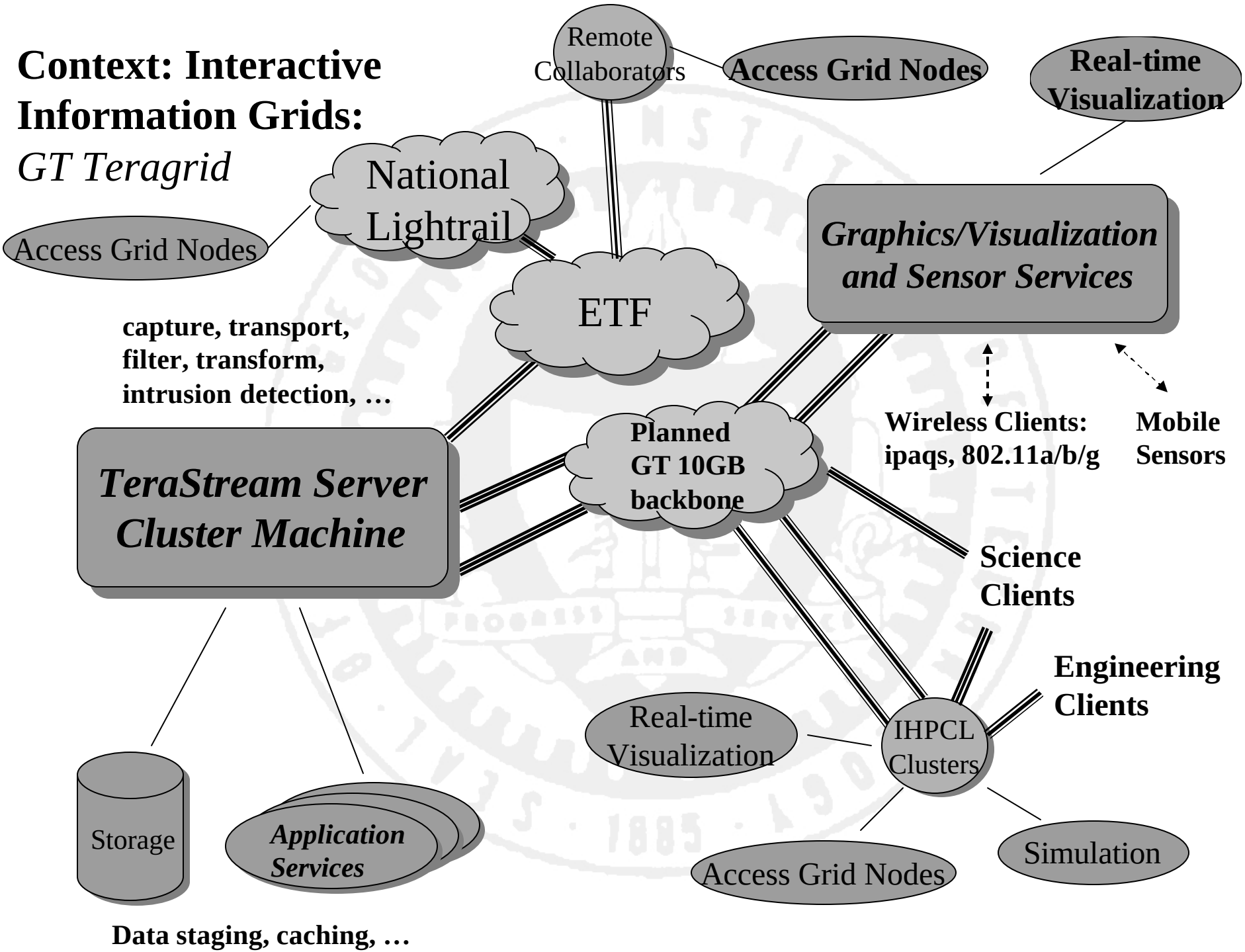
Application-level Communication Services in Edge Routers

