Realizing Reconfigurable Mesh Algorithms on Softcore Arrays

Heiner Giefers, Marco Platzner
Computer Engineering Group
University of Paderborn

{hriefers, platzner}@upb.de
Outline

1. Reconfigurable mesh model
2. Implementing reconfigurable meshes on softcore arrays
3. Execution of reconfigurable mesh algorithms
4. Case studies
5. Summary and outlook
Reconfigurable Mesh Model (R-Mesh)

- Cores execute in a lock step
  1. Switch configuration
  2. Communication
  3. (Constant time) computation

- Connection autonomy
  - 15 possible patterns

- Many model extensions

- Communication delay model
  - $O(1)$ and $O(\log(N))$

- Example: MAX
  - Complexity: $O(1)$
  - Communication distance: $3N$
Algorithm Mapping

- Basic algorithms
  - Arithmetic [Miller et al. 93]
    - Prefix sum of $N$ elements, $O(1)$ on $N \times N$ mesh
    - Minimum of $N$ elements, $O(1)$ on $N \times N$ mesh
  - Sorting of $N$ elements
    - On $N \times N$ mesh in constant time [Jang and Prasanna 95]
    - On $N^{1/2} \times N^{1/2}$ mesh in $O(N^{1/2})$ [Thompson and Kung 77]
  - Graph algorithms
  - Image processing

- Simulation of other network topologies
  - Pyramid, simulation overhead $O(1)$
  - Mesh-of-Trees, simulation overhead $O(\log(N))$
Practical Issues

- Advantages of the R-Mesh model
  - Scalable parallel programming model
  - Algorithms for numerous problems with outstanding runtime complexities

- Challenges of an R-Mesh implementation
  - Communication delays of $O(1)$ or $O(\log N)$ are unrealistic
  - Data I/O is not covered by runtime analysis

- Our goal: reduce the R-Mesh model to practice
  - (How) Can we exploit the theoretical quality while accepting multi-cycle communication?
Outline

1. Reconfigurable mesh model ✓
2. Implementing reconfigurable meshes on softcore arrays
3. Execution of reconfigurable mesh algorithms
4. Case studies
5. Summary and outlook
Building Blocks: Switch & Processing Element

- Reconfigurable switch
  1. Combinatorial switch [FPL07]
  2. Buffered switch
  3. Self timed switch
  4. Photonic switch

- Picoblaze
  - 8-bit RISC, not pipelined
  - 96 slices on Virtex-II
  - 1024 instructions in 1 BRAM
  - 2 cycles per instruction
  - 64 byte scratchpad RAM

- Wrapper
  - Storing data and switch configuration
  - Optional I/O functionality
Host Integration

- RMESH as coprocessor
- MicroBlaze 32-bit RISC core as host CPU
- Connected through FSL channels to an I/O controller
  - Low latency of FSL
  - 32-bit to 4 x 8-bit splitting
  - Load and store operations take $O(N)$ time

- Entire system runs at 100 MHz
### Synthesis Results for Virtex 4 (XCV4LX200)

<table>
<thead>
<tr>
<th>Building Blocks</th>
<th>Network Tiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>slices (utilization)</td>
<td>BRAM</td>
</tr>
<tr>
<td>PE1: pBlaze + wrapper</td>
<td>114 (0.13%)</td>
</tr>
<tr>
<td>PE2: pBlaze + wrapper + multiplier</td>
<td>131 (0.15%)</td>
</tr>
<tr>
<td>SE1: RMESH switch</td>
<td>144 (0.15%)</td>
</tr>
<tr>
<td>SE2: HVRM switch</td>
<td>67 (0.08%)</td>
</tr>
<tr>
<td>Node1: PE1+SE1</td>
<td>265 (0.30%)</td>
</tr>
<tr>
<td>Node2: PE1+SE2</td>
<td>178 (0.20%)</td>
</tr>
<tr>
<td>Node3: PE2+SE1</td>
<td>282 (0.32%)</td>
</tr>
<tr>
<td>Node4: PE2+SE2</td>
<td>195 (0.22%)</td>
</tr>
</tbody>
</table>

- **16x16 Prototype: µBlaze, FSLs, 256 Node1 (PE1+SE1)**
  - 65091 slices (73%)
  - 264 BRAMs (78%)
  - 100 MHz
**Design Tool Flow**

- **Xilinx EDK Flow**
  - HW generation
  - μBlaze programming

- **R-Mesh algorithm**

- **Assembly code generator**

- **Picoblaze assembler**

- **Bitfile modification**

- **Cycle-accurate simulator**
  - Simulate arbitrary mesh sizes
Outline

1. Reconfigurable mesh model ✓
2. Implementing reconfigurable meshes on softcore arrays ✓
3. Execution of reconfigurable mesh algorithms
4. Case studies
5. Summary and outlook
Reconfigurable Mesh Execution

R-Mesh requirements
- Cores execute in a lock step
  1. Switch configuration
  2. low latency communication
  3. bounded computation
- Communication must be synchronized

Observations
- Communication pattern often independent of static data, thus known at compile time
- Complexity of computation phase varies

