Algorithm Mapping on Massively Parallel and Reconfigurable Architectures

Part II: Algorithm Mapping on Reconfigurable Architectures
Motivation (1)

Today’s Complex Applications:

- dynamic at runtime
- sequential tasks
Motivation (2)

Problem:

- inefficient use of hardware resources
  - underutilized special cores (usable for one task only)
  - over-designed generic cores (inefficient HW use)
  - low performance generic cores (Soft-CPUs)

- with negative effect on:
  - System Cost: FPGA Size (Cost, Power Consumption)
  - Overall Throughput
Solution: Dynamic Reconfiguration

Adapt the hardware dynamically to the application!

• *but:*
  – Design Complexity
  – Configuration Time per Task
  – Configuration Data

• Advantage depends on careful system design!
Example: System Throughput vs. Reconfiguration Overhead

A: static solution
B: dynamic reconfiguration, max. throughput and high reconfiguration overhead
C: dynamic reconfiguration, high throughput and small reconfiguration overhead
Outline

1. Motivation and Problem Statement
2. Dynamic Reconfiguration Overhead
3. High Level Synthesis for low Overhead
4. Summary of Tools and Case Studies
Our Tool: High-Level-Synthesis for dynamic Reconfiguration

- Synthesis of C-based algorithms to reconfigurable cores
- Mixture of hardware reconfiguration and *multimode* hardware
- Solution optimized within a reconfiguration model

```
Algorithm 1
Algorithm 2
Algorithm 3
```
**Reconfiguration Model**

- Configuration: active hardware to execute algorithms
- Transition between Configurations → Difference-based Partial Reconfiguration

Reconfiguration State Graph (RSG)

Reconfiguration Costs: Differences between Configurations!
Configuration Differences

At Binary Level
- Configuration Data for the Device

At Structural Level
- Logic
  - Config. 1
  - Config. 2
- Interconnect
  - Config. 1
  - Config. 2

Data to Configure Device from Config. 1 to Config. 2
Configuration Differences - Lessons Learned

- Configuration data depend on:
  - Differences in logic
  - Differences in interconnect

  » Common structural model for all tasks to assess differences
    - Configuration-optimized circuit for each task
Virtual Architecture

- Mapping of algorithms to datapaths within a common structural model
- Direct assessment of configuration differences at **structural level**
- Optimized mapping for lower configuration differences

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Resource 1…</th>
<th>Resource N</th>
<th>Interconnect 1…</th>
<th>Interconnect M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm 1</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Algorithm 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Algorithm 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Average Configuration Cost: Slices xx, Interconnect yy
Optimization Potential

Examples:

- **ADPCM**: audio encoder/decoder
- **EDGE**: sobel filter horizontal/vertical/both
- **JPEG_DCT**: dct / idct
- **RGB_YUV**: color conversion forward/backward
Multimode Circuits

- Mapping of multiple Algorithms to a common datapath
- Mitigates reconfiguration cost by system design
High Level Synthesis Tool for Dynamic Reconfiguration

- Hardware-synthesis from C-based implementations
- Adjustable for easy system integration
- Optimization for partial dynamic reconfiguration with low overhead
- Tool support for automatic multimode hardware-synthesis
Applications and Demos

- Benchmarking with core algorithms from MediaBench
  - Evaluation of Resource and Reconfiguration Cost trade-off
- Reconfigurable video filter (Sobel operator) on a reconfigurable platform (ESM)
  - Demonstration of a working example
  - HLS of multimode hardware modules
- Region of interest detection:
  - Realtime detection in hardware
  - Postprocessing in Software
  - Multithreaded Hardware/Software integration with Petalinux and ReConOS on ESM