New Review Assignments

- **Were Due: Sunday, October 28, 11:59pm.**

- **Due: Tuesday, October 30, 11:59pm.**

- **Due: Thursday, November 1, 11:59pm.**
Other Readings

- **Dataflow**

- **Restricted Dataflow**
Project Milestone I Meetings

- Please come to office hours for feedback on
  - Your progress
  - Your presentation
Last Lectures

- Transactional Memory (brief)
- Interconnect wrap-up
- Project Milestone I presentations
Today

- More on Interconnects Research
- Start Dataflow
Research in Interconnects
Research Topics in Interconnects

- Plenty of topics in interconnection networks. Examples:
  
  - **Energy/power** efficient and proportional design
  - **Reducing Complexity**: Simplified router and protocol designs
  - **Adaptivity**: Ability to adapt to different access patterns
  - **QoS and performance isolation**
    - Reducing and controlling interference, admission control
  - **Co-design of NoCs with other shared resources**
    - End-to-end performance, QoS, power/energy optimization
  - **Scalable topologies** to many cores, heterogeneous systems
  - **Fault tolerance**
  - **Request prioritization, priority inversion, coherence, ...**
  - **New technologies** (optical, 3D)
Packet Scheduling

- Which packet to choose for a given output port?
  - Router needs to prioritize between competing flits
  - Which input port?
  - Which virtual channel?
  - Which application’s packet?

- Common strategies
  - Round robin across virtual channels
  - Oldest packet first (or an approximation)
  - Prioritize some virtual channels over others

- Better policies in a multi-core environment
  - Use application characteristics
Application-Aware Packet Scheduling

The Problem: Packet Scheduling

Network-on-Chip is a **critical** resource shared by multiple applications
The Problem: Packet Scheduling

- **Routers**
- **Processing Element**
  - (Cores, L2 Banks, Memory Controllers etc)

![Diagram of a network with routers and processing elements]

- **Routing Unit (RC)**
- **VC Allocator (VA)**
- **Switch Allocator (SA)**

- **Crossbar (5 x 5)**

- **Input Port with Buffers**
  - VC Identifier
    - VC 0
    - VC 1
    - VC 2
  - From East
  - From West
  - From North
  - From South
  - From PE
  - To East
  - To West
  - To North
  - To South
  - To PE

- **Control Logic**
The Problem: Packet Scheduling

Routing Unit (RU)

VC Allocator (VA)

Switch Allocator (SA)
The Problem: Packet Scheduling
The Problem: Packet Scheduling

Which packet to choose?
The Problem: Packet Scheduling

- Existing scheduling policies
  - Round Robin
  - Age
- Problem 1: Local to a router
  - Lead to contradictory decision making between routers: packets from one application may be prioritized at one router, to be delayed at next.
- Problem 2: Application oblivious
  - Treat all applications’ packets equally
  - But applications are heterogeneous
- Solution: Application-aware global scheduling policies.
Motivation: Stall Time Criticality

- Applications are not homogenous

- Applications have different criticality with respect to the network
  - Some applications are network latency sensitive
  - Some applications are network latency tolerant

- Application’s Stall Time Criticality (STC) can be measured by its average network stall time per packet (i.e. NST/packet)
  - Network Stall Time (NST) is number of cycles the processor stalls waiting for network transactions to complete
Motivation: Stall Time Criticality

- Why do applications have different network stall time criticality (STC)?
  - Memory Level Parallelism (MLP)
    - Lower MLP leads to higher STC
  - Shortest Job First Principle (SJF)
    - Lower network load leads to higher STC
  - Average Memory Access Time
    - Higher memory access time leads to higher STC
STC Principle 1 {MLP}

- Observation 1: **Packet Latency != Network Stall Time**
Observation 1: *Packet Latency != Network Stall Time*

Observation 2: A low MLP application’s packets have higher criticality than a high MLP application’s
STC Principle 2 {Shortest-Job-First}

Running ALONE

Baseline (RR) Scheduling

SJF Scheduling

Overall system throughput {weighted speedup} increases by 34%
Solution: Application-Aware Policies

- Idea
  - Identify stall time critical applications (i.e. network sensitive applications) and prioritize their packets in each router.