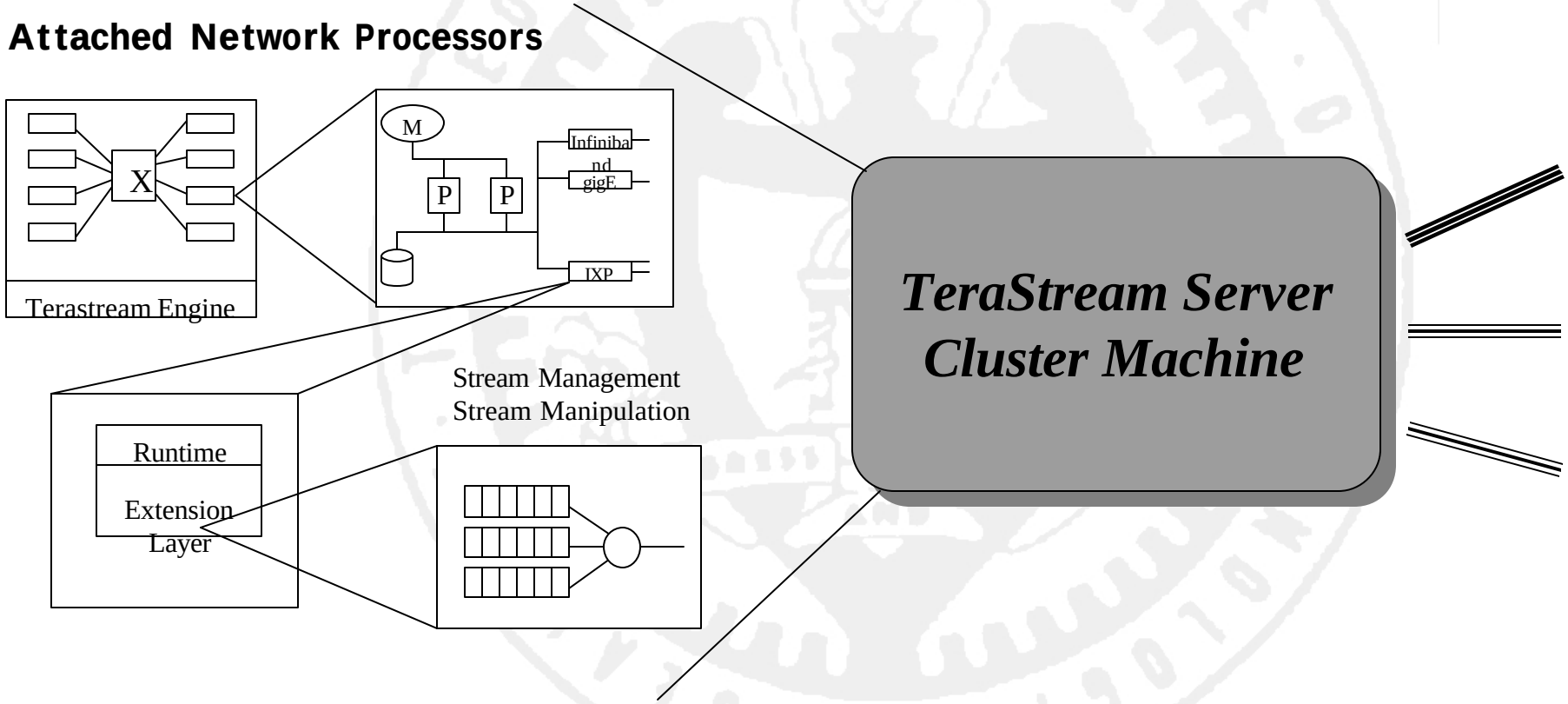
Ada Gavrilovska, Karsten Schwan,
Hailemelekot Seifu, Ola Nordstrom
www.cercs.gatech.edu/projects
W. Lee, K. Mackenzie, S. Pande, D. Schimmel and many other GT researchers

CERCS, Georgia Tech

Intel IXA Meeting, Sept. 2003


Context: Interactive Information Grids:
GT Teragrid
Remote Collaborators
Access Grid Nodes
Real-time Visualization
National Lightrail
Access Grid Nodes
ETF
Graphics/Visualization and Sensor Services
capture, transport, filter, transform, intrusion detection, …
Wireless Clients: ipaqs, 802.11a/b/g
Mobile Sensors
Planned GT 10GB backbone
TeraStream Server Cluster Machine
Science Clients
Engineering Clients
Real-time Visualization
IHPCL Clusters
Storage
Application Services
Simulation
Access Grid Nodes
Data staging, caching, …

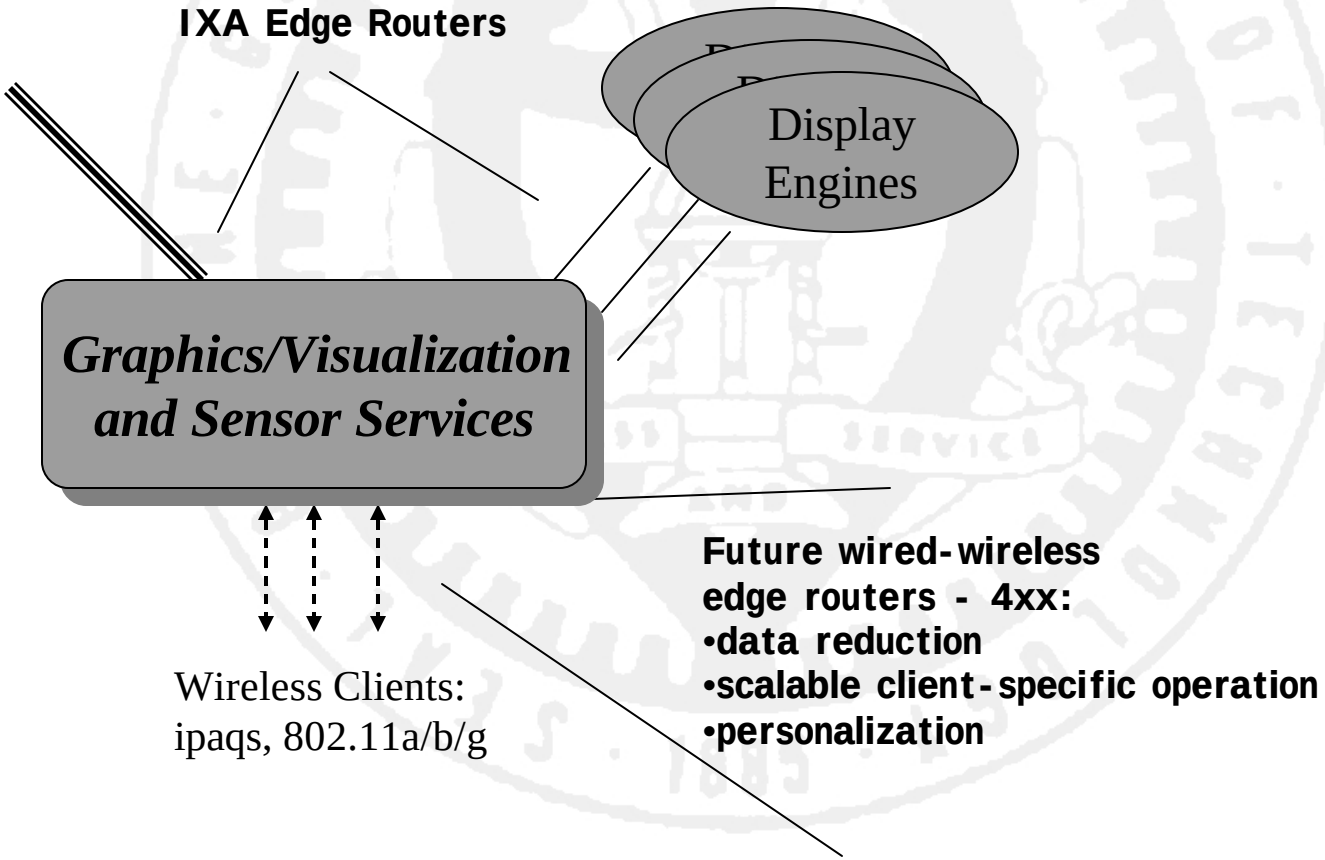
Edge Routers for Terastream Services - Cluster Machines



