ESE532: System-on-a-Chip Architecture

Day 5: September 18, 2017
Dataflow Process Model

Today

Dataflow Process Model

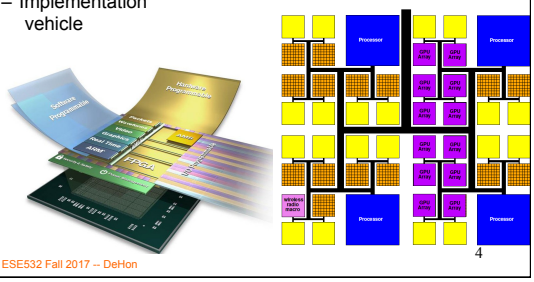
- Motivation
- Issues
- Abstraction
- Basic Approach
- Dataflow variants
- Motivations/demands for variants

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

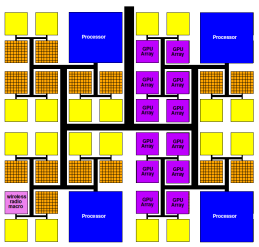
Programmable SoC

- 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)



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

Thread

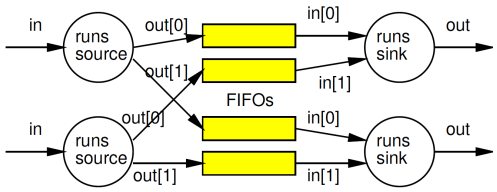
- Has a separate locus of control (PC)
- May share memory
  - Run in common address space with other threads

FIFO

- First In First Out
- Delivers inputs to outputs in order
- Data presence
  - Consumer knows when data available
- Back Pressure
  - Producer knows when at capacity
    - Typically stalls
- Decouples producer and consumer
  - Hardware: maybe even different clocks

Preclass 1

- Value of separate processes?



Process

- Processes allow expression of independent control
- Convenient for things that advance independently
- Performance optimization resource utilization

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

Operation/Operator

- **Operation** – logic computation to be performed
  - A process that communicates through dataflow inputs and outputs
- **Operator** – physical block that performs an Operation

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)

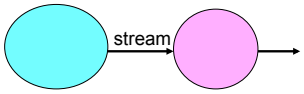
- Program is a sequence of operations
- Operation reads inputs and writes outputs into common store
- One operation runs at a time
  - defines successor

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 operators
  - Has a (single) source and sink
  - Carries data presence / flow control
  - Provides in-order (FIFO) delivery of data from source to sink



Streams

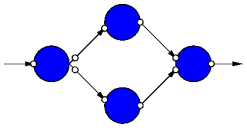
- Captures communications structure
  - Explicit producer→consumer link up
- Abstract communications
  - Physical resources or implementation
  - Delay from source to sink

Streams

- Stream: logical communication link
- What would we use
  - Two threads running on a single processor?
  - Two processes running on different processors on the same die?
  - Two processes running on different hosts
    - E.g. one at Penn, one on Amazon cloud

Dataflow Process Network

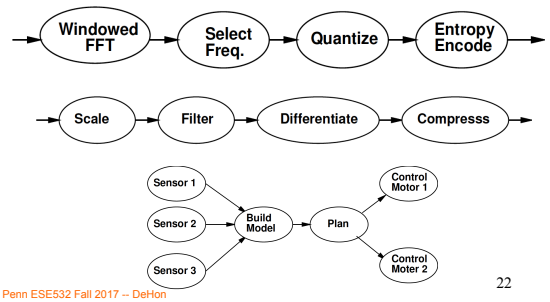
- Collection of Operators
- Connected by Streams
- Communicating with Data Tokens



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

Some Task Graphs



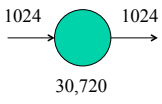
Synchronous Dataflow (SDF) with fixed operators

- 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
  - Assume one primitive operation per cycle
- Could embelish
  - Different time per operation type
  - Multiple memories with different timings

Time for Graph Iteration

- Single processor $T_{one} = \sum_i Nops_i$
- One processor per Operator

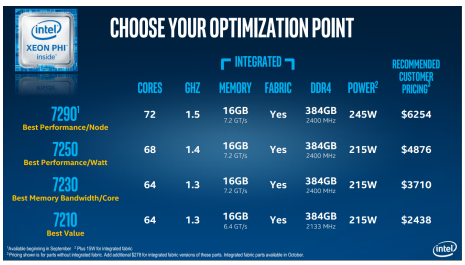
$$T_{each} = \max(Nop_1, Nop_2, Nop_3, \ldots)$$

- General

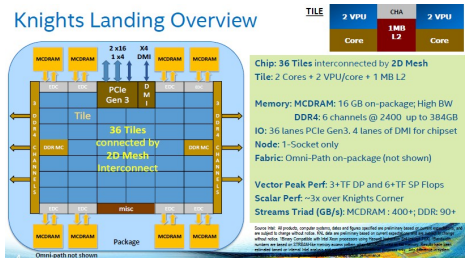
$$T_{map} = \max\left(\sum_i c(1,i) \times Nops_i, \sum_i c(2,i) \times Nops_i, \sum_i c(3,i) \times Nops_i, \ldots\right)$$

c(x,y) – 1 if Processor x runs task y

Intel Xeon Phi Pricing



Intel Knights Landing



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

[Intel, Micro 2016]

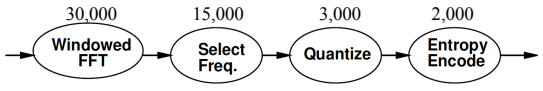
GRVI/Phallanx

- 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]

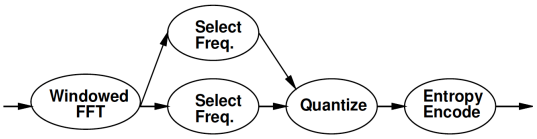
Map to different processors



- Map to
  - One processor performance?
  - One process per processor performance?
  - Two processors
    - How
    - Performance?

