A High-Performance Overlay Architecture for Pipelined Execution of Data Flow Graphs

Davor Capalija and Tarek Abdelrahman

FPL 2013
Porto, Portugal
FPGAs as accelerators

• Heterogeneous high-performance computing
  – Hardware is available

• Programmability wall
  – difficulty of hardware design
  – low level of abstraction
  – long compile times

• Need solutions
  – to compete with GPUs and DSPs
  – traditional solution: high-level synthesis
Overlays

• Alternative solution: overlay architecture
• **Programmable pre-synthesized FPGA circuit**
  – aka virtual fabric, programmable high-level layer
  – classic example: soft processor
• High-level software abstraction
• Short compile-to and reconfiguration time
  – fast development cycles
  – a library of pre-synthesized overlays
• Key challenge:
  – high-performance overlay architecture
Our approach

- Programming model:  
  - data flow graphs (DFGs)

- Overlay architecture:  
  - Mesh-of-FUs for pipelined DFG execution

- Mapping:  
  - DFG-to-overlay place-and-route tool

- Distributed and modular design based on elastic pipelines
Proof of concept

- Designed two example overlays
  - integer
    - 355 MHz
    - 24x16 mesh
  - floating-point
    - 312 MHz
    - 18x16 mesh

- DFGs from 7 real applications
  - each maps under 6 seconds
  - integer DFGs achieve up to 35 GOPS
  - floating-point DFGs achieve up to 22 GFLOPS
Outline

- DFGs
- Elastic mesh-of-FUs architecture
- DFG-to-overlay mapping algorithm
- Experimental evaluation
- Related Work
- Conclusions
- Future work
Data flow graphs (DFGs)

- Abstraction for expressing parallelism
  - Streaming, DSP, HLS
- DFGs of data-parallel kernels
  - OpenCL and parallel loops

DFG instances operate on independent data
Mesh-of-FUs overlay architecture

Overlay cell

 FU

programmable coarse-grain FU (e.g., 32-bits)

heterogeneous functional layout

overlay I/O
DFG-to-overlay mapping

- Spatial mapping
  - Compute nodes to FUs
  - I/O nodes to overlay I/O
  - Edges to programmable routing
Pipelined DFG execution

- Accelerate the kernel/DFG by:
  - maximizing the throughput of pipelined execution of DFG instances
- Example: 1 DFG instance (5 ops) per cycle
- Higher $f_{\text{MAX}} \rightarrow$ higher throughput

DFG instances operate on independent data
Mesh-of-FUs overlay challenges

- (1) How to schedule 100's of FUs
  - data-driven execution: dynamic and distributed
  - elastic pipelines built with elastic buffers (EBs)
Mesh-of-FUs overlay challenges

- (2) Latency balancing of overlay pipelines
  - deeper elastic buffers → 32-deep FIFOs
  - balancing place-and-route algorithm
Mesh-of-FUs overlay challenges

- (3) Scalability to 100's of FUs
  - modular design – self-contained overlay cell
- (4) Flexibility – map various DFG topologies
  - 4-NN overlay topology
  - overlay cells with full connectivity and fan-out
  - FU bypass
Overlay cell architecture

- Use EBs to design a self-contained overlay cell
  - modular overlay
- Sufficiently deep EBs for balancing
- EBs can be expensive in FPGA fabric
  - tailor EBs to FPGAs
  - mix with regular inelastic pipeline
  - FU inelastic

- Data-driven pipeline units (DDPUs)
  - D-EB (2-deep)
  - D-FIFO (32-deep)
  - D-REG
Overlay cell routing connectivity

FU-bypass routing connections

FU-feeding routing connections

• Routing-only cells to increase routability
  – overlay cell without FUs
DFG-to-overlay mapping

- Goal: latency balance the DFG as much as possible
  - achieve higher throughput
- Based on a place and route algorithm similar to FPGA PAR
- The problem is simplified:
  - fixed overlay $f_{MAX}$
  - smaller problem size - 100's of DFG nodes
  - latency margins of deeper elastic buffers (e.g., 32-deep)
DFG-to-overlay mapping

1: `procedure DFG_PAR(start_seed)`
2: `current_seed = start_seed`
3: `balanced = False`
4: `while not balanced do`  ▷ Balancing loop
5: `routable = False`
6: `while not routable do`  ▷ PAR loop
7: `SA_placer(current_seed)`
8: `routable = pathfinder_router()`
9: `current_seed = next_seed(current_seed)`
10: `end while`
11: `balanced = check_balanced()`
12: `end while`
13: `end procedure`
Experimental evaluation

- DE4 board with Altera Stratix IV
- Benchmark DFGs
- Two example overlay instances
- Metrics
  - $f_{\text{MAX}}$ and scalability
  - DFG throughput
  - DFG-to-overlay mapping time
  - Overlay resource usage and overhead
# Benchmarks DFGs