Examples:

- **Stream scheduling for real-time response**
- **Data mirroring for 24/7 operation**

Edge Routers for Terastream Services - Wireless Clients



Programmable Edge Routers

- Focus on Attached Network Processors (ANPs):
  - Real-time collaboration, delivering camera- or sensor-captured data, enterprise services (e.g., OIS)
  - Application-specific stream customization occurs at nodes in overlay networks mapped to suitable host/NP (ANP) pairs
- Host/ANP services address dynamically changing application needs and platform resources with application-specific stream customization:
  - Data mirroring, selection, downsampling
  - Selectively lossy data exchange and stream scheduling
  - Scalable, client-specific functionality
  - **New services**:
    - **Intrusion detection**
    - **Remote graphics**
    - **`XML' support**

Why `Push' Application Services into Network Infrastructure?

Cost/Performance

- NPs have optimized hardware:
  - Efficient access to and movement of network packets
- Services can be implemented on packets' fast path, using available headroom
  - existing work provides network-centric services: routing, network monitoring, intrusion detection, differentiated services, …
  - our research focuses on application-specific functionality

This talk: New Services:

- Remote graphics, `XML'

Technical Approach

Stream Handlers

Use Stream Handlers – computational units which implement application-level services on NPs

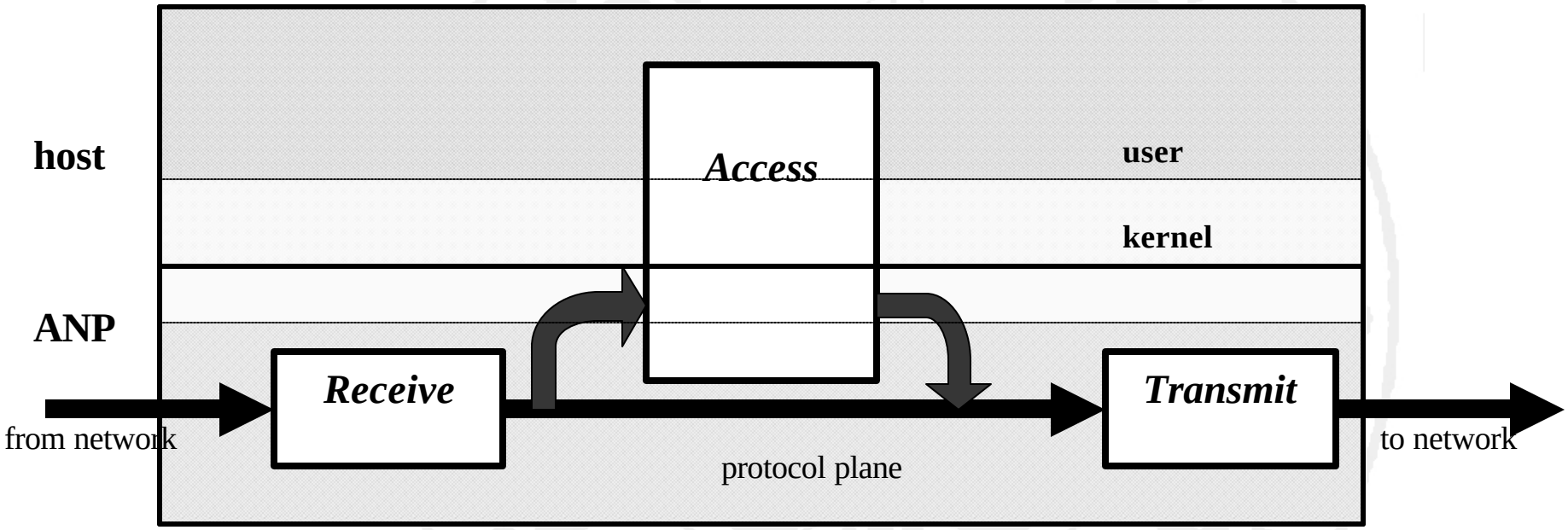
Split execution

Split execution of application-level services across stream handlers on ANPs and host kernel- or host user-level based resource needs

