Advanced Crossbar Scheduling

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst
Approximate LOOFA

- Alternate proof that $\text{minSlack}_i$ increases during xfer
  - separate xfer phase into two steps - move cells and sort
  - $\text{slack}_{ij}$ increases by 1 during move-cells step for all $i,j$ so $\text{minSlack}_i$ increases by 1
  - break sort step into compare & swap steps between consecutive VOQs in ordered list
  - let $V_{ij}$ come just before $V_{ih}$ in ordered list and let $q_i < q_j$ and let $s_{ij}$ and $s_{ih}$ be slack values after the swap
    - then $s_{in} > \text{slack}_{in}$ and $s_{ij} > \text{slack}_{ih}$
    - so, the minimum slack for $V_{ij}$ and $V_{ih}$ increases by 1
  - over multiple compare & swap steps, $\text{minSlack}_i$ does not decrease, so move-cells and swap step increase $\text{minSlack}_i$
    - is true, even if we don’t do a complete sort
Implementing ALOOFA

- Difficult to implement LOOFA at high speed
  - number of iterations, sorting of outputs
- Hardware implementation of maximal matching allows $n$ iterations to complete quickly
  - $4n$ gate delays: 6.4 ns for 32 port switch @50 ps per gate
Maintaining Output Ordering

- Not enough time to fully sort outputs
- Partial sorting effective for slowly changing lengths

initial state

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

queue lengths change by at most 1

compare & swap adjacent columns

final state

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
<td>f</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

note that entire columns are swapped
Fair Treatment of Inputs

- For speedups $< 2$, unfair treatment of inputs becomes issue
- Resolve by performing random row permutations
Connecting to Inputs and Outputs

- Scheduling component communicates with input, output line cards through fixed I/O pins
- To allow row/column swaps add IO xbars
- Swap rows of input xbar with rows of matching array
- Swap cols. of output xbar with cols. of matching array

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**Input pins**

**Output pins**

**Input xbar**

**Output xbar**
Implementation Data

- Circuit for 8 port scheduler synthesized for FPGA

<table>
<thead>
<tr>
<th></th>
<th>VHDL lines</th>
<th>LUTS</th>
<th>FFs</th>
</tr>
</thead>
<tbody>
<tr>
<td>matcher</td>
<td>160</td>
<td>745</td>
<td>72</td>
</tr>
<tr>
<td>lxbar</td>
<td>95</td>
<td>307</td>
<td>64</td>
</tr>
<tr>
<td>oxbar</td>
<td>79</td>
<td>240</td>
<td>64</td>
</tr>
<tr>
<td>top</td>
<td>232</td>
<td>941</td>
<td>368</td>
</tr>
<tr>
<td>total</td>
<td>614</td>
<td>2,233</td>
<td>570</td>
</tr>
</tbody>
</table>

- Synthesized for Xilinx Virtex 5 XC5VFX30-2
  - 6 input LUTs, 19,200 LUTs, 19,200 FFs
  - Clock period of 5.7 ns, 7 clock tick cycle, so 40 ns operating cycle
Stress Test Performance

- Results for 64 phase stress test
Emulating Ideal PIFO Switch

- Emulation of ideal switch with PIFO queues
  - in a push-in, first-out queue, arriving packets are inserted at an arbitrary point, and packets depart from front
    - can model any queueing discipline in which the relative order of two packets is decided when the “second” one arrives
    - all the common fair-queueing disciplines are PIFO
  - objective is to use a crossbar to give exactly the same behavior as ideal output-queued switch with a specified PIFO queueing discipline

- Can achieve emulation with a speedup of 2
  - for unbuffered crossbars, requires stable matchings
NVF PIFO

- NVF-P is variant of newest VOQ first scheduler
  - each input keeps list of VOQs with "newest" VOQs at front
    - inactive VOQs kept at end of list
    - arriving cells inserted into VOQ in order determined by specified PIFO queueing discipline
  - each output keeps list of inputs with order determined by first cell in inputs’ VOQs for output $i$
    - that is, output $j$ lists input $h$ before input $i$ if first cell in $V_{hj}$ comes before first cell in $V_{ij}$ in PIFO ordering
    - inputs with no cells for output $j$ come last in its list
  - crossbar scheduler computes a stable matching using lists defined by inputs and outputs
  - each output transmits cells using specified PIFO discipline
Stable Matchings

