SNACKNOC: PROCESSING IN THE COMMUNICATION LAYER

Karthik Sangaiah, Michael Lui, Ragh Kuttappa,
Baris Taskin, and Mark Hempstead

Feb 25th 2020

VLSI and Architecture Lab

Opportunistic Resources for Graduate Students

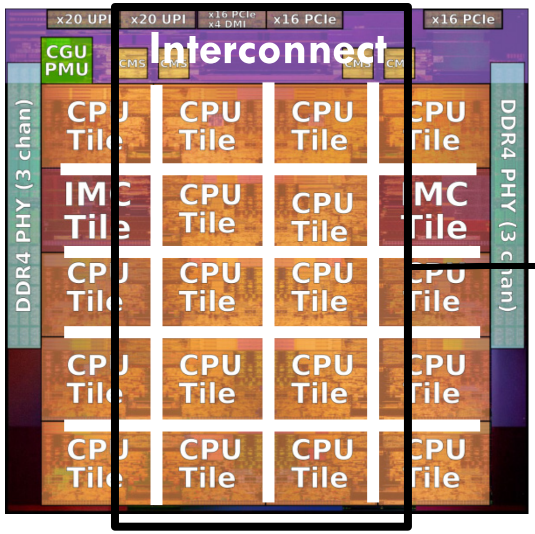
Free leftovers

toward

Steak dinner

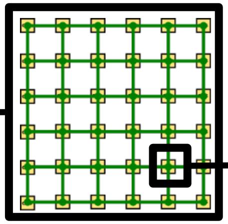
Opportunistically collecting snacks towards a meal.

Opportunistic Resources in the CMP

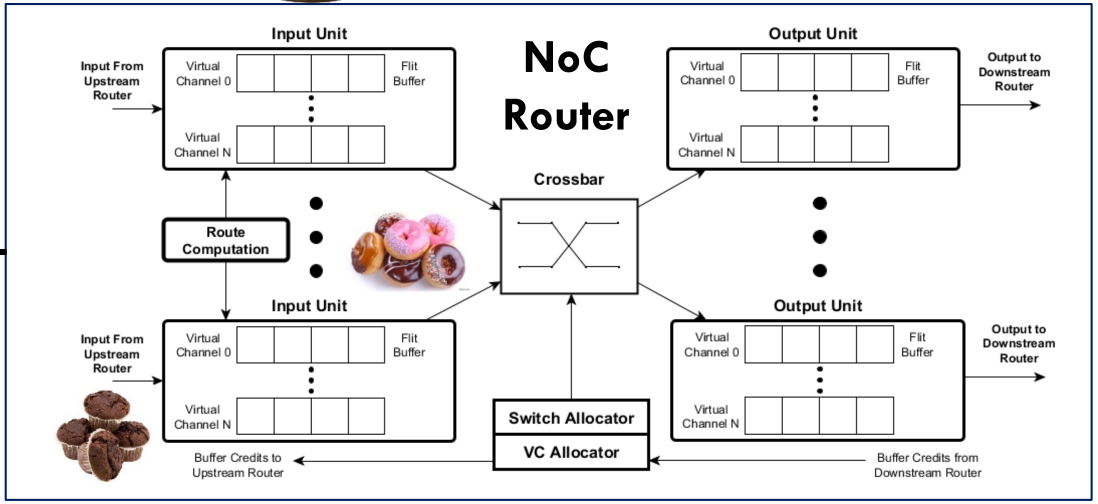


Intel Skylake 8180 HCC [1]

Communication Interconnect



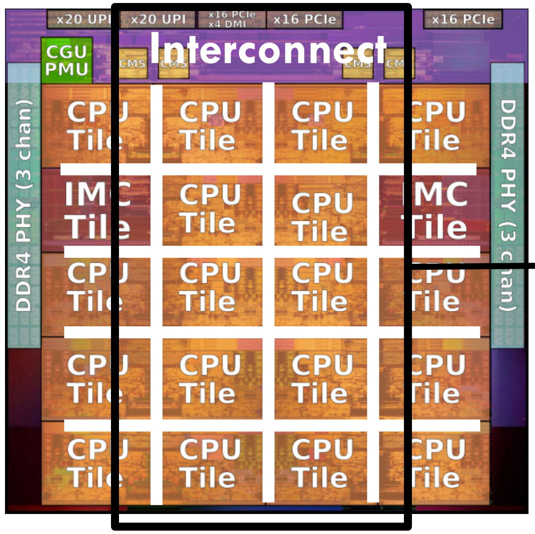
"Free leftovers"



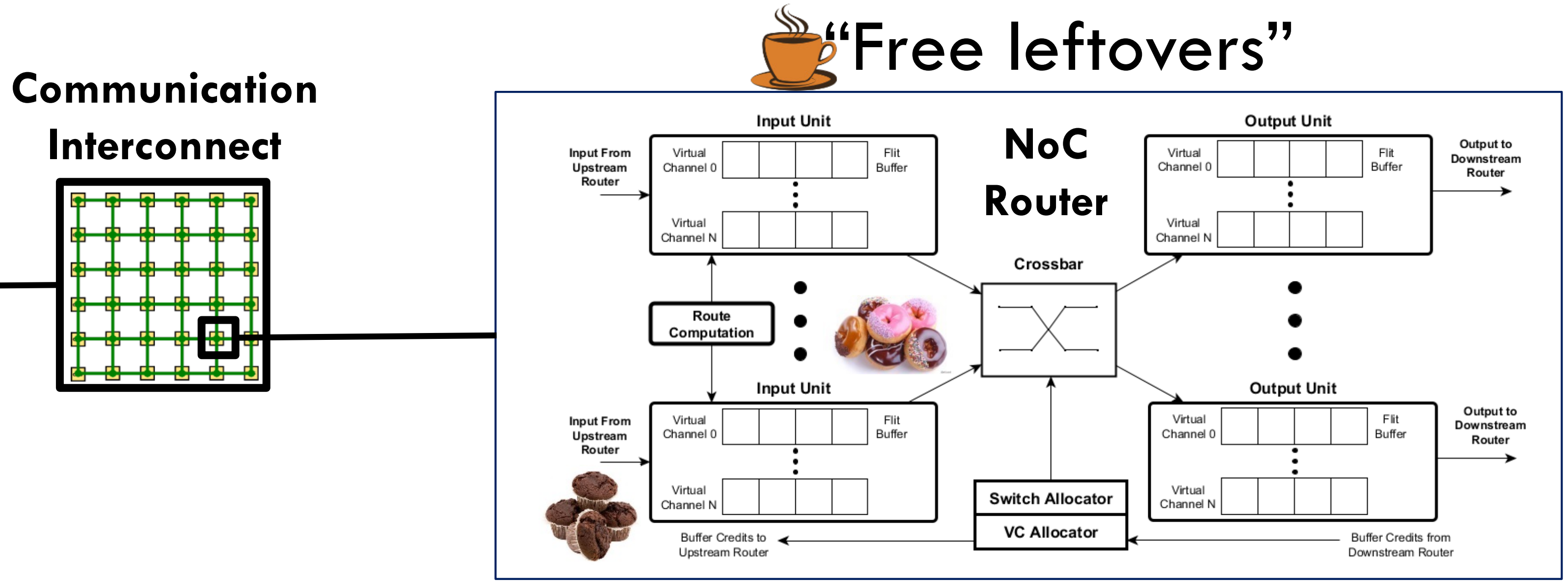
Opportunistically collecting "snacks" to make a "meal".

[1] Intel Skylake SP HCC, Wikichip.

Opportunistic Resources in the CMP



Intel Skylake 8180 HCC [1]



What is the performance gain we add by opportunistically "snacking" on CMP resources?

[1] Intel Skylake SP HCC, Wikichip.

Quantifying Design Slack in the NoC

- NoC designed to minimize latency during **heavy** traffic
  - NoC implementation can account for 60% to 75% of the miss latency[2]

[2] Sanchez et al., ACM TACO, 2010.

Quantifying Design Slack in the NoC

- NoC designed to minimize latency during **heavy** traffic
  - NoC implementation can account for 60% to 75% of the miss latency[2]

- Study of NoC resource utilization on recent NoCs designs
  - 3 selected best paper nominated NoCs have similar performance:
    - DAPPER[3], AxNoC[4], BiNoCHS[5]
  - Reducing resources, substantially reduced performances
    - Further details of study is in our paper

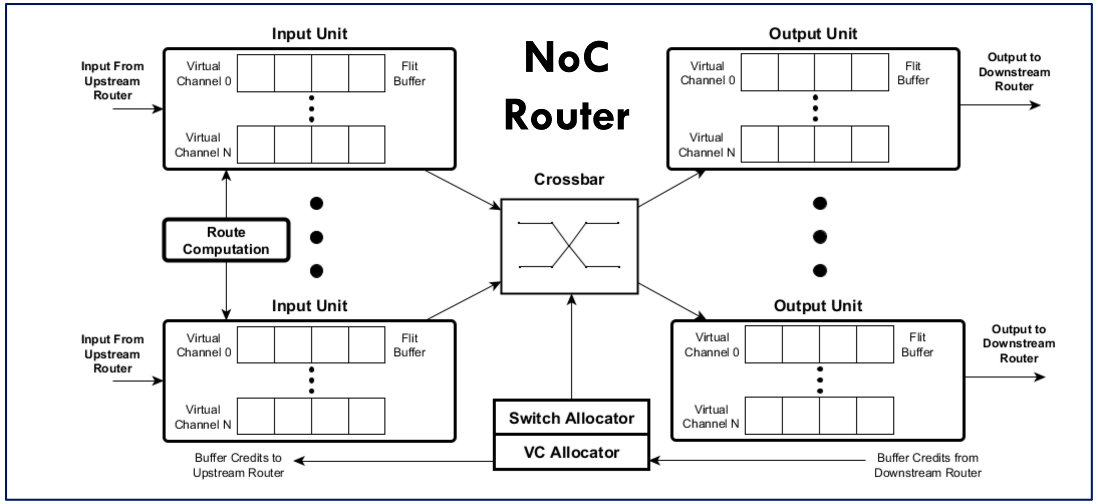
[2] Sanchez et al., ACM TACO, 2010.
[3] Raparti et al., IEEE/ACM NOCS, 2018.
[4] Ahmed et al., IEEE/ACM NOCS, 2018.
[5] Mirhosseini et al, IEEE/ACM NOCS, 2017.

Quantifying Design Slack in the NoC

- NoC designed to minimize latency during **heavy** traffic
  - NoC implementation can account for 60% to 75% of the miss latency[2]

- Study of NoC resource utilization on recent NoCs designs
  - 3 selected best paper nominated NoCs have similar performance:
    - DAPPER[3], AxNoC[4], BiNoCHS[5]
  - Reducing resources, substantially reduced performances
    - Further details of study is in our paper

- Opportunities in Network-on-Chip Slack

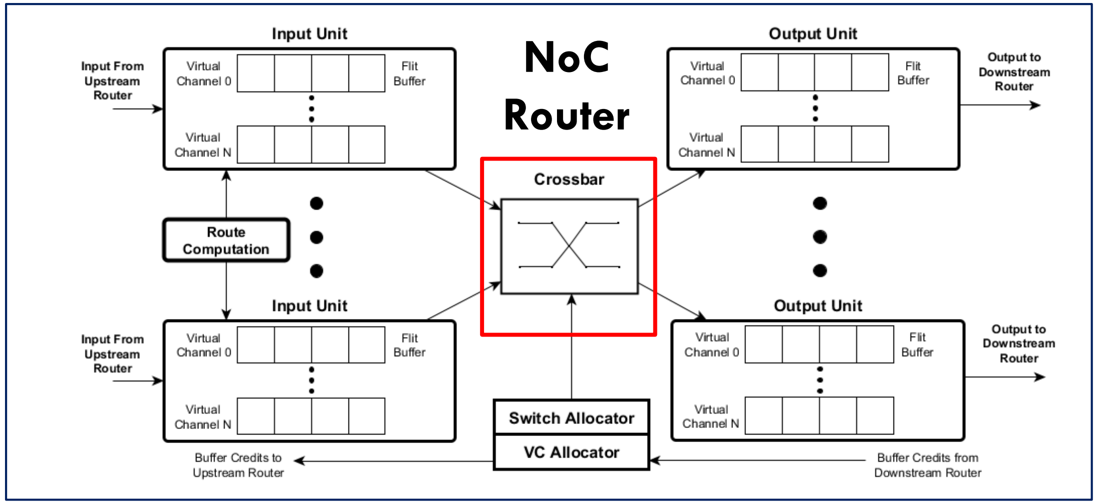


[2] Sanchez et al., ACM TACO, 2010.
[3] Raparti et al., IEEE/ACM NOCS, 2018.
[4] Ahmed et al., IEEE/ACM NOCS, 2018.
[5] Mirhosseini et al, IEEE/ACM NOCS, 2017.

Quantifying Design Slack in the NoC

- NoC designed to minimize latency during **heavy** traffic
  - NoC implementation can account for 60% to 75% of the miss latency[2]
- Study of NoC resource utilization on recent NoCs designs
  - 3 selected best paper nominated NoCs have similar performance:
    - DAPPER[3], AxNoC[4], BiNoCHS[5]
  - Reducing resources, substantially reduced performances
    - Further details of study is in our paper

- Opportunities in Network-on-Chip Slack
  - Crossbar

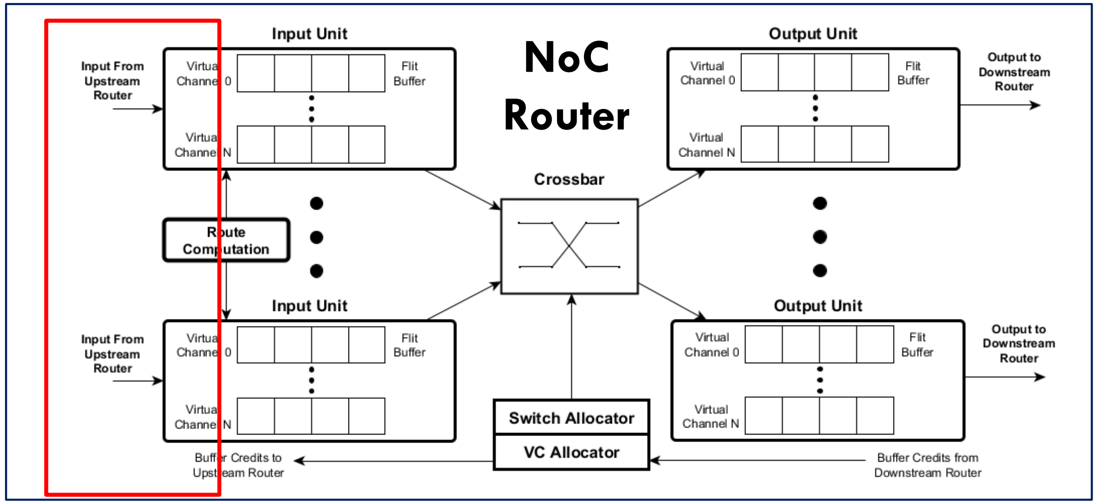


[2] Sanchez et al., ACM TACO, 2010.
[3] Raparti et al., IEEE/ACM NOCS, 2018.
[4] Ahmed et al., IEEE/ACM NOCS, 2018.
[5] Mirhosseini et al, IEEE/ACM NOCS, 2017.

Quantifying Design Slack in the NoC

- NoC designed to minimize latency during **heavy** traffic
  - NoC implementation can account for 60% to 75% of the miss latency[2]
- Study of NoC resource utilization on recent NoCs designs
  - 3 selected best paper nominated NoCs have similar performance:
    - DAPPER[3], AxNoC[4], BiNoCHS[5]
  - Reducing resources, substantially reduced performances
    - Further details of study is in our paper

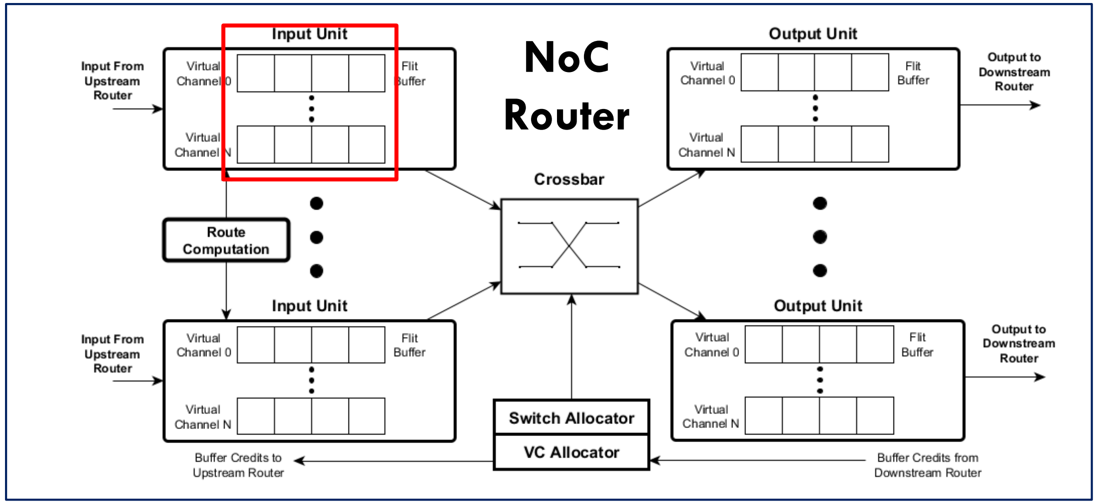
- Opportunities in Network-on-Chip Slack
  - Crossbar
  - Network Links



[2] Sanchez et al., ACM TACO, 2010.
[3] Raparti et al., IEEE/ACM NOCS, 2018.
[4] Ahmed et al., IEEE/ACM NOCS, 2018.
[5] Mirhosseini et al, IEEE/ACM NOCS, 2017.

Quantifying Design Slack in the NoC

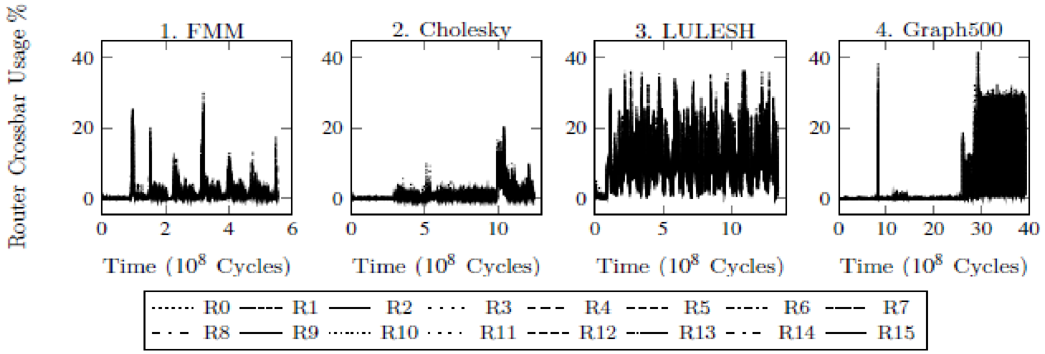
- NoC designed to minimize latency during **heavy** traffic
  - NoC implementation can account for 60% to 75% of the miss latency[2]
- Study of NoC resource utilization on recent NoCs designs
  - 3 selected best paper nominated NoCs have similar performance:
    - DAPPER[3], AxNoC[4], BiNoCHS[5]
  - Reducing resources, substantially reduced performances
    - Further details of study is in our paper
- Opportunities in Network-on-Chip Slack
  - Crossbar
  - Network Links
  - Internal Buffers



[2] Sanchez et al., ACM TACO, 2010.
[3] Raparti et al., IEEE/ACM NOCS, 2018.
[4] Ahmed et al., IEEE/ACM NOCS, 2018.
[5] Mirhosseini et al, IEEE/ACM NOCS, 2017.

Quantifying Design Slack in the NoC

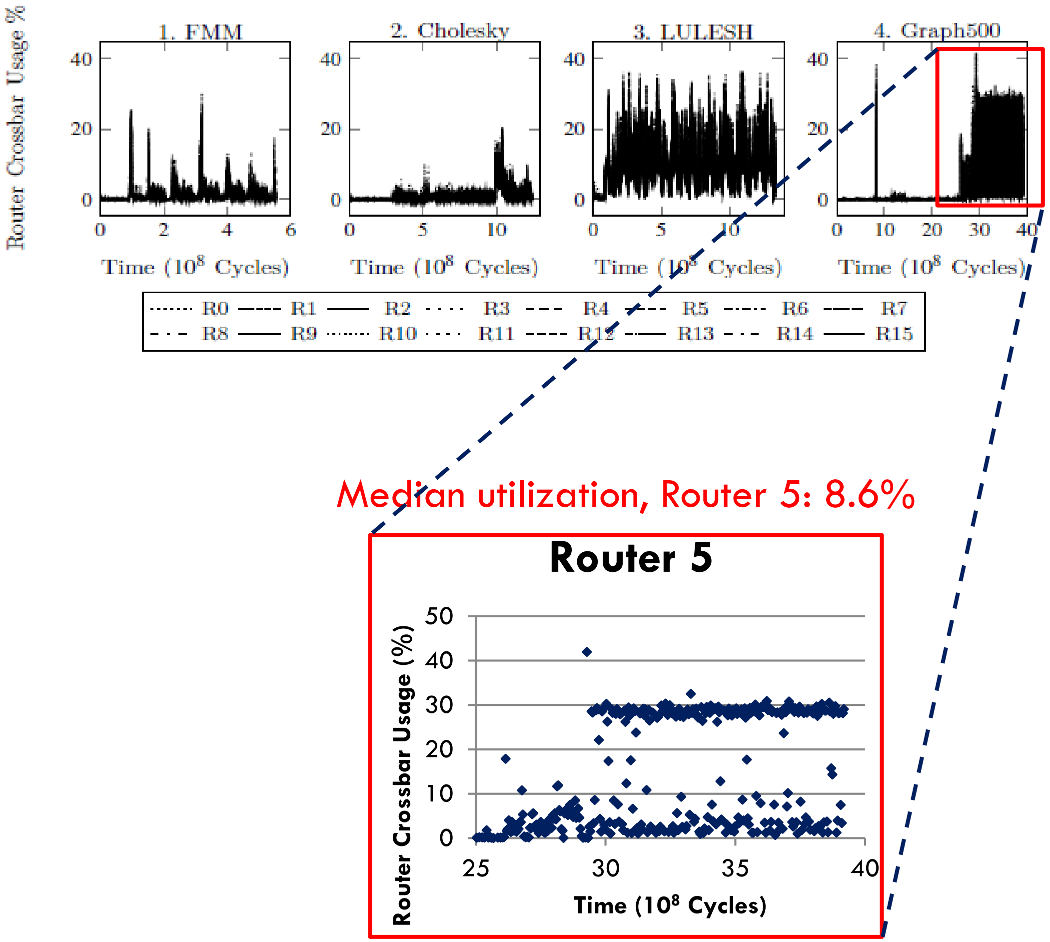
Crossbar Utilization



- Simulated 16 core CMP with 4 benchmarks representing "low", "medium", "medium-high", "high" traffic
- Crossbar Utilization:
  - Peak utilization (Graph 500): 42% utilization
  - Highest median (Graph 500): 13.3% utilization

Quantifying Design Slack in the NoC

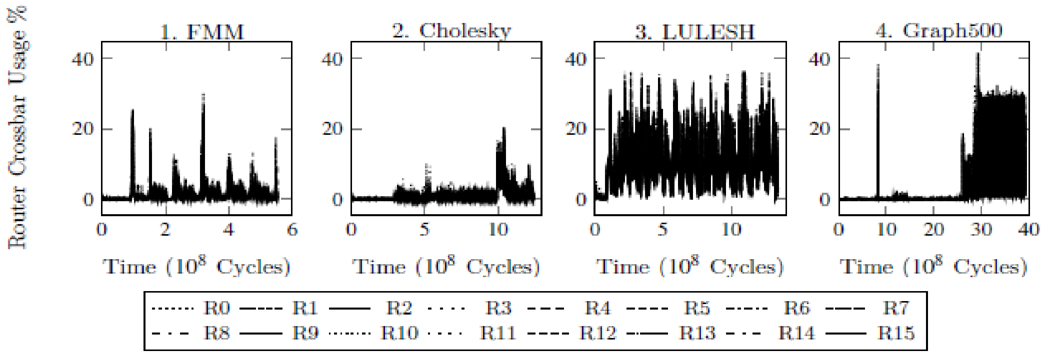
Crossbar Utilization



- Simulated 16 core CMP with 4 benchmarks representing "low", "medium", "medium-high", "high" traffic
- Crossbar Utilization:
  - Peak utilization (Graph 500): 42% utilization
  - Highest median (Graph 500): 13.3% utilization