Dynamic configuration

Dynamically create, configure, and deploy stream handlers

`Split' Architecture

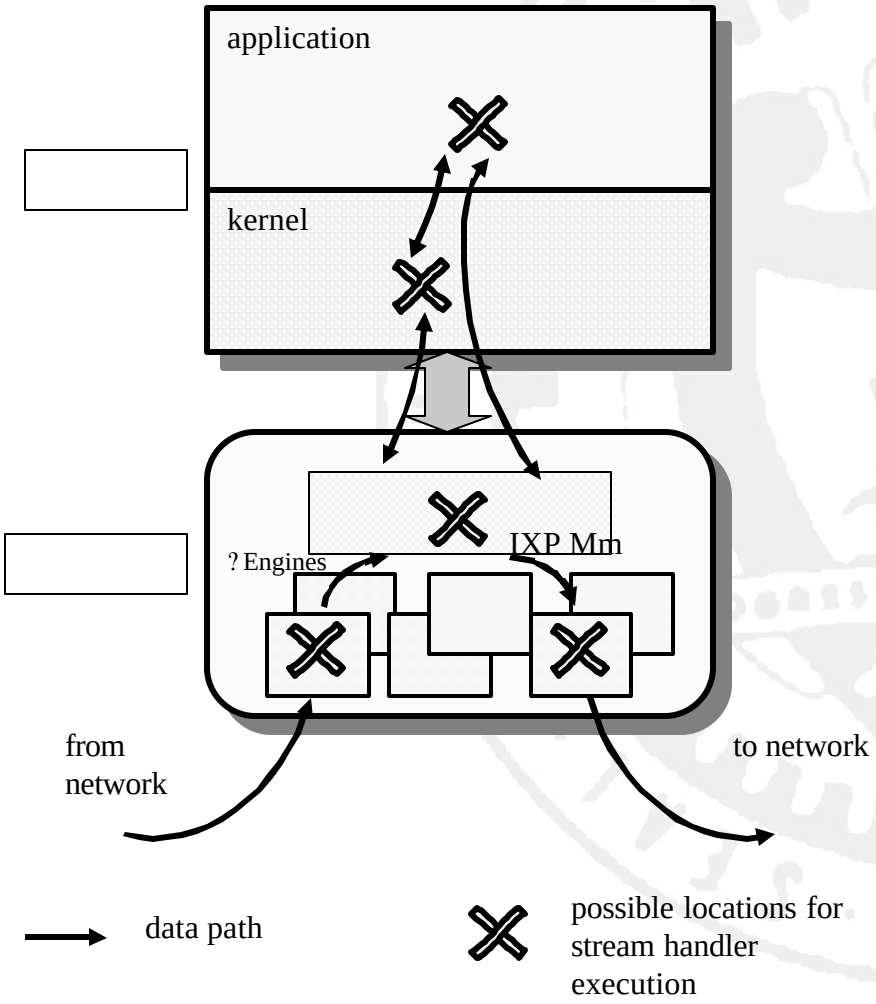


- IXP-level receive- and transmit- blocks fragment/re-assemble **application-level messages** and execute application-specific functions
- Additional functionality is implemented via data accesses at IXP or host level

IXP-level Stream Handlers

- Lightweight, composable, parameterizable, computational units, executed by the NPs; can access information 'beyond' packet headers, i.e., message headers **and** payloads
- Implementation utilizes:
  - Efficient protocol to assemble application-level data (RUDP) - Future: utilize NP-resident UDP/TCP stacks
  - Self-describing portable data formats (PBIO) that define payload structure
- Stream handler execution can be linked with host-based kernel or user actions

`Split' Operation



- IXP-side:
  - At protocol receive- or transmit-side, or in IXP memory
  - Using limited IXP resources
- Host-side:
  - At kernel- or user-level
  - Necessary to support functionality of arbitrary complexity under varying conditions
- Compositions of handlers can implement more complex services

Experimental Evaluation

Viability:

- Low overheads of stream handler implementation in terms of latency and bandwidth - previous work

New services:

- Efficient implementations of services such as client-customized multicast

Performance benefits:

