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

Davor Capalija and Tarek Abdelrahman

FPGA Seminar, June 2013
FPGAs as accelerators

- Heterogeneous high-performance computing
  - Hardware is available!
  - Nallatech, Pico Computing, Convey, Maxeler

- Programmability wall
  - difficulty of hardware design
  - low level of abstraction
  - long compile times
    - slow development cycle

- Need solutions
  - to compete with GPUs and DSPs
High-level synthesis

- Traditional solution
- High-level language into LUTs
- Decades of research
- Recent success:
  - Xilinx Vivado HLS, Altera OpenCL, UofT LegUp
- Long compile times
  - FPGA CAD is at the back-end of tools
Overlays

• Alternative solution

• **Programmable pre-synthesized FPGA circuits**
  – classic example: a soft processor

• High-level software abstraction

• Short compile-to and reconfiguration time
  – overlays are pre-synthesized ahead of time
  – fast development cycles
Overlay research challenge

- High-performance FPGA-efficient overlay
  - expose massive FPGA parallelism
  - achieve high Fmax
  - scale to a large size
  - ability to customize for an application domain
  - low resource cost
Overlay design issues

• Which programming model to use?
• What should the overlay architecture be?
• Does an architecture have the potential for an efficient FPGA implementation?

• Potential answers?
  – vector, VLIW, many-cores, mesh-of-FUs
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
- Overlay synthesis: Bottom-up tile-based methodology

- Designed two example overlay instances
  - 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
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 (OC)

programmable routing

heterogeneous functional layout

overlay I/O

mesh of FU overlay architecture

Overlay cell (OC)

programmable routing

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
- Execution latency of a single DFG instance is not optimized

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

- **FU execution scheduling**
  - 100's of FUs
  - static (compile-time) vs. dynamic (run-time)
  - control logic: centralized vs. distributed
FU execution scheduling

- Data-driven execution
  - Dynamic and distributed scheduling
- Elastic pipelines
  - *elastic buffers (EBs)* instead of registers between stages
  - EBs are locally synchronized, no global control
  - all synchronization signals are pipelined
Mesh-of-FUs architecture challenges

- **Pipeline latency balancing**
  - Non-linear pipeline (branches and joins)
  - Unequal latencies of two joining paths cause stalls
  - Reduces FU throughput
Pipeline latency balancing

- Elastic pipelines can tolerate pipeline latency imbalance
- Deep EB can hold a variable number of data items
  - Two paths are balanced by “stretching” the shorter one
Mesh-of-FUs architecture challenges

• Scalable overlay architecture
  – 100's to 1000's of FUs
Scalable overlay architecture

- Use EBs to design a self-contained overlay cell
  - modular overlay
- 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
  - D-BUF
  - D-FIFO
  - D-REG
Mesh-of-FUs architecture challenges

- Mapping various DFG topologies
  - reductions
  - crossovers
  - diverge-converge
Mapping various DFG topologies

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 $f_{\text{MAX}}$
  • smaller problem size - 100's of DFG nodes
  • latency margins of deep elastic buffers
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
Realizing the overlay on the FPGA

- Push-button flat synthesis
  - up to 3x Fmax drop for large overlay compared to small overlays

- Assist the PAR of the FPGA CAD tool
  - Match functional overlay layout to FPGA layout
  - Physical overlay floorplanning

- Leads to modular bottom-up overlay synthesis
  - enabled by modular overlay architecture
Realizing the overlay on the FPGA
Realizing the overlay on the FPGA
Realizing the overlay on the FPGA
Realizing the overlay on the FPGA

- Overlay cell based physical floorplanning
  - achieves high Fmax
  - resource-inefficient

10 x 8 overlay cell-based floorplan
Realizing the overlay on the FPGA

- Tile-based physical floorplanning
  - tile: a unit of overlay floorplanning
  - flat synthesis of individual tiles

<table>
<thead>
<tr>
<th>5 x 4 tile</th>
<th>5 x 4 tile</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 x 4 tile</td>
<td>5 x 4 tile</td>
</tr>
</tbody>
</table>

10 x 8 overlay tile-based floorplan

Overlay cells
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 overlay instances

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

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>S</th>
<th>M</th>
<th>S</th>
<th>S</th>
<th>M</th>
<th>A</th>
<th>S</th>
<th>S</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>S</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>S</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>S</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>M</td>
<td>S</td>
<td>S</td>
<td>M</td>
<td>A</td>
<td>S</td>
<td>S</td>
<td>M</td>
<td></td>
</tr>
</tbody>
</table>

- **M** mult
- **A** addsub-abs
- **S** shift
- **routing-only cell**
Example overlay instances

- OV-f: single-precision floating point
  - 18 x 16 mesh
  - total cells: 288
  - FUs: 104
  - 64-deep FIFO EBs (D-FIFOs)
  - 3 x 2 patterns

- ideal overlay I/O bandwidth
\( f_{\text{MAX}} \) and scalability

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

- Floating-point overlay
  - \( 18 \times 16 \rightarrow 312 \text{ MHz} \)
  - \( 5 \times 4 \rightarrow 312 \text{ MHz (for comparison)} \)
  - no Fmax 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><strong>22.46</strong></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><strong>34.79</strong></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

• Independent data is hard to come by

• Overlay cost over FUs-only:
  – OV-i → 10.6x
  – OV-f → 3.4x
Research directions (immediate)

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

- Reducing resource overhead

- Memory system and interfacing with overlay I/O
Research directions (near future)

- Overlay instance design
  - Automated synthesis of large overlays instances
- Overlay architecture
  - Full support for predicated control flow
  - Time-multiplexing of large DFGs on the overlay
  - Increase routability
- DFG-to-overlay mapping
  - Better PAR heuristics for latency balancing and routing congestion
Research directions (long term)

• Map OpenCL applications to the overlay
  – Full tool flow, OpenCL front-end
  – Further architectural extensions to the overlay

• Library of application-domain specialized overlay instances

• Looking forward to Stratix 10
  – 4M LUTs → 1000's of FUs
Thank you

• Questions and feedback