Quantifying Design Slack in the NoC

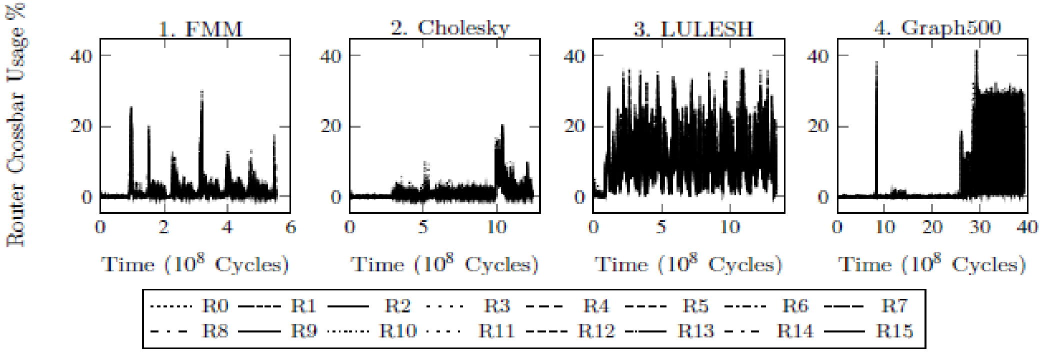
Crossbar Utilization



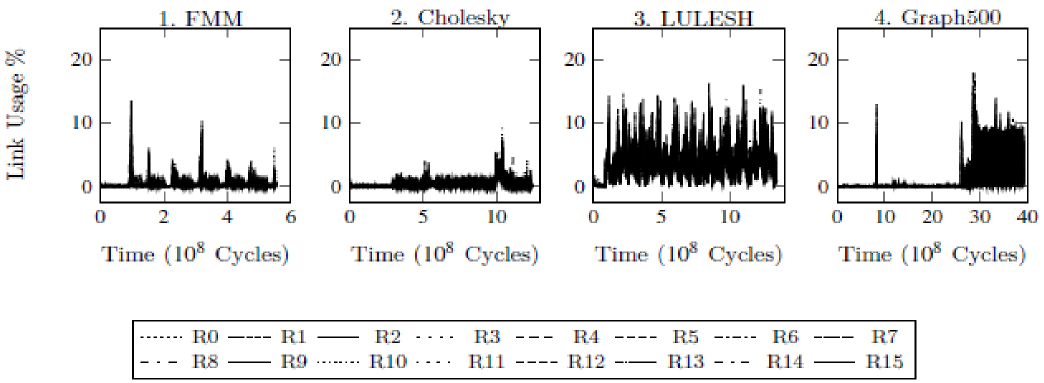
- Simulated 16 core CMP with 4 benchmarks representing "low", "medium", "medium-high", "high" traffic
- Crossbar Utilization:
  - Peak utilization (Graph 500): 42% utilization
  - Highest median (Graph 500): 13.3% utilization

Quantifying Design Slack in the NoC

Crossbar Utilization



Link Utilization



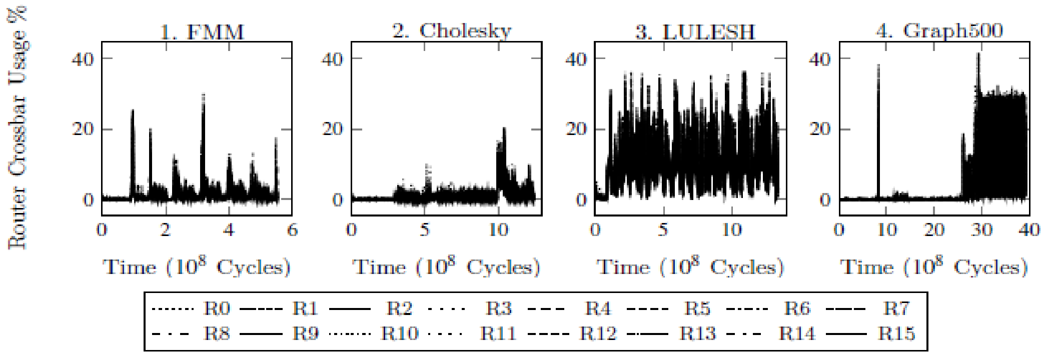
- Simulated 16 core CMP with 4 benchmarks representing “low”, “medium”, “medium-high”, “high” traffic

- Crossbar Utilization:
  - Peak utilization (Graph 500): 42% utilization
  - Highest median (Graph 500): 13.3% utilization

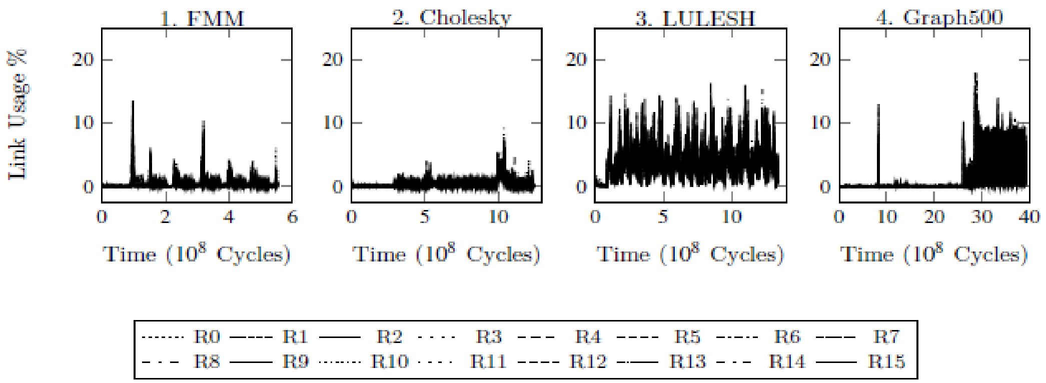
- Link Utilization
  - Peak utilization link (Graph500): 18% utilization
  - Highest median link utilization (LULESH): 3.3% utilization

Quantifying Design Slack in the NoC

Crossbar Utilization



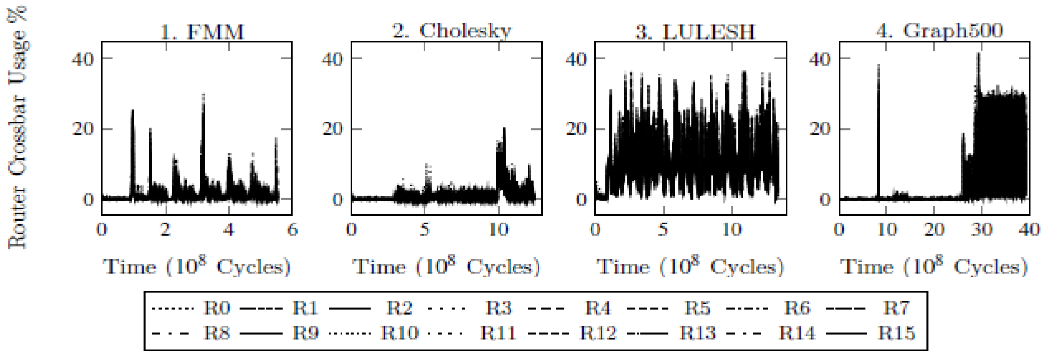
Link Utilization



- Simulated 16 core CMP with 4 benchmarks representing “low”, “medium”, “medium-high”, “high” traffic
- Crossbar Utilization:
  - Peak utilization (Graph 500): 42% utilization
  - Highest median (Graph 500): 13.3% utilization
- Link Utilization
  - Peak utilization link (Graph500): 18% utilization
  - Highest median link utilization (LULESH): 3.3% utilization
- Buffer Utilization
  - Raytrace : 4% of cycles have localized contention
  - 10% utilization during contention
  - 3M flits of the 2.4T flits forwarded: buffer utilization reaches 30-55% of the total capacity

Quantifying Design Slack in the NoC

Crossbar Utilization



- Simulated 16 core CMP with 4 benchmarks representing “low”, “medium”, “medium-high”, “high” traffic
- Crossbar Utilization:
  - Peak utilization (Graph 500): 42% utilization
  - Highest median (Graph 500): 13.3% utilization
- Link Utilization
  - Peak utilization link (Graph500): 18% utilization
  - Highest median link utilization (LULESH): 3.3% utilization

n reaches

The SnackNoC platform improves **efficiency** and **performance** of the CMP by offloading data-parallel workloads and “snacking” on network resources.

Overview

- “Slack” of the Communication Fabric
- **The SnackNoC Platform**
- Experimental Results
- Conclusion and Future Considerations

SnackNoC Platform Overview

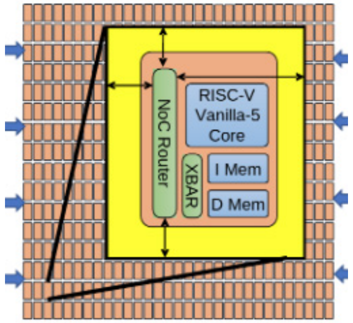
- Goals:
  - Opportunistically “Snack” on existing network resources for additional performance
  - Limited additional overhead to uncore
  - Minimal or zero interference to CMP traffic

- Opportunistic NoC-based compute platform
  - Limited dataflow engine
  - Applications:
    - Data-parallel workloads used in scientific computing, graph analytics, and machine learning

SnackNoC Platform Overview

- Goals:
  - Opportunistically “Snack” on existing network resources for additional performance
  - Limited additional overhead to uncore
  - Minimal or zero interference to CMP traffic

- Opportunistic NoC-based compute platform
  - Limited dataflow engine
  - Applications:
    - Data-parallel workloads used in scientific computing, graph analytics, and machine learning

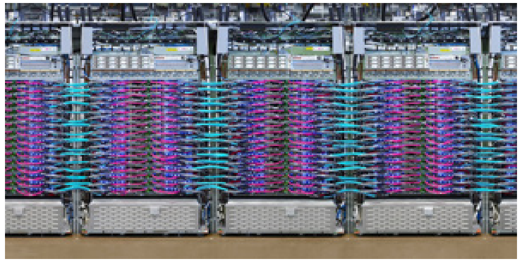


Celerity RISC-V SoC[6]

[6] S. Davidson et al., IEEE Micro, 2018.

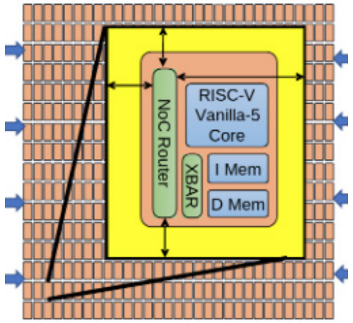
SnackNoC Platform Overview

- Goals:
  - Opportunistically “Snack” on existing network resources for additional performance
  - Limited additional overhead to uncore
  - Minimal or zero interference to CMP traffic



Google Cloud TPU[7]

- Opportunistic NoC-based compute platform
  - Limited dataflow engine
  - Applications:
    - Data-parallel workloads used in scientific computing, graph analytics, and machine learning



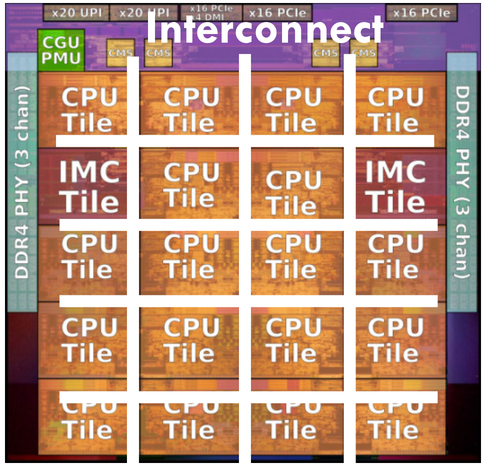
Celerity RISC-V SoC[6]

[6] S. Davidson et al., IEEE Micro, 2018.
[7] Jouppi et. al, IEEE/ACM ISCA, 2017.

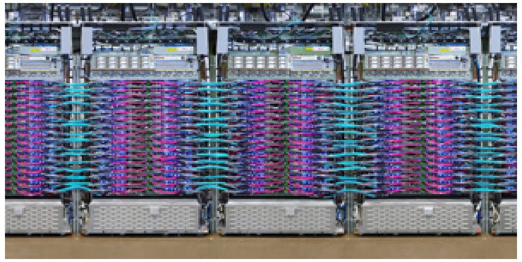
SnackNoC Platform Overview

- Goals:
  - Opportunistically "Snack" on existing network resources for additional performance
  - Limited additional overhead to uncore
  - Minimal or zero interference to CMP traffic

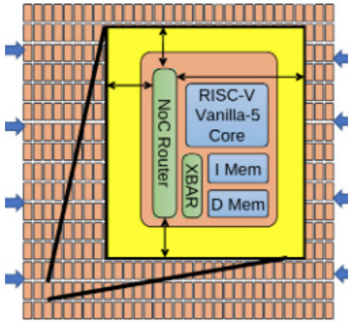
- Opportunistic NoC-based compute platform
  - Limited dataflow engine
  - Applications:
    - Data-parallel workloads used in scientific computing, graph analytics, and machine learning



Intel Skylake 8180 HCC[1]



Google Cloud TPU[7]



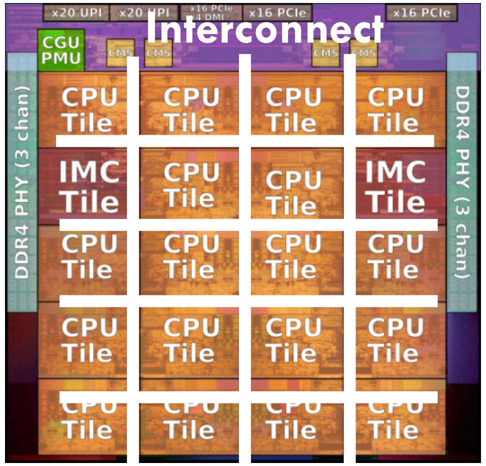
Celerity RISC-V SoC[6]

[1] Intel Skylake SP HCC, Wikichip.
[6] S. Davidson et al., IEEE Micro, 2018.
[7] Jouppi et. al, IEEE/ACM ISCA, 2017.

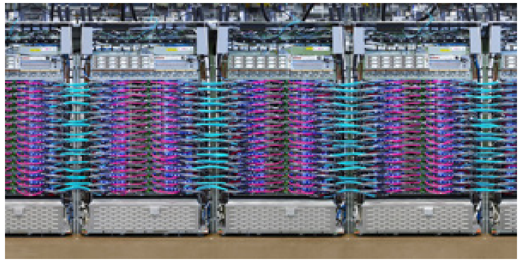
SnackNoC Platform Overview

- Goals:
  - Opportunistically “Snack” on existing network resources for additional performance
  - Limited additional overhead to uncore
  - Minimal or zero interference to CMP traffic

- Opportunistic NoC-based compute platform
  - Limited dataflow engine
  - Applications:
    - Data-parallel workloads used in scientific computing, graph analytics, and machine learning

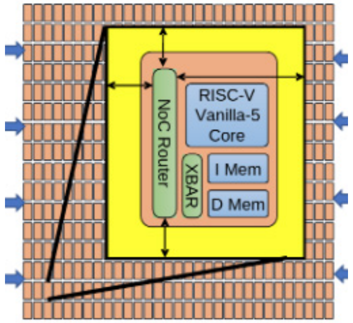


Intel Skylake 8180 HCC[1]



Google Cloud TPU[7]

Steak dinner



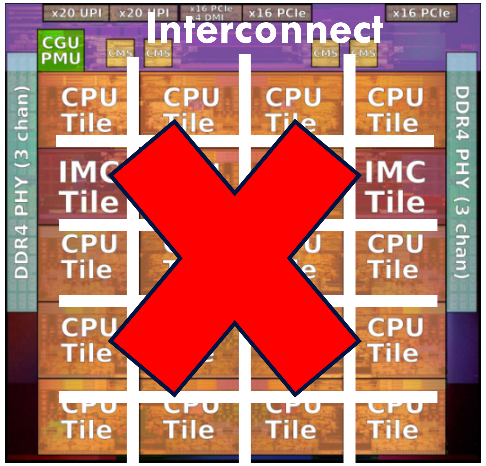
Celerity RISC-V SoC[6]

[1] Intel Skylake SP HCC, Wikichip.
[6] S. Davidson et al., IEEE Micro, 2018.
[7] Jouppi et. al, IEEE/ACM ISCA, 2017.

SnackNoC Platform Overview

- Goals:
  - Opportunistically "Snack" on existing network resources for additional performance
  - Limited additional overhead to uncore
  - Minimal or zero interference to CMP traffic

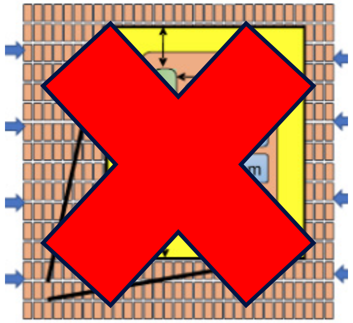
- Opportunistic NoC-based compute platform
  - Limited dataflow engine
  - Applications:
    - Data-parallel workloads used in scientific computing, graph analytics, and machine learning



Intel Skylake 8180 HCC[1]

Google Cloud TPU[7]

Steak dinner

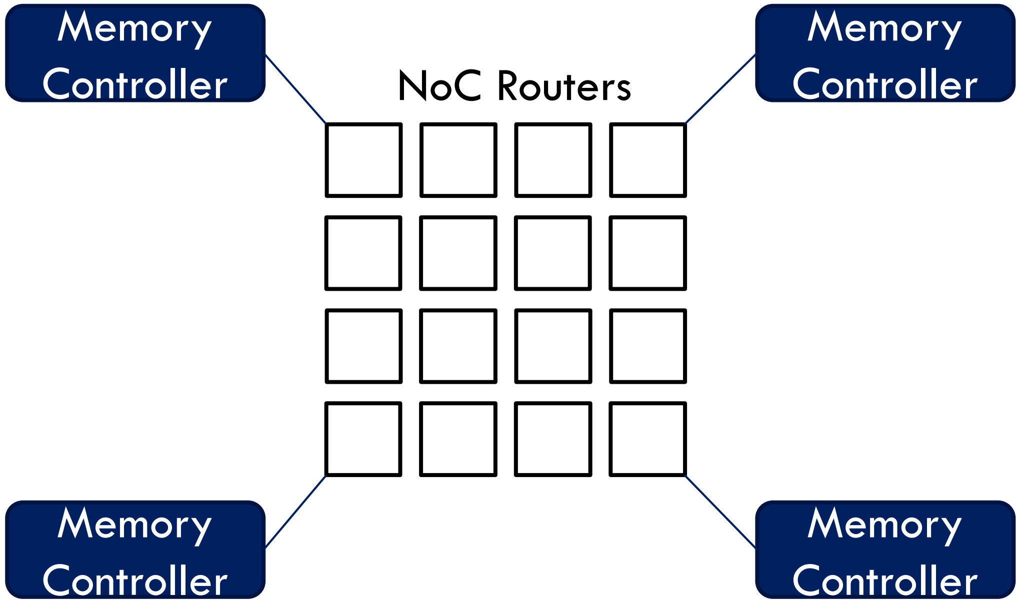


Celerity RISC-V SoC[6]

[1] Intel Skylake SP HCC, Wikichip.
[6] S. Davidson et al., IEEE Micro, 2018.
[7] Jouppi et. al, IEEE/ACM ISCA, 2017.

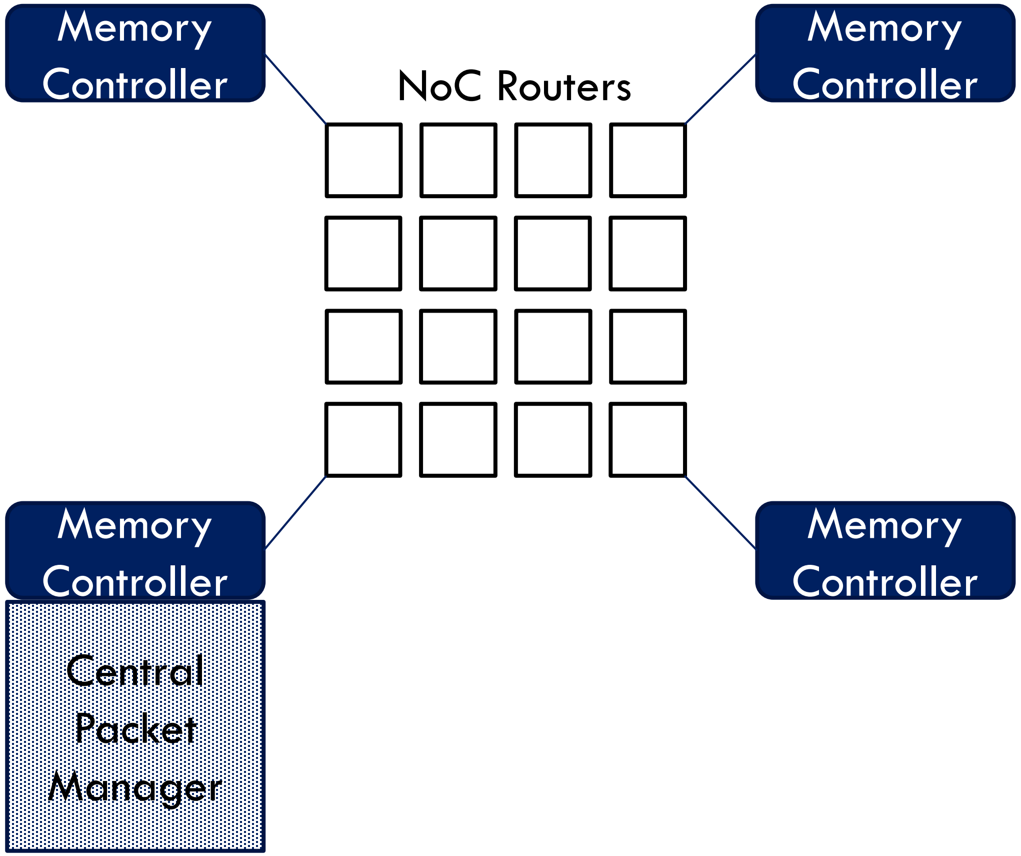
SnackNoC System Overview

- Added components to a traditional NoC



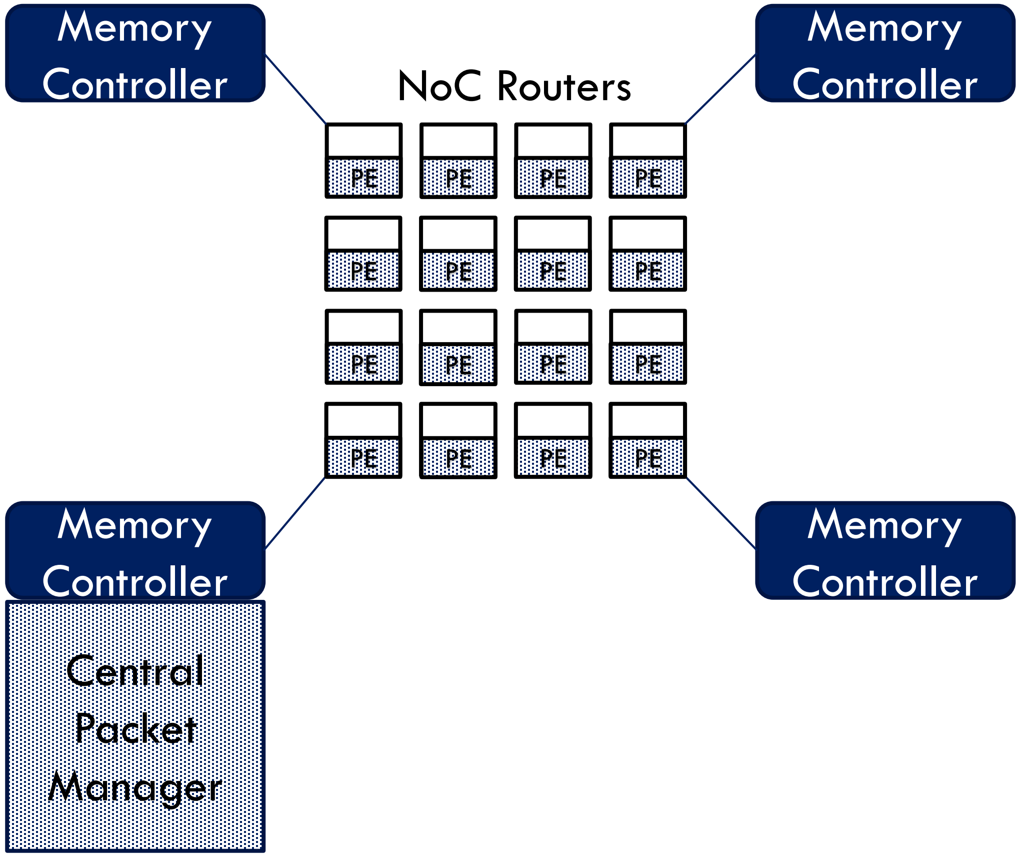
SnackNoC System Overview

- Added components to a traditional NoC
  - Central Packet Manager
    - Assemble and issue instruction packets
    - Manages execution state of kernels
    - Located at Memory Controller



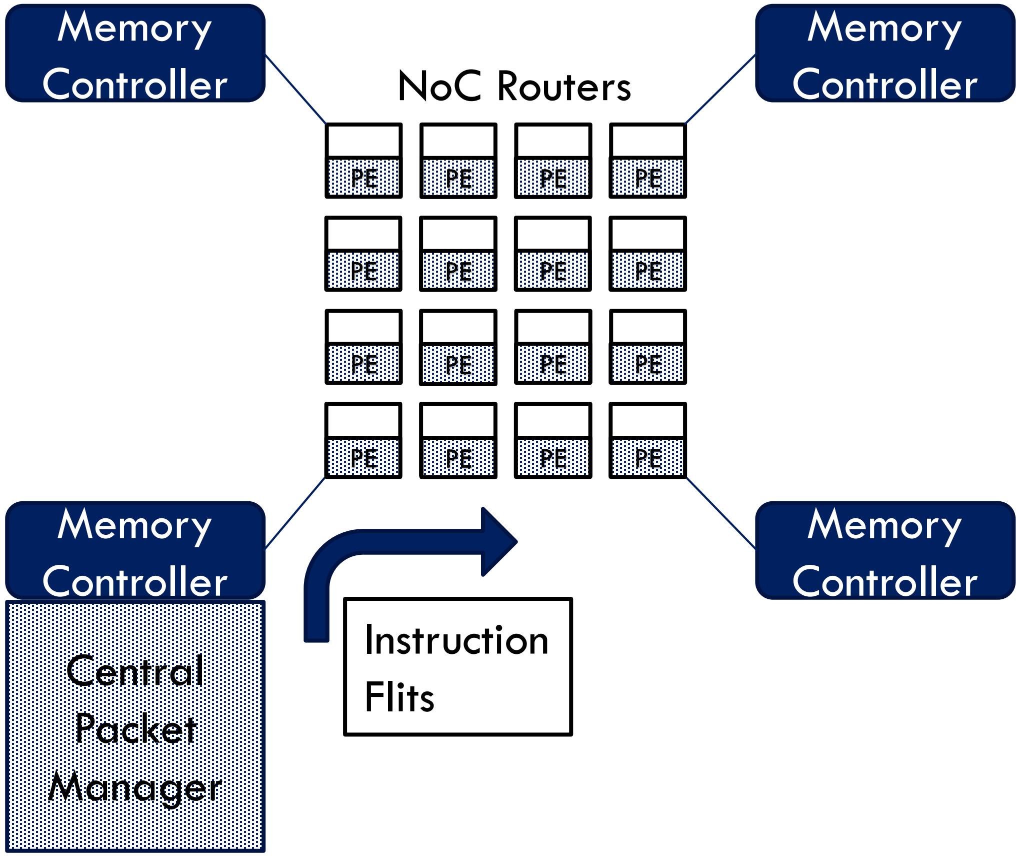
SnackNoC System Overview

- Added components to a traditional NoC
  - Central Packet Manager
    - Assemble and issue instruction packets
    - Manages execution state of kernels
    - Located at Memory Controller
  - Router Compute Units (RCU)
    - Light-weight accumulator-based processing element (PE)
      - Instruction buffering
      - ALU
    - Located in router pipeline



