ESE532: System-on-a-Chip Architecture

Day 6: February 1, 2017
Process and Threads

Today

Dataflow Process Model
- Finish Models
- Aspirational Recommended Approach

Threads
- Hazards
- Discipline

Message

- Abstract the problem agnostic from implementation
- Map based on
  - Computational demand
  - Platform
- Use restricted patterns to avoid shared-memory pitfalls

Dataflow Variants

Synchronous Dataflow (SDF)

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

Dynamic Dataflow

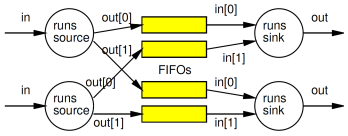
- (Less) restricted form of dataflow
- Each operation
  - Conditionally consume input based on data value
  - Conditionally produce output based on data value
  - When full set of inputs are available
    - Can (optionally) produce output
  - Can fire any (all) operations with data-specified necessary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

No Peaking

- 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);

When peaking necessary?

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



When Peak Optimization?

- What are cases where asking about data presence might allow performance optimization?

Peaking

- Removed model restriction (no peaking)
- 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 no peak	Y	N	Y
DDF w/ peak	N	N	Y

Aspirational 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. SIMD changes making hw3
  - model potential hardware

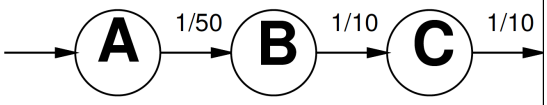
Approach (2)

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

Approach

- Process Network agnostic to operator implementation
- Implementation explores
  - operator mappings
    - What fixed SoC provides
    - Where might allocate hardware for novel SoC
  - Communication mappings
  - Memory allocation
    - Isolated memories→communication explicit

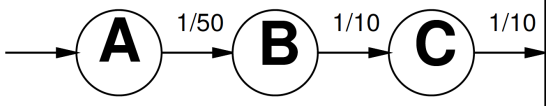
Preclass 1a



Data Parallel

- Throughput for 1:1 mapping of processes to SPs?
- Possibly we didn't know throughputs initially – map and measure

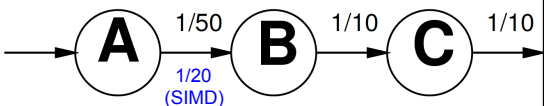
Preclass 1b



Data Parallel

- Refine: better mapping to 3 SPs?
- Throughput?

Preclass 1c



Data Parallel

- Mapping 1 SIMD and 1 SP?
- Throughput?

Approach

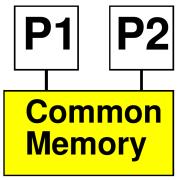
- Once have base process network (ideally), we have reduced to working in this mapping space
- Spend our time (in this class)
  - Enriching and elaborating this mapping space

Threads

Thread

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

Now: look at shared memory



Issues

- **Communication** – how move data between processes?
- **Synchronization** – how define how they advance relative to each other?
- **Determinism** – for the same inputs, do we get the same outputs?

Issues

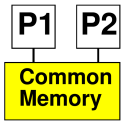
- **Communication** – read/write in common address space
- **Synchronization** – get to choose
  - Semaphores, locks, barriers, data presence
- **Determinism** – not unless you're very careful

Shared Memory

- Good
  - Sharing easy to express
  - Data movement implicit
- Bad
  - Easy to introduced unwanted non-determinism
  - Expensive abstraction to support
    - Hardware, energy
  - Harder to understand communication

What to watch for

- Thread 1:
  - Do stuff
  - M[123]=12
  - Do more stuff
  - A=M[123]*2
- Thread 2:
  - Do different stuff
  - M[123]=0
  - Do other stuff

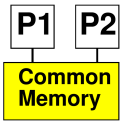


What value does A hold?

Preclass 2a

P1:
- for (i=0;i<N/2;i++)
  v=f(i);
  h[v]=h[v]+1;

P2:
- for (i=N/2+1;i<N;i++)
  v=f(i);
  h[v]=h[v]+1;

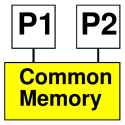


What can go wrong here?

Preclass 2b

P1:
- for (i=0;i<N/2;i++)
  v=f(i);
  h[v]=h[v]+1;

P2:
- for (i=N/2+1;i<N;i++)
  v=f(i);
  h[v]=h[v]+1;



How might we avoid hazard?

Disciplines

Don't really share
- Exploit actual sharing in limited ways

- Privatize
  - Local memory not really shared
  - Server thread to sequentialize access to shared state
- Use data presence
  - Buffer
  - Explicitly – presence variable and check

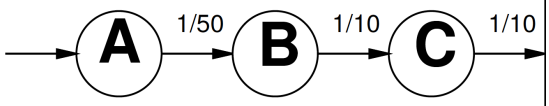
Disciplines

- Read-Only sharing OK
- Barrier
  - Force rendezvous at point before continue
  - Like a clock edge in a circuit

Expression and Implementation not 1:1

Examples
- Pure DPN implemented on shared-memory processors
  - Private memory, implement buffers in SM
- Single threaded DP with conditionals
  - Implement in multiple threads
- SDF with many tasks
  - Implement schedule of operations on single instruction stream

SDF in single thread



Data Parallel

```
While(input)
    A=Astep(in());
    B=Bstep(A);
    C=Cstep(B);
```


Multiple Ways to Decompose

- Searching for 2 targets strings in a single document
- Run on 2 processor
- How decompose and map?
- Resource implications?
- Data access implications?

Approach

- Identify design space of mappings
  - Decomposition, hardware mapping
- Analyze which meet our platform restrictions (are most efficient)
- Experiment as necessary

Big Ideas

- Abstract the problem agnostic from implementation
- Map based on
  - Computational demand
  - Platform
  - Seldom 1:1 → rich options to explore
- Use restricted patterns to avoid shared-memory pitfalls

Admin

- Reading for Day 7 on web
- HW3 due Friday