<table>
<thead>
<tr>
<th>DFG</th>
<th>Data type</th>
<th># compute nodes</th>
<th># in / out nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-Body</td>
<td>float 32b</td>
<td>20</td>
<td>11 / 3</td>
</tr>
<tr>
<td>N-Body-2x</td>
<td>float 32b</td>
<td>40</td>
<td>15 / 3</td>
</tr>
<tr>
<td>N-Body-3x</td>
<td>float 32b</td>
<td>60</td>
<td>26 / 6</td>
</tr>
<tr>
<td>BlackScholes</td>
<td>float 32b</td>
<td>68</td>
<td>19 / 2</td>
</tr>
<tr>
<td>MatMul</td>
<td>float 32b</td>
<td>63</td>
<td>24 / 9</td>
</tr>
<tr>
<td>MatMulandAdd</td>
<td>float 32b</td>
<td>72</td>
<td>33 / 9</td>
</tr>
<tr>
<td>RGB2YIQ</td>
<td>int 32b</td>
<td>21</td>
<td>14 / 3</td>
</tr>
<tr>
<td>RGB2YIQ-2x</td>
<td>int 32b</td>
<td>42</td>
<td>17 / 6</td>
</tr>
<tr>
<td>RGB2YIQ-4x</td>
<td>int 32b</td>
<td>84</td>
<td>34 / 12</td>
</tr>
<tr>
<td>SAD</td>
<td>int 32b</td>
<td>47</td>
<td>32 / 1</td>
</tr>
<tr>
<td>SAD-2x</td>
<td>int 32b</td>
<td>94</td>
<td>36 / 2</td>
</tr>
<tr>
<td>GaussianBlur</td>
<td>int 32b</td>
<td>49</td>
<td>31 / 1</td>
</tr>
<tr>
<td>GaussianBlur-2x</td>
<td>int 32b</td>
<td>98</td>
<td>36 / 2</td>
</tr>
<tr>
<td>MatMul</td>
<td>int 32b</td>
<td>63</td>
<td>24 / 9</td>
</tr>
<tr>
<td>MatMulandAdd</td>
<td>int 32b</td>
<td>72</td>
<td>33 / 9</td>
</tr>
</tbody>
</table>
Example overlays

• OV-i: 32-bit integer
  - 24 x 16 mesh
  - total cells: 384
  - FUs: 170
  - 3 x 2 patterns
  - 32-deep FIFO EBs (D-FIFO)
  - ideal overlay I/O bandwidth
Example overlays

- OV-f: single-precision floating point
  - 18 x 16 mesh
  - total cells: 288
  - FUs: 104
  - 3 x 2 patterns
  - 64-deep FIFO EBs (D-FIFOs)
  - ideal overlay I/O bandwidth
\( f_{\text{MAX}} \) and scalability

- 32-bit integer overlay
  - 24 x 16 \( \rightarrow \) 355 MHz
  - 5 x 4 \( \rightarrow \) 380 MHz (for comparison)
  - only 7\% \( F_{\text{max}} \) drop

- Floating-point overlay
  - 18 x 16 \( \rightarrow \) 312 MHz
  - 5 x 4 \( \rightarrow \) 312 MHz (for comparison)
  - no \( F_{\text{max}} \) drop
## DFG throughput

<table>
<thead>
<tr>
<th>DFG</th>
<th>Overlay</th>
<th>DFG throughput</th>
<th>GFLOPS/GOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-Body</td>
<td>OV-f</td>
<td>1.00</td>
<td>6.24</td>
</tr>
<tr>
<td>N-Body-2x</td>
<td>OV-f</td>
<td>1.00</td>
<td>12.48</td>
</tr>
<tr>
<td>N-Body-3x</td>
<td>OV-f</td>
<td>1.00</td>
<td>18.72</td>
</tr>
<tr>
<td>BlackScholes</td>
<td>OV-f</td>
<td>1.00</td>
<td>21.22</td>
</tr>
<tr>
<td>MatMul</td>
<td>OV-f</td>
<td>1.00</td>
<td>19.66</td>
</tr>
<tr>
<td>MatMulandAdd</td>
<td>OV-f</td>
<td>1.00</td>
<td>22.46</td>
</tr>
<tr>
<td>RGB2YIQ</td>
<td>OV-i</td>
<td>1.00</td>
<td>7.46</td>
</tr>
<tr>
<td>RGB2YIQ-2x</td>
<td>OV-i</td>
<td>1.00</td>
<td>14.91</td>
</tr>
<tr>
<td>RGB2YIQ-4x</td>
<td>OV-i</td>
<td>1.00</td>
<td>29.82</td>
</tr>
<tr>
<td>SAD</td>
<td>OV-i</td>
<td>1.00</td>
<td>16.69</td>
</tr>
<tr>
<td>SAD-2x</td>
<td>OV-i</td>
<td>1.00</td>
<td>33.37</td>
</tr>
<tr>
<td>GaussianBlur</td>
<td>OV-i</td>
<td>1.00</td>
<td>17.4</td>
</tr>
<tr>
<td>GaussianBlur-2x</td>
<td>OV-i</td>
<td>1.00</td>
<td>34.79</td>
</tr>
<tr>
<td>MatMul</td>
<td>OV-i</td>
<td>1.00</td>
<td>22.37</td>
</tr>
<tr>
<td>MatMulandAdd</td>
<td>OV-i</td>
<td>1.00</td>
<td>25.56</td>
</tr>
</tbody>
</table>
## PAR compile time

