CS294-6
Reconfigurable Computing

Day 22

November 5, 1998

Requirements for Computing Systems

(SCORE Introduction)

Previously

- What we need to compute
  - Primitive computational elements
    - compute, interconnect (time + space)
- How we map onto computational substrate
- What we have to compute
  - optimizing work we perform
    - generalization
    - specialization
  - directing computation
    - instruction, control

Today

- What do we expect out of a GP computing systems?
- What have we learned about software computer systems which aren't typically present in hardware?

- SCORE introduction

Desirable (from Day 3)

- We general expect a general-purpose computing platform to provide:
  - Get Right Answers :-)
  - Support large computations -> need to virtualize physical resources
  - Software support, programming tools -> higher level abstractions for programming
  - Automatically store/restore programs
  - Architecture family --> compatibility across variety of implementations
  - Speed -> … new hardware work faster

Expect from GP Compute?

- Virtualize to solve large problems
  - robust degradation?
- Computation defines computation
- Handle dynamic computing requirements efficiently
- Design subcomputations and compose

Virtualization

- Differ from sharing/reuse?
  - Compare segmentation vs. VM

Virtualization

- Functionally
  - hardware boundaries not visible to developer/user
  - (likely to be visible performance-wise)

  - write once, run “efficiently” on different physical capacities

How Achieve?

- Exploit Area-Time curves
- Generalize
  - local
  - instruction select
- Time Slice (virtualize)
- Architect for heavy serialization
  - processor, include processor(s) in resource mix

Virtualization Components

- Need to reuse for different tasks
  - store
    - state
    - instruction
  - sequence
  - select (instruction control)
    - predictability
    - lead time
    - load bandwidth

Handling Virtualization

- Alternatives
  - Compile to physical target
    - capacities/mix of resources
  - Manage physical resources at runtime

Data Dependent Computation

- Cannot reasonably take max over all possible values
  - – bounds finite, but unbounded
  - – pre allocate maximum memory?
- Consequence:
  - – Computations unfold during execution
  - – Can be dramatically different based on data
    - “shape” of computation differ based on data

Dynamic Creation

- Late bound data
  - – don’t know parameters until runtime
  - – don’t know number and types until runtime

- Implications: not known until runtime:
  - – resources (memory, compute)
  - – linkage of dataflow

Dynamic Creation

- Handle on Processors/Software
  - Malloc => allocate space
  - new, higher-order functions
    - parameters -> instance
  - pointers => dynamic linkage of dataflow

Dynamic Computation Structure

- Selection from defined dataflow
  - branching, subroutine calls
- Unbounded computation shape
  - recursive subroutines
  - looping (non-static/computed bounds)
  - thread spawning
- Unknown/dynamic creation
  - function arguments
  - cons/eval

Composition

- Abstraction is good
- Design independent of final use
- Use w/out reasoning about all implementation details (just interface)
- Link together subcomputations to build larger

Composition

- Processor/Software Solution
  - packaging
    - functions
    - classes
    - APIs
  - assemble programs from pre-developed pieces
    - call and sequence
    - link data through memory / arguments
    - mostly w/out getting inside the pieces

Resources Available

- Vary with
  - device/system implementation
  - task data characteristics
  - co-resident task set

Break
Remaining Assignments

- PROGRAM
- POWER
- Project Summary
  - class presentation

SCORE

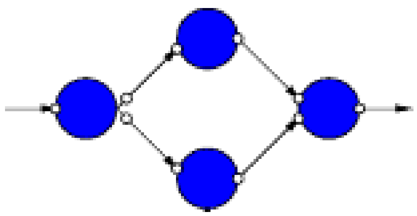
- An attempt at defining a computational model for reconfigurable systems
  - abstract out
    - physical hardware details
    - especially size / # of resources
- Goal
  - achieve device independence
  - approach density/efficiency of raw hardware
  - allow application performance to scale based on system resources (w/out human intervention)

SCORE Basics

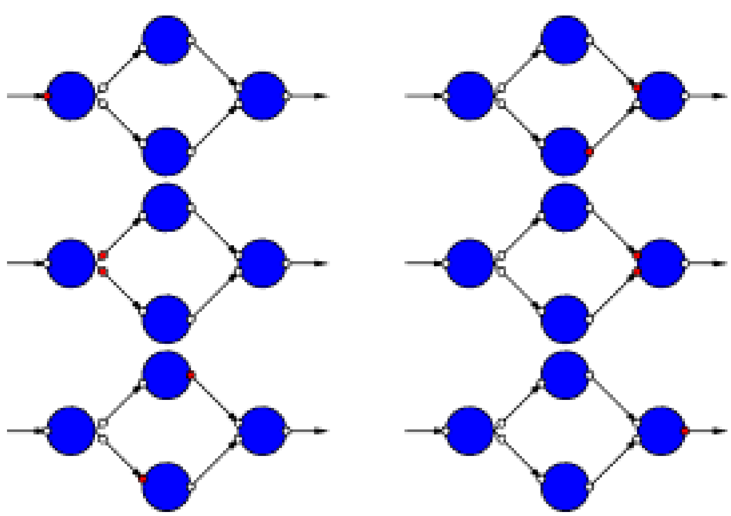
- Abstract computation is a dataflow graph
  - stream links between operators
  - dynamic dataflow rates
- Allow instantiation/modification/destruction of dataflow during execution
  - separate dataflow construction from usage
- Break up computation into compute pages
  - unit of scheduling and virtualization
  - stream links between pages
- Runtime management of resources

