CIM 기반 가속기 내 트래픽 감소를 위한 Adder 통합 라우터 구현

*김유진, 정의영
연세대학교 전기전자공학부 소속
e-mail : realyj2000@yonsei.ac.kr, eychung@yonsei.ac.kr

Implementation of an Adder-integrated Router for traffic reduction in CIM-based Accelerators

*Yujin Kim and Eui-Young Chung
Department of Electrical and Electronic Engineering
Yonsei University

Abstract

While Computing-In-Memory(CIM) architectures address traditional data movement bottlenecks, multi-core CIM accelerators using H-tree topologies in Network-on-Chip(NoC) often suffer from severe traffic congestion at the root node due to massive data transmissions. To resolve this, we propose an 'Adder-in-Router' architecture that integrates addition logic directly into the NoC router. Unlike conventional routers that merely bypass data, the proposed design performs in-network data reduction by actively merging partial data. Furthermore, a merge synchronization mechanism utilizing data packets ensures computational accuracy even in asynchronous environments. System-level simulations using a VGG-16 workload demonstrate that the proposed architecture reduces the total data packets sent to the controller by up to 50% without additional latency penalty, achieving an effective inference latency improvement of up to 28%. Ultimately, the proposed design fundamentally mitigates network congestion and improves the efficiency of CIM-based AI accelerators.

I. 서론

최근 Convolutional Neural Networks(CNNs) 모델의 복잡도와 규모가 지속적으로 증가함에 따라, 데이터 이동 병목 현상으로 인한 전통적인 폰 노이만 아키텍처의 한계를 극복하기 위한 방안으로 Computing-In-Memory(CIM) 기반 가속기가 주목받고 있다. 그러나 최근 AI 모델의 크기가 개별 연산 수용 용량을 초과함에 따라, 연구의 초점은 단일 모듈 설계를 넘어 Host CPU, Main memory, Bus와 CIM tile이 결합된 시스템 수준의 확장형 아키텍처로 이동하고 있다[1].

이러한 멀티 코어 기반 CIM 시스템에서 Network-on-Chip(NoC)은 타일 내 방대한 데이터 교환을 관리하며 시스템의 전체 성능과 에너지 효율을 결정하는 핵심적인 역할을 수행한다. NoC의 성능은 라우터와 코어의 물리적 배치를 정의하는 토폴로지에 의해 크게 좌우되는데, 다양한 토폴로지 중 H-tree 구조는 대칭적인 구조와 다른 토폴로지에 비해 낮은 홉 지연(hop latency) 덕분에 AI 가속기 설계에서 널리 채택되어 왔다[2]. 특히 모든 노드에 대해 균등한 경로를 제공한다는 점은 병렬 연산이 중요한 가속기 환경에서 큰 장점으로 작용한다.

그러나 연산 코어의 수가 증가함에 따라 H-tree 구조는 상위 계층 노드로 트래픽이 집중되는 심각한

병목 현상에 직면하게 된다. CIM 연산의 특성상 각 코어에서 생성된 수많은 데이터는 최종 결과를 도출하기 위해 컨트롤러로 동시에 전송되는 경향이 있다. 이 과정에서 상위 노드로 갈수록 데이터 충돌과 불필요한 트래픽 중복 이동이 발생하며, 이는 대역폭 활용도를 저하시키고 시스템 전체의 지연 시간을 급격히 증가시키는 결과를 초래한다.

본 논문에서는 이러한 문제를 해결하기 위해 라우터 내부에 합산 로직(Adder Function)을 통합한 'Adder-in-Router' 기반 H-tree NoC 구조를 제안한다. 단순히 데이터를 전달하는 역할에 그쳤던 기존 라우터와 달리, 제안하는 아키텍처는 라우터 내 데이터 병합(In-router data reduction) 기능을 수행하여 개별 데이터 패킷을 최종 목적지까지 각각 전송할 필요를 없앤다. 이를 통해 네트워크 혼잡도를 완화하고 시스템 전체의 처리량을 개선함으로써, 시스템 레벨 CIM 가속기를 위한 효율적인 확장성을 제공한다. 본 논문에서는 제안하는 구조의 전체적인 동작 원리와 이를 검증하기 위한 시뮬레이션 체계에 집중하여 기술하고자 한다.