- Key components of scheduling policy:
  - Application Ranking
  - Packet Batching

- Propose low-hardware complexity solution
Component 1: Ranking

- Ranking distinguishes applications based on Stall Time Criticality (STC).
- Periodically rank applications based on Stall Time Criticality (STC).
- Explored many heuristics for quantifying STC (Details & analysis in paper)
  - Heuristic based on outermost private cache Misses Per Instruction (L1-MPI) is the most effective
    - Low L1-MPI => high STC => higher rank
- Why Misses Per Instruction (L1-MPI)?
  - Easy to Compute (low complexity)
  - Stable Metric (unaffected by interference in network)
Component 1 : How to Rank?

- Execution time is divided into fixed “ranking intervals”
  - Ranking interval is 350,000 cycles
- At the end of an interval, each core calculates their L1-MPI and sends it to the Central Decision Logic (CDL)
  - CDL is located in the central node of mesh
- CDL forms a ranking order and sends back its rank to each core
  - Two control packets per core every ranking interval
- Ranking order is a “partial order”

- Rank formation is not on the critical path
  - Ranking interval is significantly longer than rank computation time
  - Cores use older rank values until new ranking is available
Component 2: Batching

- **Problem:** Starvation
  - Prioritizing a higher ranked application can lead to starvation of lower ranked application

- **Solution:** Packet Batching
  - Network packets are grouped into finite sized batches
  - **Packets of older batches are prioritized over younger batches**

- Alternative batching policies explored in paper

- **Time-Based Batching**
  - New batches are formed in a periodic, synchronous manner across all nodes in the network, every $T$ cycles
Putting it all together

- Before injecting a packet into the network, it is tagged by
  - Batch ID (3 bits)
  - Rank ID (3 bits)

- Three tier priority structure at routers
  - Oldest batch first (prevent starvation)
  - Highest rank first (maximize performance)
  - Local Round-Robin (final tie breaker)

- Simple hardware support: priority arbiters

- Global coordinated scheduling
  - Ranking order and batching order are the same across all routers
STC Scheduling Example

Packet Injection Order at Processor

Batch 0

Batch 1

Batch 2

Batching interval length = 3 cycles

Ranking order =  

Core1  Core2  Core3
STC Scheduling Example

Batch 0

Batch 1

Batch 2

Applications

Injection Cycles

Router

Scheduler
STC Scheduling Example

Router

5
8
4
3
7
1
6
2
2

Scheduler

Round Robin

3
2
8
7
6

STALL CYCLES

<table>
<thead>
<tr>
<th>STALL CYCLES</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>8</td>
</tr>
<tr>
<td>Age</td>
<td></td>
</tr>
<tr>
<td>STC</td>
<td></td>
</tr>
</tbody>
</table>
STC Scheduling Example

Router

Scheduler

Round Robin

Time

Age

STALL CYCLES | Avg
---|---
RR | 8 6 11 | 8.3
Age | 4 6 11 | 7.0
STC |
STC Scheduling Example

Router

Scheduler

Round Robin

Time

Age

STC

STALL CYCLES

<table>
<thead>
<tr>
<th></th>
<th>RR</th>
<th>Age</th>
<th>STC</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>8</td>
<td>6</td>
<td>11</td>
<td>8.3</td>
</tr>
<tr>
<td>Time</td>
<td>4</td>
<td>6</td>
<td>11</td>
<td>7.0</td>
</tr>
<tr>
<td>Time</td>
<td>1</td>
<td>3</td>
<td>11</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Qualitative Comparison

- **Round Robin & Age**
  - Local and application oblivious
  - Age is biased towards heavy applications
    - Heavy applications flood the network
    - Higher likelihood of an older packet being from heavy application