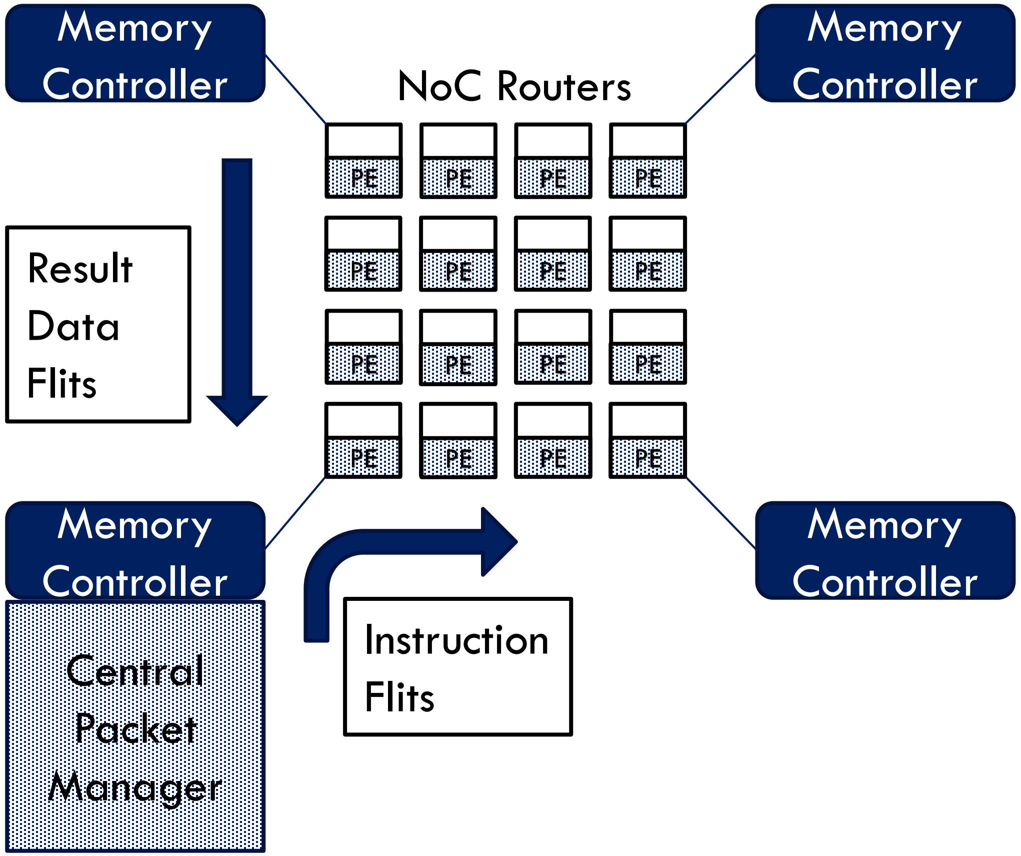
SnackNoC System Overview

- Added components to a traditional NoC
  - Central Packet Manager
    - Assemble and issue instruction packets
    - Manages execution state of kernels
    - Located at Memory Controller
  - Router Compute Units (RCU)
    - Light-weight accumulator-based processing element (PE)
      - Instruction buffering
      - ALU
    - Located in router pipeline



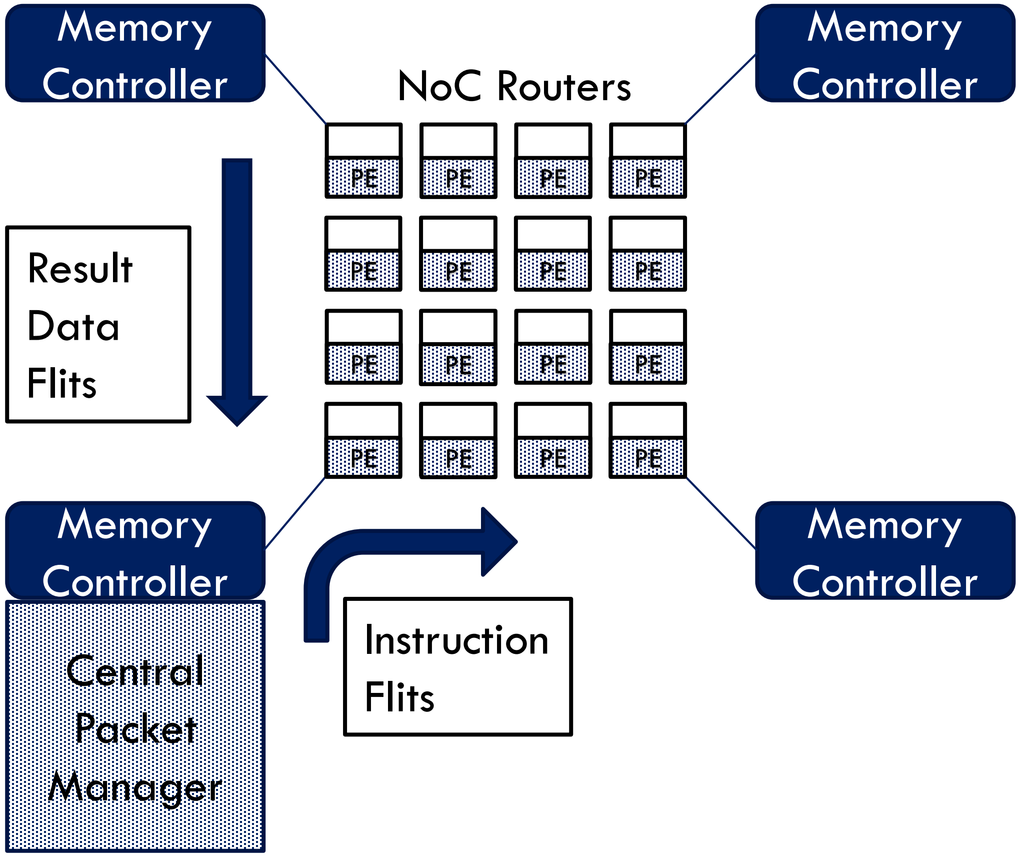
SnackNoC System Overview

- Added components to a traditional NoC
  - Central Packet Manager
    - Assemble and issue instruction packets
    - Manages execution state of kernels
    - Located at Memory Controller
  - Router Compute Units (RCU)
    - Light-weight accumulator-based processing element (PE)
      - Instruction buffering
      - ALU
    - Located in router pipeline



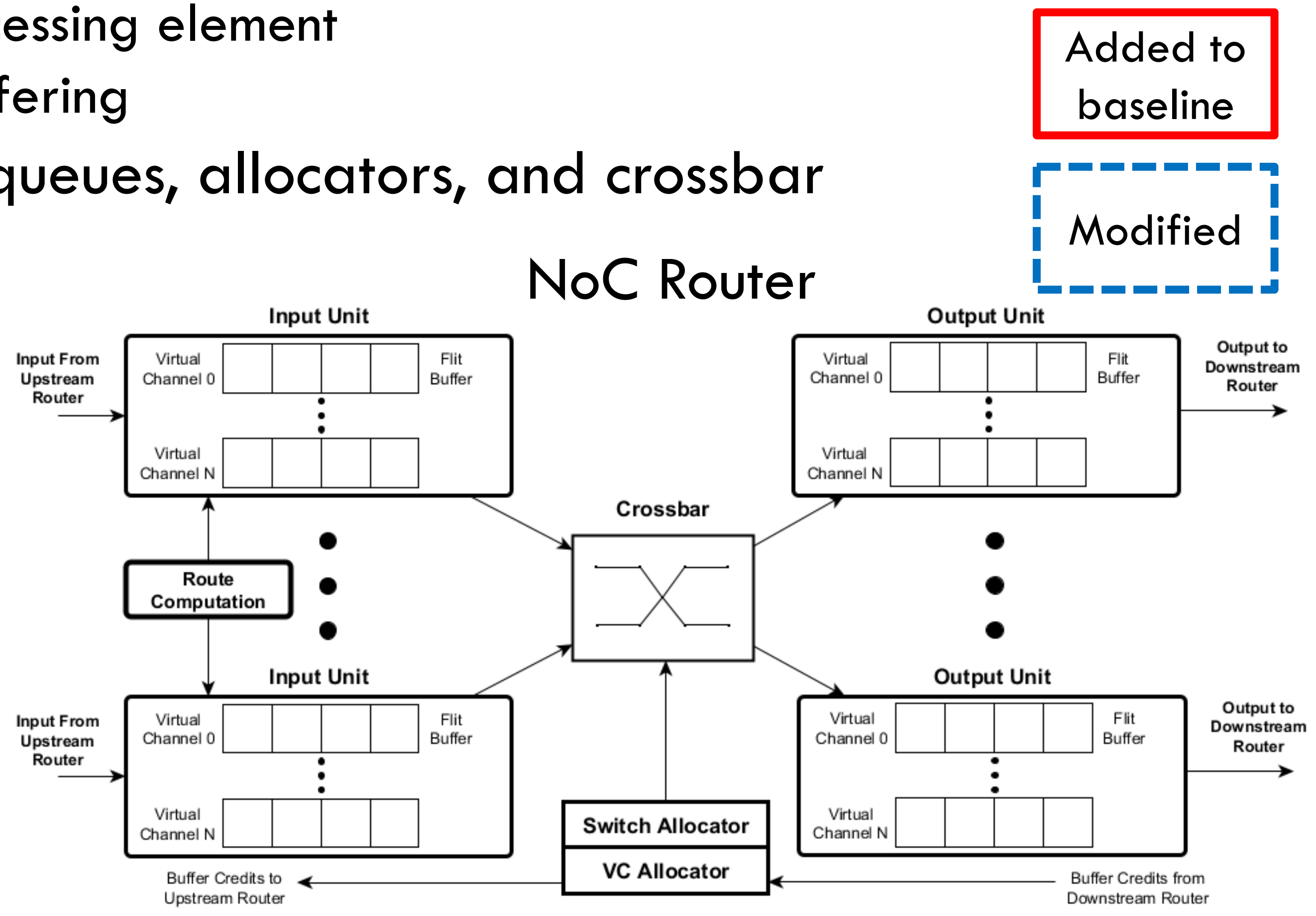
SnackNoC System Overview

- Added components to a traditional NoC
  - Central Packet Manager
    - Assemble and issue instruction packets
    - Manages execution state of kernels
    - Located at Memory Controller
  - Router Compute Units (RCU)
    - Light-weight accumulator-based processing element (PE)
      - Instruction buffering
      - ALU
    - Located in router pipeline
- Added features to a traditional NoC:
  - CPU traffic priority arbitration
  - Available NoC buffers as transient data storage



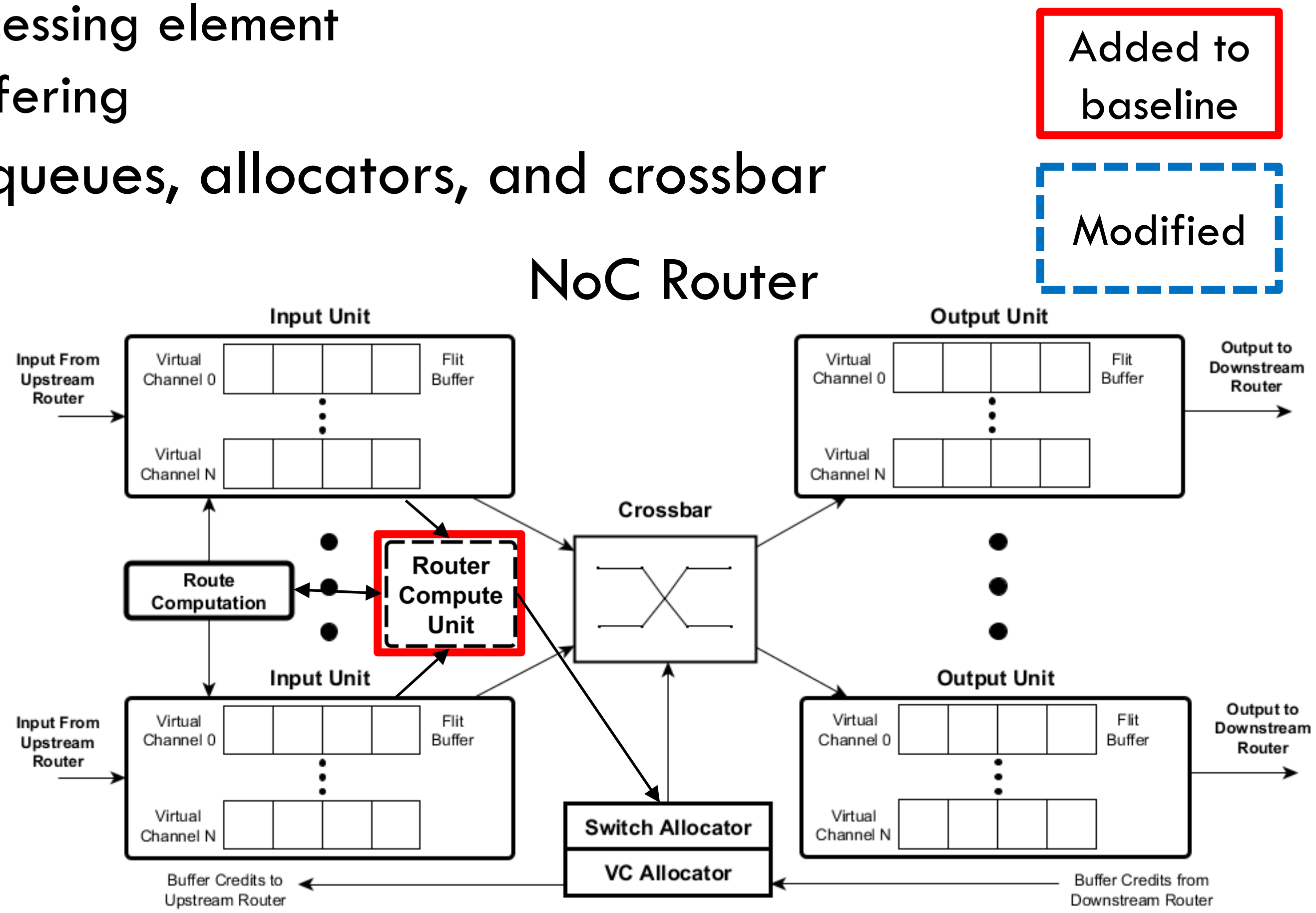
NoC Router Modification and RCU Additions

- Router Compute Units (RCUs)
  - 32-bit accumulator-based processing element
  - Instruction re-ordering and buffering
- Modifications to input buffer queues, allocators, and crossbar



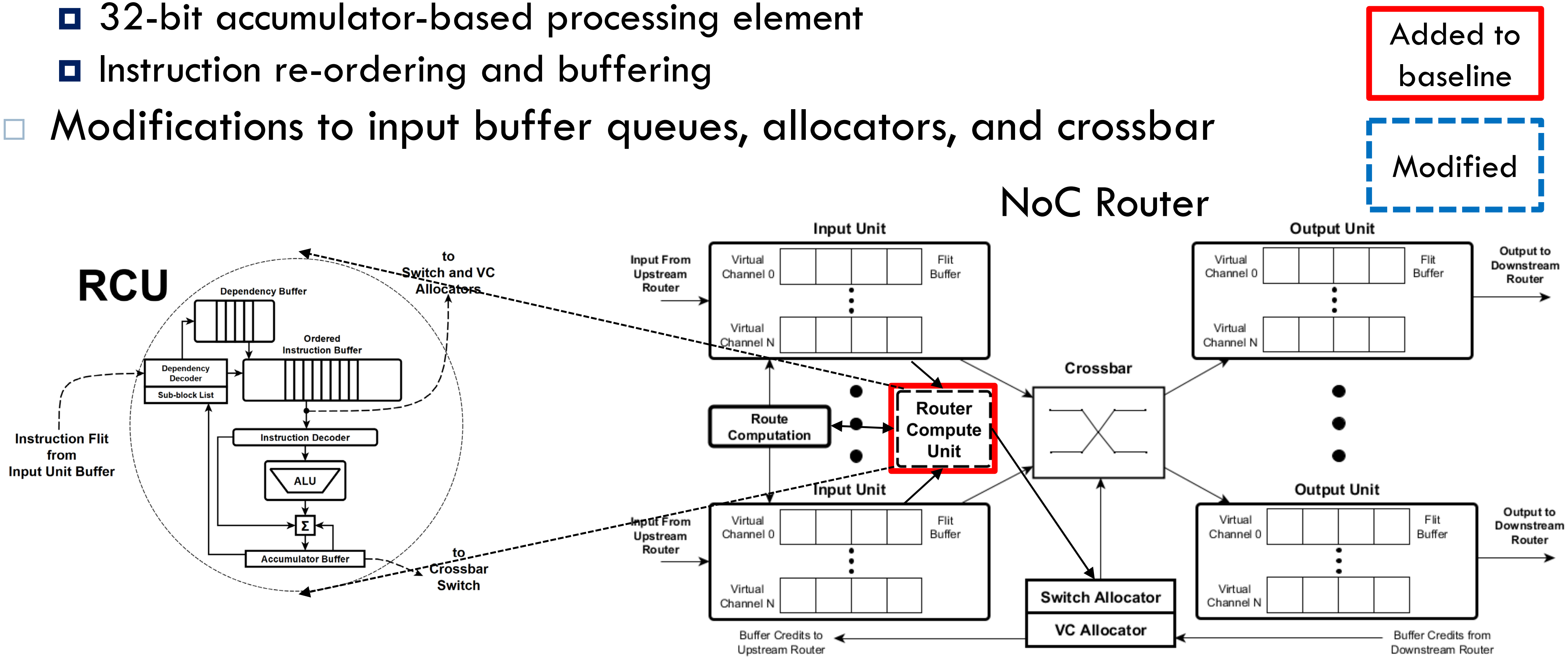
NoC Router Modification and RCU Additions

- Router Compute Units (RCUs)
  - 32-bit accumulator-based processing element
  - Instruction re-ordering and buffering
- Modifications to input buffer queues, allocators, and crossbar



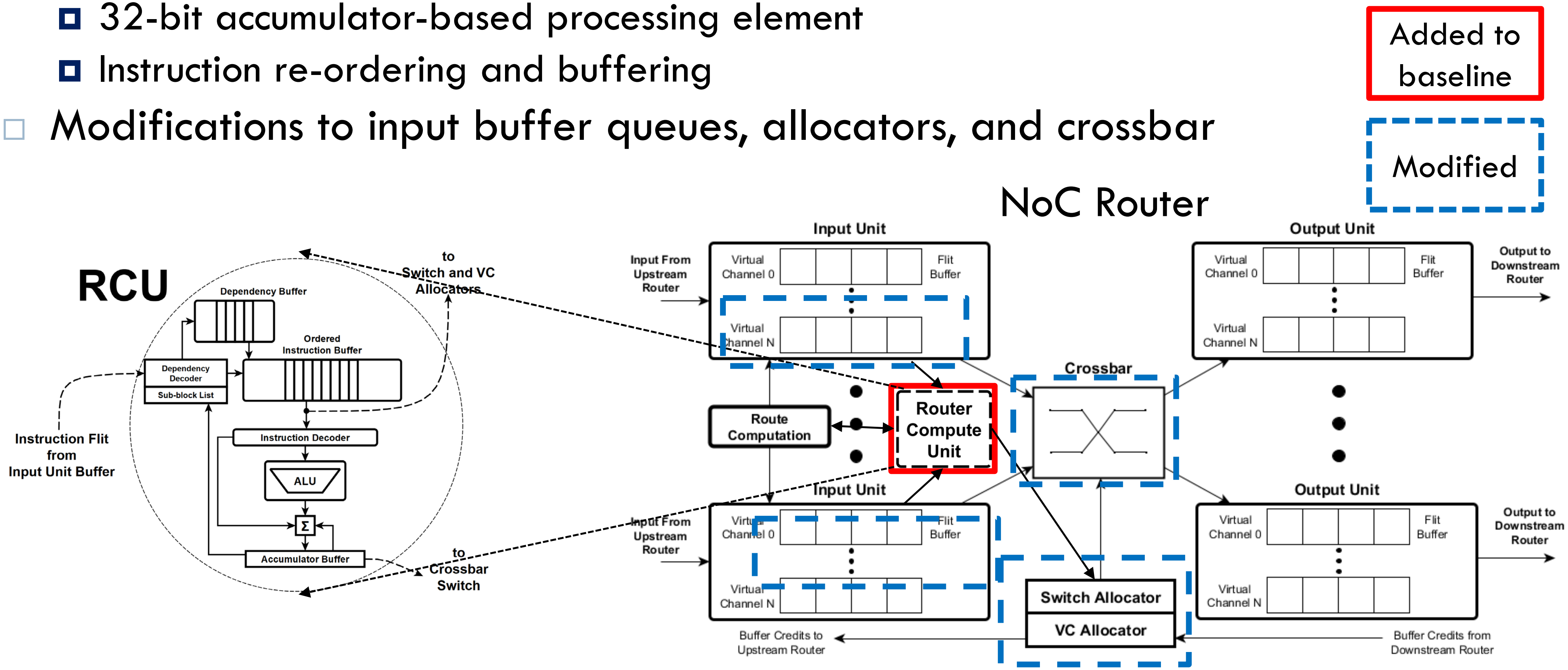
NoC Router Modification and RCU Additions

- Router Compute Units (RCUs)
  - 32-bit accumulator-based processing element
  - Instruction re-ordering and buffering
- Modifications to input buffer queues, allocators, and crossbar



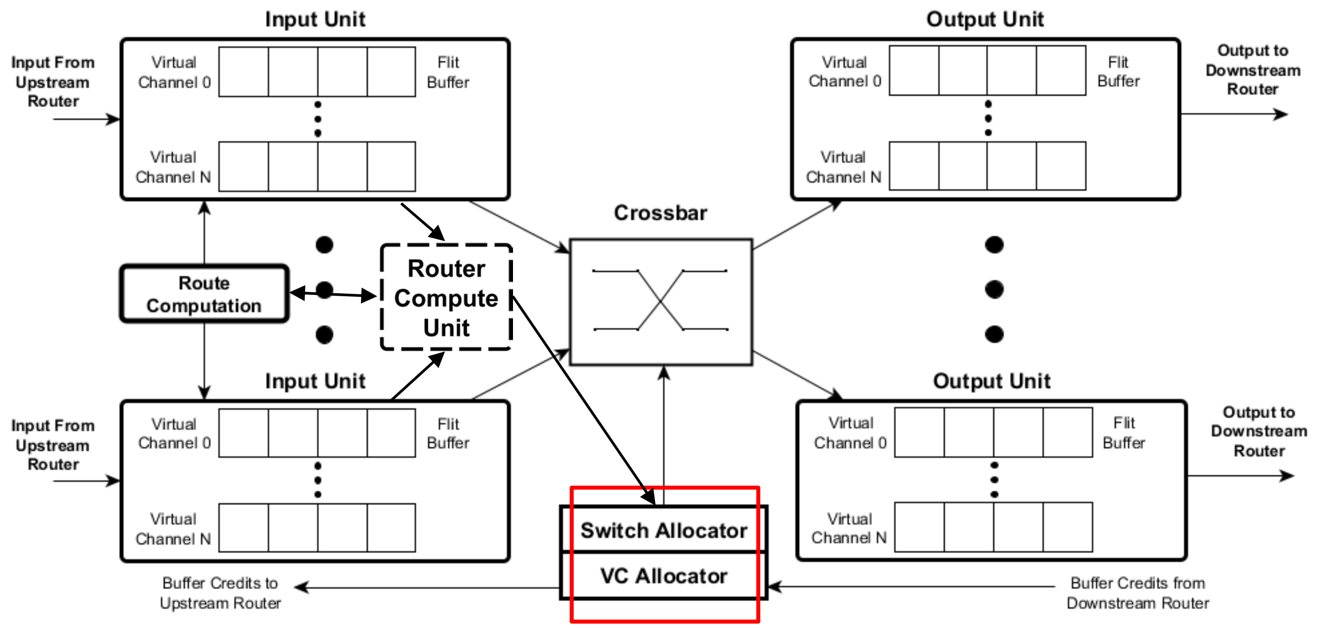
NoC Router Modification and RCU Additions