II. 본론

2.1 Computing-in-Memory (CIM)

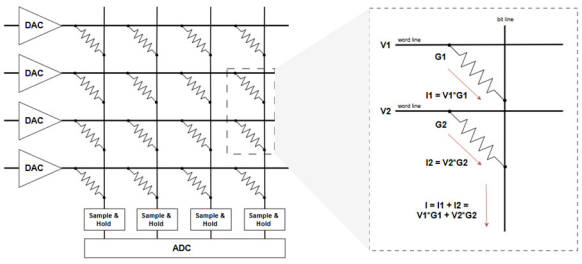


그림 1. Analog CIM operation

기존의 폰 노이만 아키텍처는 연산 장치인 CPU와 메모리가 물리적으로 분리된 구조적 특성으로 인해, 연산 시 대량의 데이터 이동으로 인한 메모리 병목 현상을 야기한다[3]. 이를 해결하기 위해, 데이터 위주의 연산을 CPU에서 처리하는 대신 메모리 셀에서 처리하는 CIM 기술이 주된 해결책으로 떠오르고 있다.

그림 1과 같이 CIM core 내부에는 가중치(weight) 데이터가 메모리 셀의 컨덕턴스(G) 형태로 저장된 크로스바 어레이가 배치된다. 연산 시, 입력 데이터는 전압 신호(V)로 변환되어 각 행의 Word Line에 인가된다. 이 때, 각 셀에서는 옴의 법칙(I = V x G)에 따라 입력 전압과 가중치 값의 곱에 비례하는 전류가 발생한다. 각 열을 구성하는 Bit Line은 해당 열에 연결된 모든 셀에서 발생한 전류를 합산하여 벡터-행렬 곱셈(VMM) 결과를 도출한다. 최종적으로 Bit Line 끝단에 위치한 sample-and-hold 회로와 ADC를 거쳐 디지털 형태의 연산 결과 데이터를 변환한다. 이렇게 생성된 디지털 데이터는 패킷화 과정을 거쳐 라우터로 전달되며, 네트워크를 통해 컨트롤러로 이동한다.

2.2 System Architecture

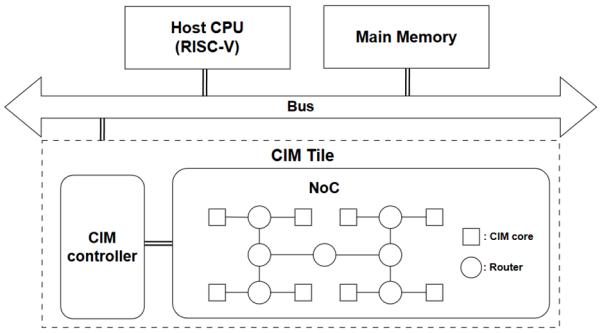


그림 2. Top Architecture

본 연구에서 제안하는 CIM 기반 AI 가속기의 전체적인 구조는 그림 2와 같다. 전체 시스템은 스케줄링을 담당하는 Host CPU, 연산 데이터 및 모델 가중치를 보관하는 Main memory, 구성 요소 간 통신을 중재하는 Bus, 그리고 실제 연산을 수행하는 CIM tile로 구성된다. 가속기의 전반적인 동작은 다음과 같다. Host CPU에서 WRITE, GEMV(General Matrix-Vector Multiplication) 연산 명령을 생성하면, Bus는 메모리 맵 기반 주소를 참조하여 해당 요청을 CIM tile 내의 컨트롤러(CIM controller)로 전달한다. 명령을 수신한 컨트롤러는 DMA(Direct Memory Access)를 활성화하여 Main memory로부터 연산에 필요한 데이터를 읽어온 후, 이를 CIM tile 내부로 공급하여 연산을 진행한다.

