CprE / ComS 583 Reconfigurable Computing

Prof. Joseph Zambreno
Department of Electrical and Computer Engineering
Iowa State University

Lecture #21 – HW/SW Codesign

Outline

- HW/SW Codesign
  - Motivation
  - Specification
  - Partitioning
  - Automation

Hardware/Software Codesign

- Definition 1 – the concurrent and co-operative design of hardware and software components of an embedded system
- Definition 2 – A design methodology supporting the cooperative and concurrent development of hardware and software (co-specification, co-development, and co-verification) in order to achieve shared functionality and performance goals for a combined system [MicGup97A]

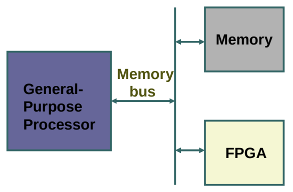
Motivation

- Not possible to put everything in hardware due to limited resources
- Some code more appropriate for sequential implementation
- Desirable to allow for parallelization, serialization
- Possible to modify existing compilers to perform the task

Why put CPUs on FPGAs?

- Shrink a board to a chip
- What CPUs do best:
  - Irregular code
  - Code that takes advantage of a highly optimized datapath
- What FPGAs do best:
  - Data-oriented computations
  - Computations with local control

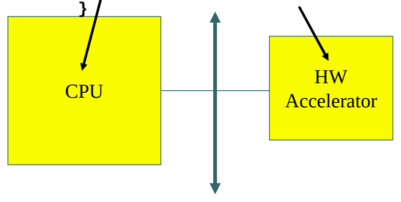
Computational Model



- Most recent work addressing this problem assumes relatively slow bus interface
- FPGA has direct interface to memory in this model

Hardware/Software Partitioning

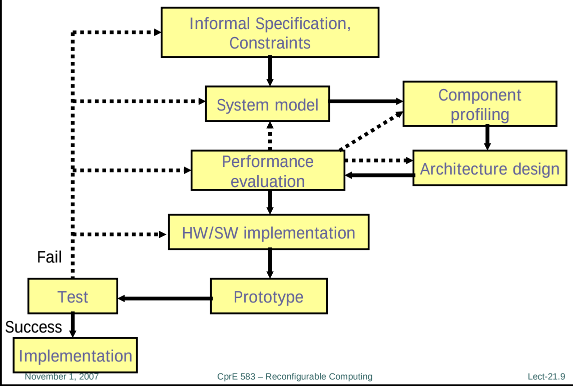
```
if (foo < 8) {
        for (i=0; i<N; i++)
                x[i] = y[i]*z[i];
}
```



Methodology

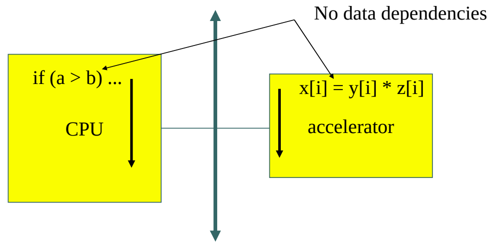
- Separation between function, and communication
- Unified refinable formal specification model
  - Facilitates system specification
  - Implementation independent
  - Eases HW/SW trade-off evaluation and partitioning
- From a more practical perspective:
  - Measure the application
  - Identify what to put onto the accelerator
  - Build interfaces

System-Level Methodology



Concurrency

- Concurrent applications provide the most speedup

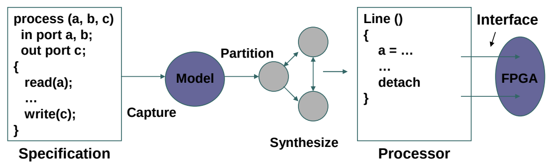


Partitioning

- Can divide the application into several processes that run concurrently
- Process partitioning exposes opportunities for parallelism

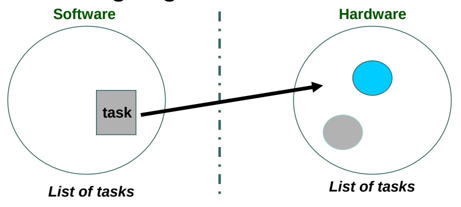
if (i>b) …	Process 1
for (i=0; i<N; i++) …	Process 2
for (j=0; j<N; j++) ...	Process 3

Automating System Partitioning



- Good partitioning mechanism:
  1) Minimize communication across bus
  2) Allows parallelism → both hardware (FPGA) and processor operating concurrently
  3) Near peak processor utilization at all times (performing useful work)

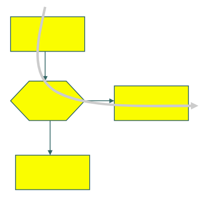
Partitioning Algorithms



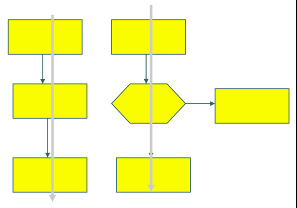
- Assume everything initially in software
- Select task for swapping
- Migrate to hardware and evaluate cost
  - Timing, hardware resources, program and data storage, synchronization overhead
- Cost evaluation and move evaluation similar to what we've seen regarding mincut and simulated annealing

