

# Active SANs: Hardware Support for Integrating Computation and Communication

Craig Ulmer, Chris Wood, and Sudhakar Yalamanchili

Center for Experimental Research in Computer Systems

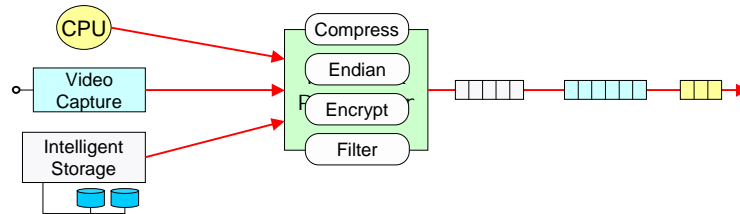


## Overview

- Motivation
- Active SANs
  - Computations on messages “in-transit”
  - FPGA-based processing
- Implementation with commodity parts
- Performance
- Conclusions

## Motivation

- SAN hardware increasingly more powerful
  - Fast network processors
  - Example: Intel IXP Network Processor
- Clusters utilize embedded devices
  - Intelligent storage, video capture
- Can network processors process application data?
  - Transformations, Compression, Cryptography



## Active SANs

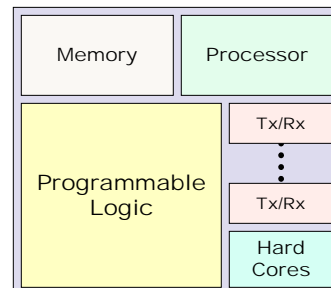
- Place processing elements in network
  - Perform computations on messages passing through network
- Benefits:
  - Offload host CPUs
  - Embedded devices in SAN to perform computations without relying on host CPU
- *Related work: Active Networks*

## Active SAN Requirements

1. Network processors operate at link speeds
2. Usable programming model
3. Customize computations to application

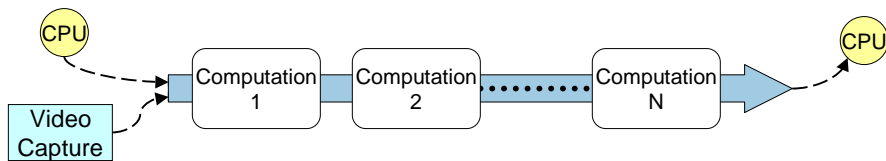
### 1. High Speed Network Processors: FPGAs

- Reconfigurable hardware to process messages
- New architecture features
  - Multiple Gbps transceivers
  - Emulate 10 million gates
  - Processor cores
  - Other dedicated hardware
- FPGA-based NI
  - Hardware computations at wire



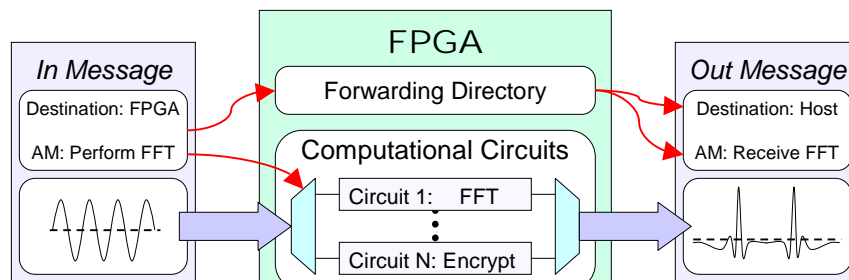
## 2. Programming Model

- Streaming computations
  - Load FPGAs with computational circuits
  - Stream data through FPGA and NI
- Extension: Route through multiple FPGAs



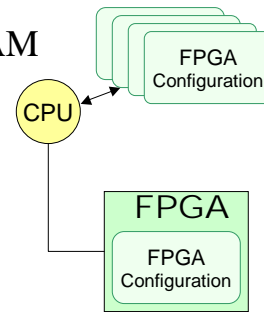
## Streaming Fundamentals

- Computation: How is a computation performed?
  - Active message approach
- Forwarding: Where are results transmitted?
  - Programmable *forwarding directory*



