Recap – Genetic Pattern Matching

- Comparing strings by edit distance
- Motivation: The Human Genome Project
  - Do two genetic strings match?
  - How are they related?
- When biologists characterize a new sequence, they want to compare it to the (growing) database of known sequences
- Abstraction:
  - What is the cost of transforming $s$ into $t$
  - Given – costs for insertion, deletion, substitution

Alphabet and Costs

- Alphabet
  - Letters in the string. For DNA, there are four:
    - A (Adenine)
    - C (Cytosine)
    - T (Thymine)
    - G (Guanine)
- Transformation Costs
  - Insert: 1, Delete: 1, Substitute: 2, match: 0
- Type of comparison
  - One target to many sources
  - One target to one source

Substitution Example

<table>
<thead>
<tr>
<th>Word</th>
<th>Move</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>baboon</td>
<td>Delete 'o'</td>
<td>1</td>
</tr>
<tr>
<td>bab</td>
<td>on</td>
<td>Substitute 'o'</td>
</tr>
<tr>
<td>bobon</td>
<td>Insert 'u'</td>
<td>1</td>
</tr>
<tr>
<td>bourbon</td>
<td>Insert 'r'</td>
<td>1</td>
</tr>
<tr>
<td>bourbon</td>
<td>Match?</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Total cost: 5</td>
</tr>
</tbody>
</table>

Dynamic Programming Solution

- Source sequence: $s_1, s_2, ... s_m$
- Target sequence: $t_1, t_2, ... t_n$
- $d_{ij}$ = distance between subsequence $s_1, s_2, ... s_i$ and subsequence $t_1, t_2, ... t_j$
  - $d_{0,0} = 0$
  - $d_{i,0} = d_{i-1,0} + \text{Delete}(s_i)$
  - $d_{0,j} = d_{0,j-1} + \text{Insert}(t_j)$
  - $d_{ij} = \min \begin{cases} 
    d_{i,j-1} + \text{Delete}(s_i) \\
    d_{i-1,j} + \text{Insert}(t_j) \\
    d_{i-1,j-1} + \text{Substitute}(s_i, t_j) 
  \end{cases}$
- Distance(Source, Target) = $d_{m,n}$
Parallelism on the Anti-Diagonal

Bidirectional PE Example

If (SCin != 0) and (TCin != 0)
PEDist ← \text{min}(PEDist + \text{Substitute}(SCin, TCin))

else-if (SCin != 0)
PEDist ← SDin
else-if (TCin != 0)
PEDist ← TDin
endif

SCout ← SCin
TCout ← TCin
SDout ← PEDist
TDout ← PEDist

Bidirectional Summary
- 16 CLBs/PE
- 384 PEs/Board
- 2,100 Million Cells/sec
- Requires \(2 \times (m+n)\) PEs
- Uses only half the processors at any one time
- Must stream both source and target for each comparison
  - Makes comparison against large DB impractical

Genetic Search Performance
- Nearly linear scaling in cell updates per second (CUPS)
- Need to reuse array for large patterns

<table>
<thead>
<tr>
<th>Hardware</th>
<th>CUPS</th>
<th>(\lambda)</th>
<th>Area</th>
<th>CUP/(\lambda^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Splash 2 x16</td>
<td>43.000M</td>
<td>0.60(\mu)</td>
<td>500M(\lambda^2) x17x16</td>
<td>0.32</td>
</tr>
<tr>
<td>Splash 2</td>
<td>3.000M</td>
<td>0.60(\mu)</td>
<td>500M(\lambda^2) x16</td>
<td>0.38</td>
</tr>
<tr>
<td>Splash 1</td>
<td>370M</td>
<td>0.60(\mu)</td>
<td>420M(\lambda^2) x32</td>
<td>0.028</td>
</tr>
<tr>
<td>P-NAC (34)</td>
<td>500M</td>
<td>2.0(\mu)</td>
<td>7.8M(\lambda^2) x34</td>
<td>1.9</td>
</tr>
<tr>
<td>CM-2 (64K)</td>
<td>150M</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>CM-5 (32)</td>
<td>33M</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>SPARC 10</td>
<td>1.2M</td>
<td>0.40(\mu)</td>
<td>1.6G(\lambda^2)</td>
<td>0.00075</td>
</tr>
<tr>
<td>SPARC 1</td>
<td>0.87M</td>
<td>0.75(\mu)</td>
<td>273M(\lambda^2)</td>
<td>0.0032</td>
</tr>
</tbody>
</table>

Outline
- Recap – Pattern Matching on Splash-2
- The Field-Programmable Port Extender (FPX)
- FPX Architecture
- FPX Programming Model
- FPX Applications
  - Pattern Matching
  - Packet Classification
  - Rule Processing

Application – Network Processing
- Networking applications well-suited for reconfigurable hardware
  - Target signatures change often
  - Massive quantities of stream-based data
  - Repetitive operations
- Connecting up to a realistic networking environment is hard
  - Washington University experimental setup one of the best
  - Shows importance of both memory and processing capability
- Numerous experiments performed over the past five years
Network Routing with the FPX

- FPX Modules distributed across each port of a switch
- IP packets (over ATM) enter and depart line card
- Packet fragments processed by modules
- Advantages:
  - New protocols implemented directly in silicon
  - Easy to upgrade in the field

FPX Hardware Device

FPX Hardware in a WUGS-20 Switch

FPGA-based Router

Reprogrammable Application Device

- Spatial Re-use of FPGA Resources
  - Modules implemented using FPGA logic
  - Module logic can be individually reprogrammed
- Shared Access to off-chip resources
  - Memory Interfaces to SRAM and SDRAM
  - Common Datapath to send and receive data

Architecture of the FPX

- **RAD**
  - Large Xilinx FPGA
  - Attaches to SRAM and SDRAM
  - Reprogrammable over network
  - Provides two user-defined Module Interfaces
- **NID**
  - Provides Utopia Interfaces between switch & line card
  - Forwards cells to RAD
  - Programs RAD
Architecture of the FPX (cont.)