Multi-threaded Systems

- Single thread:



- Multi-thread:

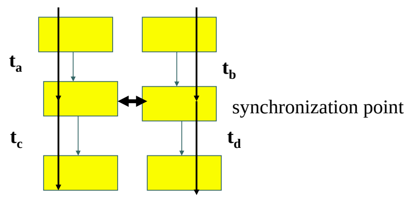


Performance Analysis

- Single threaded:
  - Find longest possible execution path
- Multi-threaded with no synchronization:
  - Find the longest of several execution paths
- Multi-threaded with synchronization:
  - Find the worst-case synchronization conditions

Multi-threaded Performance Analysis

- Synchronization causes the delay along one path to affect the delay along another

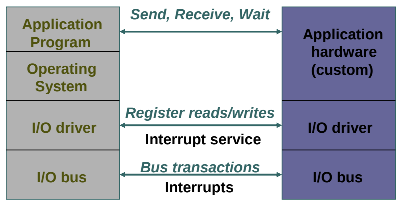


Delay = $\max(t_a, t_b) + t_d$

Control

- Need to signal between CPU and accelerator
  - Data ready
  - Complete
- Implementations:
  - Shared memory
  - Handshake
- If computation time is very predictable, a simpler communication scheme may be possible

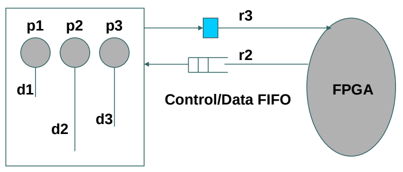
Communication Levels



- Easier to program at application level
  - (send, receive, wait) but difficult to predict
- More difficult to specify at low level
  - Difficult to extract from program but timing and resources easier to predict

Other Interface Models

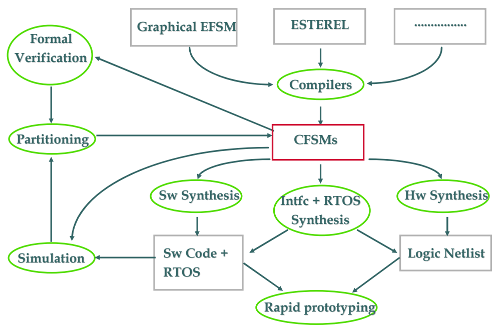
- Synchronization through a FIFO
- FIFO can be implemented either in hardware or in software
- Effectively reconfigure hardware (FPGA) to allocate buffer space as needed
- Interrupts used for software version of FIFO



Debugging

- Hard to test a CPU/accelerator system:
  - Hard to control and observe the accelerator without the CPU
  - Software on CPU may have bugs
- Build separate test benches for CPU code, accelerator
- Test integrated system after components have been tested

POLIS Codesign Methodology



Codesign Finite State Machines

- POLIS uses an FSM model for
  - Uncommitted
  - Synthesizable
  - Verifiable

  Control-dominated HW/SW specification
- Translators from
  - State diagrams,
  - Esterel, ECL, ReactiveJava
  - HDLs

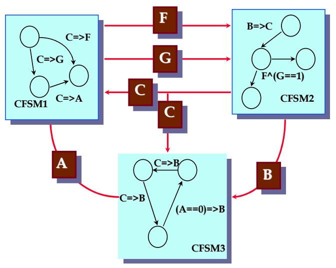
  Into a single FSM-based language

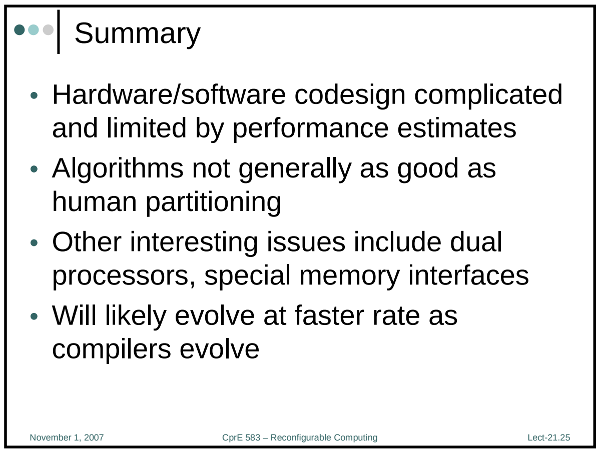
CFSM behavior

- Four-phase cycle:
  - ☆ Idle
  - Detect input events
  - Execute one transition
  - Emit output events
- Software response could take a long time:
  - Unbounded delay assumption
- Need efficient hw/sw communication primitive:
  - Event-based point-to-point communication

Network of CFSMs

- Globally Asynchronous, Locally Synchronous (GALS) model





Summary

- Hardware/software codesign complicated and limited by performance estimates
- Algorithms not generally as good as human partitioning
- Other interesting issues include dual processors, special memory interfaces
- Will likely evolve at faster rate as compilers evolve