- Router Compute Units (RCUs)
  - 32-bit accumulator-based processing element
  - Instruction re-ordering and buffering
- Modifications to input buffer queues, allocators, and crossbar



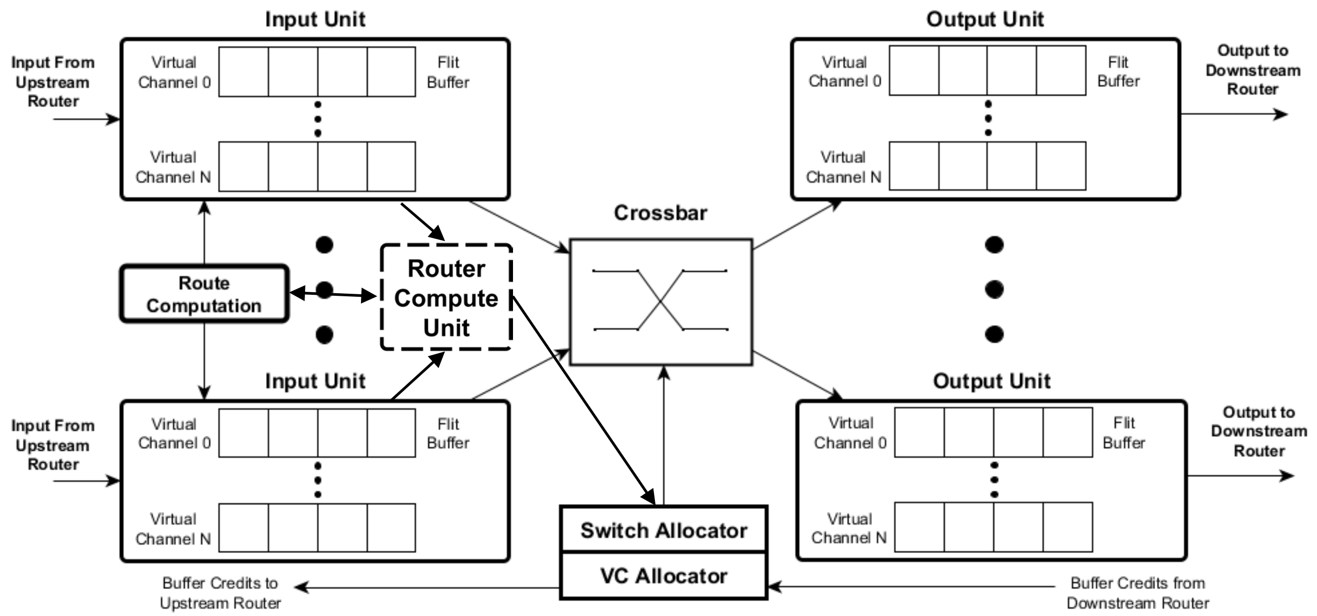
CPU Traffic Priority Arbitration

- Primary functionality of NoC is to transfer CPU core and memory traffic
  - “Fair” allocators are typically set to select traffic in round-robin
  - Allocators are modified to prioritize CPU traffic over SnackNoC instruction or data traffic



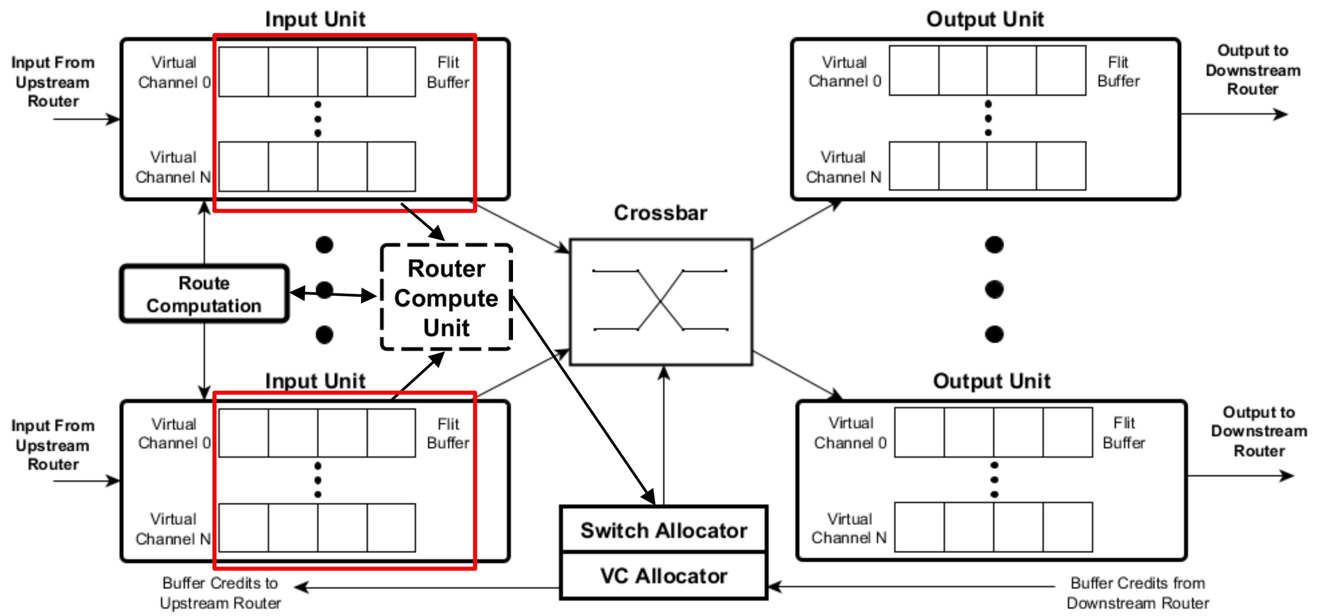
Transient Data Storage

- Input buffers typically have low contention
  - Available buffers and bandwidth can be used as transient storage
    - Useful to keep intermediate results and read-only values on chip



Transient Data Storage

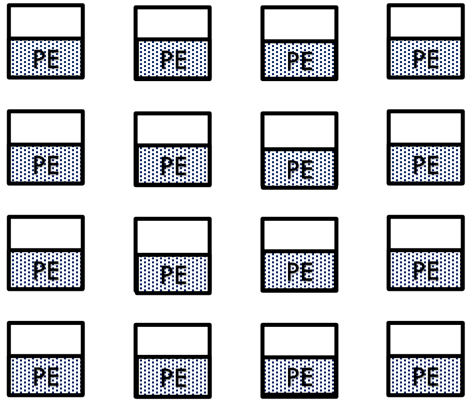
- Input buffers typically have low contention
  - Available buffers and bandwidth can be used as transient storage
    - Useful to keep intermediate results and read-only values on chip

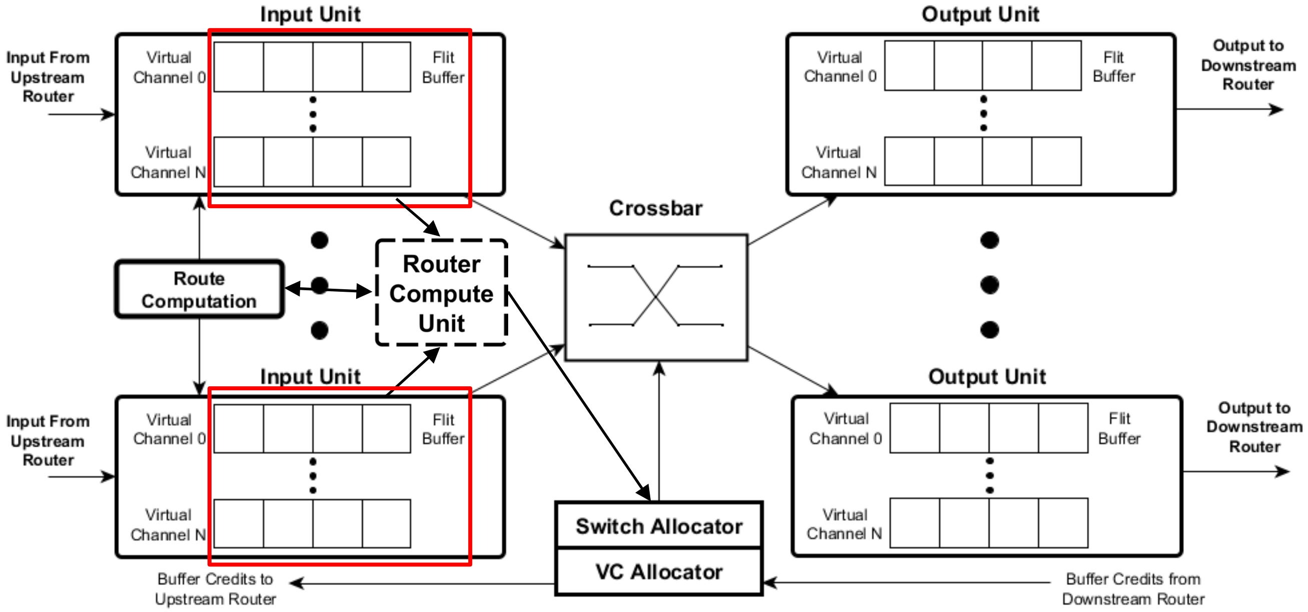


Transient Data Storage

- Input buffers typically have low contention
  - Available buffers and bandwidth can be used as transient storage
    - Useful to keep intermediate results and read-only values on chip

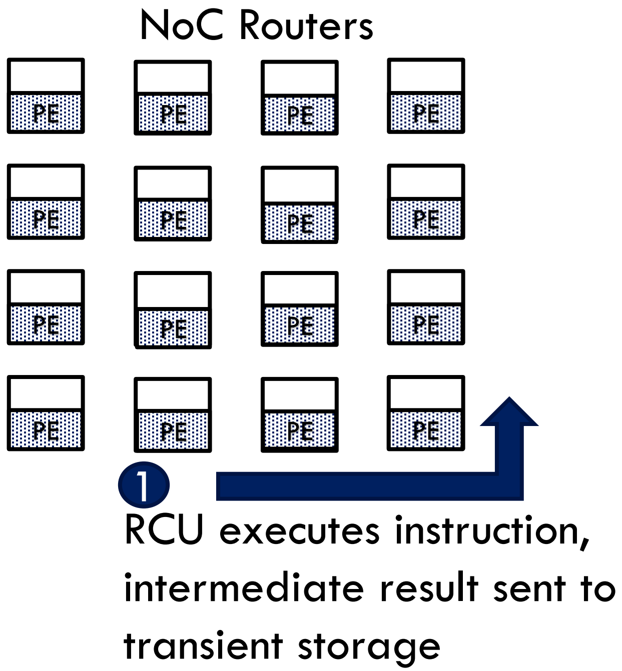
NoC Routers

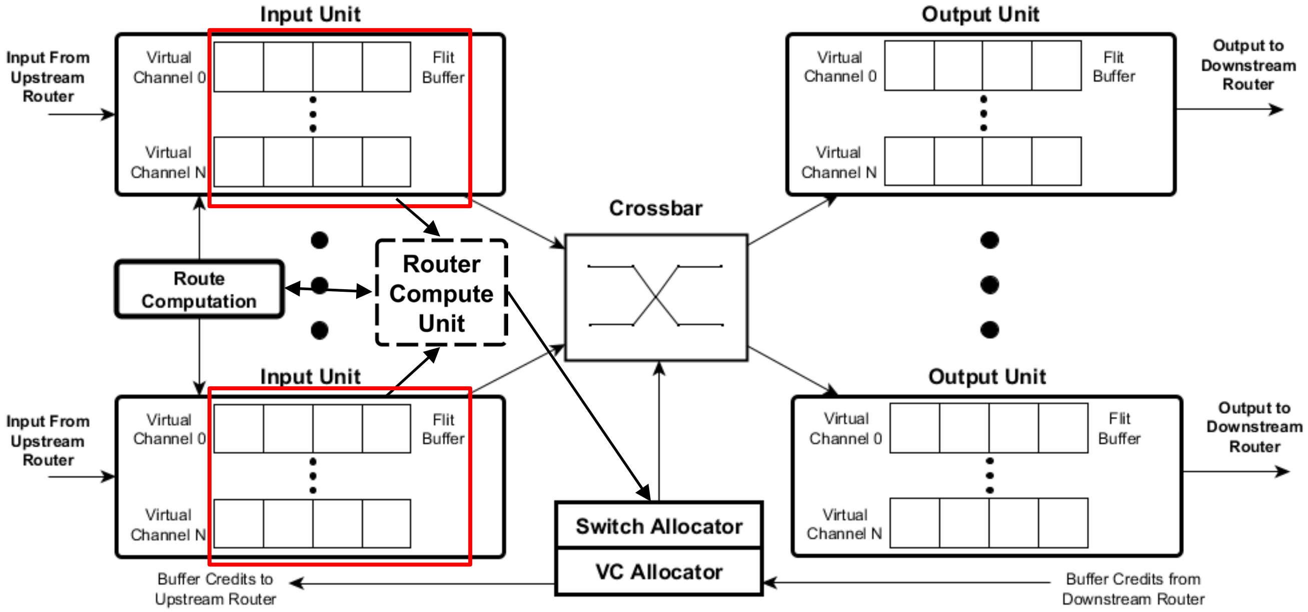




Transient Data Storage

- Input buffers typically have low contention
  - Available buffers and bandwidth can be used as transient storage
    - Useful to keep intermediate results and read-only values on chip

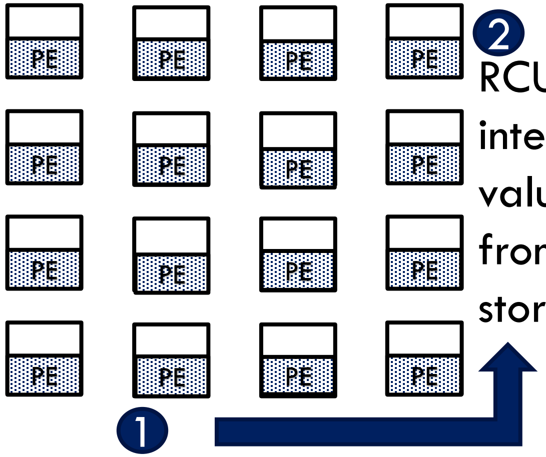




Transient Data Storage

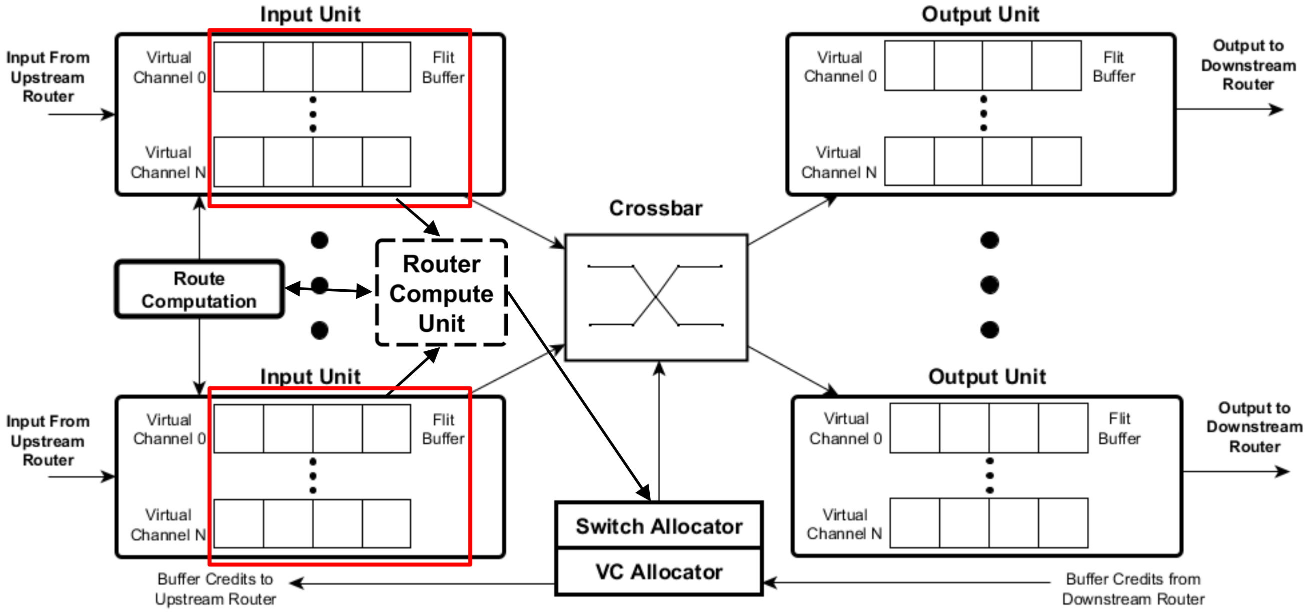
- Input buffers typically have low contention
  - Available buffers and bandwidth can be used as transient storage
    - Useful to keep intermediate results and read-only values on chip

NoC Routers



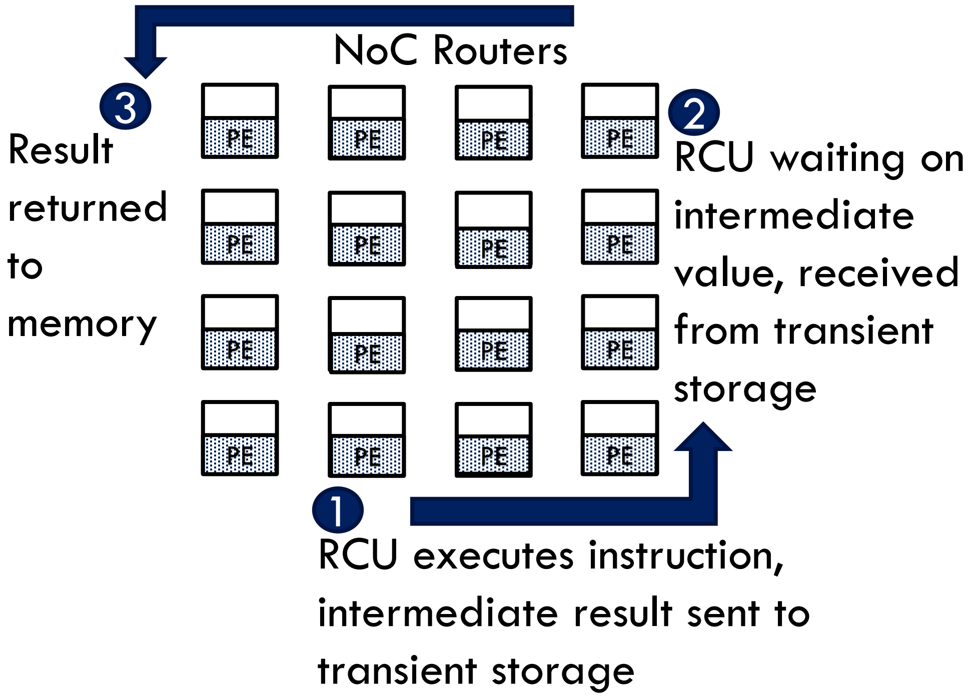
2 RCU waiting on intermediate value, received from transient storage

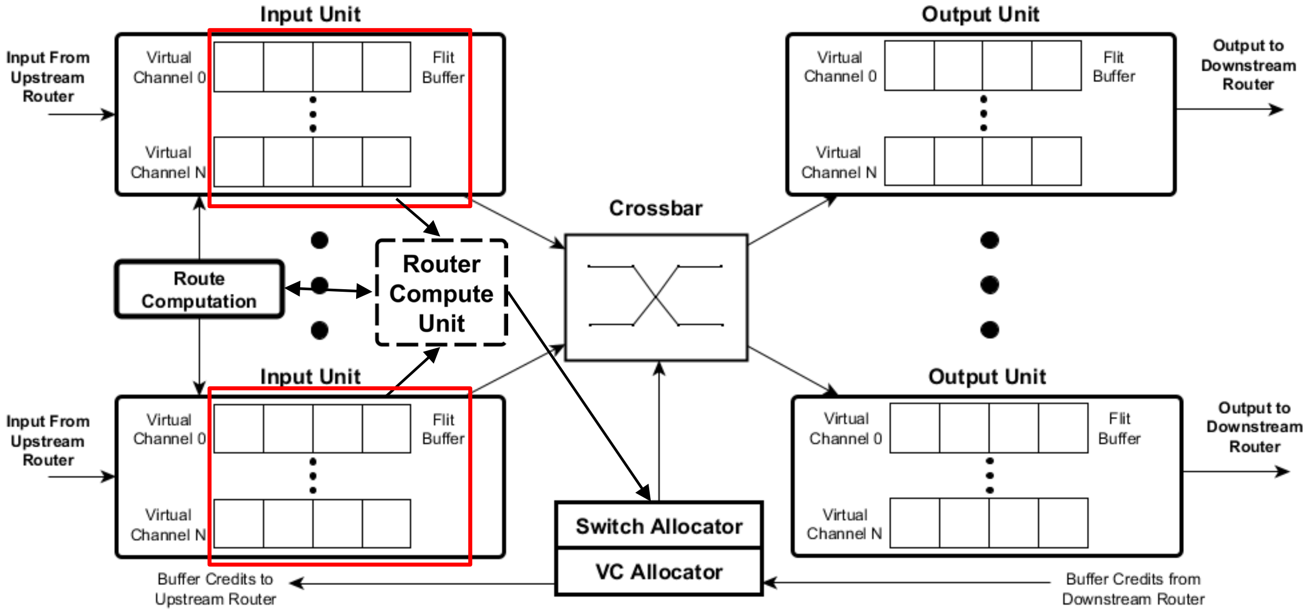
1 RCU executes instruction, intermediate result sent to transient storage



Transient Data Storage

- Input buffers typically have low contention
  - Available buffers and bandwidth can be used as transient storage
    - Useful to keep intermediate results and read-only values on chip

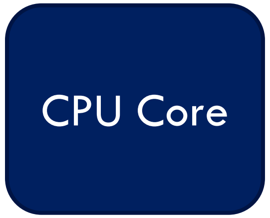




Running a SnackNoC Kernel

❶ C-code APIs for Matrix-multiply

```
...
snA = sn_create_mat(cxt, "A", A, l, m);
snB = sn_create_mat(cxt, "B", B, m, n);
snC = sn_create_mat_mul(cxt, snA, snB);
...
```

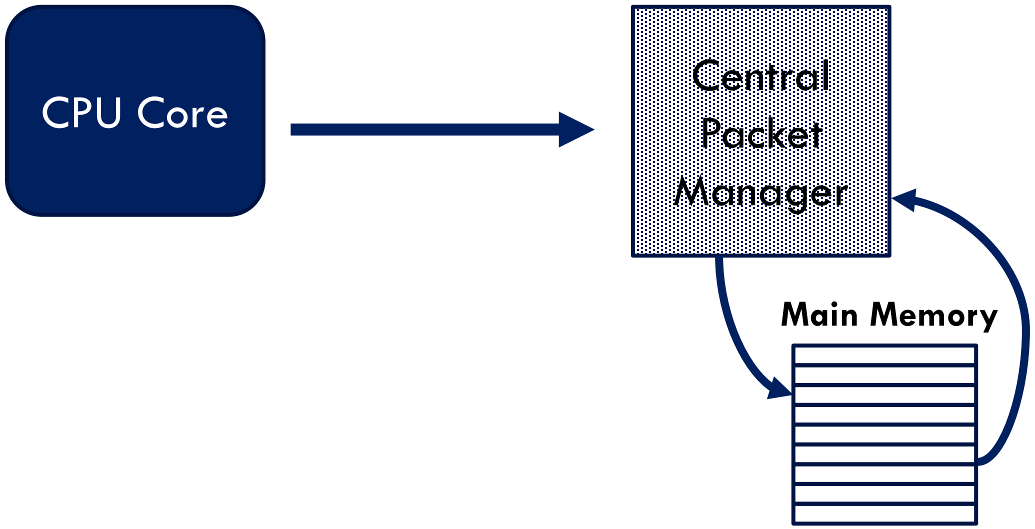


Running a SnackNoC Kernel

❶ C-code APIs for Matrix-multiply

```
…
snA = sn_create_mat(cxt, "A", A, l, m);
snB = sn_create_mat(cxt, "B", B, m, n);
snC = sn_create_mat_mul(cxt, snA, snB);
…
```