- PROM
- Cache
- Program
- Flow Buffer
- Route Filter
- Extensible Modules
- Layered Protocol Wrappers
- Switch
- SDRAM
- SRAM
- NID (FPGA)
- Memory

FPX SRAM
- Provide low latency for fast table-lookups
- Zero Bus Turnaround (ZBT) allows back-to-back read / write operations every 10ns
- Dual, Independent Memories
- 36-bit wide bus

FPX SDRAM
- Dual, independent SDRAM memories
- 64-bit wide, 100 MHz
- 64Mb / Module : 128 Mb total [expandable]
- Burst-based transactions [1-8 word transfers]
- Latency of 14 cycles to Read/Write 8-word burst

Routing Traffic Flows
- Traffic flows routed among
  - Switch
  - Line Card
  - RAD Switch
  - RAD Linecard
- Functions
  - Check packets for errors
  - Process commands
  - Control, status, & reprogramming
  - Implement per-flow forwarding

Typical Flow Configurations

Reprogramming Logic
- NID programs at boot from EPROM
- Switch Controller writes RAD configuration memory to NID
  - Configuration file for RAD arrives transmitted over network via control cells
- Switch Controller issues [Full/Partial] reconfigure command
- NID reads RAD config memory to program RAD
  - Performs complete or partial reprogramming of RAD
**FPX Interfaces Provides**
- Well defined Interface
  - Utopia-like 32-bit fast data interface
  - Flow control allows back-pressure
- Flow Routing
  - Arbitrary permutations of packet flows through ports
  - Dynamically Reprogrammable
    - Other modules continue to operate even while new module is being reprogrammed
- Memory Access
  - Shared access to SRAM and SDRAM
  - Request/Grant protocol

**Pattern Matching using the FPX**
- Use Hardware to detect a pattern in data
- Modify packet based on match
- Pipeline operation to maximize throughput

**"Hello, World" Module Function**
- Compare
- Match
- Match+Write
- Write
- Copy
- Copy

**Logical Implementation**
- VCI Match
- "Hello" Check
- "World" to payload
- Append "WORLD" to payload
- Reset
- New Cell
- Dout

**The Wrapper Concept**
- App
- Wrapper
- Wrapper

**AAL5 Encapsulation**
- Payload is packed in cells
- Padding may be added
- 64 bit Trailer at end of cell
- Trailer contains CRC-32
- Last Cell indication bit (last bit of PTI field)
HelloBob Module

Results: Performance

- Operating Frequency: 119 MHz.
  - 8.4ns critical path
  - Well within the 10ns period RAD's clock.
  - Targeted to RAD's V1000E-FG680-7
- Maximum packet processing rate:
  - (100 MHz)/(14 Clocks/Cell)
  - Circuit handles back-to-back packets
- Slice utilization:
  - 0.4% (49/12,288 slices)
  - Less than one half of one percent of chip resources
- Search technique can be adapted for other types of data matching and modification
  - Regular expressions
  - Parsing image content...

CAM-based Packet Matching

- Sample Packet:
  - Source Address = 128.252.5.5 (dotted.decimal)
  - Destination Address = 141.142.2.2 (dotted.decimal)
  - Source Port = 4096 (decimal)
  - Destination Port = 50 (decimal)
  - Protocol = TCP (6)
  - Payload = “Consolidate your loans. CALL NOW”
  - Payload Lists = { General SPAM (0), Save Money SPAM (1) }
  - Content Vector = “00000011” (binary) = x’03’ (hex)

Sample Filter

- Source Address = 128.252.0.0 / 16
- Destination Address = 141.142.0.0 / 16
- Source Port = Don't Care
- Destination Port = 50
- Protocol = TCP (6)
- Payload includes general SPAM (List 0)

Packet Classifier with FlowID

Fast IP Lookup Algorithm

- Function
  - Search for best matching prefix using Trie algorithm
Hardware Implementation in the FPX

- SRAM1
- SRAM2
- IP Lookup Engine
- Counter
- On-Chip Cell Store
- SRAM1 Interface
- Control Cell
- Processor
- Packet Reassembler
- NID FPGA

Pipelined FIPL Operations

- Generate Address
- Latch ADDR into SRAM
- SRAM D < M[A]
- Latch Data into FPGA
- Compute

- Time (cycles)

- Throughput: Optimized by interleaving memory accesses
  - Operate 5 parallel lookups
  - t_pipelined_lookup = 550ns / 5 = 110 ns
  - Throughput = 9.1 Million packets / second

Other Modules Implemented

- IPv4 CAM Filter
  - 104 Bit header matching
  - Fast IP Lookup (FIPL)
  - Longest Prefix Match
  - MAE-West at 10M pkts/second
  - Packet Content Scanner
  - Reg. Expression Search
  - Data Queueing
  - Per-flow queue in SDRAM

- IPv6 Tunneling Module
  - Tunnels IPv6 over IPv4
- Statistics Module
  - Event counter
- Traffic Generator
  - Per-flow mixing
- Video Recoder
  - Motion JPEG
- Embedded Processor
  - KCPSM

Summary

- Field Programmable Port Extender (FPX)
  - Network-accessible Hardware
  - Reprogrammable Application Device
- Module Deployment
  - Modules implement fast processing on data flow
  - Network allows Arbitrary Topologies of distributed systems
- Project Website
  - http://www.arl.wustl.edu/arl/projects/fpx/