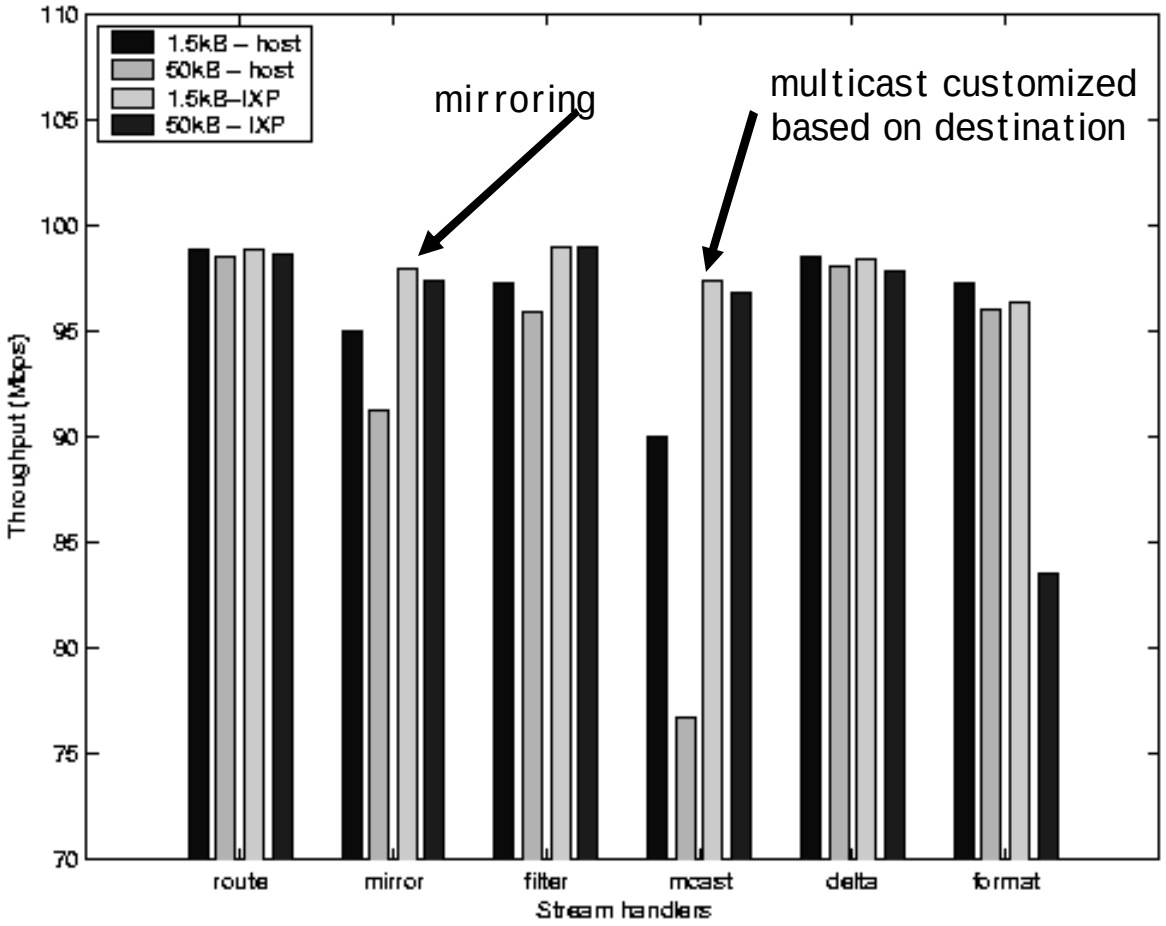
- Performance benefits include offloading the host CPUs, and load reduction on the underlying network and memory infrastructure

Performance Benefits/Viability: Improved Message Latencies

data size, u	Host-side	IXP-side
100B	32us	28us
1kB	83us	82us
1.5kB	132us	131us
10kB	896us	840us
50kB	6.8ms	4.2ms
100kB	15.4ms	8.4ms

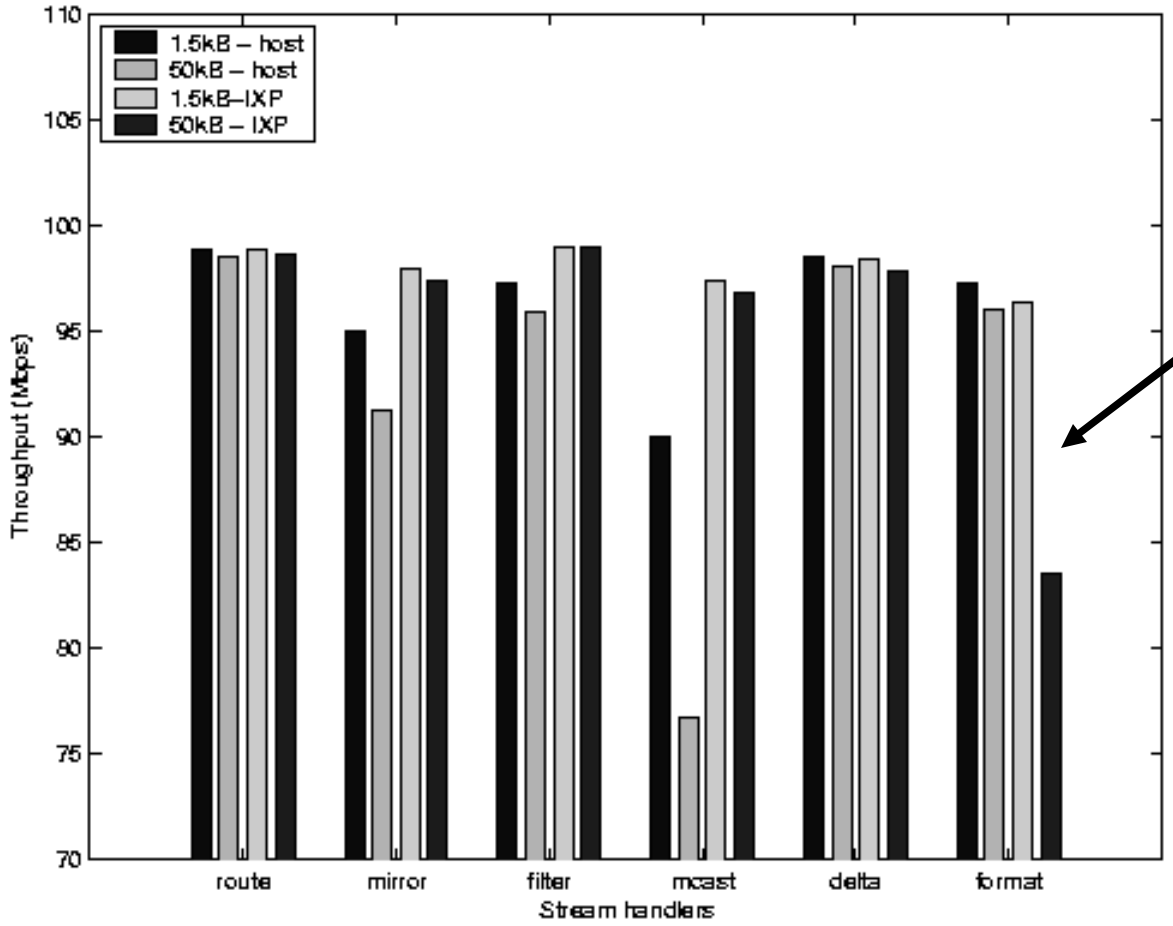
- IXP-based forwarding improves end-to-end latency:
- Comparable to host-level performance for smaller messages
- Improvements more profound as message sizes increase (i.e., consider remote visualization)

