine coordination as necessary for implementation
- Map operations and streams to resources
  - Provision hardware
  - Scheduling: Map operations to operators
  - Memories, interconnect
- Profile and tune
- Refine

Dataflow Variants

Part 3:

(coverage here depends on time available)

Turing Complete

- Can implement any computation describable with a Turing Machine
  - (theoretical model of computing by Alan Turing)
- Turing Machine – captures our notion of what is computable
  - If it cannot be computed by a Turing Machine, we don't know how to compute it

DataFlow (DF) Process Network Roundup

Model	Deterministic Result	Deterministic Timing	Turing Complete
SDF+fixed-delay operators	Y	Y	N
SDF+variable delay operators	Y	N	N
DDF blocking	Y	N	Y
DDF non-blocking	N	N	Y

Synchronous Dataflow (SDF) with fixed operators

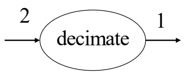
- Particular, restricted form of dataflow
- Each operation
  - Consumes a fixed number of input tokens
  - Produces a fixed number of output tokens
  - Operator performs fixed number of operations (in fixed time)
  - When full set of inputs are available
    - Can produce output
  - Can fire any (all) operations with inputs available at any point in time

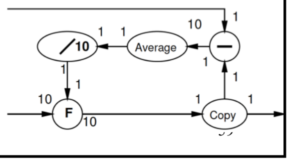
Synchronous Dataflow (SDF)

- Particular, restricted form of dataflow
- Each operation
  - Consumes a fixed number of input tokens
  - Produces a fixed number of output tokens
  - (can take variable computation for operator)
  - When full set of inputs are available
    - Can produce output
  - Can fire any (all) operations with inputs available at any point in time

Multirate Synchronous Dataflow

- Rates can be different
  - Allow lower frequency operations
  - Communicates rates to tools
    - Use in scheduling, provisioning
  - Rates must be constant
    - Data independent





Dynamic Dataflow

- (Less) restricted form of dataflow
- Each operation
  - Conditionally consume input based on data value
  - Conditionally produce output based on data value
  - When full set of inputs are available
    - Can (optionally) produce output
  - Can fire any (all) operations with data-specified necessary inputs available at any point in time

Blocking

- Key to determinism: behavior doesn't depend on timing
  - Cannot ask **if** a token is present
- If (not_empty(in))
  - Out.put(3);
- Else
  - Out.put(2);

Process Network Roundup

Model	Deterministic Result	Deterministic Timing	Turing Complete
SDF+fixed-delay operators	Y	Y	N
SDF+variable delay operators	Y	N	N
DDF blocking	Y	N	Y
DDF non-blocking	N	N	Y

Motivations and Demands for Options

Time Permitting

Variable Delay Operators

- Why might a multiplier have variable delay?
  - Hint: consider shift-and-add multiply
- Why might square root have variable delay?
- Why might memory lookup on a processor have variable delay?

Variable Delay Operators

- Operators with Variable Delay
  - Cached memory or computation
  - Shift-and-add multiply
  - Iterative divide or square-root

GCD (Preclass 3)

- What is delay of GCD computation?

- while(a!=b)
  - t=max(a,b)-min(a,b)
  - a=min(a,b)
  - b=t
- return(a);

Dynamic Rates?

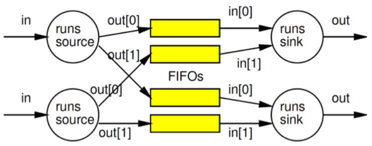
- What is implication of static rates
  - on compression?
  - Filtering?
    - (e.g. discard all spam packets)

Dynamic Rates?

- Static Rates limiting
  - Compress/decompress
    - Lossless
    - Even Run-Length-Encoding
  - Filtering
    - Discard all packets from spamRus
  - Anything data dependent

When non-blocking necessary?

- What are cases where we need the ability to ask if a data item is present?
- Consider an IP packet router:



Non-Blocking

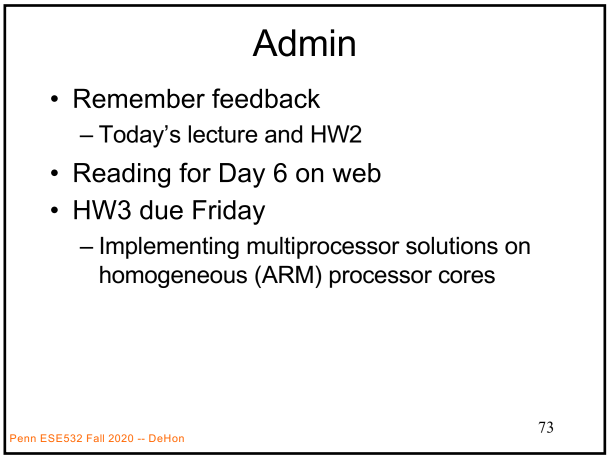
- Removed model restriction
  - Can ask if token present
- Gained expressive power
  - Can grab data as shows up
- Weaken our guarantees
  - Possible to get non-deterministic behavior

Process Network Roundup

Model	Deterministic Result	Deterministic Timing	Turing Complete
SDF+fixed-delay operators	Y	Y	N
SDF+variable delay operators	Y	N	N
DDF blocking	Y	N	Y
DDF non-blocking	N	N	Y

Big Ideas

- Capture gross parallel structure with Process Network
- Use dataflow synchronization for determinism
  - Abstract out timing of implementations
  - Give freedom of implementation
- Exploit freedom to refine mapping to optimize performance
- Minimally use non-determinism as necessary



Admin

- Remember feedback
  - Today’s lecture and HW2
- Reading for Day 6 on web
- HW3 due Friday
  - Implementing multiprocessor solutions on homogeneous (ARM) processor cores