- Given two sets $A$ and $B$ with $n$ elements each, and
  - for each element $a \in A$, a function $rank_a(b)$ that assigns each $b \in B$ a unique integer in $[1,n]$, and
  - for each element $b \in B$, a function $rank_b(a)$ that assigns each $a \in A$ a unique integer in $[1,n]$
- Find a pairing $P$ of the elements such that for all pairs $(a,b) \in P$ and $(\alpha,\beta) \in P$
  
  \[
  (rank_a(b) < rank_a(\beta) \text{ or } rank_b(\alpha) < rank_b(a)) \text{ and } \\
  (rank_b(a) < rank_b(\alpha) \text{ or } rank_a(\beta) < rank_a(b))
  \]
- Algorithm sketch – start with no pairs
  - repeat until a stable matching is achieved
    - each unpaired element in $A$ makes a “bid” for top-ranked member of $B$ which it has not bid for previously
    - each element of $B$ selects best of its current bids or stays with its current “partner” (if any)
      - selections made by elements of $B$ form the current set of pairs
Example of NVF-P

VOQ lists

0 1 2 3
2,3 d e
1,0,2,3 b c c b
2,1,3,0 c b a d c
0,2,3 a d b a

stable matching
input 0 selects 2
Input 1 selects 1
input 2 selects 2
input 3 selects 0
output 2 selects input 2

second round matches
input 0 to output 3

output queues

6 3 3 5
NVF Emulation Result

- Preliminaries
  - A cell \( b \) precedes another cell \( c \) at input \( i \), if its VOQ comes first or it is in same VOQ but is earlier in PIFO order
  - Let \( p(c) \) be 1+number of cells at input \( i \) that precede \( c \)
  - Let \( q(c) \) be number of cells in queue at the output \( c \) is going to that come before \( c \) in the PIFO order
  - \( \text{margin}(c)=q(c)-p(c) \)

- Sketch of proof
  - \( \text{margin}(c) \) never decreases while \( V_{ij} \) remains active
  - \( \text{margin}(c) \geq 0 \) while \( V_{ij} \) is active
  - If output \( j \) sends a cell \( x \) that comes later in the PIFO order than some cell \( c \) in a VOQ, then \( \text{margin}(c)<0 \)
**Margin(c) Never Decreases**

- Idealized switch operation for speedup 2
  - input phase – cells arrive and are placed in VOQs
  - transfer phase – cells move from inputs to outputs
  - output phase – cells sent from outputs
  - second transfer phase
- During each transfer phase
  - if a cell $c$ is not transferred
    - either $q(c)$ increases by one or $p(c)$ decreases by 1
    - in either case, $\text{margin}(c)$ increases by $\geq 1$
- Input phase causes $p(c)$ to increase by at most 1
- Output phase causes $q(c)$ to decrease by at most 1
- So, over full cycle, $\text{margin}(c)$ cannot decrease
NVF-P Emulates PIFO Discipline

- $\text{margin}(c) \geq 0$ before each output phase
  - right after $c$ arrives, $p(c) = 1$, so $\text{margin}(c) \geq -1$
  - next xfer phase increases $\text{margin}(c)$ by 1, so $\text{margin}(c) > 0$
  - subsequent cycles can lead to no net decrease in $\text{margin}(c)$ so long as $c$ remains at input

- No wasted output phases
  - suppose at some output phase, output $j$ sends a cell $x$ that comes later in PIFO order than some cell $c$ in $V_{ij}$
  - since output $j$ sends cells in PIFO order, $q(c) = 0$
  - since $p(c) \geq 1$, $\text{margin}(c) = q(c) - p(c) < 0$, yielding a contradiction

- So, NVF emulates PIFO for speedup of 2