Dataflow Graph

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

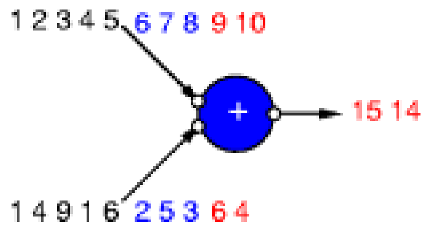


Dataflow Graph Example



Stream Links

- Sequence of data flowing between operators
  - e.g. vector, list, image
- Same
  - source
  - destination
  - processing



Operator

- Basic compute unit
- Primitive operators
  - single thread of control
  - implement basic functions
    - FIR, IIR, accumulate
- Provide parameters at instantiation time
  - new fir(8,16,{0x01,0x04,0x01})
- Operate from streams to streams

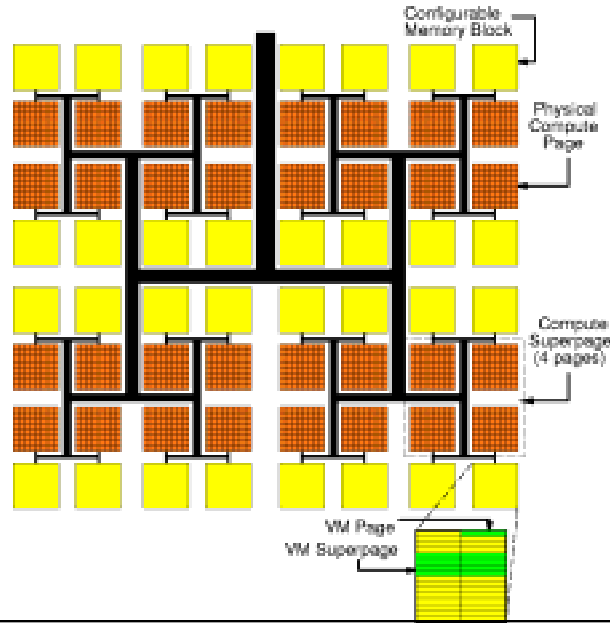
Composition

- Composite Operators: provide hierarchy
  - build from other operators
  - link up streams between operators
    - get interface (stream linkage) right and don't have to worry about operator internals
  - constituent operators may have independent control
- May compose operators dynamically
- Composition persists for stream lifetime

Compute Pages

- Primitive operators
  - broken into compute pages
    - (physical realization)
- Unit of
  - control
  - scheduling
  - virtualization
  - reconfiguration
- Canonical example:
  - HSRA Subarray (16--1024 BLB subtree)

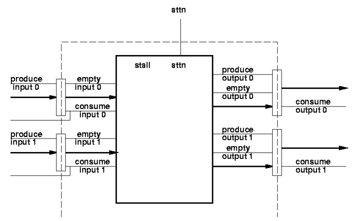
Hardware Model



Virtual/Physical

- Compute pages virtualized
- Mapped onto physical pages for execution

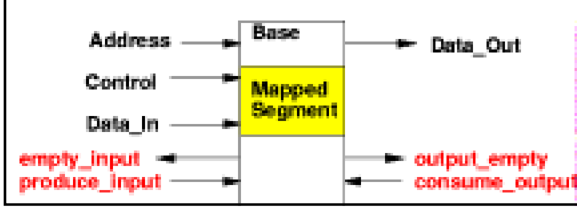
Compute Page



- Unit of Control
  - stall waiting on
    - input data present to compute
    - output path ready to accept result
  - runs together (atomicly)
  - partial reconfiguration at this level

Configurable Memory Block

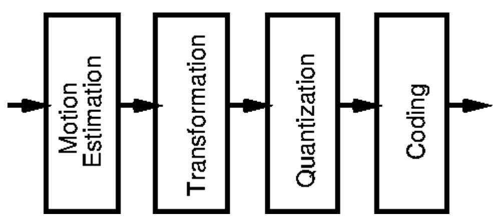
- Physical memory resource
  - serves
    - compute page configuration/state data
    - stream buffers
    - mapped memory segments