- **Globally Synchronized Frames (GSF)** [Lee et al., ISCA 2008]
  - Provides bandwidth fairness at the expense of system performance
  - Penalizes heavy and bursty applications
    - Each application gets equal and fixed quota of flits (credits) in each batch.
    - Heavy application quickly run out of credits after injecting into all active batches & stall till oldest batch completes and frees up fresh credits.
    - Underutilization of network resources
System Performance

- STC provides 9.1% improvement in weighted speedup over the best existing policy (averaged across 96 workloads)
- Detailed case studies in the paper
Slack-Driven Packet Scheduling

Packet Scheduling in NoC

- Existing scheduling policies
  - Round robin
  - Age

- Problem
  - Treat all packets equally
  - Application-oblivious

- Packets have different criticality
  - Packet is critical if latency of a packet affects application’s performance
  - Different criticality due to memory level parallelism (MLP)
MLP Principle

Packet Latency $\neq$ Network Stall Time

Different Packets have different criticality due to MLP

$\text{Criticality}(\square) > \text{Criticality}(\square) > \text{Criticality}(\square)$
Outline

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aërgia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion
What is Aégria?

- Aégria is the spirit of laziness in Greek mythology
- Some packets can afford to slack!
Outline

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Áërgia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion
Slack of Packets

- What is slack of a packet?
  - Slack of a packet is number of cycles it can be delayed in a router without (significantly) reducing application’s performance
  - Local network slack

- Source of slack: Memory-Level Parallelism (MLP)
  - Latency of an application’s packet hidden from application due to overlap with latency of pending cache miss requests

- Prioritize packets with lower slack
Concept of Slack

Instruction Window

Load Miss Causes
Load Miss Causes

Execution Time

Latency (↑)

Latency (↓)

Slack

Stall Compute

Network-on-Chip

Slack (↑) = Latency (↑) – Latency (↓) = 26 – 6 = 20 hops

Packet(↑) can be delayed for available slack cycles without reducing performance!
Prioritizing using Slack

Core A

Load Miss
Causes
Load Miss
Causes

Core B

Load Miss
Causes
Load Miss
Causes

Packet | Latency | Slack
---|---|---
13 hops | 0 hops
3 hops | 10 hops

- Interference at 3 hops
- Slack( ) > Slack ( )
- Prioritize
Slack in Applications

- 50% of packets have 350+ slack cycles
- 10% of packets have <50 slack cycles

Non-critical

Critical
Slack in Applications

68% of packets have zero slack cycles

Percentage of all Packets (%) vs Slack in cycles

- Gems
- art

- 68% of packets have zero slack cycles
Diversity in Slack

The graph shows the percentage of all packets (%) over slack in cycles for various benchmarks. The y-axis represents the percentage of all packets, while the x-axis represents slack in cycles. Different benchmarks are indicated by distinct lines and colors, allowing for a comparison of how each benchmark behaves under varying slack conditions.
Diversity in Slack

Slack varies **between packets of different applications**

Slack varies **between packets of a single application**
Outline

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aergia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion
Estimating Slack Priority

\[ \text{Slack (P)} = \text{Max (Latencies of P’s Predecessors)} - \text{Latency of P} \]

- Predecessors(P) are the packets of outstanding cache miss requests when P is issued

- Packet latencies not known when issued

- Predicting latency of any packet Q
  - Higher latency if Q corresponds to an L2 miss
  - Higher latency if Q has to travel farther number of hops
Estimating Slack Priority

- Slack of $P = \text{Maximum Predecessor Latency} - \text{Latency of } P$

- Slack($P) = \begin{array}{ccc} \text{PredL2} & \text{MyL2} & \text{HopEstimate} \\ (2 \text{ bits}) & (1 \text{ bit}) & (2 \text{ bits}) \end{array}$

**PredL2**: Set if any predecessor packet is servicing L2 miss

**MyL2**: Set if $P$ is NOT servicing an L2 miss

**HopEstimate**: Max (# of hops of Predecessors) – hops of $P$
Estimating Slack Priority

- How to predict L2 hit or miss at core?
  - *Global Branch Predictor* based L2 Miss Predictor
    - Use Pattern History Table and 2-bit saturating counters
  - *Threshold* based L2 Miss Predictor
    - If \#L2 misses in “M” misses \(\geq\) “T” threshold then next load is a L2 miss.

