멀티 프로세서 시스템-온-칩(MPSoC)을 위한 버스 매트릭스 구조의 빠르고 정확한 성능 예측 기법

(Fast and Accurate Performance Estimation of Bus Matrix for Multi-Processor System-on-Chip(MPSoC))

김 성 찬 † 하 순 회 ††
(Sungchan Kim) (Soonhoi Ha)

요 약 본 논문은 큐잉 이론을 이용한 멀티 프로세서 시스템-온-칩(MPSoC)의 버스 매트릭스 기반 통신 구조에 대한 성능 예측 기법을 제안한다. 버스 매트릭스 기반 통신 구조는 다양한 설계 인자를 가지고 있어 이에 대한 성능 최적화는 방대한 설계 공간의 탐색을 필요로 하지만, 현재 널리 사용되고 있는 시뮬레이션에 기반한 방법은 많은 시간을 요구하기 때문에 점점 짧아지고 있는 시장 적기 출하(time-to-market) 제약 조건을 만족하기 어렵다. 이러한 문제를 해결하기 위하여 본 논문에서는 시뮬레이션보다 훨씬 빠르면서 정확하게 성능을 예측할 수 있는 기법을 개발하였다. 제안한 성능 분석 기법은 고성능의 버스 매트릭스를 위해 사용되는 버스 프로토콜인 multiple-outstanding transaction을 고려한다. 또한 지수 분포(exponential distribution)를 이용하여 비현실적으로 메모리 시스템을 모델하였던 기존의 연구들과 달리 실제적인 메모리 시스템 모델을 위하여 일반 분포(general distribution)를 이용하였다. 제안한 성능 예측 기법의 정확도 및 효율성을 검증하기 위하여 무작위로 생성된 버스 트랜잭션들과 4-채널 DVR 예제에 적용하였을 때, 사이클 단위의 정확도를 갖는 시뮬레이션과 비교하여 10^5배 이상 빠르면서 평균 94% 이상의 정확도를 갖는 것으로 분석되었다.

키워드 : 멀티 프로세서 시스템-온-칩, 통신 구조, 버스 매트릭스, 성능 예측, 큐잉 이론

Abstract This paper presents a performance estimation technique based on queuing analysis for on-chip bus matrix architectures of Multi-Processor System-on-Chips(MPSoCs). Previous works relying on time-consuming simulation are not able to explore the vast design space to cope with increasing time-to-market pressure. The proposed technique gives accurate estimation results while achieving faster estimation time than cycle-accurate simulation by order of magnitude. We consider the followings for the modeling of practical memory subsystem: (1) the service time with the general distribution instead of the exponential distribution and (2) multiple-outstanding transactions to achieve high performance. The experimental results show that the proposed analysis technique has the accuracy of 94% on average and much shorter runtime (10^5 times faster at least) compared to simulation for the various examples: the synthetic traces and real-time application, 4-channel DVR.

Key words : multi-processor system-on-chip, communication architecture, bus matrix, performance estimation, queuing theory

· 이 논문은 제34회 추계학술대회에서 '멀티 프로세서 시스템-온-칩(MPSoC)을 위한 버스 매트릭스 구조의 빠르고 정확한 성능 예측 기법'의 제목으로 발표된 논문을 확장한 것임

† 정 회 원 : 서울대학교 전기컴퓨터공학부
sungchan.kim@iris.snu.ac.kr
†† 정 회 원 : 서울대학교 전기컴퓨터공학부 교수
sha@iris.snu.ac.kr

논문접수 : 2007년 12월 6일
심사완료 : 2008년 10월 20일

정보과학회논문지: 시스템 및 이론 제35권 제11호(2008.12)

1. 서 론

반도체 공정 기술의 발달로 인해 점점 많은 수의 프로세서 및 메모리 코어들을 하나의 칩에 집적하는 것이 가능해지면서 많은 임베디드 시스템들이 멀티 프로세서 시스템-온-칩(Multi-Processor System-on-Chip, MPSoC)의 형태로 구현되고 있다. 이로 인해 칩 안에서 프로세서 코어들 사이의 통신은 급격하게 증가하였고 그 특성 또한 다양해졌다. 결과적으로 프로세서 코어들을 연결하는 통신 구조는 성능, 전력 소모 및 하드웨어 면적 등의 측면에서 큰 영향을 끼치게 되었고 이의 최적화가 MPSoC 설계의 중요한 문제가 되었다.