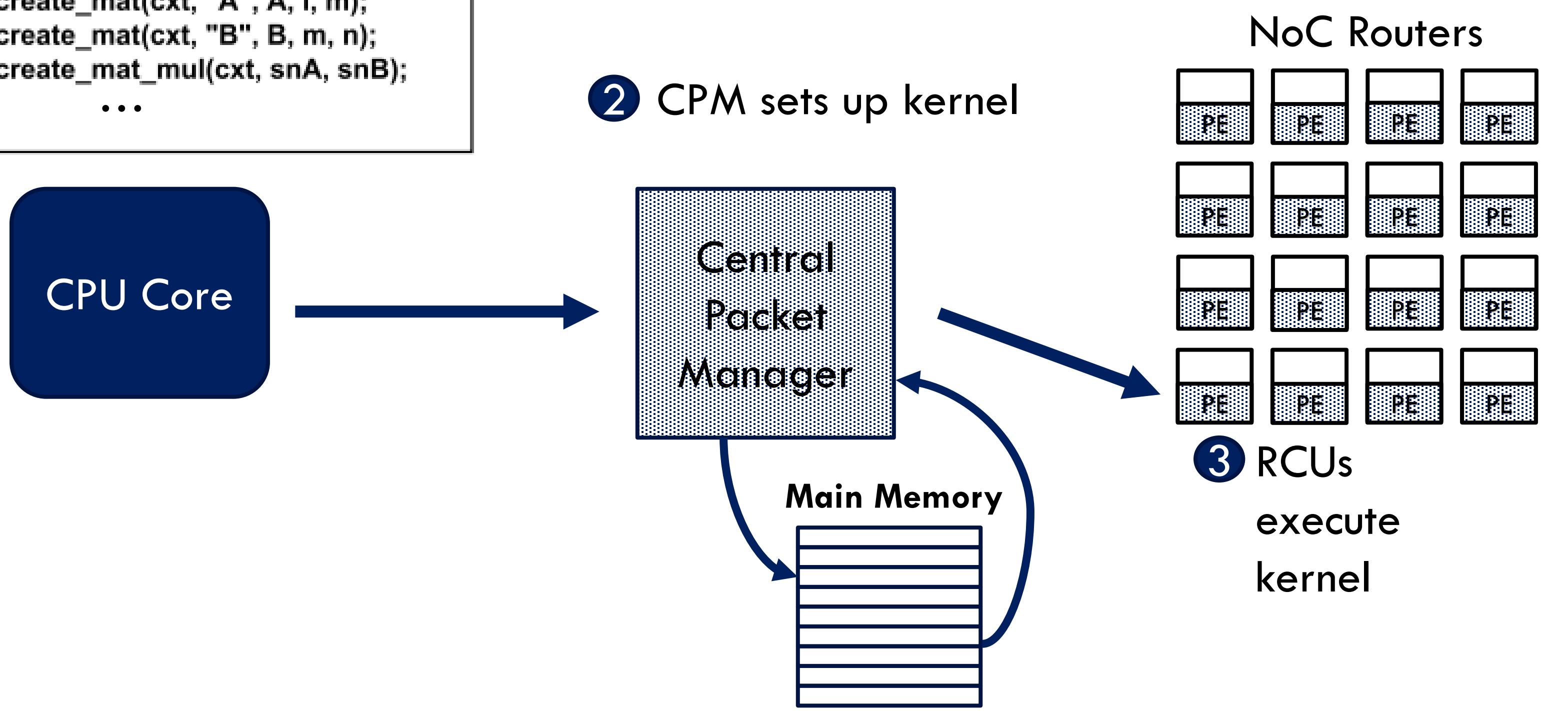
❷ CPM sets up kernel



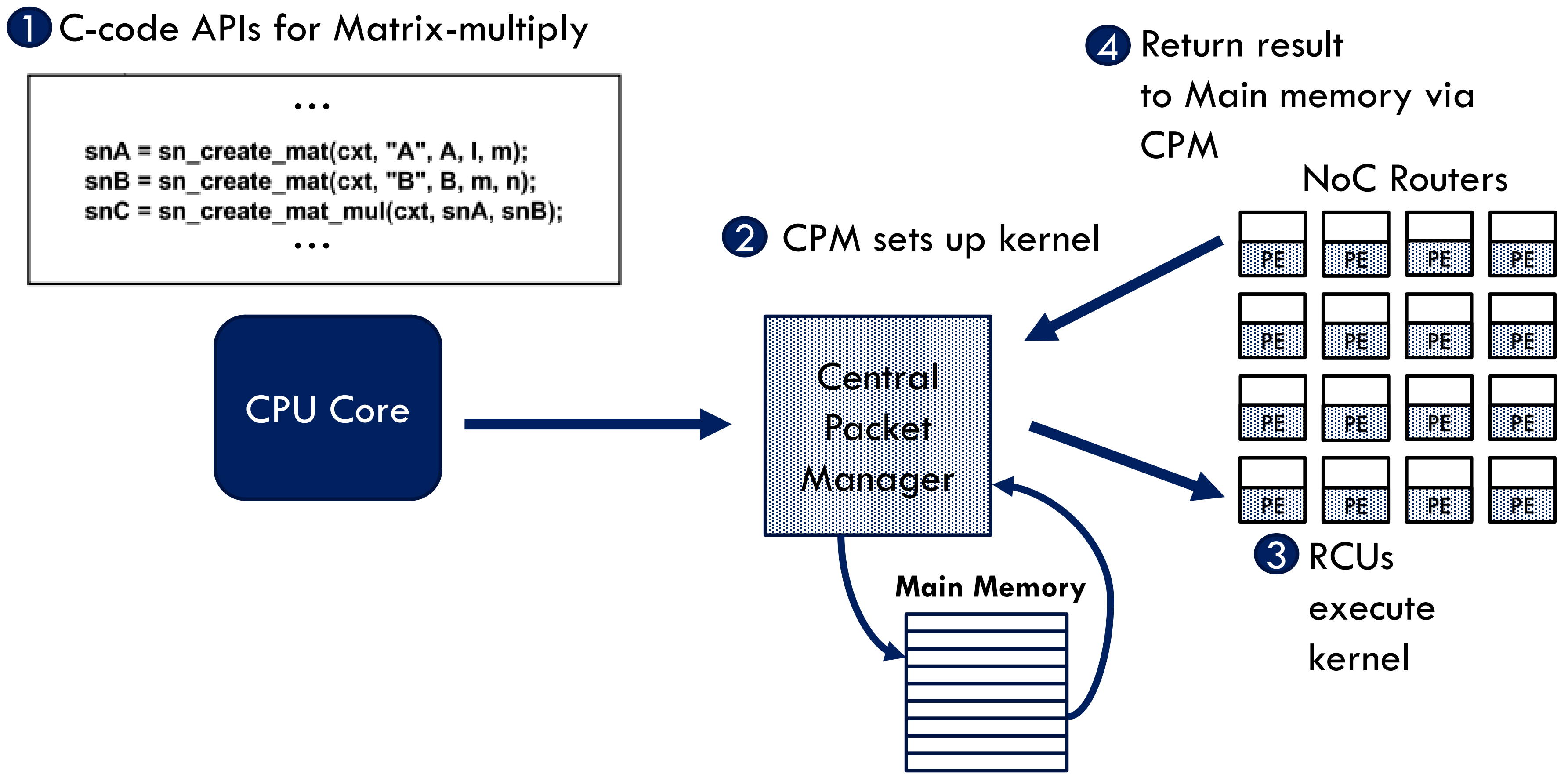
Running a SnackNoC Kernel

❶ C-code APIs for Matrix-multiply

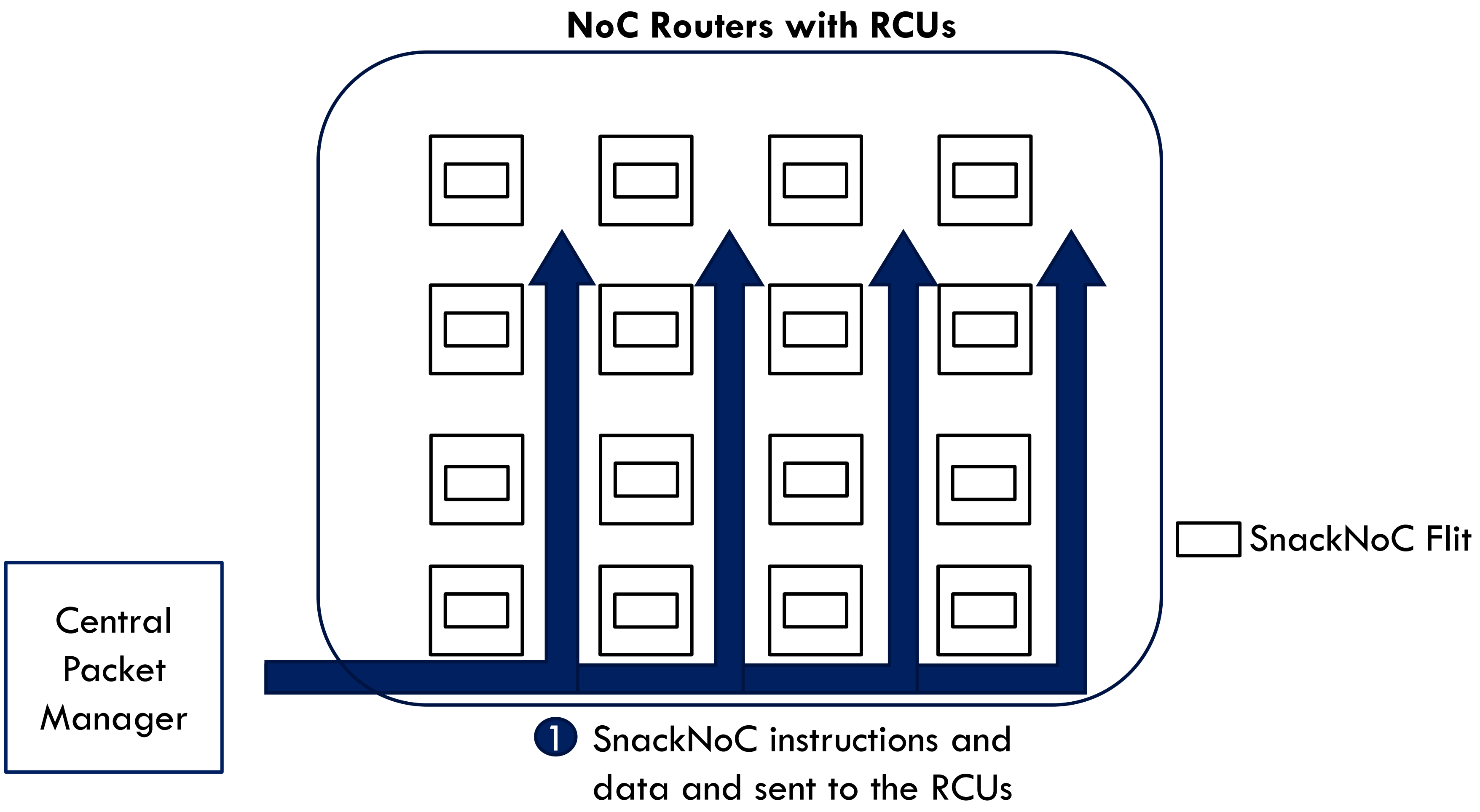
```
…
snA = sn_create_mat(cxt, "A", A, l, m);
snB = sn_create_mat(cxt, "B", B, m, n);
snC = sn_create_mat_mul(cxt, snA, snB);
…
```



Running a SnackNoC Kernel



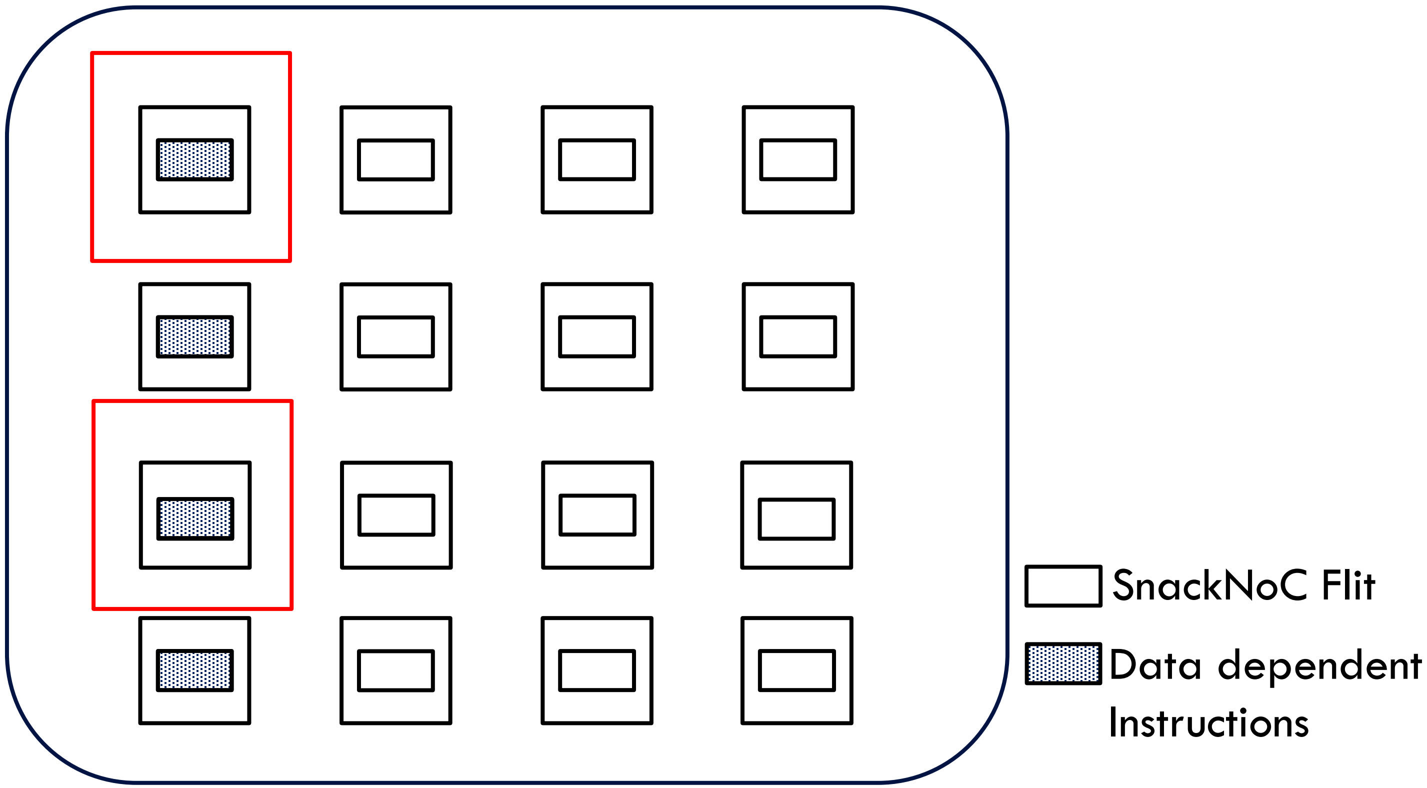
Example of a Reduction Kernel



Example of a Reduction Kernel

NoC Routers with RCUs

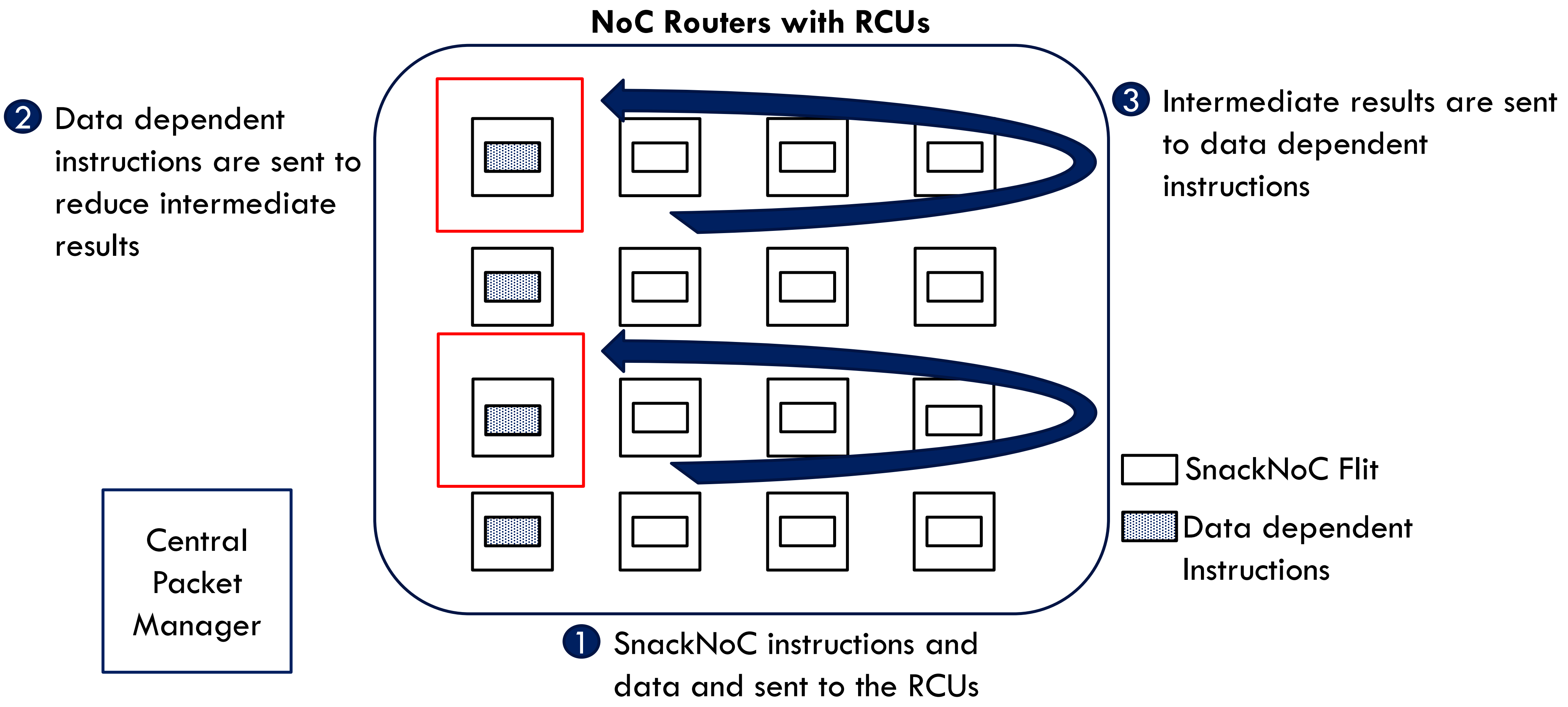
❷ Data dependent instructions are sent to reduce intermediate results



Central Packet Manager

❶ SnackNoC instructions and data and sent to the RCUs

Example of a Reduction Kernel

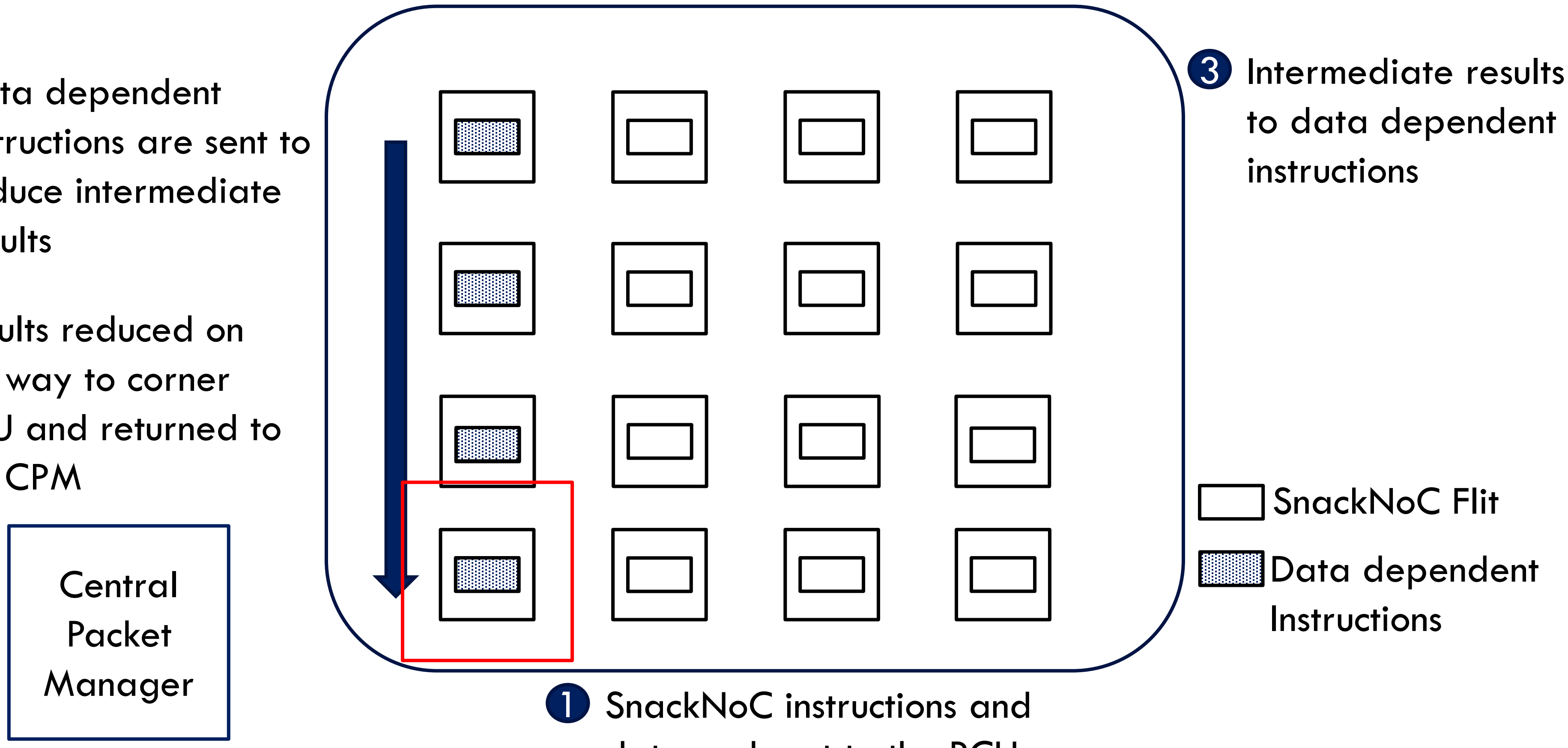


Example of a Reduction Kernel

NoC Routers with RCUs

❷ Data dependent instructions are sent to reduce intermediate results

❹ Results reduced on the way to corner RCU and returned to the CPM



❸ Intermediate results are sent to data dependent instructions

❶ SnackNoC instructions and data and sent to the RCUs

Example of a Reduction Kernel

NoC Routers with RCUs

❷ Data dependent instructions are sent to reduce intermediate results

❹ Results reduced on the way to corner RCU and returned to the CPM

Central Packet Manager

❸ Intermediate results are sent to data dependent instructions

Repurposed our NoC router crossbar, network links, and internal buffers to compute this kernel.

SnackNoC Flit

Data dependent Instructions

❶ SnackNoC instructions and data and sent to the RCUs

Overview

- “Slack” of the Communication Fabric
- The SnackNoC Platform
- **Experimental Results**
- Conclusion and Future Considerations

Methodology

- Experiments:
  1. Assess the performance of SnackNoC
     - How many additional cores worth of performance can SnackNoC provide opportunistically?
  2. Quantify the performance interference of operating SnackNoC on the CPU cores

Implemented four SnackNoC kernels (SGEMM, Reduction, MAC, SPMV)

Executed 16 multi-threaded benchmarks from PARSEC3, Splash2X, FastForward2 to assess performance interference

Methodology – Quantifying SnackNoC Performance

Methodology – Quantifying SnackNoC Performance

- SnackNoC is modeled in the gem5 simulation framework

Methodology – Quantifying SnackNoC Performance

- SnackNoC is modeled in the gem5 simulation framework
- To quantify performance, four SnackNoC kernels executed on:
  1. Simulated CMP with the SnackNoC platform
     - Compiled to SnackNoC instructions

Methodology – Quantifying SnackNoC Performance

- SnackNoC is modeled in the gem5 simulation framework
- To quantify performance, four SnackNoC kernels executed on:
  1. Simulated CMP with the SnackNoC platform
     - Compiled to SnackNoC instructions

SnackNoC Parameters	Configuration
RCU Count	16 RCUs
RCU Freq.	1 GHz
Flit Priority Arbitration	ON/OFF

Methodology – Quantifying SnackNoC Performance

- SnackNoC is modeled in the gem5 simulation framework
- To quantify performance, four SnackNoC kernels executed on:
  1. Simulated CMP with the SnackNoC platform
     - Compiled to SnackNoC instructions

SnackNoC Parameters	Configuration
RCU Count	16 RCUs
RCU Freq.	1 GHz
Flit Priority Arbitration	ON/OFF
Simulated CMP Parameters	**Configuration**
Core Count	16 in-order cores
Core Frequency	2GHz
L1 I&D Cache	32KB, 4-way
L2 Cache	256KB, 4-way
NoC Topology	2D 4x4 Mesh, 4 Memory Controllers
NoC Flit Size	32B
# Virtual Channels	4
# Buffers	4

Methodology – Quantifying SnackNoC Performance

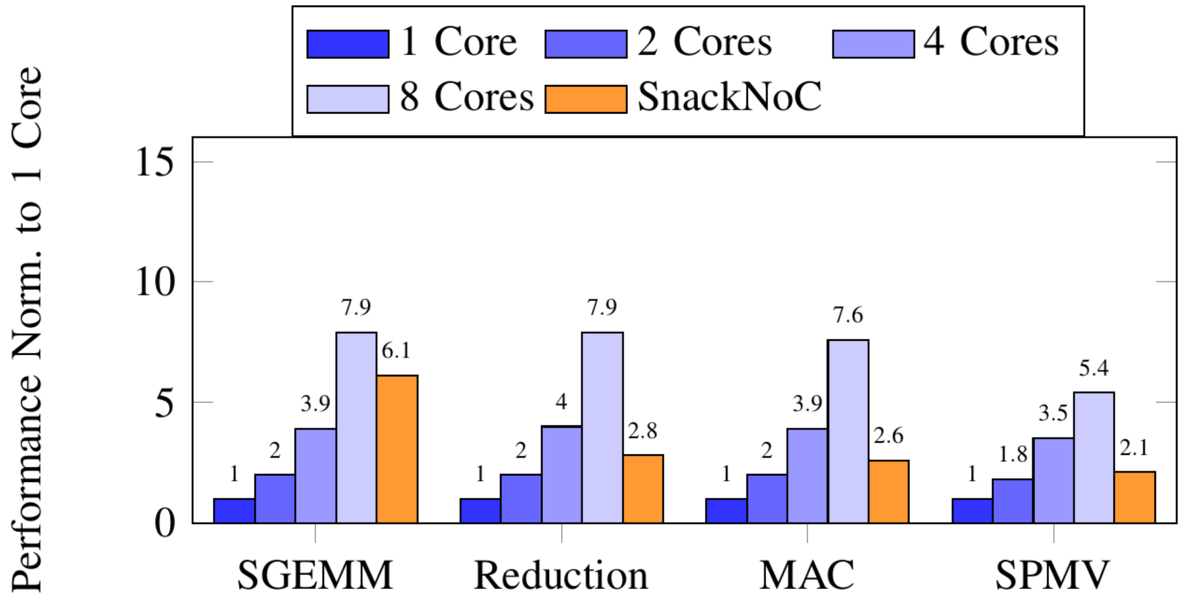
- SnackNoC is modeled in the gem5 simulation framework
- To quantify performance, four SnackNoC kernels executed on:
  1. Simulated CMP with the SnackNoC platform
     - Compiled to SnackNoC instructions
  2. Native Dell server with Intel Xeon E5-2660
     - C++ multi-threaded with OpenMP