CIM Tile은 타일 내 데이터 흐름을 제어하는 컨트롤러와 통신 및 연산 데이터 취합을 담당하는 NoC(Network-on-Chip)로 이루어진다. NoC는 다수의 코어(CIM core)와 이를 연결하는 라우터(Router)로 구성된다. 컨트롤러는 수신한 명령어를 해석하여 각 코어별로 연산을 할당하며, 메모리에서 읽어온 데이터를 패킷화하여 NoC를 통해 해당 코어로 전송한다. 데이터 패킷은 목적지 주소, 연산 명령어, 연산 데이터 등으로 구성된다. 각 코어는 전달받은 패킷 정보를 바탕으로 크로스바 어레이 내에 WRITE나 GEMV 연산을 수행하며, 도출된 결과 데이터는 다시 패킷화되어 네트워크로 송출한다.

2.3 Router Architecture

본 연구는 H-tree 토폴로지를 기반으로 NoC를 구성하였다.

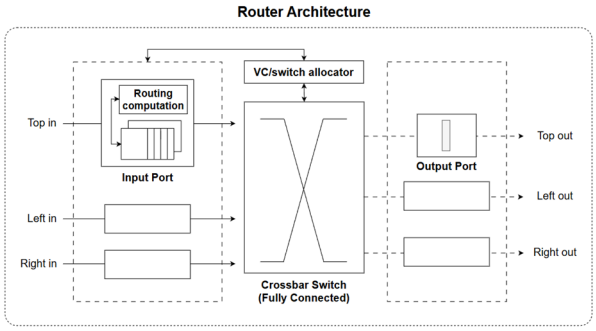


그림 3. Baseline Router Architecture

코어에서 송출된 데이터 패킷의 전달은 라우터에 의해 수행된다. 그림 3은 데이터를 단순히 전달(bypass)하는 데 초점을 맞춘 기존 라우터의 내부 구조를 보여준다. 라우터는 크게 입력 포트(Input Port), 크로스바 스위치(Crossbar Switch), 그리고 출력 포트(Output Port)로 구성된다. 코어나 인접 노드로부터 유입된 데이터 패킷은 먼저 입력 포트 내의 FIFO 버퍼에 임시로 저장된다. 이후 라우팅 연산(Routing computation) 로직을 통해 패킷 정보를 분석하여 목적지 방향을 결정하고, Allocator의 제어에 따라 크로스바 스위치를 통과하여 해당 방향의 출력 포트로 연결된다[4],[5].

H-tree 토폴로지 특성상 라우터 포트 방향은 연결 노드 위치에 따라 명확히 구분된다. 상위 계층인 컨트롤러 혹은 루트 노드의 라우터와 연결된 방향을 Top, 하위 계층인 리프 노드의 라우터 혹은 코어와 연결된 방향을 각각 Left 및 Right 포트로 정의한다.

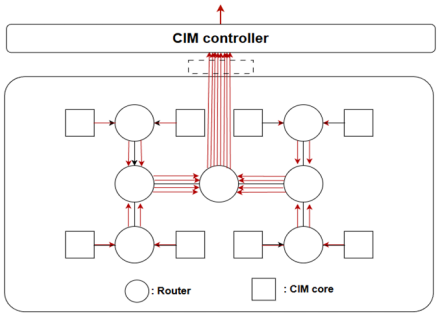


그림 4. Traffic congestion problem

기존의 라우터 구조는 정해진 경로에 따라 데이터를 단순 이동시키는 기능에만 최적화되어 있어, 상위 노드로 트래픽이 집중되는 H-tree 토폴로지에서는 한계에 부딪힌다. 그림 4와 같이 컨트롤러에 인접한 라우터일수록 처리해야 할 데이터 패킷 수가 기하급수적으로 증가하며, 이는 대역폭 내에서 네트워크 정체(traffic congestion)를 유발한다. 결과적으로 패킷의 전송 지연시간(latency)이 길어져 시스템 전체의 추론 속도(inference latency)와 성능을 저하시키게 된다.

III. 구현

3.1 Adder-in-Router

본 논문에서는 H-tree 토폴로지에서 생기는 트래픽 혼잡을 완화하고 전체 네트워크의 데이터 이동량을 감소시키기 위해, 데이터 패킷 라우팅과 동시에 내부에서 직접적으로 연산을 수행할 수 있는 'Adder-in-Router' 아키텍처를 제안한다.