- Number of miss predecessors?
  - List of outstanding L2 Misses

- Hops estimate?
  - Hops \(\Rightarrow\) \(\Delta X + \Delta Y\) distance
  - Use predecessor list to calculate slack hop estimate
Starvation Avoidance

- Problem: Starvation
  - Prioritizing packets can lead to starvation of lower priority packets

- Solution: Time-Based Packet Batching
  - New batches are formed at every $T$ cycles
  - Packets of older batches are prioritized over younger batches
Putting it all together

- Tag header of the packet with priority bits before injection

\[
\text{Priority (P) = Batch (3 bits) \cdot PredL2 (2 bits) \cdot MyL2 (1 bit) \cdot HopEstimate (2 bits)}
\]

- Priority(P)?
  - P’s batch (highest priority)
  - P’s Slack
  - Local Round-Robin (final tie breaker)
Outline

- Introduction
  - Packet Scheduling
  - Memory Level Parallelism
- Aérgia
  - Concept of Slack
  - Estimating Slack
- Evaluation
- Conclusion
Evaluation Methodology

- **64-core system**
  - x86 processor model based on Intel Pentium M
  - 2 GHz processor, 128-entry instruction window
  - 32KB private L1 and 1MB per core shared L2 caches, 32 miss buffers
  - 4GB DRAM, 320 cycle access latency, 4 on-chip DRAM controllers

- **Detailed Network-on-Chip model**
  - 2-stage routers (with speculation and look ahead routing)
  - Wormhole switching (8 flit data packets)
  - Virtual channel flow control (6 VCs, 5 flit buffer depth)
  - 8x8 Mesh (128 bit bi-directional channels)

- **Benchmarks**
  - Multiprogrammed scientific, server, desktop workloads (35 applications)
  - 96 workload combinations
Qualitative Comparison

▪ Round Robin & Age
  ▪ Local and application oblivious
  ▪ Age is biased towards heavy applications

▪ Globally Synchronized Frames (GSF)
  [Lee et al., ISCA 2008]
  ▪ Provides bandwidth fairness at the expense of system performance
  ▪ Penalizes heavy and bursty applications

▪ Application-Aware Prioritization Policies (SJF)
  [Das et al., MICRO 2009]
  ▪ Shortest-Job-First Principle
  ▪ Packet scheduling policies which prioritize network sensitive applications which inject lower load
System Performance

- SJF provides 8.9% improvement in weighted speedup
- Àergia improves system throughput by 10.3%
- Àergia+SJF improves system throughput by 16.1%
Network Unfairness

- SJF does not imbalance network fairness
- Aergia improves network unfairness by 1.5X
- SJF+Aergia improves network unfairness by 1.3X
Conclusions & Future Directions

- Packets have different criticality, yet existing packet scheduling policies treat all packets equally.
- We propose a new approach to packet scheduling in NoCs.
  - We define Slack as a key measure that characterizes the relative importance of a packet.
  - We propose Áergia, a novel architecture to accelerate low slack critical packets.
- Result
  - Improves system performance: 16.1%
  - Improves network fairness: 30.8%
Express-Cube Topologies

2-D Mesh
2-D Mesh

- **Pros**
  - Low design & layout complexity
  - Simple, fast routers

- **Cons**
  - Large diameter
  - Energy & latency impact
Concentration (*Balfour & Dally, ICS ‘06*)

- **Pros**
  - Multiple *terminals* attached to a router node
  - Fast nearest-neighbor communication via the crossbar
  - Hop count reduction proportional to concentration degree

- **Cons**
  - Benefits limited by crossbar complexity
Concentration

- Side-effects
  - Fewer channels
  - Greater channel width
Replication

- Benefits
  - Restores bisection channel count
  - Restores channel width
  - Reduced crossbar complexity

CMesh-X2
Flattened Butterfly (Kim et al., Micro ’07)