Performance Effects: Application-level Services



Mirroring & destination-specific multicast more efficient on ANP, as part of the Rx/Tx code

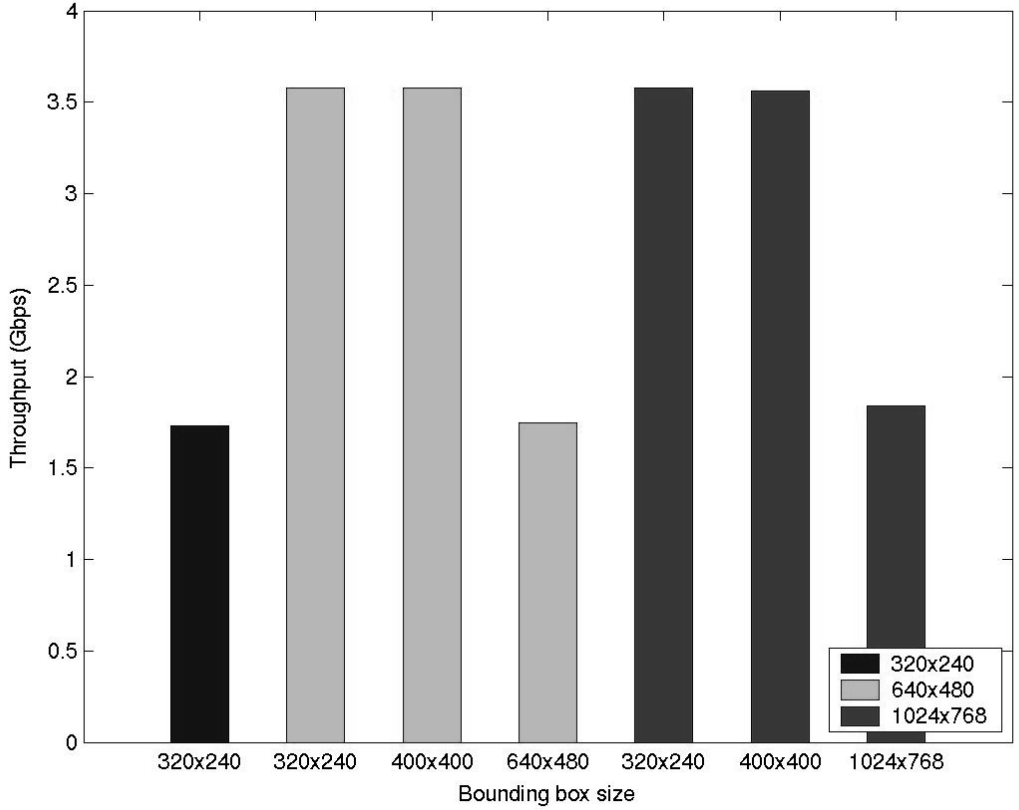
Need for 'Split' Handlers: Complex Handlers and 'Headroom'



intensive computation

- Complexity of 'format' increases with data size, available headroom is exceeded, and performance degrades
- **Need for intermediate threads/processing**