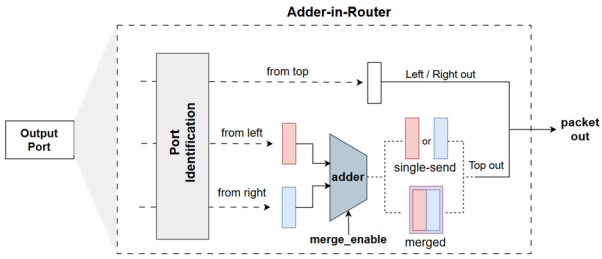


그림 5. Proposed Architecture

그림 5와 같이 제안하는 라우터 구조는 기존 크로스바 스위치 후단의 출력 포트(Output Port)에 데이터 병합을 위한 Adder와 제어 로직(merge_enable)이 추가된 형태이다. 라우터로 유입되는 데이터 패킷은 입력 방향 및 제어 로직의 판단에 따라 두 가지 경로로 처리된다. 상위 노드(Top)에서 하위 코어 방향으로 내려오는 데이터는 Adder를 거치지 않고 개별적으로 전달되는 단일 전송(single-send) 방식으로 처리된다. 반면, 하위 노드인 왼쪽(Left)과 오른쪽(Right) 포트에서 올라오는 데이터 패킷은 제어 로직의 승인 하에 Adder를 통해 하나의 데이터 패킷으로 합쳐져(merge) 상위 노드로 전달된다.

Algorithm 1 Data Merging and Synchronization Logic in Adder-in-Router

Input: Packets from left (P_L) and right (P_R) input ports
Output: Trasmitted packet (P_{out})
1: **function** Check_Merge_Enable (P, router_id) :
2: **if** P.*ps_num* == 0 **then**
3: **return** FALSE
4: **if** router_id == HOST_ID **then**
5: **return** (P.*ps_num* > HOST_POSITION)
6: **else**
7: **return** (0 < router_id < P.*ps_num*)
8: **end function**
9:
10: **Procedure** Forward_Packets :
11: **if** Check_Merge_Enable(P_L) is TRUE **then**
12 Wait until both P_L and P_R arrive in internal FIFOs
13: **if** P_L.*slice_id* == P_R.*slice_id* **then**
14: P_{out}.*data* = P_L.*data* + P_R.*data*;
15: Transmit P_{out} and discard original P_L, P_R (Merge)
16: **else**
17: Wait for the matching packet event
18: **else**
19: Transmit P_L and P_R individually (Single-Send)

그림 6. Data Merging and Synchronization Logic

유효한 데이터 병합 연산을 수행하기 위해, 패킷 내에는 연산에 참여하는 전체 코어 개수를 나타내는

정보(ps_num)와 레이어 정보를 나타내는 슬라이스 ID(slice_id)를 포함시킨다. 그림 6은 라우터 내부에서 동기화를 맞추고 병합을 제어하는 로직을 보여준다. 제어 로직은 가장 먼저 연산에 참여하는 코어의 개수가 0인지 확인하여 연산이 진행 중인 코어가 없으면 병합을 생략한다. 이후 현재 라우터의 위치(router id)와 연산 코어 개수를 비교하여 해당 라우터의 양쪽 하위 코어에서 모두 유효한 데이터가 오는지 구조적으로 판단한다.

병합 조건이 충족된 경우, 네트워크 트래픽 상황에 따른 데이터 도착 시간의 차이를 해결하기 위해 병합 동기화 메커니즘을 수행한다. 라우터는 출력 포트 내부의 방향별 FIFO를 활용하여, 한쪽 포트의 패킷이 먼저 도착하더라도 반대쪽 패킷이 준비될 때까지 대기한다. 두 패킷이 모두 준비되고, 두 데이터가 동일한 레이어의 연산 결과임이 최종 확인된 시점에 제어 로직이 활성화되어 Adder 연산을 수행한다. 이를 통해 연산의 정확성을 보장하는 동시에 상위 노드로 향하는 데이터 패킷 수를 획기적으로 줄임으로써 네트워크 내부의 병목 현상을 해소하고 전체적인 추론 지연 시간을 효과적으로 단축한다.

3.2 Experimental Setup

제안하는 'Adder-in-Router' 기반 NoC 아키텍처의 성능 검증을 위해 Synopsys Platform Architect 기반의 시스템 수준 시뮬레이션 환경을 구축하였다. 전체 시스템은 SystemC TLM (Transaction-Level Modeling)으로 구현되었다. 시뮬레이션 툴에서 제공하는 IP를 활용하여 Host CPU는 RISC-V, Main memory는 DDR IP로 구성하였고, Bus는 AXI protocol을 적용하였다.