이렇게 빠르게 증가하는 온-칩 통신을 위해 좀 더 높은 병렬성 및 고성능을 제공하기 위해 최근 MPSoC[1] 뿐 아니라 Chip-Multi-Processor(CMP)[2]에서 버스 매트릭스 기반 통신 구조가 널리 사용되고 있다. 특히 MPSoC 설계 분야에서는 버스 매트릭스를 적용하기 위해 현재 널리 사용되는 버스 프로토콜과의 호환성을 유지하면서 고성능을 제공할 수 있는 상용화된 IP들이 활발하게 이용되고 있다[3].

공유 버스 기반의 통신 구조의 탐색에 대한 기존 연구들에 의하면, 다양한 설계 인자들에 의해 매우 방대한 설계 공간이 형성되기 때문에 주어진 개발 시간 내에 최적의 통신 구조를 얻기 위해서는 각 구조들에 대해 정확하고 빠르게 성능을 예측하는 것이 필요하다[4,5]. 최근 사용되는 버스 프로토콜은 보다 높은 성능을 얻기 위해 multiple-outstanding transaction[1]과 out-of-order completion을 사용한다[6]. 또한 보다 높은 동작 속도를 얻기 위해 다단계 버스 매트릭스(cascaded bus matrix) 구조를 사용하기도 한다[7]. 이렇게 다양한 설계 인자들을 고려하여 최적화된 버스 매트릭스 기반의 통신 구조를 설계하고자 한다면 공유 버스 기반의 통신 구조보다 훨씬 넓은 설계 공간을 고려해야 할 것이다.

그러나 현재까지 개발된 버스 매트릭스 통신 구조의 설계 공간 탐색 방법은 많은 시간이 걸리는 시뮬레이션에 의존하고 있다[8,9]. 따라서 제한된 개발 시간에 시뮬레이션을 이용하여 탐색할 수 있는 설계 공간의 크기는 한정적이므로 최적화된 혹은 최적에 가까운 통신 구조를 찾아내는 것은 쉽지 않다. 이러한 문제를 해결하기 위하여 본 논문에서는 주어진 응용 예제의 메모리 접근 패턴의 통계 정보에 기반한 큐잉 이론을 이용하여 버스 매트릭스 통신 구조의 성능 예측 기법을 제시하고자 한다. 큐잉 이론을 이용한 기존의 성능 예측 기법들은 모델의 복잡도를 낮추어 계산을 쉽게 하기 위해 메모리 시스템에 대한 접근 시간(혹은 큐잉 모델에서의 서비스 시간)이 지수 분포(exponential distribution)를 갖는다고 가정하였지만, 이는 실제 시스템에서의 메모리 접근 시간을 반영할 수 없으므로 비현실적이다. 간단한 예로, 워드당 한 사이클의 접근 시간을 갖는 SRAM은 지수 분포로 표현할 수 없다. 반면 본 논문의 성능 예측 기법은 실제 메모리 시스템을 고려하여 접근 시간을 모델하기 위해 일반 분포(general distribution)을 갖는다고 가정한다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구들을 살펴보고 이를 통해 본 논문의 의의를 기술한다. 3장에서는 제안한 성능 예측 기법을 구체적으로 설명한다. 먼저 단일 버스 구조에 대한 성능 예측 기법을 제시하고 이를 버스 매트릭스로 확장한다. 4장에서는 실험을 통해 제안한 성능 예측 기법을 시뮬레이션과 비교하여 성능 예측의 정확도 및 실행 시간의 측면에서 검증하고, 5장에서 결론 및 향후 과제를 제시하며 본 논문을 맺는다.