New Services: Client-specific OpenGL Image Cropping on the IXP

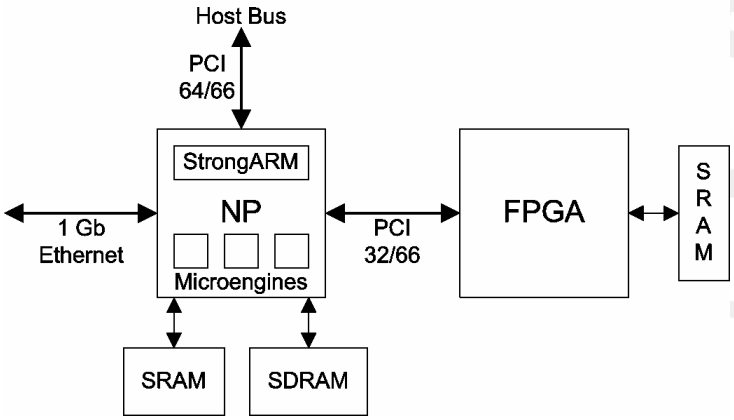


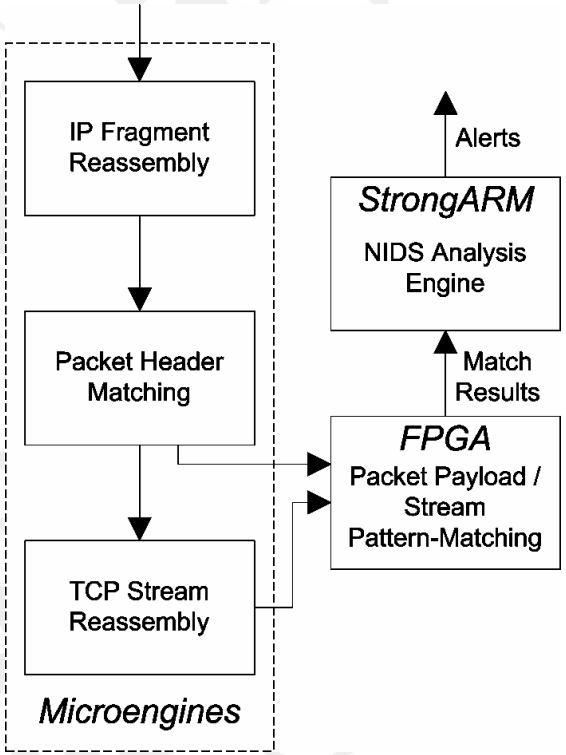
- Can perform computationally intensive tasks like image cropping efficiently
- Performance Benefits: CPU load when performed at host: 99.95%

`Split' Handlers and Additional Resources: NIDS System Design

A Layered and pipelined architecture:

- Maximize performance by assigning tasks to the most appropriate device:
  - StrongArm/Xscale: configuration, control, I/O
  - Microengines: sequential, repetitive packet processing
  - FPGA: massively concurrent processing





–Prototype system developed for 1 Gbps networks using IXP1200 and Xilinx Virtex FPGA

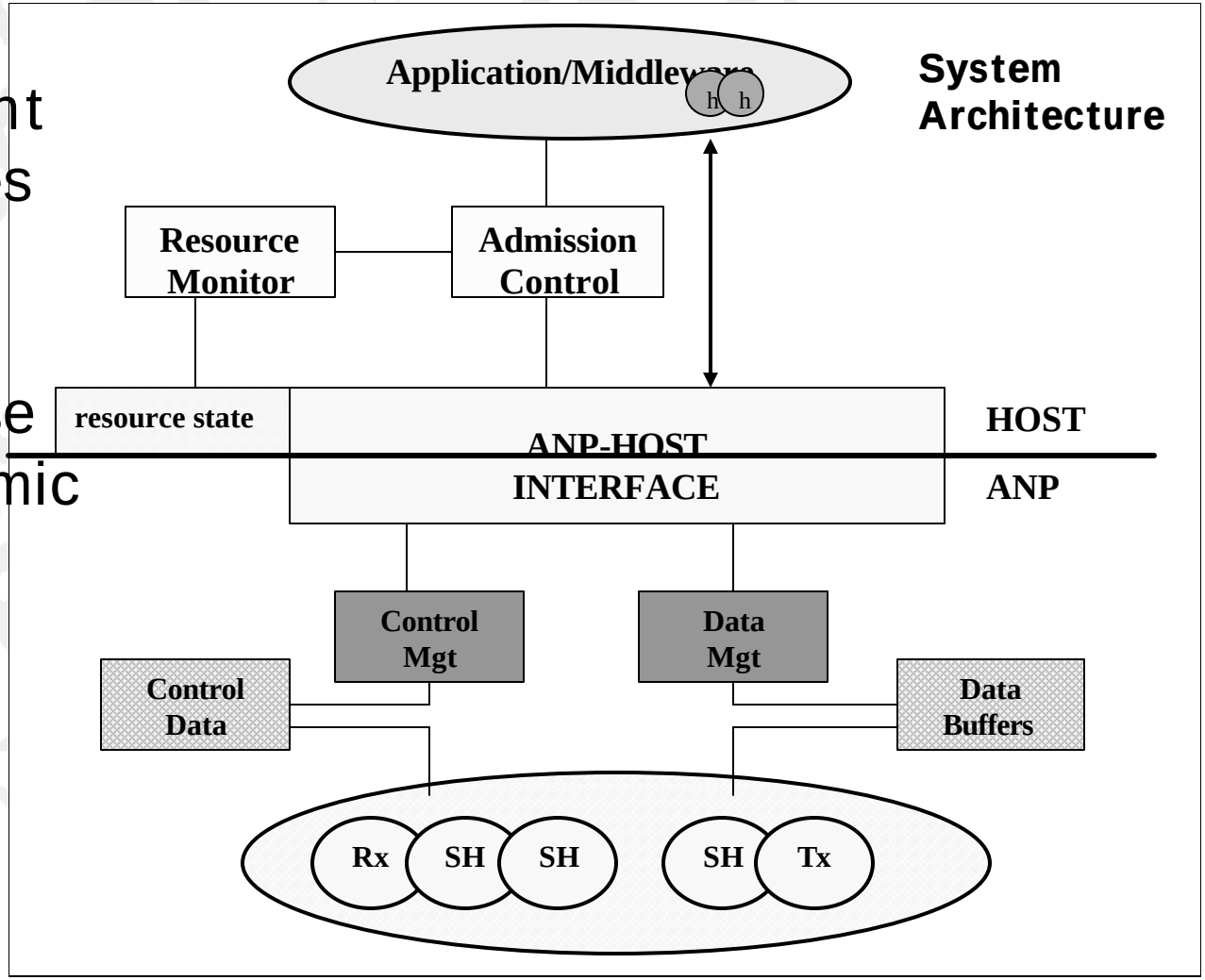
–Moving to IXP2400 and Virtex2 to support faster networks