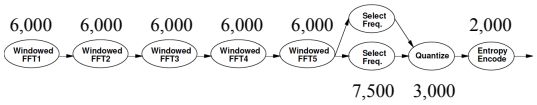
Refine Data Parallel

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



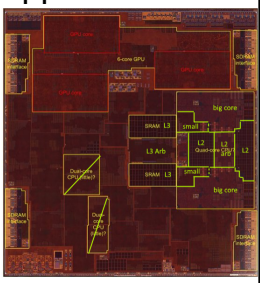
Refine Pipeline

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

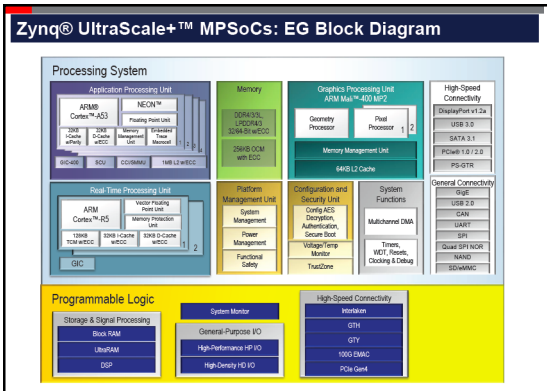


Today SoC: Apple A10

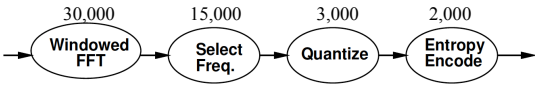
- 3.3B transistors
- Quad=Dual Dual Core
  - Dual 64-bit ARM 2.3GHz
  - Dual 64-bit ARM low energy
- 3MB L2 cache
- 6 GPU cores
- Custom accelerators
  - Image Processor?
- 125mm^2 16nm FinFET



Chipworks Die Photo



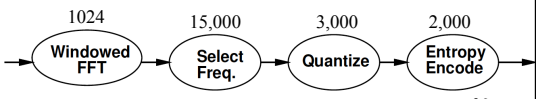
Heterogeneous Processor



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

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

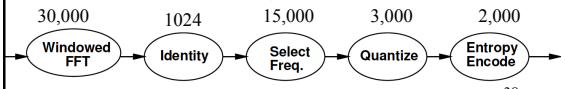


Operations

- 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
- May itself be described as
  - Dataflow process network, sequential, hardware register transfer language

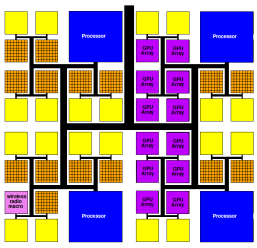
Add Delay

- What do to computation if add an operator that copies inputs to outputs with some latency?
  - Function?
  - Performance if Identity operator has
    - Latency 10, throughput 1 value per cycle?



Communication Latency

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

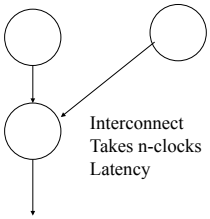


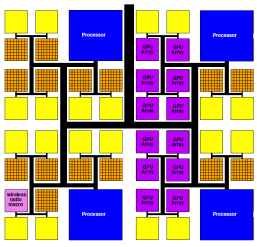
Day 3

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





Semantics

- 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 operators software
  - Focus on correctness
- Identify flow rates, computation per operator, parallelism needed
- Refine operators
  - Decompose further parallelism?
  - E.g. data parallel split, ILP implementations
  - model potential hardware

Approach (2)

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

Dataflow Variants

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

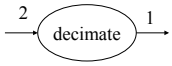
- Particular, restricted form of dataflow
- Each operation
  - Consumes a fixed number of input tokens
  - Produces a fixed number of output tokens
  - Operator performs fixed number of operations (in fixed time)
  - When full set of inputs are available
    - Can produce output
  - Can fire any (all)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

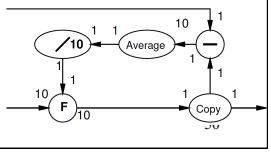
Synchronous Dataflow (SDF)

- 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

- What are example of variable delay operators we might have?

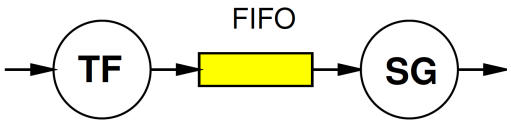
GCD (Preclass 3)

- Compute Data Parallel GCD
- What is delay of GCD computation?

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

Preclass 4

- How long to process each input?
- Correlation in delays?
- What benefit from FIFO and processes?

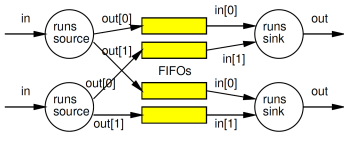


Dynamic Rates?

- When might static rates be limiting? (prevent useful optimizations?)

When non-blocking necessary?

- What are cases where we need the ability to ask if a data item is present?



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

- Reading for Day 6 on web
- HW3 due Friday