<table>
<thead>
<tr>
<th>DFG</th>
<th>Overlay</th>
<th># PAR loop iter. (avg.)</th>
<th># balancing loop iter. (avg.)</th>
<th>total compile time (sec, avg.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-Body</td>
<td>OV-f</td>
<td>1.00</td>
<td>1.32</td>
<td>0.11</td>
</tr>
<tr>
<td>N-Body-2x</td>
<td>OV-f</td>
<td>1.00</td>
<td>1.19</td>
<td>0.24</td>
</tr>
<tr>
<td>N-Body-3x</td>
<td>OV-f</td>
<td>1.00</td>
<td>1.19</td>
<td>0.44</td>
</tr>
<tr>
<td>BlackScholes</td>
<td>OV-f</td>
<td>1.13</td>
<td>2.86</td>
<td>1.33</td>
</tr>
<tr>
<td>MatMul</td>
<td>OV-f</td>
<td>2.68</td>
<td>1.25</td>
<td>1.05</td>
</tr>
<tr>
<td>MatMulandAdd</td>
<td>OV-f</td>
<td>5.53</td>
<td>1.73</td>
<td>3.80</td>
</tr>
<tr>
<td>RGB2YIQ</td>
<td>OV-i</td>
<td>1.00</td>
<td>1.00</td>
<td>0.07</td>
</tr>
<tr>
<td>RGB2YIQ-2x</td>
<td>OV-i</td>
<td>1.00</td>
<td>1.00</td>
<td>0.15</td>
</tr>
<tr>
<td>RGB2YIQ-4x</td>
<td>OV-i</td>
<td>2.10</td>
<td>1.00</td>
<td>0.91</td>
</tr>
<tr>
<td>SAD</td>
<td>OV-i</td>
<td>1.00</td>
<td>1.00</td>
<td>0.17</td>
</tr>
<tr>
<td>SAD-2x</td>
<td>OV-i</td>
<td>1.00</td>
<td>1.00</td>
<td>0.41</td>
</tr>
<tr>
<td>GaussianBlur</td>
<td>OV-i</td>
<td>1.00</td>
<td>1.00</td>
<td>0.19</td>
</tr>
<tr>
<td>GaussianBlur-2x</td>
<td>OV-i</td>
<td>10.26</td>
<td>1.00</td>
<td>5.33</td>
</tr>
<tr>
<td>MatMul</td>
<td>OV-i</td>
<td>3.98</td>
<td>1.72</td>
<td>2.11</td>
</tr>
<tr>
<td>MatMulandAdd</td>
<td>OV-i</td>
<td>4.20</td>
<td>1.46</td>
<td>2.32</td>
</tr>
</tbody>
</table>
Resource usage

• Measured in adaptive logic modules (ALMs)
  – Adaptive LUT and two FFs
• Each overlay uses 75% of total device ALMs

• Breakdown:
  – FUs
    • int 32b: 90 ALMs
    • float 32b: 453 ALMs
  – DDPUs, routing and synchronization
    • compute cell: 440 ALMs
    • routing only cell: 355 ALMs
Resource overhead

• Multi-dimensional problem
  – design time
  – compile time
  – performance
  – resource cost

• Overlay cost over FUs-only:
  – OV-i → 10.6x
  – OV-f → 3.4x
Conclusions

• FPGA as accelerators
  – programmability wall
  – overlays as a solution

• Elastic mesh-of-FUs overlay
  – pipelined DFG execution
  – elastic pipelines, modular design
  – place and route algorithm with balancing

• Experimental evaluation
  – high-performance
  – fast compile to the overlay
Future work

● Measure resource overhead of using an overlay
  – Compare to hand-crafted HDL and HLS
  – Compare different overlays against each other

● Reducing resource overhead

● Memory subsystem(s)

● Automated design and compilation of overlays
Thank you

• Questions and feedback