Conclusions

- `Split' Architecture:
  - Use headroom to implement middleware- and application-level services on fast path through NPs
  - Benefit from network-near execution of stream handlers and flexible mapping across host-ANP
- Deliver new functionality and performance gains to applications while meeting network performance requirements
- Issue: `Vertical' system programming

Ongoing and Future Work

- Dynamic deployment of complex services across ANP-host boundaries.
- Focus on Enterprise Applications: dynamic XML-format interpretation and code generation.
- Admission control
- Request: host/NP proximity: beyond PCI



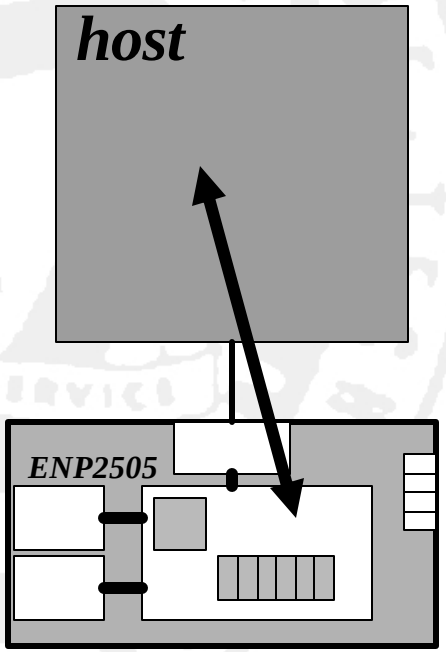
Research Overview

- `Split' Services: K. Mackenzie, K. Schwan, S. Yalamanchili
- NIDS System: D.Contis, D. Schimmel, W. Lee
- Efficient Host/ANP Intrusion Detection - W. Lee
- Automatic Register Allocation for Micro-engine Code - S. Pande

Support Tools: GT IXP Driver

kenmac@cc, austen@cc, ganev@cc

- User interfaces: 2 so far (host side)
  - faux "ethernet" interface (in-kernel)
  - DEC "CLF" message system (user)
- "Hacker's Driver" (host side)
  - exposes all ENP2505 card resources to host kernel and/or user
- Msg-over-PCI protocol (host & uEngine)
- Extensible NI (uEngine)



- *IXP2400 operational soon*

IXP Driver - Some Detail

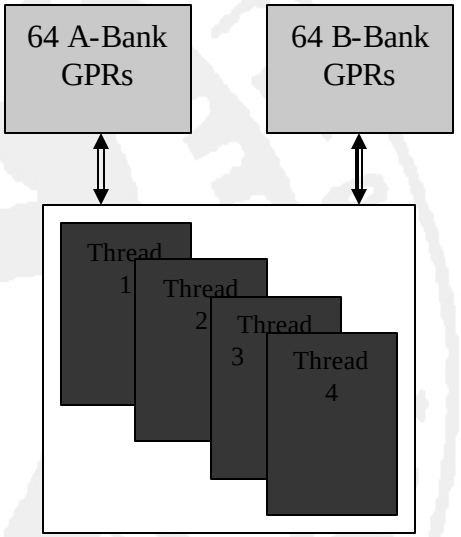
- Currently supports:
  - IXP1200 boards (Radisys ENP-2505)
  - IXP2400 boards (Radisys ENP-2611)
- Exports hardware resources to host kernel/user space code:
  - PCI bridge config/status registers
  - IXP chip config/status registers
  - IXP SDRAM
- Provides physically contiguous host SDRAM to user/kernel space code
- Integrates Intel's pciDg driver on top
  - Completed for IXP1200 boards
  - In progress for IXP2400 boards

Related Work

- Extensible network architectures
  - SPINE, VCM, WUGS/DHP, ANTS, CANEs…
  - IXP1200: Princeton Vera, Columbia Netbind, microACE, IXP as NIC…
- Composable computation
  - microprotocols, CANs, Protocol Boosters…
- Stream customization
  - publish/subscribe (Echo/Jecho, Gryphon…) and peer-to-peer (Chord, Pastry…)

Dual-bank Register Constraint

- *Dual-bank Constraint*
  - *Only for ALU instructions*
  - *Two source operands must come from different banks*
  - *Why—fetch them in parallel to achieve 1 cycle latency for all ALU instructions*




```
ALU[dest_op,source_op_a,+,source_op_b]
```

OR

```
source_op_a ⇒ Bank A, source_op_b ⇒ Bank B
source_op_a ⇒ Bank B, source_op_b ⇒ Bank A
```

Our Approaches

Problem modeling

Build *Register Conflict subGraph (RCG),* then detect and break all odd-cycles on the *RCG*.

Two observations

Breaking smaller cycles may break bigger cycles as well.

Most odd-cycles are small.

Algorithm Complexity

Brute-force algorithm takes exponential time. Based on our algorithm, in most cases, it is polynomial-time solvable.

Combine with Register Allocation

We propose 3 algorithms: *Pre-RA, Post-RA, Combined,* depending on the phase-ordering of our algorithm and the register allocation. Current results show *Post-RA* is best, but more potential improvements are possible for the *Combined* approach.