2. 관련 연구

공유 버스 기반의 통신 구조의 성능 예측을 위한 기존의 연구들에서는 정적 분석 기법, 시뮬레이션 및 이들의 장점을 모두 이용하기 위한 혼합된 기법 등을 이용하였다[10,11]. 공유 버스 기반의 통신 구조의 성능을 예측하기 위해 주로 두 가지 방법이 사용된다. 정적 분석 기법은 버스 구조의 동작 속도, 버스의 데이타 폭, 전송할 데이타의 크기 등의 설계 인자들을 고려하여 정적으로 데이타 전송에 걸리는 시간을 예측하는 것이다[12,13]. 이러한 기법은 빠른 시간에 성능을 예측할 수 있으나 버스 중재 등의 버스에서의 동적인 상황에 의한 지연 시간을 예측할 수 없어 정확한 성능 예측이 어렵다는 문제가 있다. 반면 시뮬레이션을 이용하면 이러한 동적 지연 시간을 분석할 수 있으므로 정확한 성능 예측이 가능하지만 실행 시간이 너무 오래 걸리는 단점이 있다[8,9]. 이러한 성능 예측 기법들을 이용하여 버스 토폴로지, 프로세서 및 메모리 코어들의 버스에 대한 할당, 동작 속도, 전력 소모 및 상위 수준에서의 레이아웃 등을 고려한 설계 공간 탐색 기법들이 연구되었다[4,5,14].

버스 매트릭스 기반의 통신 구조에 대한 연구로서는 매트릭스 내부의 버스 개수, 버스 중재 정책, 다단계 버스 매트릭스 구조에서의 토폴로지 등을 고려하여 최적의 구조를 탐색하는 방법들이 최근 연구되었다[7,8]. 주어진 성능 제약 조건을 만족하면서 최소의 버스 개수를 갖는 온-칩 버스 매트릭스 구조를 찾아내기 위해 ILP (Integer Linear Program)과 휴리스틱을 이용한 2가지 방법이 [15]에서 제안되었으나, multiple-outstanding trans-

1) transaction (트랜잭션): 여러 개의 word로 이루어진 데이타 블록을 가져오기 위한 연속적인 메모리 접근들의 집합. 본 논문에서 트랜잭션과 메모리 접근은 같은 의미이며 별도의 구분 없이 사용된다.

action이나 out-of-completion과 같은 버스 프로토콜을 고려하지 않았다. Globally-Asynchronous-Locally-Synchronous(GALS) 통신을 위한 온-칩 버스 구조가 [16]에서 제안되었고 multiple-outstanding transaction 및 out-of-order completion을 사용한 버스 프로토콜이 데이타 전송 처리량과 지연 시간의 측면 모두에서 버스 매트릭스의 성능을 향상시킬 수 있다는 것을 보여 주었다.

버스 매트릭스 및 NoC의 성능 예측을 위해 분석적인 기법들도 널리 연구되고 있다. [17]에서는 버스 매트릭스의 throughput 및 지연 시간을 구하기 위한 수식들을 유도하였으나 multiple-outstanding transaction 및 out-of-order completion 등을 고려하지 않았다. [18]에서는 NoC의 라우터에서 사용되는 버퍼의 크기를 최적화하기 위해 큐잉 이론에 기반을 둔 분석 기법을 제시하였으나 지수 분포를 갖는 메모리 시스템의 접근 시간을 가정하였다. 이는 서론에서 살펴보았듯이 현실적인 시스템의 메모리 구조에 대한 접근 시간을 고려할 수 없는 문제점이 있다.

기존 연구들과 비교해 본 논문은 두 가지 의의를 갖는다. 첫 번째, 제시된 성능 예측 기법은 multiple-outstanding transaction을 고려하였고, 메모리 시스템의 접근 시간이 일반 확률 분포를 가질 수 있다고 가정하여 실제적인 버스 매트릭스 구조 분석을 가능하게 한다. 또한 제안하는 기법은 multiple-outstanding transaction을 지원하는 버스 매트릭스에서 동시에 실행되고 있는 트랜잭션들의 평균 개수를 예측할 수 있어, 실제의 버스 매트릭스 구현에서 버스 트랜잭션을 저장해야 하는 버퍼의 크기를 최소화할 수 있도록 해준다.