본 실험의 워크로드로는 CIFAR-10 데이터셋으로 학습된 VGG-16 모델을 채택하였다. 각 코어에서 발생하는 연산 결과 데이터가 라우터를 거쳐 컨트롤러로 집계되는 과정을 모델링하였으며, 채널 대역폭(bandwidth)을 변화시키며 전체 추론 지연 시간 및 데이터 처리량(throughput)을 분석하였다. 데이터 전송은 패킷 단위로 모델링하였으며, 플릿 크기(flit size)는 대역폭과 패킷 전송 지연을 조절하는 파라미터로 사용하였다. 상세한 하드웨어 파라미터는 다음 표 1과 같다.

CIM array size	512 x 512
Core num	16
Data transport size	512 bytes

표 1. Hardware parameters

3.3 Experimental Results

앞서 기술한 실험 환경을 바탕으로, 기존 baseline 구조와 제안하는 'Adder-in-Router' 기반 H-tree NoC 아키텍처 간의 성능을 비교 분석하였다. 실험은 VGG-16 모델을 활용하여 레이어별 네트워크 트래픽 변화와 대역폭 변화에 따른 지연 시간을 중심으로 수행되었다.

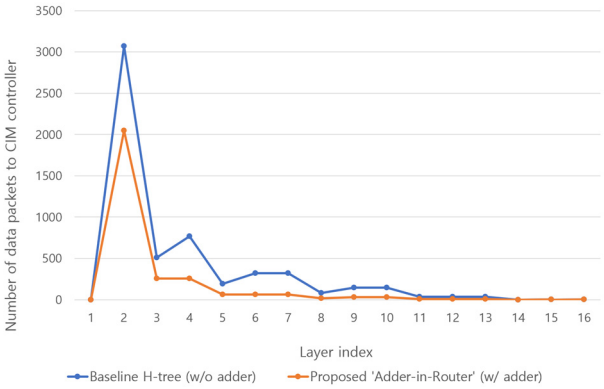


그림 7. Packet Count Comparison per layer

그림 7은 VGG-16 모델의 각 레이어에서 컨트롤러로 전송되는 데이터 패킷 수의 측정 결과를 보여준다. 실험에 포함한 16개 레이어 중 12개의 레이어에서 baseline 대비 트래픽 감소 효과를 보였다. 데이터 발생량이 가장 큰 레이어 2의 경우 기존 3072개의 패킷이 2048개로 감소하여 약 33.3%의 트래픽 절감 효과를 나타냈으며, 이후 레이어 전반에서도 최대 약 50%에 달하는 전송 데이터 패킷 수 감소가 확인되었다. 이는 컨트롤러에 도달하기 전 라우터 단계에서 데이터 병합이 수행될 경우 NoC 내부의 혼잡을 완화할 수 있음을 보여준다.

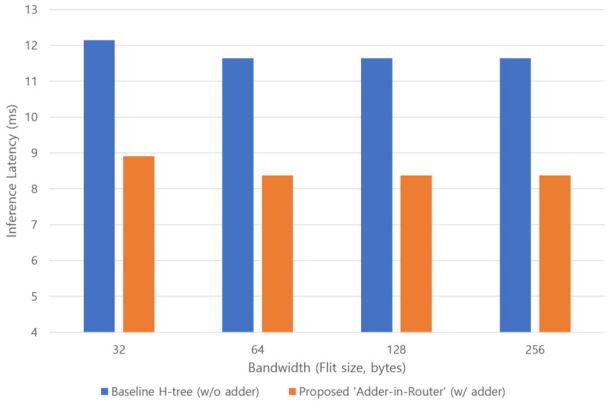


그림 8. Inference Latency Comparison

플릿 크기(flit size)의 변화에 따른 전체 추론 지연 시간을 분석한 결과는 그림 8과 같다. 제안하는 아키텍처는 라우터 내부에 Adder 로직이 통합되었음에도 불구하고 추가적인 지연 시간 오버헤드 없이 모든 대역폭 구간에서 기존 baseline