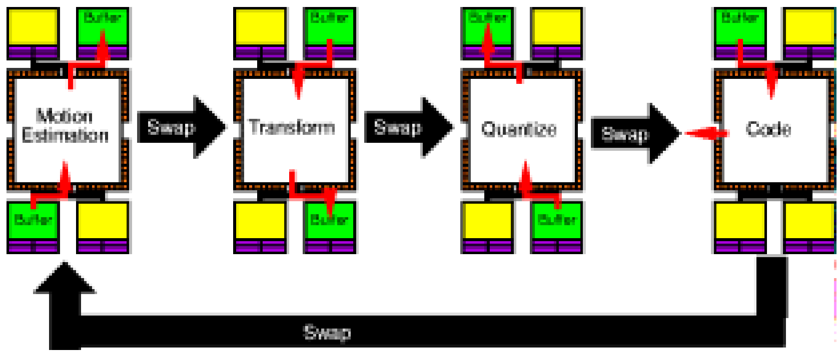
Stream Links

- Connect up
  - compute pages
  - compute page and processor / off chip io
- Two realizations
  - physical link through network
  - buffer in CMB between production and consumption

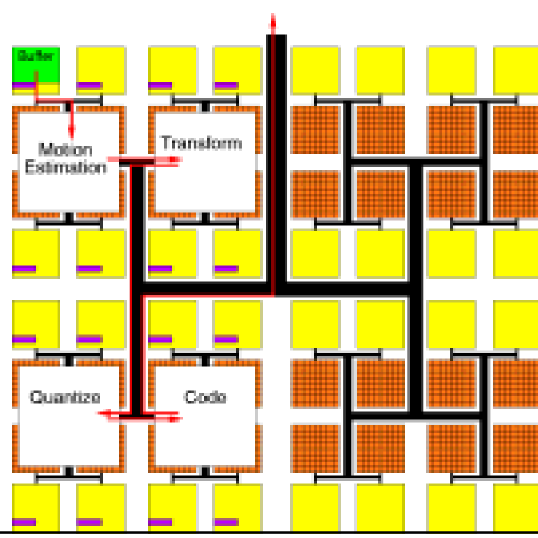
Example



Serial Implementation



Spatial Implementation



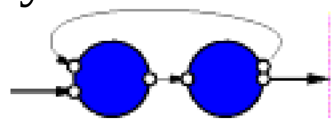
Dynamic Flow Rates

- Operator not always producing results at same rate
- data presence
  - throttle downstream operator
  - prevent write into stream buffer
- output data backup (buffer full)
  - throttle upstream operator
- stall page to throttle
  - persistent stall, may signal need to swap

Pragmatics

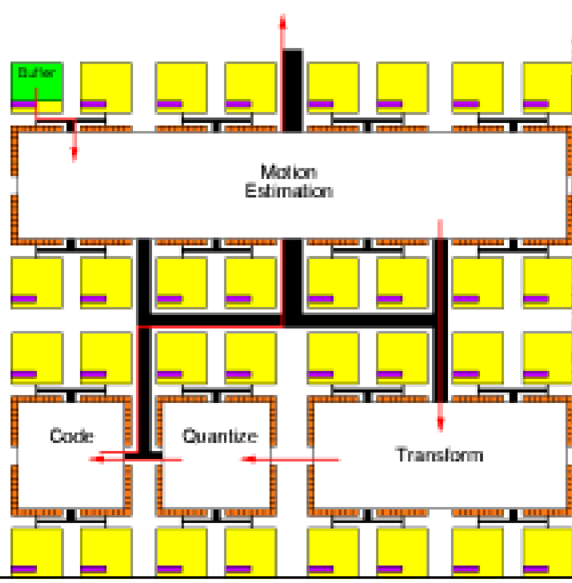
- Processor execute run-time management
- Attn notify processor
  - specialization/uncommon case fault
  - data stall
- Operator alternatives
  - run on processor / array
  - different area/time points, superpage blockings
  - specializations
- Locking on mapped memory pages

Pragmatics / Cycles



- Cycles spanning pages
  - will limit number of cycles can run page before stalls on its own downstream data
- Limit (short) cycles to page/superpage
  - unit guaranteed to be co-resident
  - state fine as long as limit to (super)page
- HSRA w/ on-chip DRAM
  - 100s of cycles for reconfig.
    - Want to be able to run 1000's of cycles before swap

Alternative Example



Computational Components

Element of Computation	Description Language	SCORE RT Model	Implementation
Transform	• RTL • Program operators	• Operators	• Pages (instructions or LUTs)
Interconnect: space	• dataflow links • stream links	• streams	• stream buffers • wires
Interconnect: time	• RTL registers • variables • arrays	• registers • streams • segments	• registers • CMBs
Synchronization	• dataflow	• dataflow/presence	• data presence
Creation	• allocate memory (new) • allocate/run thread	• new segment • new operator	• memory allocation (sbrk) • operator creation/instantiation
Control Flow	• sequencing/dataflow • threading	• dataflow • separate control per operator	• dataflow • runtime scheduling of operators
Data Types	• define/overload operators	• hide in operator definition	• not visible

tied up with what an operator is

Summary

- On to computing systems
  - virtualization
  - dynamic creation/linkage/composition and requirements
  - composability
- SCORE
  - fill out computational model for RC
    - capturing additional system features