SnackNoC Parameters	Configuration
RCU Count	16 RCUs
RCU Freq.	1 GHz
Flit Priority Arbitration	ON/OFF
Simulated CMP Parameters	**Configuration**
Core Count	16 in-order cores
Core Frequency	2GHz
L1 I&D Cache	32KB, 4-way
L2 Cache	256KB, 4-way
NoC Topology	2D 4x4 Mesh, 4 Memory Controllers
NoC Flit Size	32B
# Virtual Channels	4
# Buffers	4

Native CPU Parameters	Configuration
Processor	Intel Xeon E5-2660 v3
Core Frequency	2.6GHz
L1 I&D Cache	32KB, 8-way
L2 Cache	256KB, 8-way
L3 Cache	20MB, 20-way

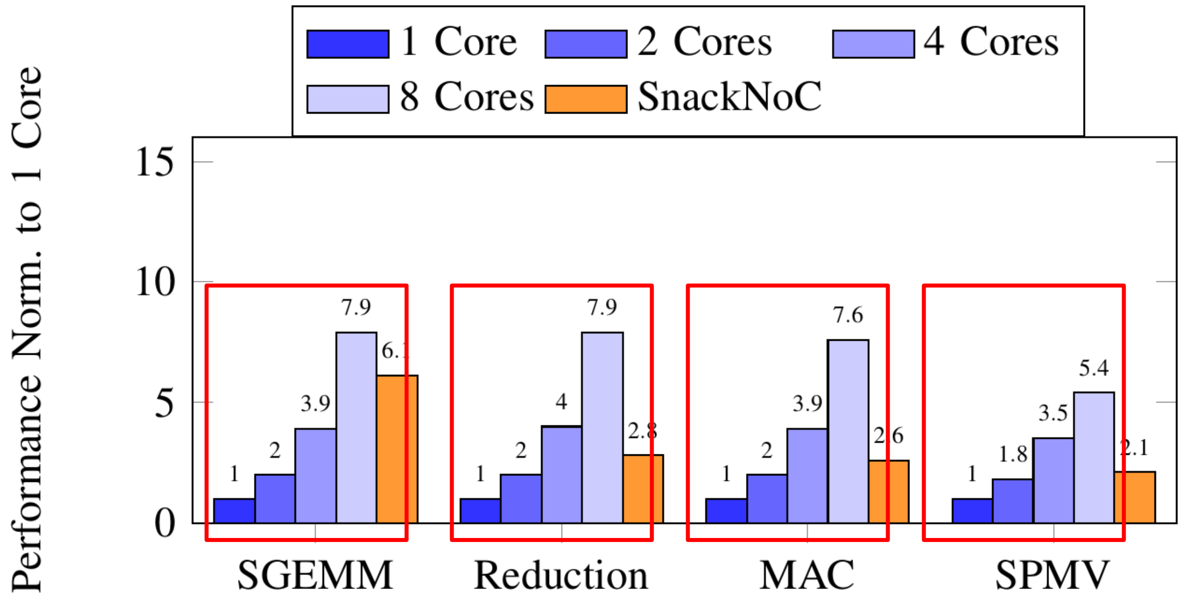
Quantifying SnackNoC Performance Gain

- SnackNoC kernels are executed on an increasing number of cores to determine comparable performance of SnackNoC



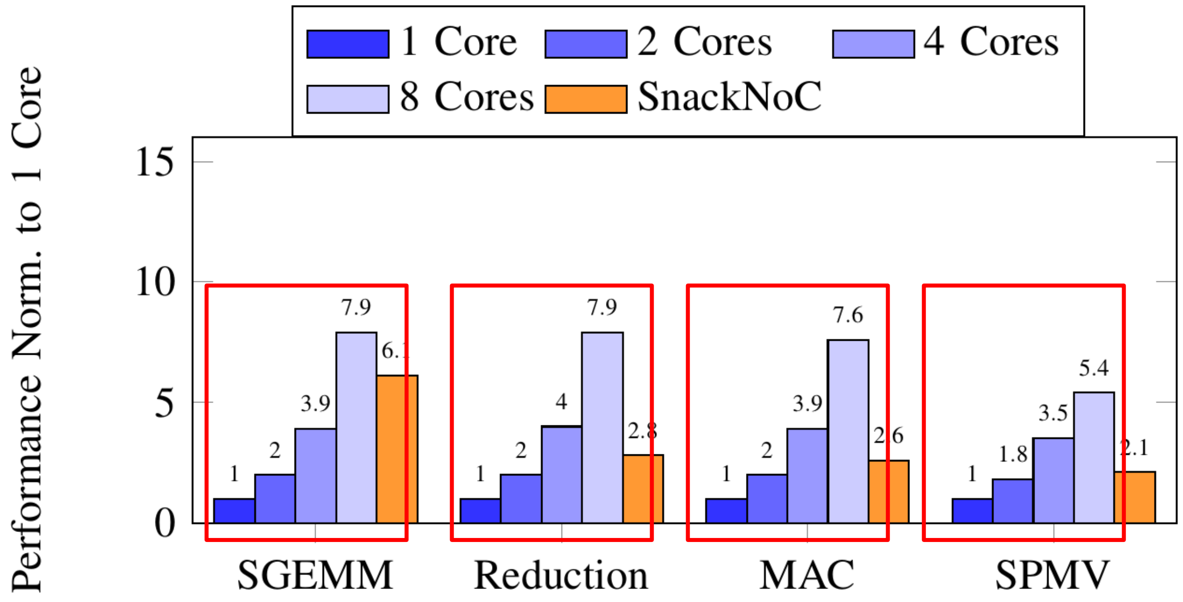
Quantifying SnackNoC Performance Gain

- SnackNoC kernels are executed on an increasing number of cores to determine comparable performance of SnackNoC



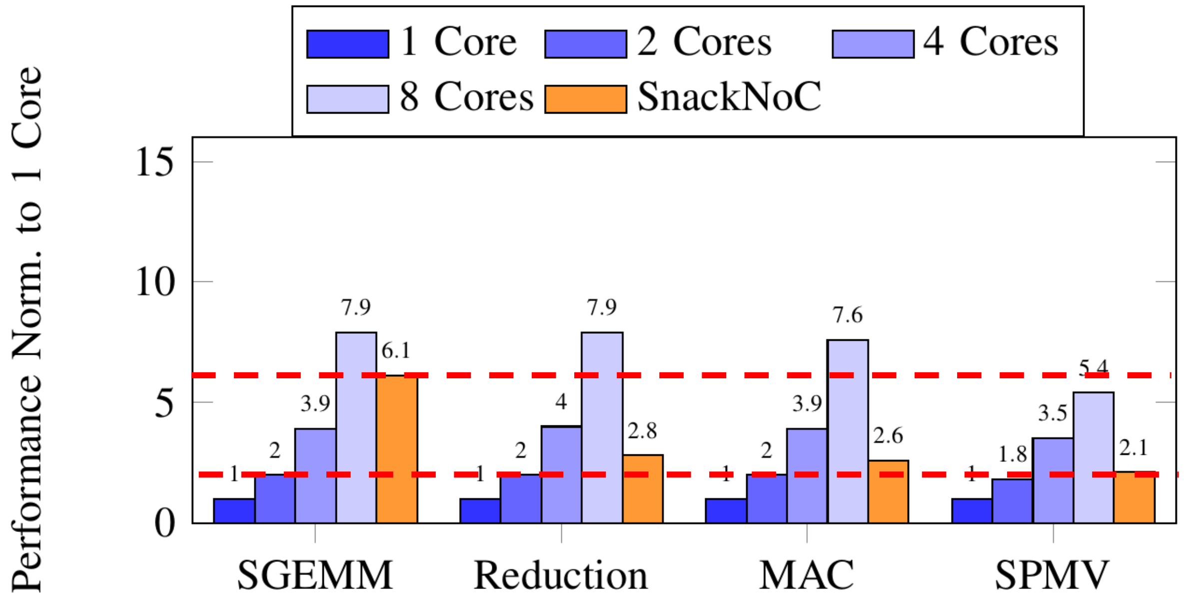
Quantifying SnackNoC Performance Gain

- SnackNoC kernels are executed on an increasing number of cores to determine comparable performance of SnackNoC
- CMP performance roughly linear increase with increasing cores, with exception to SPMV



Quantifying SnackNoC Performance Gain

- SnackNoC kernels are executed on an increasing number of cores to determine comparable performance of SnackNoC
- CMP performance roughly linear increase with increasing cores, with exception to SPMV
- Performance gain between 2 and 6 x86 OOO cores



SnackNoC Area and Power Overhead

- SnackNoC components' RTL implemented, synthesized with Synopsis Design Compiler:
  - 45nm NCSU technology node
  - Operating Freq. 1GHz

Router Control Unit (RCU)	Additional Power (%)	Additional Area (%)
32-bit Parallel Adder	1.14%	1.15%
32-bit Parallel Subtractor	1.14%	1.15%
32-bit Multiply and Accumulate (MAC)	2.05%	1.73%
Ordered Instruction Buffer	2.05%	2.30%
Dependency Buffer	2.51%	1.15%
Accumulator Buffer	0.68%	0.12%
Sub Block List	0.23%	1.73%
Total	**9.81%**	**9.33%**

Central Packet Manager	Additional Power (%)	Additional Area (%)
Assembly Logic and Buffers	0.08%	2.43%
Kernel State	0.16%	0.10%
Instruction Buffer	10.71%	25.75%
Offload Data Memory Buffer	0.95%	2.28%
Output Result FIFO	0.95%	2.28%
Total	**12.85%**	**33.04%**

SnackNoC Area and Power Overhead

- SnackNoC components' RTL implemented, synthesized with Synopsis Design Compiler:
  - 45nm NCSU technology node
  - Operating Freq. 1GHz
- Single RCU per NoC router
  - Under 10% additional power and area per router

Router Control Unit (RCU)	Additional Power (%)	Additional Area (%)
32-bit Parallel Adder	1.14%	1.15%
32-bit Parallel Subtractor	1.14%	1.15%
32-bit Multiply and Accumulate (MAC)	2.05%	1.73%
Ordered Instruction Buffer	2.05%	2.30%
Dependency Buffer	2.51%	1.15%
Accumulator Buffer	0.68%	0.12%
Sub Block List	0.23%	1.73%
Total	**9.81%**	**9.33%**

Central Packet Manager	Additional Power (%)	Additional Area (%)
Assembly Logic and Buffers	0.08%	2.43%
Kernel State	0.16%	0.10%
Instruction Buffer	10.71%	25.75%
Offload Data Memory Buffer	0.95%	2.28%
Output Result FIFO	0.95%	2.28%
Total	**12.85%**	**33.04%**

SnackNoC Area and Power Overhead

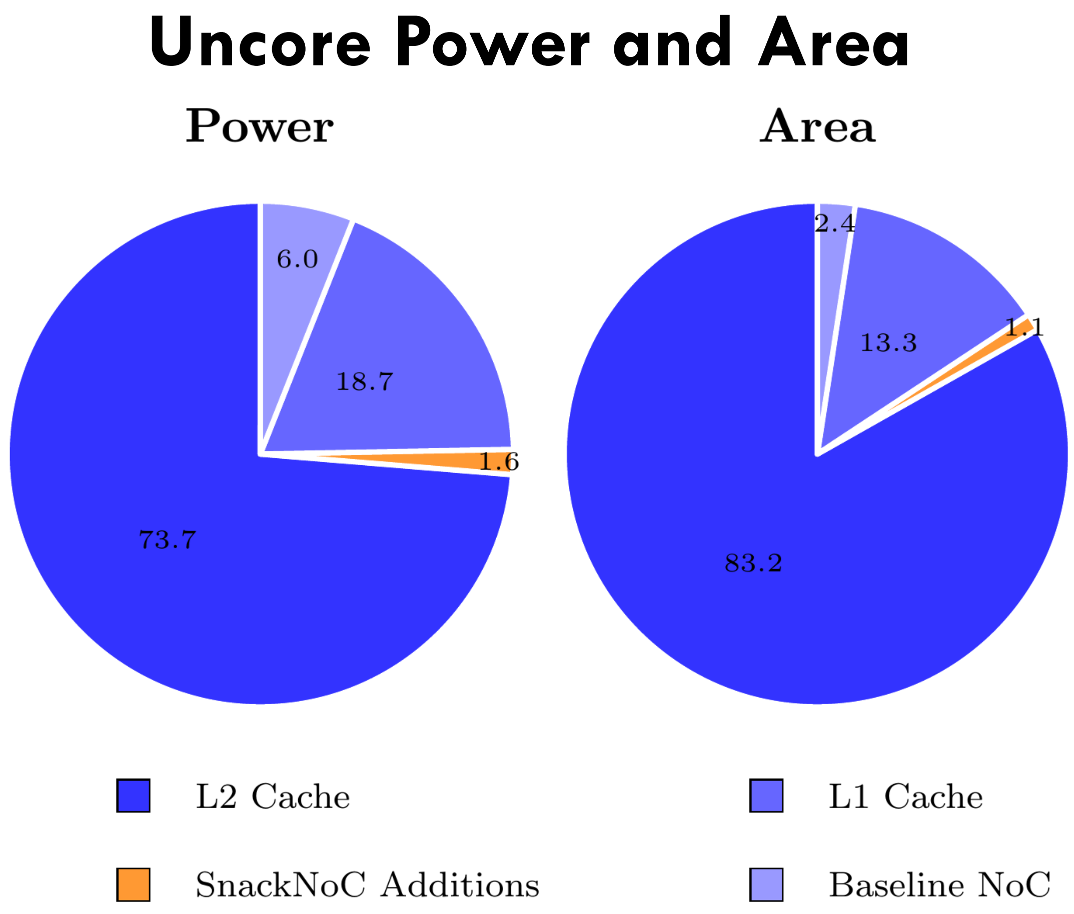
- SnackNoC components' RTL implemented, synthesized with Synopsis Design Compiler:
  - 45nm NCSU technology node
  - Operating Freq. 1GHz
- Single RCU per NoC router
  - Under 10% additional power and area per router
- Single CPM per NoC
  - 12.85% additional power per NoC
  - 33.04% additional area per NoC
    - Largest contributor is instruction buffer

Router Control Unit (RCU)	Additional Power (%)	Additional Area (%)
32-bit Parallel Adder	1.14%	1.15%
32-bit Parallel Subtractor	1.14%	1.15%
32-bit Multiply and Accumulate (MAC)	2.05%	1.73%
Ordered Instruction Buffer	2.05%	2.30%
Dependency Buffer	2.51%	1.15%
Accumulator Buffer	0.68%	0.12%
Sub Block List	0.23%	1.73%
Total	**9.81%**	**9.33%**

Central Packet Manager	Additional Power (%)	Additional Area (%)
Assembly Logic and Buffers	0.08%	2.43%
Kernel State	0.16%	0.10%
Instruction Buffer	10.71%	25.75%
Offload Data Memory Buffer	0.95%	2.28%
Output Result FIFO	0.95%	2.28%
Total	**12.85%**	**33.04%**

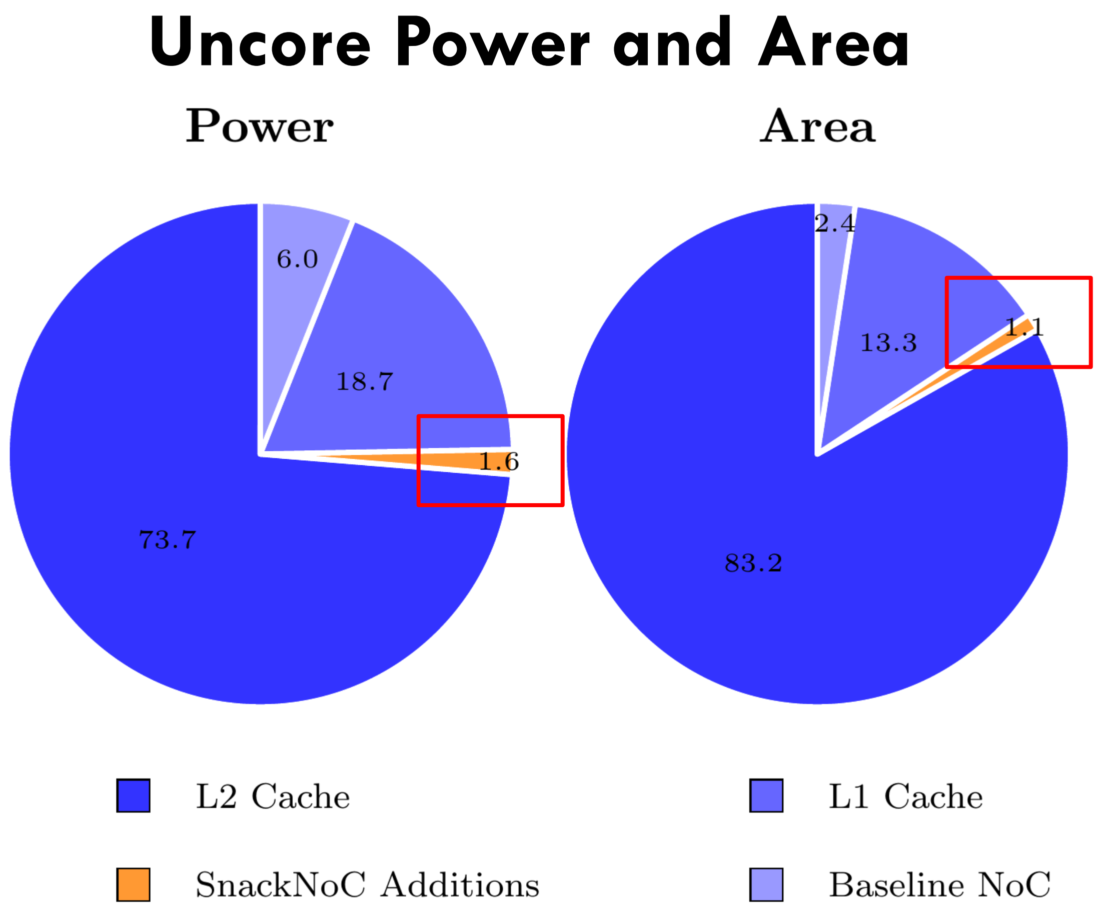
SnackNoC's Small Contribution to the Total Uncore

- Full uncore of 16 core CMP is modeled in 45nm with Cacti 7.0 and Orion 3.0.



SnackNoC's Small Contribution to the Total Uncore

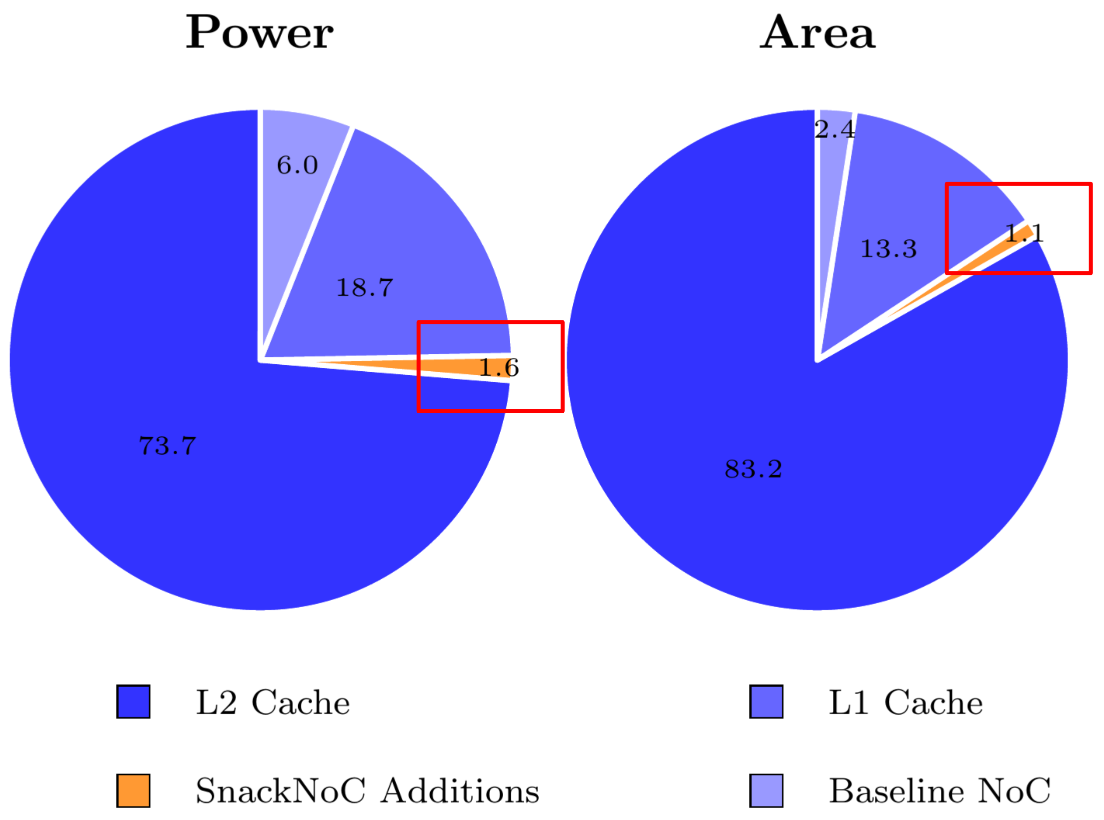
- Full uncore of 16 core CMP is modeled in 45nm with Cacti 7.0 and Orion 3.0.



SnackNoC's Small Contribution to the Total Uncore

- Full uncore of 16 core CMP is modeled in 45nm with Cacti 7.0 and Orion 3.0.
- 16 RCU SnackNoC only contributes 1.6% and 1.1% power and area, respectively.

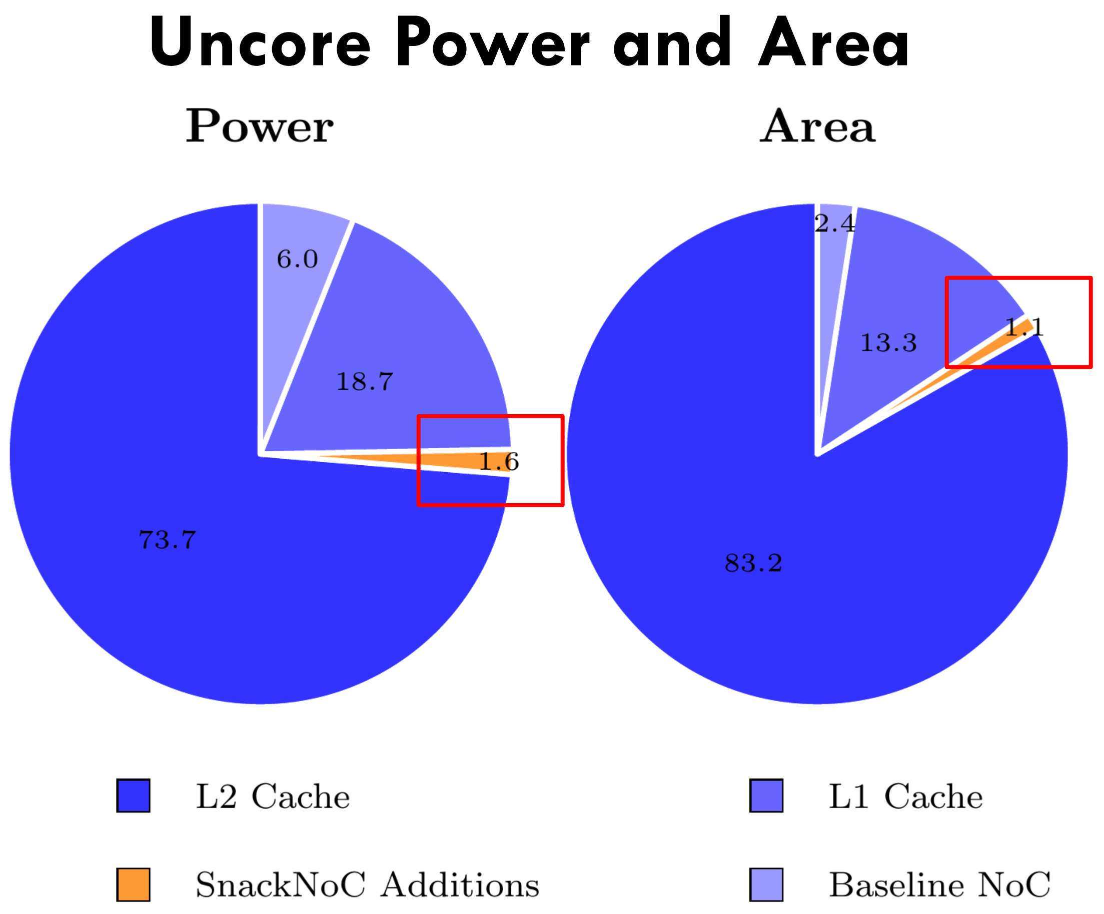
Uncore Power and Area



SnackNoC's Small Contribution to the Total Uncore

- Full uncore of 16 core CMP is modeled in 45nm with Cacti 7.0 and Orion 3.0.
- 16 RCU SnackNoC only contributes 1.6% and 1.1% power and area, respectively.