대비 약 27~28%의 추론 성능 향상을 달성하였다. 특히 주목할 점은 플릿 크기가 256 bytes까지 증가한 고대역폭 환경에서도 약 3.3ms 수준의 성능 개선이 나타난다는 것이다. 실제 그래프 상에서도 대역폭 증가와 무관하게 추론 지연 시간 차이가 일정하게 유지되는 양상을 확인할 수 있다. 이는 극심한 트래픽 혼잡 상황에서는 단순히 물리적인 대역폭을 늘리는 것만으로는 추론 지연 시간을 온전히 해소할 수 없으며, 수많은 데이터 패킷을 개별적으로 라우팅하며 발생하는 오버헤드가 주된 병목의 원인으로 작용함을 시사한다. 결과적으로 제안된 'Adder-in-Router' 아키텍처는 데이터 패킷 병합 메커니즘을 통해 불필요한 데이터 이동을 최소화하여 전반적인 시스템 성능 개선에 기여함을 보여준다.

IV. 결론 및 향후 연구 방향

본 논문에서는 CIM 기반 AI 가속기의 H-tree NoC에서 발생하는 트래픽 병목 현상을 해결하기 위해, 라우터 내부에서 데이터를 직접 병합하는 'Adder-in-Router' 아키텍처를 제안하였다. 제안하는 구조는 단순 데이터 전달을 넘어, 패킷 내의 데이터를 활용하여 유효한 연산 데이터를 식별하고 출력 포트 내부의 Adder와 제어 메커니즘을 통해 데이터 병합을 수행하도록 설계하였다.

SystemC TLM 기반의 시뮬레이션 환경에서 VGG-16 모델을 워크로드로 적용하여 성능을 검증한 결과, 제안하는 아키텍처는 하드웨어 로직 추가에 따른 지연 시간 오버헤드 없이 안정적으로 동작하는 것을 확인하였다. 제안 구조는 즉각적인 데이터 병합을 통해 컨트롤러로 향하는 전체 데이터 패킷 수를 최대 50%까지 절감할 뿐만 아니라, 약 28% 수준의 유의미한 추론 지연 시간 단축 성과를 거두었다. 이는 제안된 라우터 구조가 네트워크 혼잡도를 근본적으로 완화하고 전체 CIM 기반 AI 가속기 시스템의 효율을 극대화할 수 있는 방안임을 보여준다. 특히 연산에 참여하는 코어의 수가 많아질수록 병합되는 데이터 패킷의 비율이 증가하는 이론적 특성을 고려할 때, 향후 대규모 확장형 멀티 코어 시스템에서 본 제안 구조의 트래픽 절감 및 성능 개선 효과는 더욱 극대화될 것으로 기대된다.

향후 연구에서 본 연구의 성능 평가가 시스템 수준에서 진행된 점을 보완하기 위해, 제안된 라우터를 RTL(Register-Transfer Level) 모듈로 구현하고 게이트 수준에서의 PPA(Power, Performance, Area) 물리적 오버헤드를 정량적으로 검증할 계획이다. 또한 VGG-16 모델 외에도 AlexNet, ResNet 등의 다른 CNN 모델들과 ViT (Vision Transformer) 계열 모델로 워크로드를 확대하여 제안하는 아키텍처의 구조적 확장성을 검증하고자 한다.

Acknowledgment

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT). (No.2710096985, Development of memory module and memory compiler for non-volatile PIM optimized for data characteristics and data access characteristics of AI processor)

참고문헌

[1] A. Siemieniuk et al., "OCC: An automated end-to-end machine learning optimizing compiler for computing-in-memory," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 41, no. 6, pp. 1674-1686, June 2022.

[2] Z. Zhu et al., "MNSIM 2.0: A behavior-level modeling tool for processing-in-memory architectures," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 11, pp. 4112-4125, November 2023.

[3] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA), pp. 14-26, June 2016.

[4] A. Banerjee et al., "An energy and performance exploration of network-on-chip architectures," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no. 3, pp. 319-329, Mar 2009.

[5] Jiang, Nan et al., "A detailed and flexible cycle-accurate network-on-chip simulator," in Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS), pp. 86-96, April 2013.