Sharing, Protection, and Compatibility for Reconfigurable Fabric with *AmorphOS*

AHMED KHAWAJA\(^1\), JOSHUA LANDGRAF\(^1\), ROHITH PRAKASH\(^1\), MICHAEL WEI\(^2\), ERIC SCHKUFZA\(^2\), CHRISTOPHER J. ROSSBACH\(^1,2\)

\(^1\)UT AUSTIN

\(^2\)VMWARE RESEARCH GROUP
Motivation

Bigger, faster FPGAs deployed in the cloud
- Microsoft Catapult/Azure
- Amazon F1

FPGAs: Reconfigurable Accelerators
- ASIC Prototyping, Video & Image Proc., DNN, Blockchain
- Potential solution to *accelerator provisioning challenge*

Our position: FPGAs will be shared
- Sharing requires protection
- Abstraction layers provide compatibility
- Beneficiary: provider → consolidation
FPGA Background

Field Programmable Gate Array (FPGA)
- Reconfigurable interconnect → custom data paths
- FPGAs attached as coprocessors to a CPU

FPGA Build Cycle
- Synthesis: HDL → Netlist (~seconds)
- Place and Route: Netlist → Bitstream (~min--hours)
- Reconfiguration/Partial Reconfiguration (PR)

Production systems: No multi-tenancy

Emerging/Research Systems use fixed slots/PR
- Fixed-sized slots → fragmentation (50% or more)
- Elastic resource management needed
AmorphOS Goals

Protected Sharing/Isolation
- Mutually distrustful applications

Compatibility / Portability
- HDL written to AmorphOS interfaces
- **14 benchmarks run unchanged on Microsoft Catapult and Amazon F1**

Elasticity
- User logic scales with resource availability
- Sharing density scales with availability
AmorphOS Abstractions

- **Zone**: Allocatable Unit of Fabric
  - 1 Global zone
  - N dynamically sized, sub-dividable PR zones

- **Hull**: OS/Protection Layer
  - Memory Protection, I/O Mediation

- **Morphlet**: Protection Domain
  - Extends Process abstraction
  - Encapsulate user logic **on PR or global zone**

- **Registry**: Bitstream Cache
  - Hides latency of place-and-route (PaR)

<table>
<thead>
<tr>
<th>Morphlets</th>
<th>Bitstream</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;A,B&gt;</td>
<td>0x0a1...</td>
</tr>
<tr>
<td>&lt;A,B,C&gt;</td>
<td>0x0fb01...</td>
</tr>
<tr>
<td>&lt;B,C&gt;</td>
<td>0x11ad...</td>
</tr>
</tbody>
</table>
Scheduling Morphlets

Tradeoff
- Fixed zones + PR $\rightarrow$ fast, fragmentation
- Global zone + PaR $\rightarrow$ eliminates fragmentation, slow

AmorphOS: best of both worlds
- Low Latency Mode
  - Fixed zones + PR
  - Default Morphlet bitstream
- High Throughput Mode
  - Combine multiple Morphlets
  - Co-schedule on a global zone
Scheduling Case Study

Low-Latency Mode

Host
DRAM
App A
Morphlet A
FPGA Fabric
App B

High-Throughput Mode

Host
DRAM
App A
Morphlet A
App B
Morphlet B

Low-Latency Mode

Host
DRAM
App A
Morphlet A
App B
Morphlet B

High-Throughput Mode

Host
App A
Morphlet A
App B
Morphlet B
App C
Morphlet C
App D
Morphlet D
DRAM

T₀
T₁
T₂
T₃

Low-Latency Mode
High-Throughput Mode
AmorphOS Hull

Hardens and extends vendor Shells
- Microsoft Catapult
- Amazon F1

AmorphOS Interfaces
- Control: \textit{CntrlReg}
- Virtual Memory: \textit{AMI}
- Bulk Data Transfer: Simple-\textit{PCIe}
AmorphOS Hull

Hardens and extends vendor Shells
- Microsoft Catapult
- Amazon F1

AmorphOS Interfaces
- Control: \textit{CntrlReg}
- Virtual Memory: \textit{AMI}
- Bulk Data Transfer: Simple-\textit{PCIe}