Satisfies goal of limited overhead



Methodology – Quantifying SnackNoC Interference

- To quantify performance interference, the performance of the CMP is compared with and without SnackNoC Traffic

Simulated CMP Parameters	Configuration
Core Count	16 in-order cores
Core Frequency	2GHz
L1 I&D Cache	32KB, 4-way
L2 Cache	256KB, 4-way
NoC Topology	2D 4x4 Mesh, 4 Memory Controllers
NoC Flit Size	32B
# Virtual Channels	4
# Buffers	4

SnackNoC Parameters	Configuration
RCU Count	16 RCUs
RCU Freq.	1 GHz
Flit Priority Arbitration	ON/OFF

Methodology – Quantifying SnackNoC Interference

- To quantify performance interference, the performance of the CMP is compared with and without SnackNoC Traffic
  - Simulated 16 core CMP with benchmarks from PARSEC3, Splash2X, and FastForward2

Simulated CMP Parameters	Configuration
Core Count	16 in-order cores
Core Frequency	2GHz
L1 I&D Cache	32KB, 4-way
L2 Cache	256KB, 4-way
NoC Topology	2D 4x4 Mesh, 4 Memory Controllers
NoC Flit Size	32B
# Virtual Channels	4
# Buffers	4

SnackNoC Parameters	Configuration
RCU Count	16 RCUs
RCU Freq.	1 GHz
Flit Priority Arbitration	ON/OFF

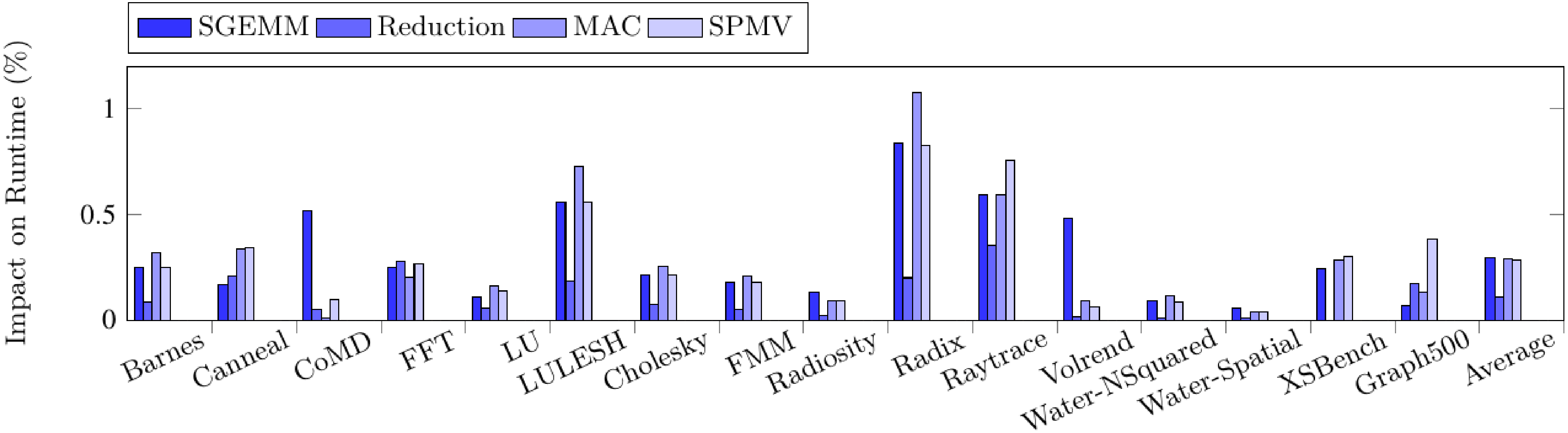
Methodology – Quantifying SnackNoC Interference

- To quantify performance interference, the performance of the CMP is compared with and without SnackNoC Traffic
  - Simulated 16 core CMP with benchmarks from PARSEC3, Splash2X, and FastForward2
  - SnackNoC kernels are simultaneously executed

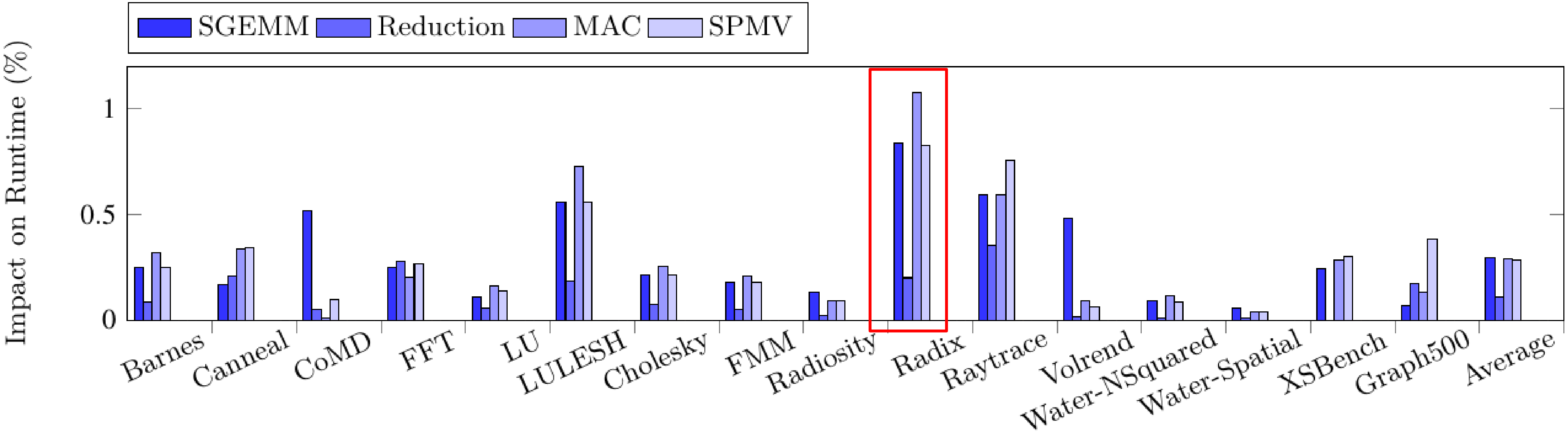
Simulated CMP Parameters	Configuration
Core Count	16 in-order cores
Core Frequency	2GHz
L1 I&D Cache	32KB, 4-way
L2 Cache	256KB, 4-way
NoC Topology	2D 4x4 Mesh, 4 Memory Controllers
NoC Flit Size	32B
# Virtual Channels	4
# Buffers	4

SnackNoC Parameters	Configuration
RCU Count	16 RCUs
RCU Freq.	1 GHz
Flit Priority Arbitration	ON/OFF

Minimal impact of “Snacking” on CMP performance

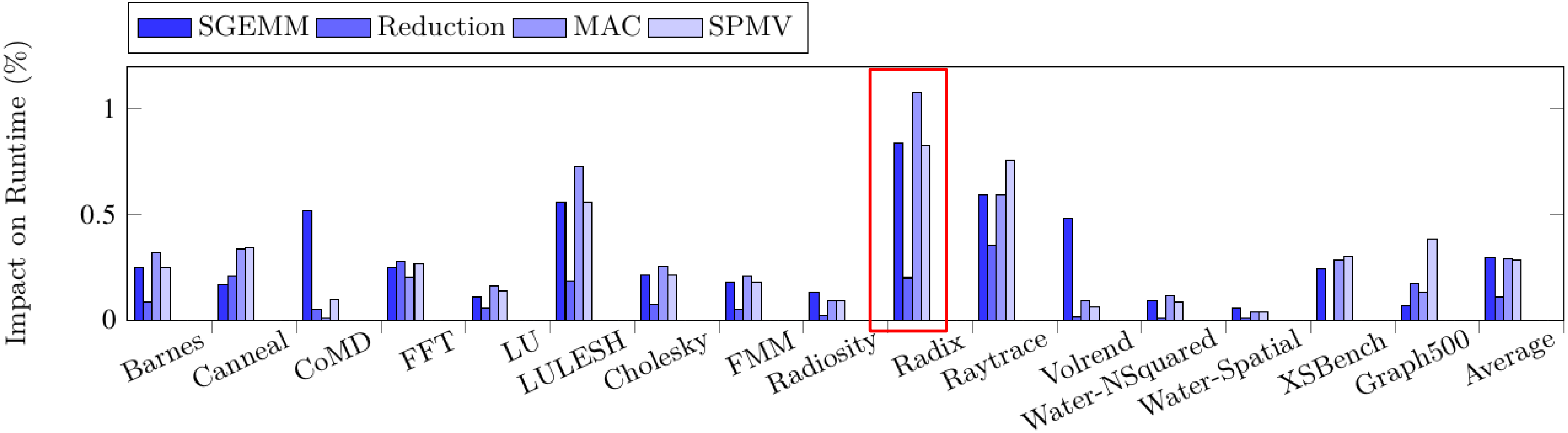


Minimal impact of “Snacking” on CMP performance



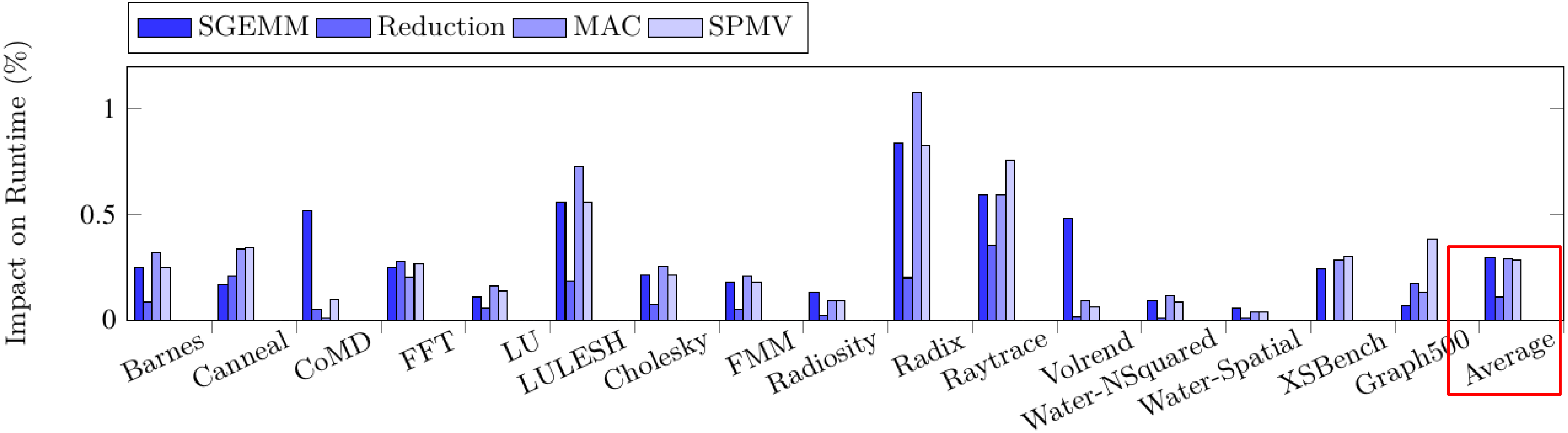
- Performance impact varies based on NoC utilization

Minimal impact of "Snacking" on CMP performance



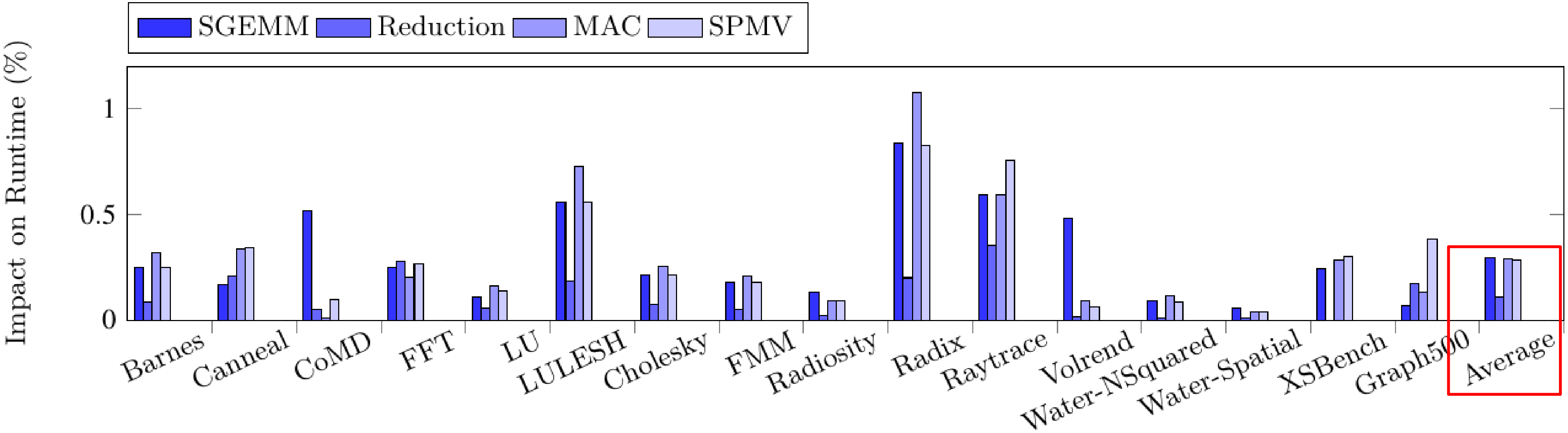
- Performance impact varies based on NoC utilization
  - Peak 1.1% performance impact on CMP cores

Minimal impact of “Snacking” on CMP performance



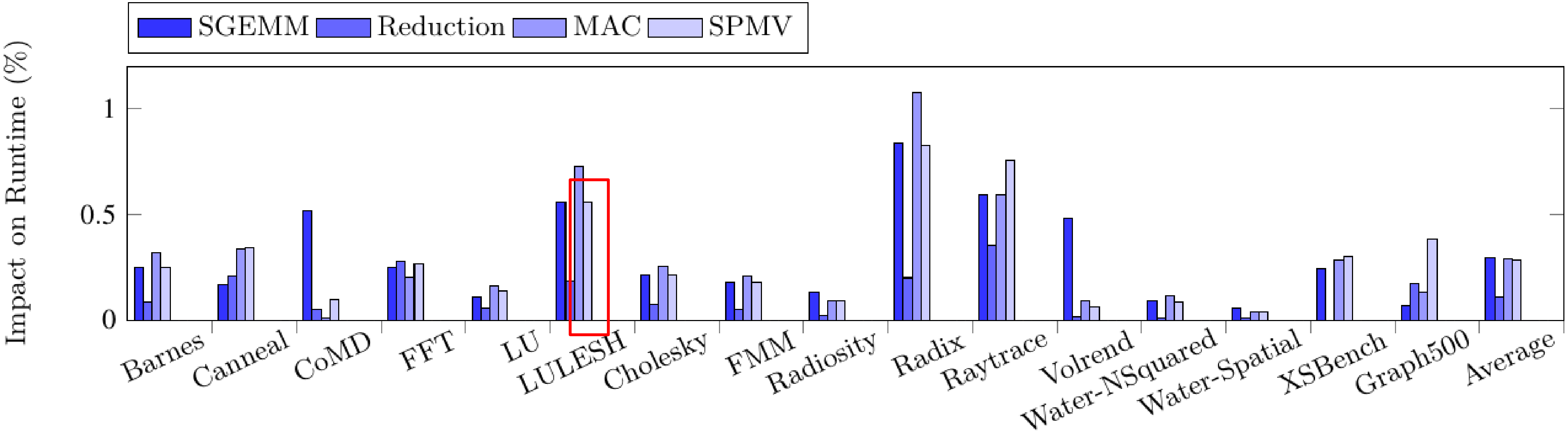
- Performance impact varies based on NoC utilization
  - Peak 1.1% performance impact on CMP cores
  - On average ~0.30% for SGEMM, MAC, SPMV. On average 0.11% for Reduction

Minimal impact of "Snacking" on CMP performance

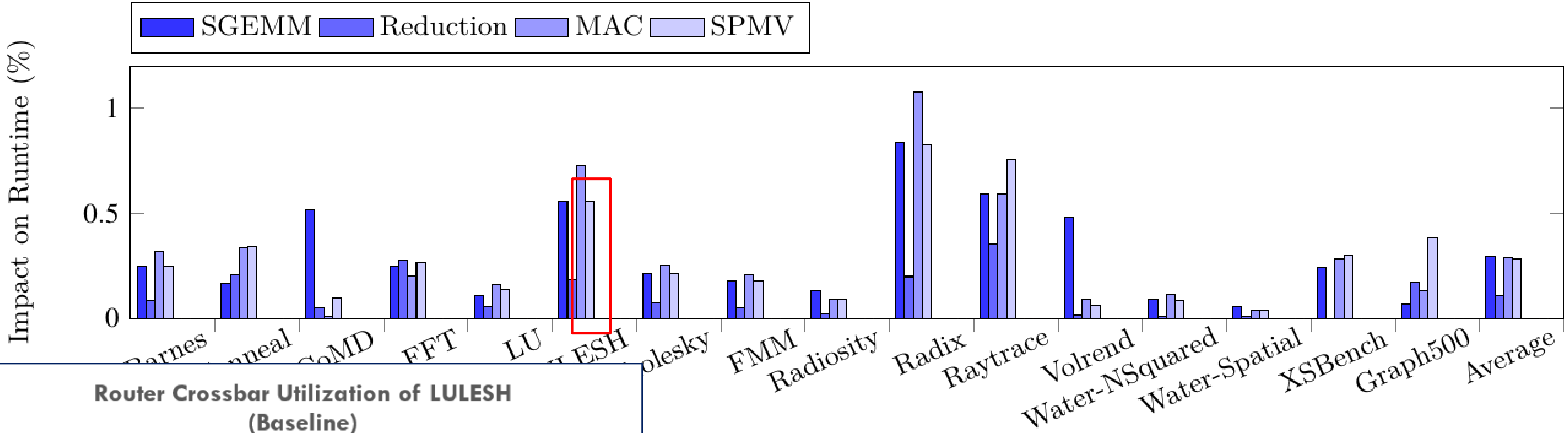


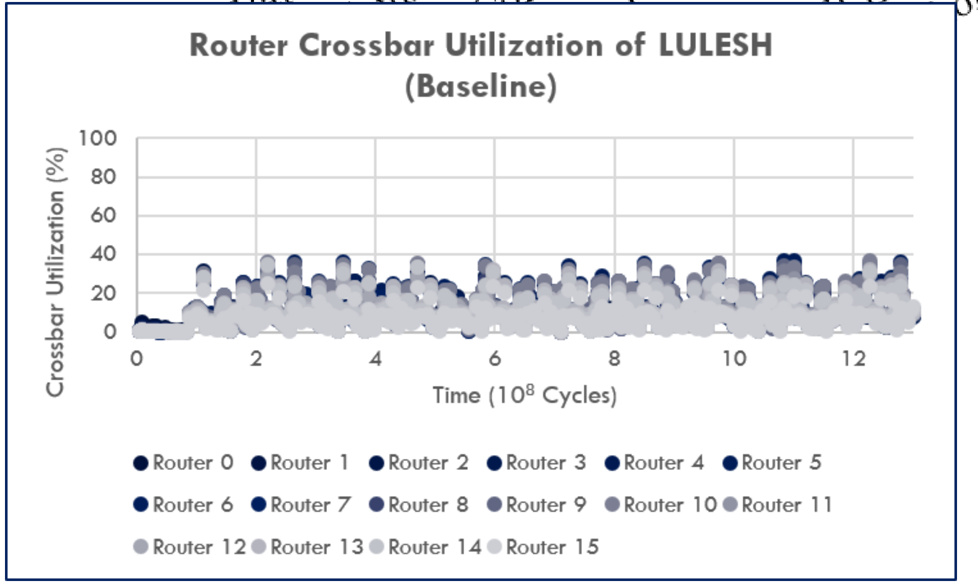
- Performance impact varies based on NoC utilization
  - Peak 1.1% performance impact on CMP cores
  - On average ~0.30% for SGEMM, MAC, SPMV. On average 0.11% for Reduction
- SnackNoC kernel completion time impacted at most 3.9% with fair arbitration

Minimal impact of "Snacking" on CMP performance

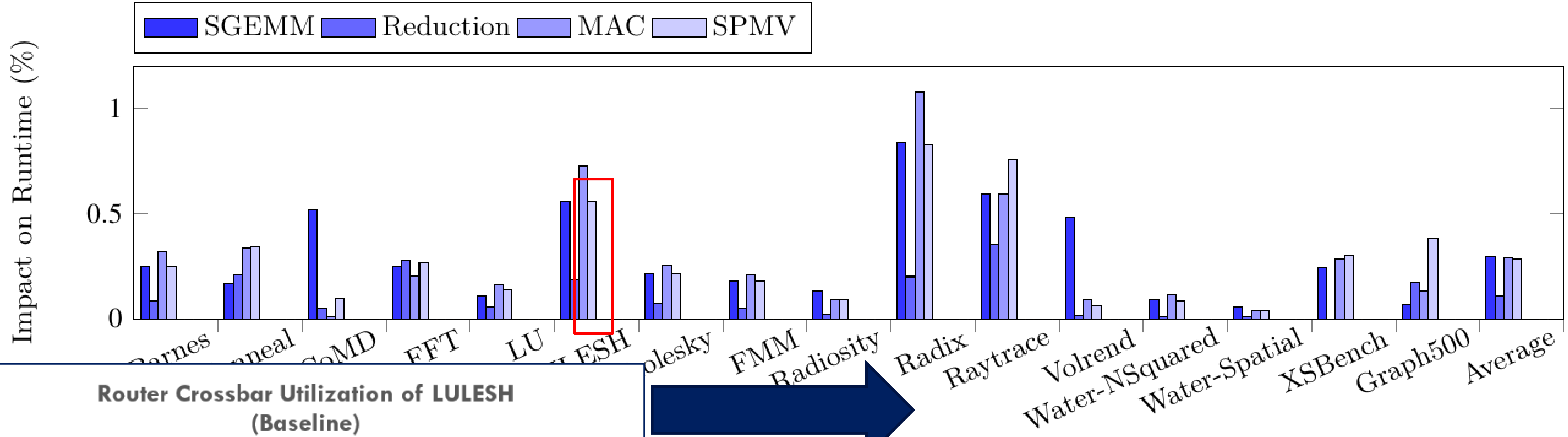


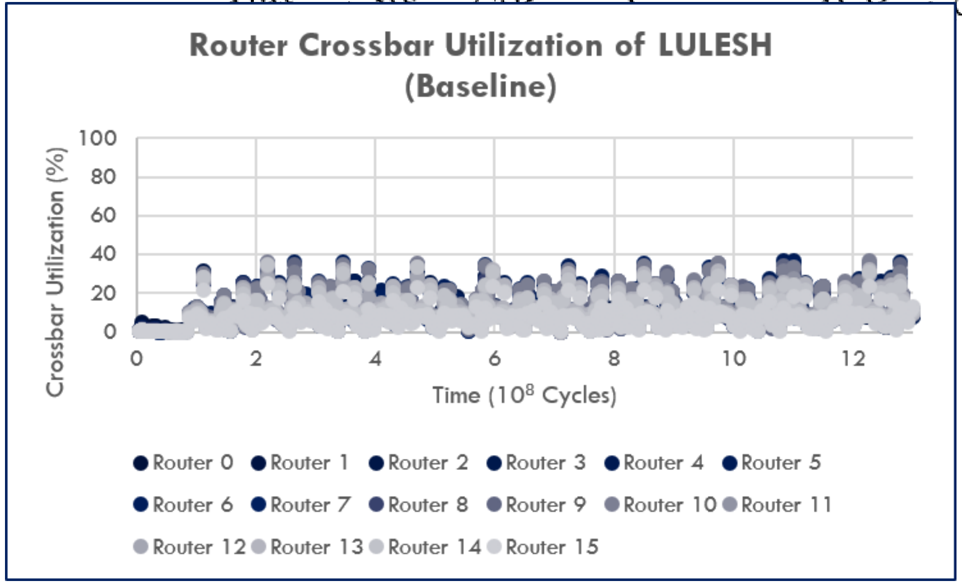
Minimal impact of "Snacking" on CMP performance





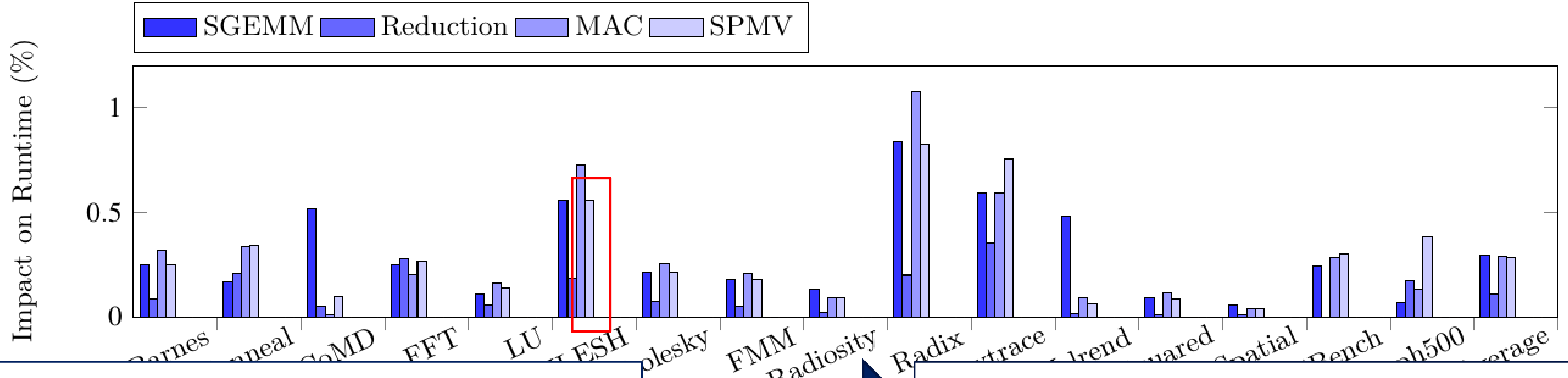
Minimal impact of “Snacking” on CMP performance