Implementation principles
- Very low communication overhead
  - Configure the switch with an atomic instruction
  - Write registers directly to the mesh
- Select algorithms that match our architecture
  - Keep linear communication overhead in mind
  - Increase computation/communication ratio
Synchronisation

1. Communication in a single (clock) cycle
   - Inappropriate for large scale meshes

2. Combinatorial switches with multi-cycle paths
   - Tedious synthesis process (at least with Xilinx place&route tools)

3. Buffered switches
   - Extend data with valid bit
   - More robust and flexible
Algorithm Selection

- Complexity Analysis bases on constant time delay
  - Algorithms with best computational complexity usually make heavy use of broadcasts
  - Our architecture profits from short distance communication
  - Select / develop proper algorithms

- Example: 3 maximum finding algorithms
  1. Classical R-Mesh algorithm with constant runtime
  2. Computes the Maximum of N values on an R-Mesh of size N by sequentially comparing all k bits of all values
  3. Simulates a binary tree algorithm for maximum finding on R-Mesh

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Datasize</th>
<th>Area</th>
<th>Complexity</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>N</td>
<td>N x N</td>
<td>O(1)</td>
<td>3N [cycles]</td>
</tr>
<tr>
<td>#2</td>
<td>N</td>
<td>N</td>
<td>O(k)</td>
<td>2kN^{1/2} [cycles]</td>
</tr>
<tr>
<td>#3</td>
<td>N</td>
<td>N</td>
<td>O(log(N^{1/2}))</td>
<td>2N^{1/2} [cycles]</td>
</tr>
</tbody>
</table>
Outline

1. Reconfigurable mesh model ✓
2. Implementing reconfigurable meshes on softcore arrays ✓
3. Execution of reconfigurable mesh algorithms ✓
4. Case studies
5. Summary and outlook
### Case Study: Sparse Matrix Multiplication

- **RMESH algorithm for column/row sparse matrix multiplication** [Middendorf et al. 99]
- **Column sparse ($\leq k$ non-zero Elements/column) Algorithm:**
  - Repeat (at most $k (=2)$ times)
    - Broadcast the top-most non-zero elements of A to the appropriate rows of B
    - Multiply the received A-element with the local B-element
    - Route the Products to the row index given by the A-element ($\leq k$ steps)
    - Discard the top-most A-element

![Matrix Multiplication Example](image)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C = AxB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 1 0 0 0 0 0</td>
<td>0 0 4 0 1 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 1 0 0 0 0 0 0</td>
<td>0 0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>5 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>5</td>
<td>0 0 0 0 0 0 0</td>
<td>0 0 0 0 1 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 7</td>
<td>0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

**Row Index:**
- **A** (top-most non-zero (row index))
- **B**
- **C = AxB**

![Example Matrix Diagram](image)
Runtime Results

- Multiplication of 2 16x16 sparse matrices on a 16x16 mesh
  - Algorithm complexity is $O(k^2)$
  - Matrices are fully loaded → constant I/O overhead of 2530 cycles
  - (Straight-forward) matrix multiplication on Microblaze takes 14788 cycles
  - Preloaded data and $k=1 \rightarrow$ Speedup 74 ($k=2 \rightarrow$ Speedup 11.5)

![Diagram showing runtime results](image.png)
**Scalability through Self Simulation**

- Simulate an RM algorithm for \( N \) nodes on an RM instance (of the same model) consisting of \( P \) nodes (normally: \( P < N \))
  - Simulation is dependent on RM model, not on the simulated algorithm
  - Self simulation slowdown \( O(f^* N/P) \), with simulation overhead \( f \)
    - \( f = O(1) \) → optimal self simulation for HV-RM, L-RM
    - \( f = O(P) \) → strong self simulation for FR-RM (\{NSWE\} and \{NS, WE\} only)
    - \( f = O(\log N) \) → weak self simulation for general RM

- HV-RM simulation uses contraction mapping
  - Shortens communication distances
Self Simulation Example

- Add up $64 \times 64 = 4096$ elements
  - Assume preloaded data
  1. On single Microblaze (best Optimization): 24696 cycles
  2. On 64x64 RMESH: 528 cycles
    - Speedup 46.8
  3. On 16x16 RMESH: 308 cycles
    - Each node ‘simulates’ a $4 \times 4$ submesh
    - Speedup 80.2

- Self simulation can gain speedup!

<table>
<thead>
<tr>
<th></th>
<th>Datasize</th>
<th>Runtime [cycles]</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microblaze</td>
<td>$2^{14}$</td>
<td>24696</td>
<td>1</td>
</tr>
<tr>
<td>64x64 R-Mesh</td>
<td>$2^{14}$</td>
<td>528</td>
<td>46.8</td>
</tr>
<tr>
<td>16x16 R-Mesh</td>
<td>$2^{14}$</td>
<td>308</td>
<td>80.2</td>
</tr>
</tbody>
</table>
Summary & Future Work

- **Summary**
  - Implementation of reconfigurable mesh algorithms on FPGA
  - Synchronization methods
  - Case study on sparse matrix multiplication
  - Scaling by self simulation

- **Ongoing work includes**
  - Improvement of the RMESH compilation flow
  - Integration of self simulation methods into the compiler
  - Analysis of our softcore array under energy aspects
Thanks for your attention!

Questions?