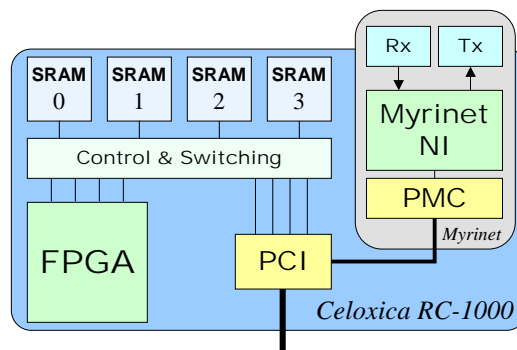
### 3. Customization

- Demand driven dynamic reconfiguration
  - Host manages FPGA configurations
- If requested circuit not loaded:
  - FPGA Stores dynamic state in SRAM
  - FPGA Generates *function fault*
  - Host loads new configuration
  - FPGA loads state, restarts



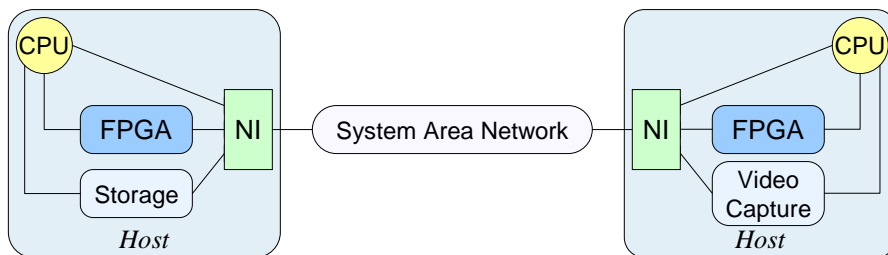
### Implementation: FPGA-based NI

- **Celoxica RC-1000**
  - Xilinx Virtex-1000 FPGA
  - 8 MB SRAM
- **Myricom Myrinet NI**
  - LANai 4.3 / 33MHz / 1MB
  - PMC form factor

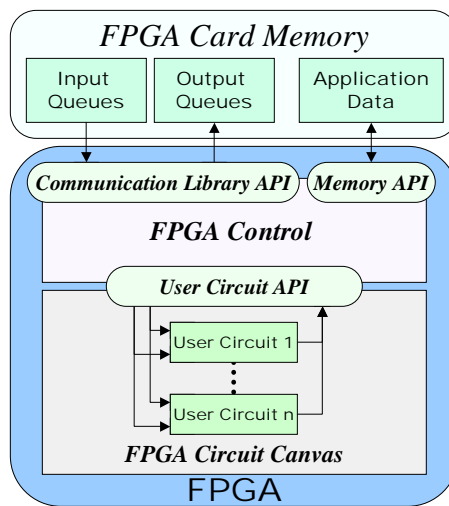


# Communication Library: GRIM

- **GRIM: General-purpose Reliable In-order Messages**
  - Supports inter-/intra- host communication
  - NI-based flow control, Logical Channels, and Active Messages