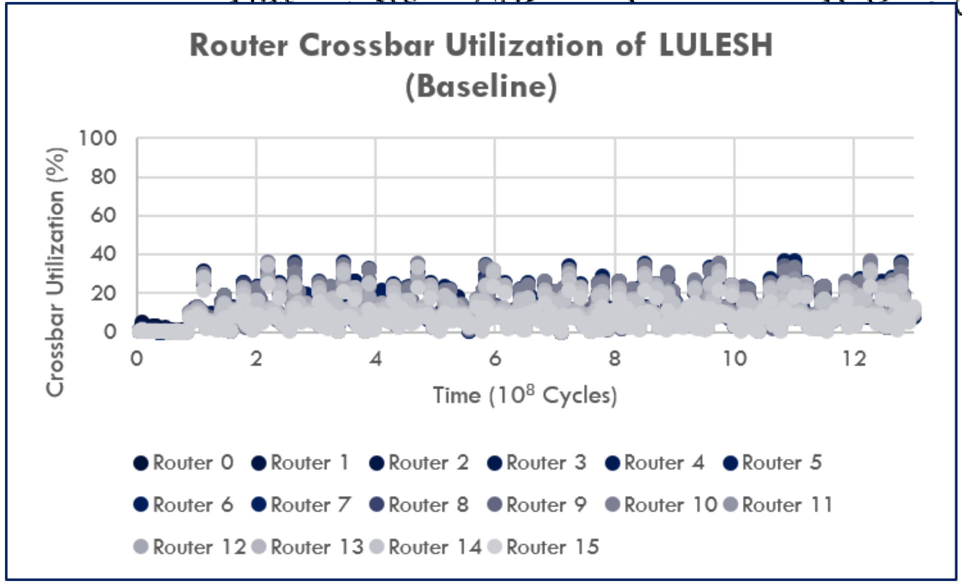




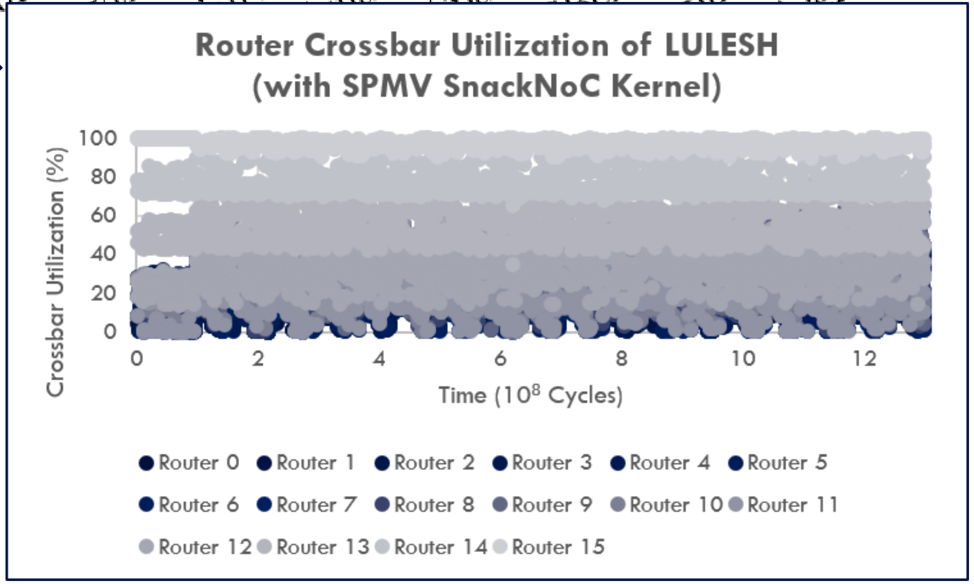
SnackNoC traffic added to LULESH

Minimal impact of "Snacking" on CMP performance

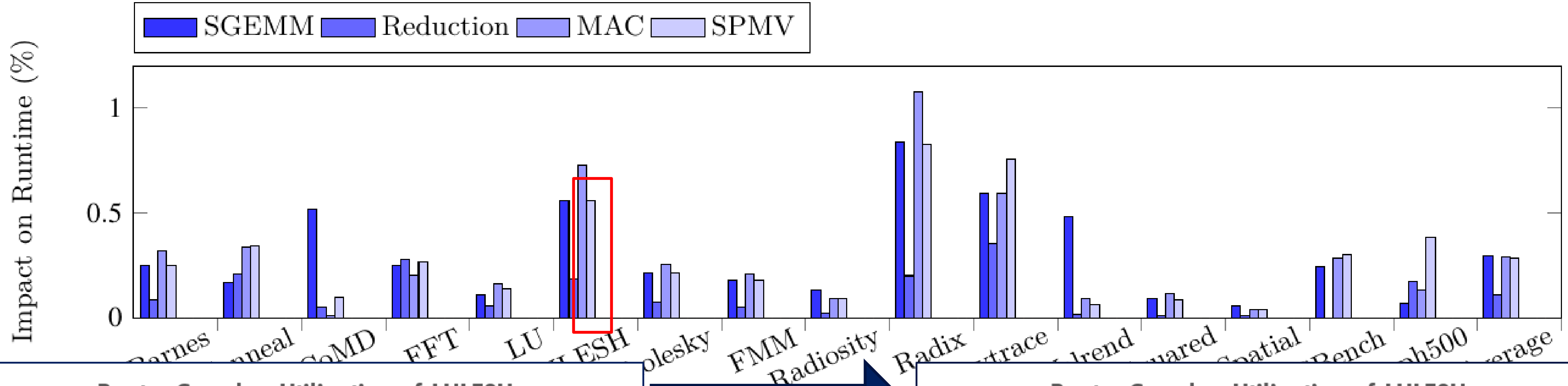


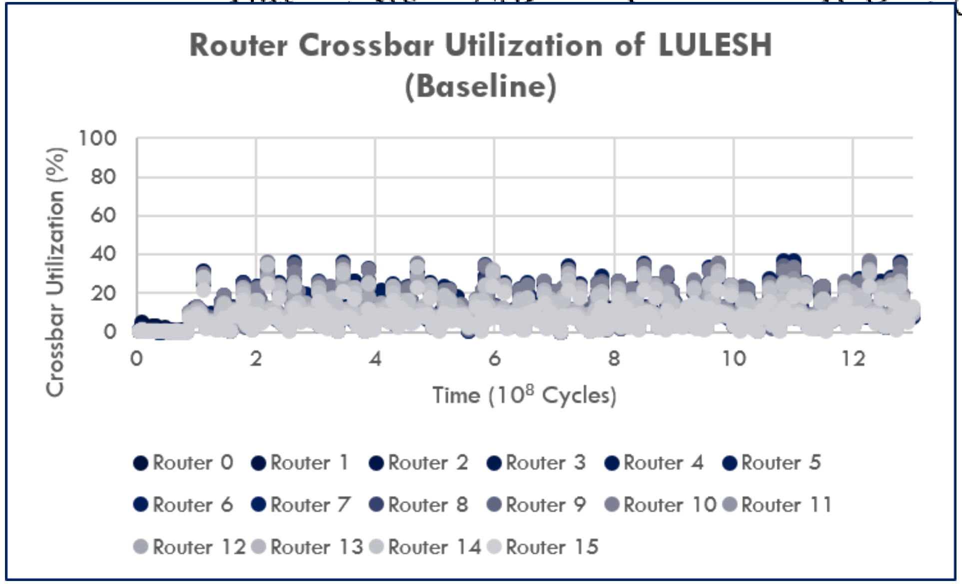


SnackNoC traffic added to LULESH



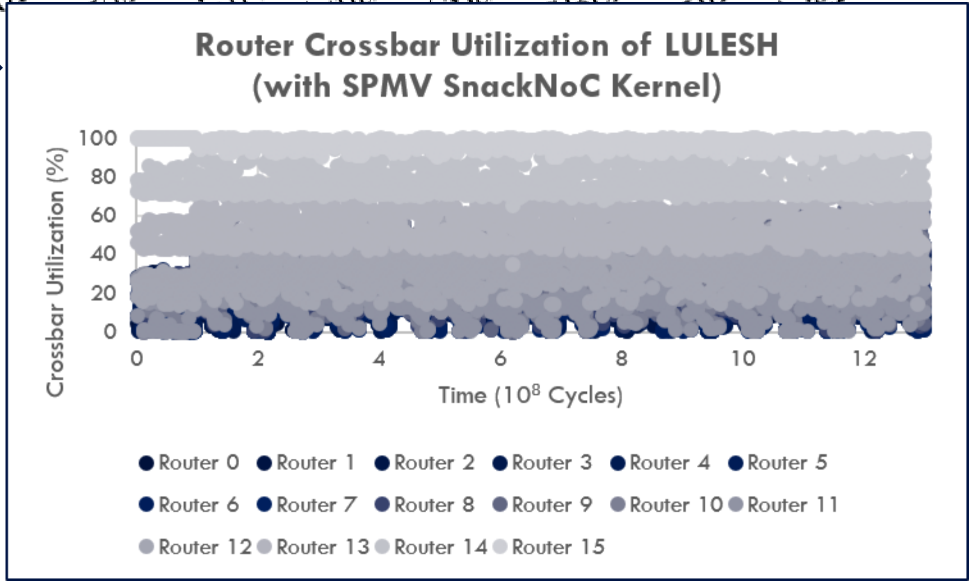
Minimal impact of "Snacking" on CMP performance



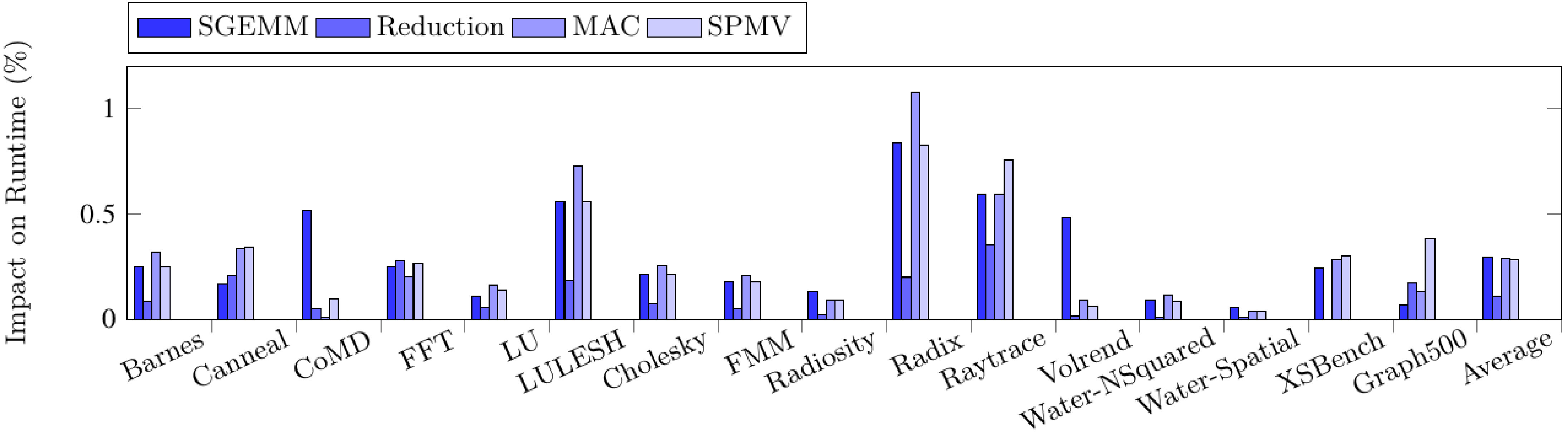


SnackNoC traffic added to LULESH

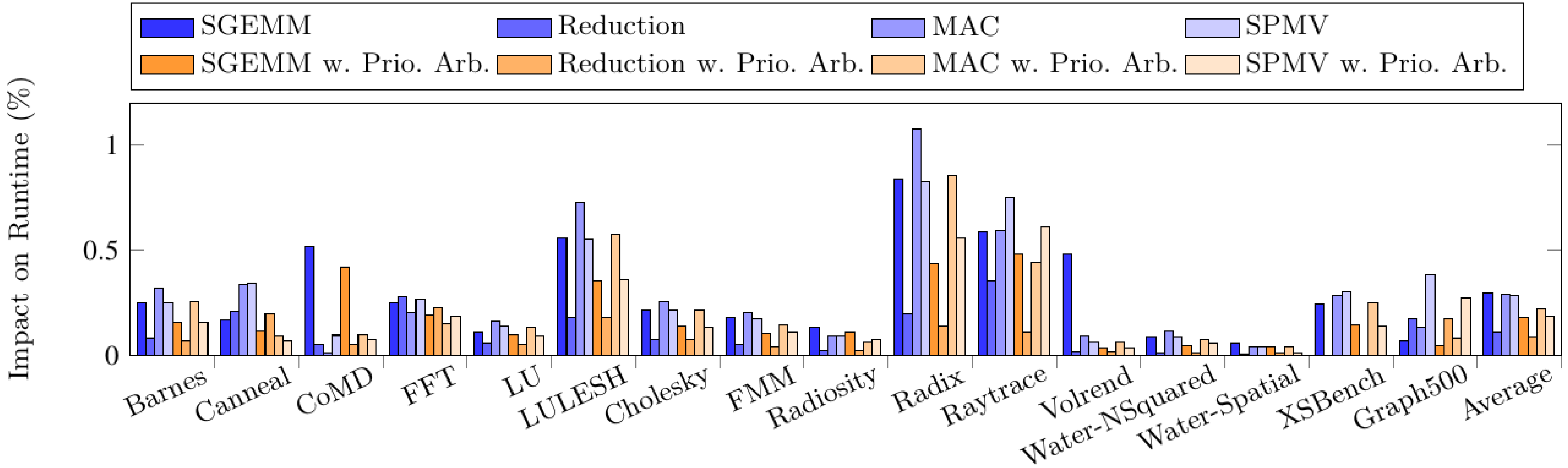
Minimal impact to CMP Performance



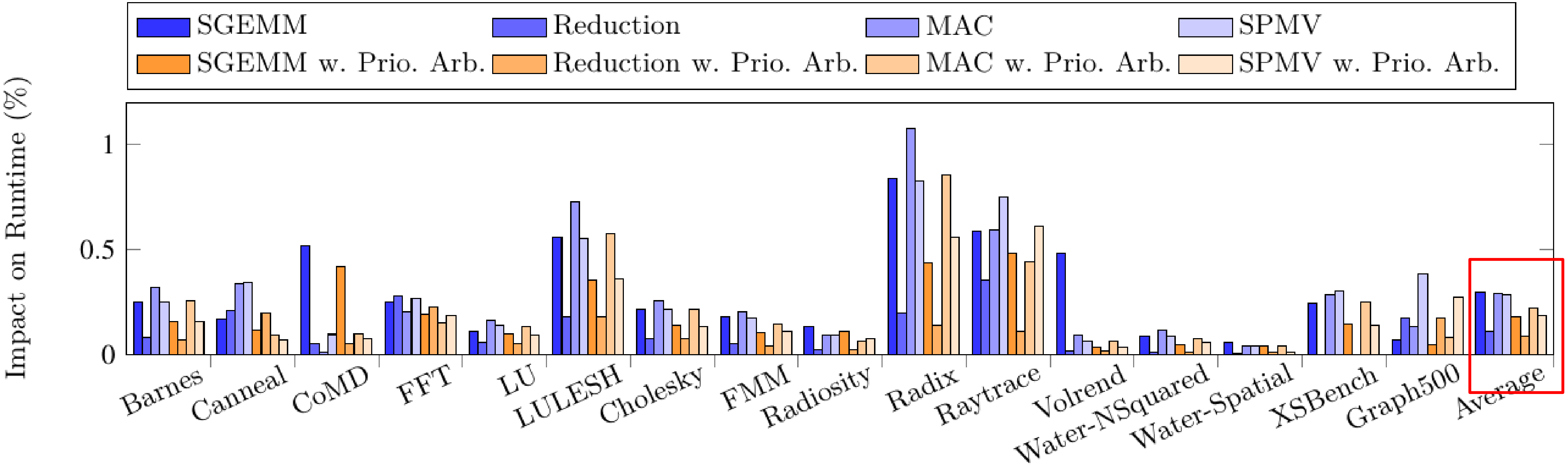
Further Reducing Impact with Priority Arbitration



Further Reducing Impact with Priority Arbitration

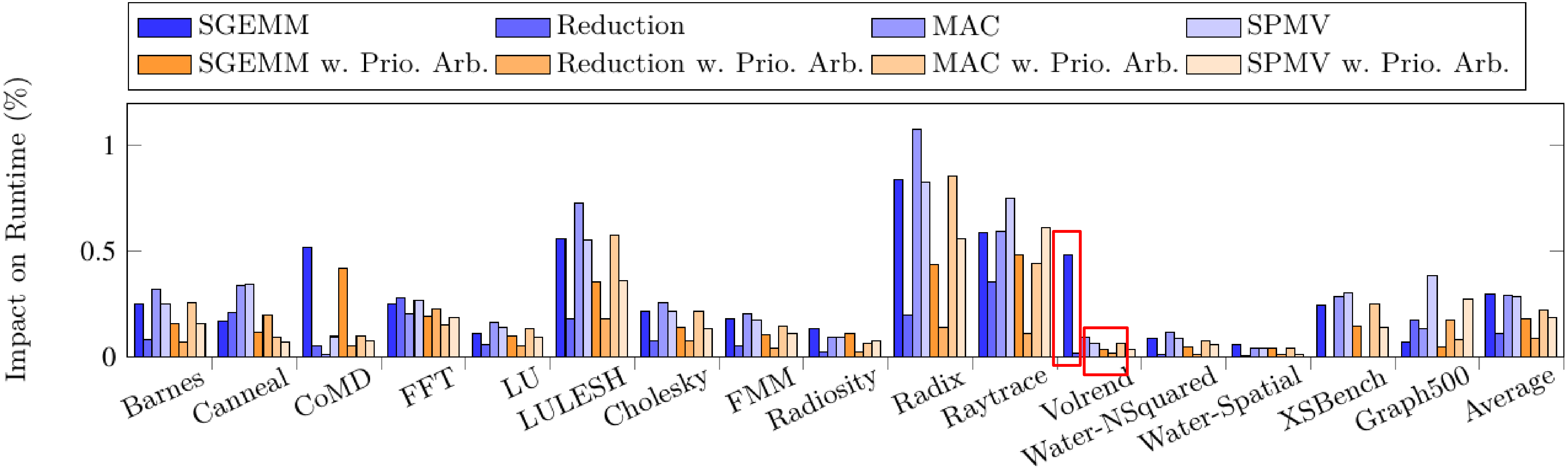


Further Reducing Impact with Priority Arbitration



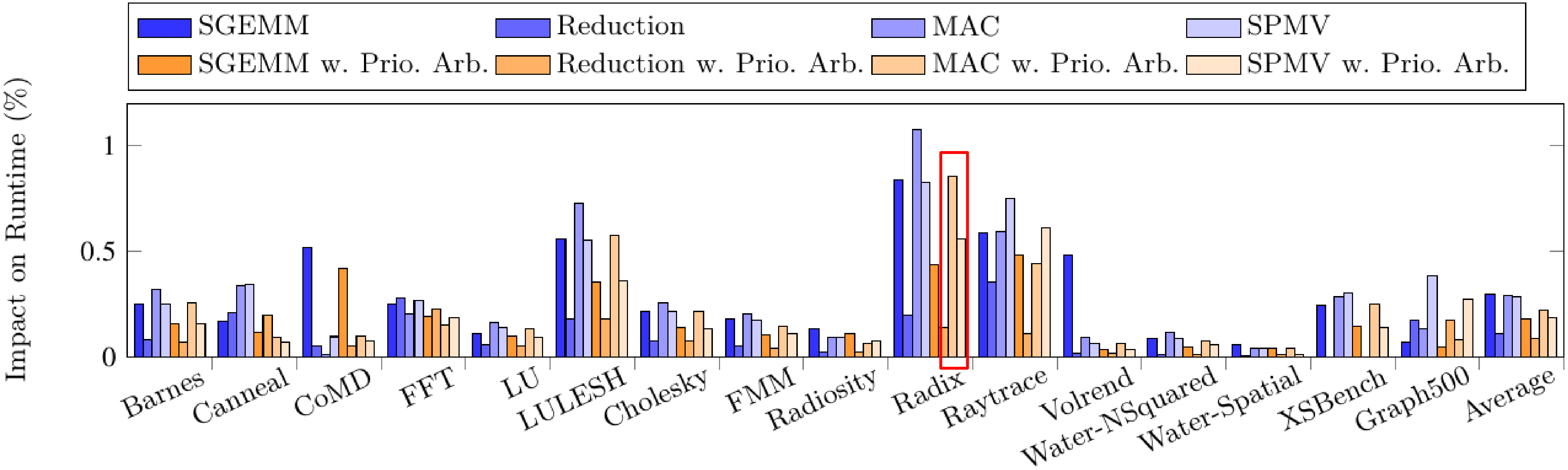
- Adding priority flit arbitration for CMP traffic:
  - Average performance impact drops from 0.25% to 0.17%

Further Reducing Impact with Priority Arbitration



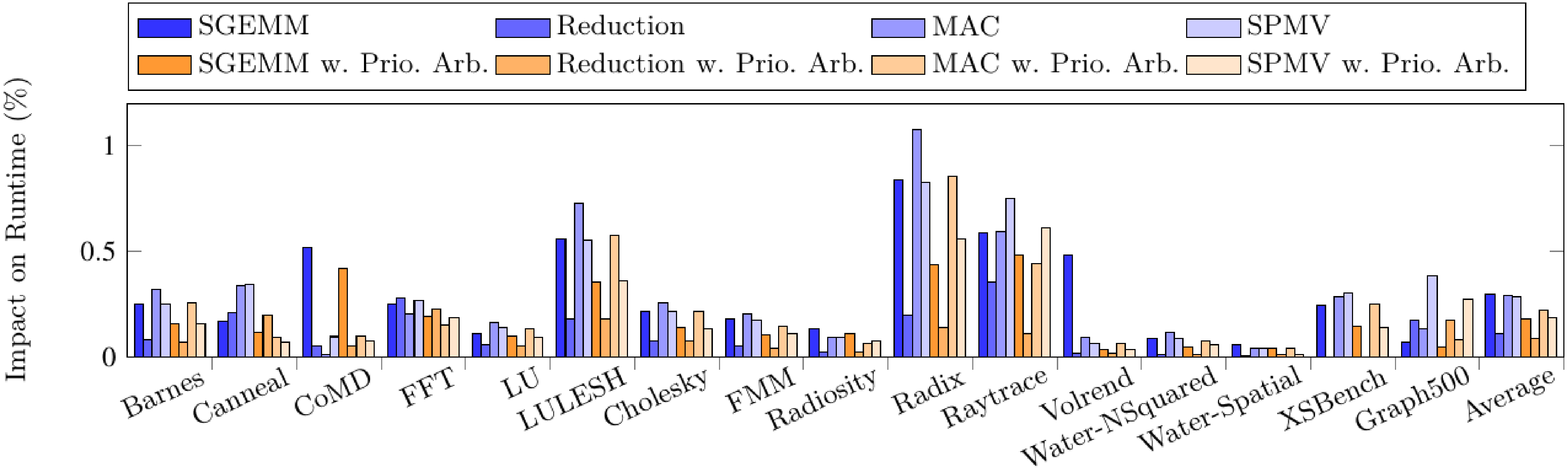
- Adding priority flit arbitration for CMP traffic:
  - Average performance impact drops from 0.25% to 0.17%
  - Improves flit interference by up to 92%

Further Reducing Impact with Priority Arbitration



- Adding priority flit arbitration for CMP traffic:
  - Average performance impact drops from 0.25% to 0.17%
  - Improves flit interference by up to 92%
  - Peak performance impact with priority arbitration is 0.83%

Further Reducing Impact with Priority Arbitration



- Adding priority flit arbitration for CMP traffic:
  - Average performance impact drops from 0.25% to 0.17%
  - Improves flit interference by up to 92%
  - Peak performance impact with priority arbitration is 0.83%

Satisfies goal of limited performance impact

Overview

- “Slack” of the Communication Fabric
- The SnackNoC Platform
- Experimental Results
- **Conclusion and Future Considerations**

Conclusion and Future Considerations

- Opportunistically “snacking” on NoC resources can add performance to our CMPs
  - Added 2 to 6 cores of performance with only a 1.3% increase of the uncore area

Conclusion and Future Considerations

- Opportunistically "snacking" on NoC resources can add performance to our CMPs
  - Added 2 to 6 cores of performance with only a 1.3% increase of the uncore area

- Further tradeoffs we're investigating:
  1. Growing application coverage
  2. Scaling compute density
  3. Supporting future topologies

Questions?

Main Contributions:

- Quantified design slack in the communication fabric
- Opportunistically adds 2 to 6 core performance to the CMP by repurposing NoC resources with low overhead

Karthik Sangaiah, Michael Lui, Ragh Kuttappa, Baris Taskin [Drexel University], and Mark Hempstead [Tufts University], "SnackNoC: Processing in the Communication Layer", Proceedings of the IEEE international Symposium on High Performance Computer Architecture (HPCA), February 2020.

http://vlsi.ece.drexel.edu/ & https://sites.tufts.edu/tcal/