- Objectives:
  - Improve connectivity
  - Exploit the wire budget
Flattened Butterfly \textit{(Kim et al., Micro '07)}
Flattened Butterfly (Kim et al., Micro ‘07)
Flattened Butterfly (*Kim et al., Micro* ‘07)
Flattened Butterfly (Kim et al., Micro '07)
Flattened Butterfly (*Kim et al.*, Micro '07)

- **Pros**
  - Excellent connectivity
  - Low diameter: 2 hops

- **Cons**
  - High channel count: \( k^2/2 \) per row/column
  - Low channel utilization
  - Increased control (arbitration) complexity
Multidrop Express Channels (MECS)

- Objectives:
  - Connectivity
  - More scalable channel count
  - Better channel utilization
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)
Multidrop Express Channels (MECS)

- **Pros**
  - One-to-many topology
  - Low diameter: 2 hops
  - $k$ channels row/column
  - Asymmetric

- **Cons**
  - Asymmetric
  - Increased control (arbitration) complexity
Partitioning: a GEC Example

MECS

MECS-X2

Partitioned MECS

Flattened Butterfly
## Analytical Comparison

<table>
<thead>
<tr>
<th></th>
<th>CMesh</th>
<th>FBfly</th>
<th>MECS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network Size</strong></td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td><strong>Radix (conctr’ d)</strong></td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>Diameter</strong></td>
<td>6</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Channel count</strong></td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td><strong>Channel width</strong></td>
<td>576</td>
<td>144</td>
<td>288</td>
</tr>
<tr>
<td><strong>Router inputs</strong></td>
<td>4</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td><strong>Router outputs</strong></td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
# Experimental Methodology

<table>
<thead>
<tr>
<th>Topologies</th>
<th>Mesh, CMesh, CMesh-X2, FBFly, MECS, MECS-X2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network sizes</td>
<td>64 &amp; 256 terminals</td>
</tr>
<tr>
<td>Routing</td>
<td>DOR, adaptive</td>
</tr>
<tr>
<td>Messages</td>
<td>64 &amp; 576 bits</td>
</tr>
<tr>
<td>Synthetic traffic</td>
<td>Uniform random, bit complement, transpose, self-similar</td>
</tr>
<tr>
<td>PARSEC benchmarks</td>
<td>Blackscholes, Bodytrack, Canneal, Ferret, Fluidanimate, Freqmine, Vip, x264</td>
</tr>
<tr>
<td>Full-system config</td>
<td>M5 simulator, Alpha ISA, 64 OOO cores</td>
</tr>
<tr>
<td>Energy evaluation</td>
<td>Orion + CACTI 6</td>
</tr>
</tbody>
</table>

---

UTCS HPCA '09
64 nodes: Uniform Random

Latency (cycles) vs. injection rate (%) for different network topologies:
- mesh
- cmesh
- cmesh-x2
- fbfly
- mecs
- mecs-x2
256 nodes: Uniform Random

Latency (cycles) vs. Injection rate (%)

- mesh
- cmesh-x2
- fbfly
- mecs
- mecs-x2

Injection rate (%) range from 1 to 25.
Energy (100K pkts, Uniform Random)

![Graph showing energy consumption for different network configurations at 64 nodes and 256 nodes. The graph compares link energy (green) and router energy (red).](image-url)
64 Nodes: PARSEC

Router Energy  Link Energy  latency

Blackscholes  Canneal  Vip  x264
Summary

- MECS
  - A new one-to-many topology
  - Good fit for planar substrates
  - Excellent connectivity
  - Effective wire utilization

- Generalized Express Cubes
  - Framework & taxonomy for NOC topologies
  - Extension of the k-ary n-cube model
  - Useful for understanding and exploring on-chip interconnect options
  - Future: expand & formalize
Kilo-NoC: Topology-Aware QoS

Motivation

- Extreme-scale chip-level integration
  - Cores
  - Cache banks
  - Accelerators
  - I/O logic
  - Network-on-chip (NOC)
- 10-100 cores today
- 1000+ assets in the near future
Kilo-NOC requirements