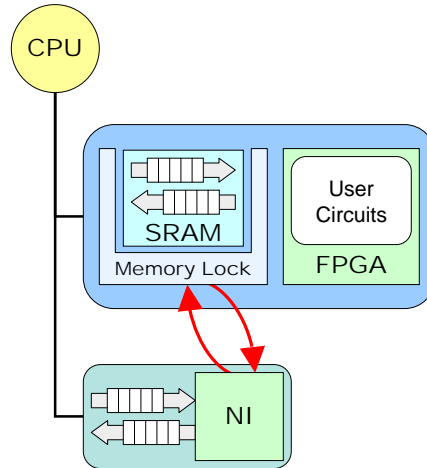


# FPGA Interfaces



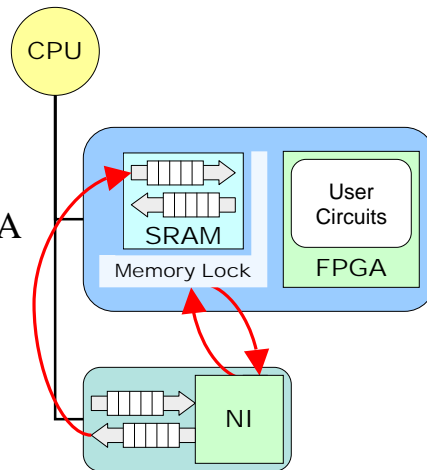
## RC-1000 Challenges

- **Hardware implementation**
  - Queue state machines
- **Memory locking**
  - SRAM single ported
  - Arbitrate for use
- **CPU / NI contention**
  - NI manages FPGA lock

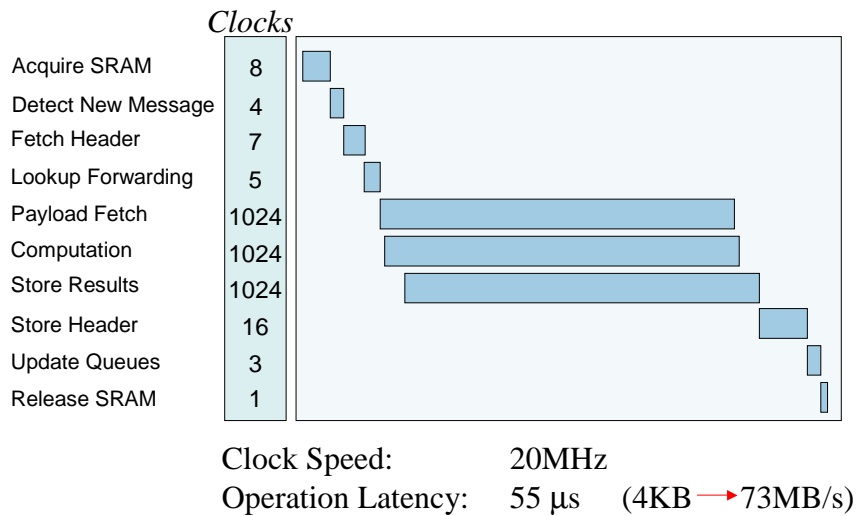


## Performance: Card Interactions

- **Acquire FPGA SRAM**
  - CPU-NI: 20  $\mu$ s
  - NI: 8  $\mu$ s
- **Inject 4 KB message to FPGA**
  - CPU: 58  $\mu$ s (70 MB/s)
  - NI: 32  $\mu$ s (128 MB/s)
- **Release FPGA SRAM**
  - CPU-NI: 8  $\mu$ s
  - NI: 5  $\mu$ s

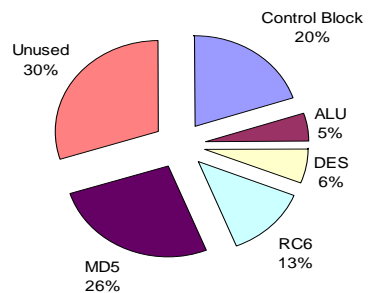


## Performance: FPGA Computations



## Performance: FPGA Configurations

- Cryptography configuration
  - DES, RC6, MD5, and ALU
- Full FPGA Reconfiguration
  - Configure Time: 90 ms
  - 700KB+ configuration
  - PIO transfers



## Implementation Observations

- Performance bottlenecks
  - FPGA memory lock: prevents overlap
  - Two PCI traversals to get to wire for FPGA-NI
  - Lengthy reconfiguration times
- Promising environment
  - FPGA able to process at PCI speeds
  - Fault mechanisms suitable for partial reconfiguration
  - Bottlenecks removed with emerging FPGAs

## Conclusions

- FPGAs useful for processing in Active SAN
  - Streaming mechanisms
- Implement with current technology
  - Nontrivial because of interfaces
  - Desire better overlap
- Future FPGAs provide single chip NI
  - Processing model applicable