Multiplexing of interfaces
- Isolation/data protection
- Scalable, 32 accelerators
  - Tree of multiplexers
Implementation & Methodology

Catapult Prototype
- Altera Mt. Granite Stratix V GS 2x4GB DDR3, Windows Server
- Segment-based protection, **partial reconfiguration (PR)**

Amazon F1 Prototype
- Xilinx UltraScale+ VU9P, 4x16GB GDDR4, CentOS 7.5
- **No PR, but much more fabric than Catapult**

Workloads
- DNNWeaver – **DNN inference**
- MemDrive – **Memory Bandwidth**
- Bitcoin – **blockchain hashing**
- CHStone – **11 accelerators (e.g. AES, jpeg, etc)**
Scalability

- MemDrive: 5X throughput
- DNNWeaver: 32X density
- Bitcoin: compute-bound

**Scalability**

- **F1**: Xilinx UltraScale+ VU9P, 4x16GB GDDR4, CentOS 7.5
- **Higher is better, Homogenous Morphlets**

**Takeaway**: Massive throughput/density improvement possible, awareness of contended resources necessary
Throughput

End-to-End Bitcoin Run Time

- No Sharing: serialized
- Two PR Zones: worse than no sharing due to down-scaling!
- AmorphOS (HT)

- 8 Bitcoin Morphlets
- Catapult Altera Stratix V GS 2x4GB DDR3, Windows
- Registry pre-populated: ctxt sw. 200ms
- Log Scale, Lower is better

Takeaway: Co-scheduling on a global zone can perform better than fixed-sized slots and PR
Partitioning Policies

**Single Sharing (baseline)**
- Everything runs serially
- Single context

**Global Zone**
- Multiple Morphlets
- No fixed size zones
- Single-level zone policy
- Two PR zones
- One Morphlet each

**Co-schedule**
- Multiple Morphlets in a single PR zone

**Subdivide**
- Divide top-level PR zone into smaller PR zones
Partitioning Policies

- **Bitcoin Morphlets**
- **Catapult: Altera Mt. Granite Stratix V GS 2x4GB DDR3, Windows**
- Registry pre-populated: ctxt sw. 200ms
- Higher is better

**Takeaway:**
- Hierarchical PR on limited HW not worth it
- See paper for projections on F1

**Graph:**
- Single-level partitioning better than recursive subdivision
  (cause: downscaling)
- Global zone best
  - Co-schedule on global zone
  - Single-level: only two fixed slots
  - Co-schedule: morph multiple in fixed slot
  - Subdivide: hierarchical partitions

**Bar Chart:**
- Relative Speedup
- Categories: Non-sharing, Global, Single-Level, Co-schedule, Subdivide
Related Work

Access to OS-managed resources
- Borph: So [TECS ’08, Thesis ’07]
- Leap: Adler [FPGA ’11]
- CoRAM: Chung [FPGA ’11]

First-class OS support
- HThreads: Peck [FPL’06], ReconOS: Lübbers [TECS ’09] -- extend threading to FPGA SoCs
- MURAC: Hamilton [FCCM ’14] – extend process abstraction to FPGAs

Single-application Frameworks
- Catapult: Putnam [ISCA ‘14] / Amazon F1

Fixed-slot + PR
- OpenStack support: Chen [CF ‘14], Byma [FCCM ’14]; Fahmy [CLOUDCOM ‘15];
- Disaggregated FPGAs: Weerasinghe [UIC-ATC-ScalCom ‘15]

Overlays
- Zuma: Brant [FCCM ‘12],
- Hoplite: Kapre [FPL ‘15],
- ReconOS+Zuma: [ReConfig ’14]
Conclusions & Future Work

Compatibility Improved
- without restricting programming model
- Comprehensive set of stable interfaces
- Port AmorphOS *per platform not each accelerator per platform*

Scalability achieved *within and across accelerators*
- AmorphOS transparently scales morphlets up/down
- Powerful combination of slots/Partial Reconfiguration and full FPGA bitstreams

Future work
- Transparently scale across multiple FPGAs
- Scale across more than just FPGAs
- Open source AmorphOS/port to more platforms

Questions?