- High efficiency
  - Area
  - Energy
- Good performance
- Strong service guarantees (QoS)
Topology-Aware QoS

- Problem: QoS support in each router is expensive (in terms of buffering, arbitration, bookkeeping)

- Goal: Provide QoS guarantees at low area and power cost

- Idea:
  - Isolate shared resources in a region of the network, support QoS within that area
  - Design the topology so that applications can access the region without interference
Baseline QOS-enabled CMP

Multiple VMs sharing a die

- Shared resources (e.g., memory controllers)
- VM-private resources (cores, caches)
- QOS-enabled router
Contestion scenarios:

- **Shared resources**
  - memory access
- **Intra-VM traffic**
  - shared cache access
- **Inter-VM traffic**
  - VM page sharing
Conventional NOC QOS

Contention scenarios:
- Shared resources
  - memory access
- Intra-VM traffic
  - shared cache access
- Inter-VM traffic
  - VM page sharing

Network-wide guarantees *without* network-wide QOS support
Kilo-NOC QOS

- Insight: leverage rich network connectivity
  - Naturally reduce interference among flows
  - Limit the extent of hardware QOS support

- Requires a low-diameter topology
  - This work: Multidrop Express Channels (MECS)

Grot et al., HPCA 2009
Dedicated, QOS-enabled regions
- Rest of die: QOS-free

Richly-connected topology
- Traffic isolation

Special routing rules
- Manage interference
Topology-Aware QOS

- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference
Dedicated, QOS-enabled regions
- Rest of die: QOS-free

Richly-connected topology
- Traffic isolation

Special routing rules
- Manage interference
Topological Aware QOS

- Dedicated, QOS-enabled regions
  - Rest of die: QOS-free
- Richly-connected topology
  - Traffic isolation
- Special routing rules
  - Manage interference
- Topology-aware QOS support
  - Limit QOS complexity to a fraction of the die

- Optimized flow control
  - Reduce buffer requirements in QOS-free regions
## Evaluation Methodology

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>15 nm</td>
</tr>
<tr>
<td>Vdd</td>
<td>0.7 V</td>
</tr>
<tr>
<td>System</td>
<td>1024 tiles: 256 concentrated nodes (64 shared resources)</td>
</tr>
<tr>
<td>Networks:</td>
<td></td>
</tr>
<tr>
<td>MECS+PVC</td>
<td>VC flow control, QOS support (PVC) at each node</td>
</tr>
<tr>
<td>MECS+TAQ</td>
<td>VC flow control, QOS support only in shared regions</td>
</tr>
<tr>
<td>MECS+TAQ+EB</td>
<td>EB flow control outside of SRs, Separate Request and Reply networks</td>
</tr>
<tr>
<td>K-MECS</td>
<td>Proposed organization: TAQ + hybrid flow control</td>
</tr>
</tbody>
</table>
Area comparison

Area (mm²)

<table>
<thead>
<tr>
<th>Configuration</th>
<th>SR Routers</th>
<th>Routers</th>
<th>Link EBs</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>MECS+PVC</td>
<td></td>
<td>25</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>MECS+TAQ</td>
<td>5</td>
<td></td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>MECS+TAQ+EB</td>
<td>10</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K-MECS</td>
<td>15</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
Energy comparison

![Graph showing energy comparison between different network configurations.](image)

- **MECS**
- **MECS EB**
- **MECS hybrid**

**Axes:**
- X-axis: Load (%)
- Y-axis: Average packet latency (cycles)

**Legend:**
- Green: SR Routers
- Pink: Routers
- Blue: Link EBs
- Orange: Links

**Network Configurations:**
- MECS+PVC
- MECS+TAQ
- MECS+EB+TAQ
- K-MECS
Summary

Kilo-NOC: a heterogeneous NOC architecture for kilo-node substrates

- Topology-aware QOS
  - Limits QOS support to a fraction of the die
  - Leverages low-diameter topologies
  - Improves NOC area- and energy-efficiency
  - Provides strong guarantees