두 번째, 제안하는 성능 예측 기법은 사이클 단위의 정확도를 갖는 시뮬레이션과 비교해 훨씬 빠르면서 작은 성능 예측 오차를 갖기 때문에 방대한 설계 공간에 대한 효율적인 탐색을 가능하게 해 준다. 가령 제안하는 성능 예측 기법과 시뮬레이션을 같이 이용한다면 설계 공간 탐색 시간을 크게 줄일 수 있다. 먼저 성능 예측 기법으로 전체 설계 공간을 탐색하여 의미 있는 설계 공간을 빠르게 가려낼 수 있다. 그 후 이에 대해서 시뮬레이션을 이용하여 정확하게 성능을 평가한다. 이러한 방법으로 시간이 오래 걸리는 시뮬레이션을 가능한 적게 이용하면서 우수한 해를 찾아낼 수 있다. 성능 예측 기법에 의해 가려낼 수 있는 설계 공간의 크기는 성능 예측 기법의 정확도에 따라 달라진다. 탐색 방법에 대한 하나의 예로써, 만약 예측 기법이 90%의 정확도를 가진다면 성능 예측 기법에 의해 가장 우수한 성능을 갖는 통신 구조와 비교해 10% 이하의 성능 차이를 갖는 통신 구조들은 시뮬레이션에서 가장 우수한 성능을 가질 확률이 있는 통신 구조가 될 것이라고 간주할 수 있다. 따라서 이들에 대해서만 시뮬레이션을 적용한다. 성능 예측 오차가 적을수록 시뮬레이션에서 평가해야 하는 통신 구조들의 수는 적어질 것이다.

3. 버스 매트릭스의 성능 예측 기법

3.1 배경 지식

버스 매트릭스는 2가지 측면에서 고성능을 제공한다. 첫 번째로 그림 1(a)와 같이 버스 매트릭스는 각 프로세서 코어(마스터)들과 메모리 코어(슬레이브)들의 사이에 독립적인 버스를 사용하여 데이타 접근 경로를 제공할 수 있다. 이를 통해 각 프로세서 코어들이 같은 메모리를 접근하지 않는 한 서로 다른 메모리들을 동시에 접근할 수 있도록 해 준다.

두 번째는 multiple-outstanding transaction과 out-of-order completion을 사용하여 버스 사용 효율을 높이는 것이다. 각 마스터들은 트랜잭션을 수행하기 위해 버스가 해당 마스터의 트랜잭션을 수행하기 시작했다는 응답(그림 1(b)의 'Response')을 받아야 한다. 버스가

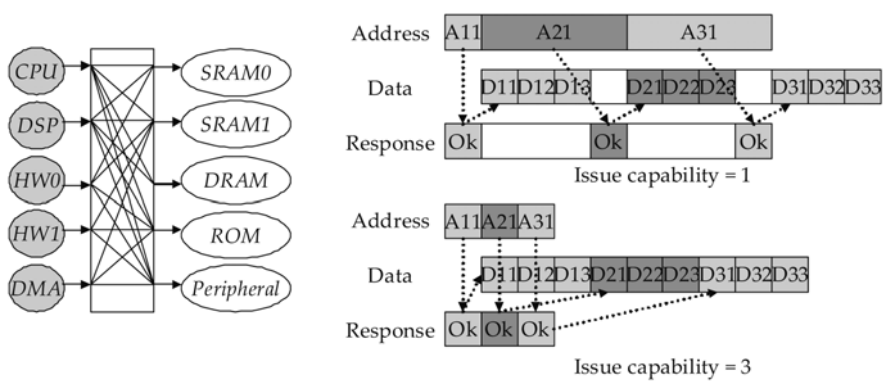


그림 1 (a) 버스 매트릭스, (b) issue capability에 따른 트랜잭션들의 수행 시간

동시에 처리할 수 있는 트랜잭션의 개수를 issue capability라고 한다. 만약 버스가 동시에 하나의 트랜잭션만 처리할 수 있을 때, (즉 issue capability가 1일 때) 버스를 사용하고자 하는 다른 마스터들은 현재 처리되고 있는 트랜잭션의 수행이 끝날 때까지 기다려야 한다. 반면 여러 개의 트랜잭션들이 동시에 수행될 때 (issue capability가 1 이상일 때) 마스터들은 현재 트랜잭션의 해당 슬레이브를 접근하는 동안 자신의 트랜잭션에 대한 응답을 버스로부터 즉시 받을 수 있어 버스 사용 효율을 높일 수 있다.

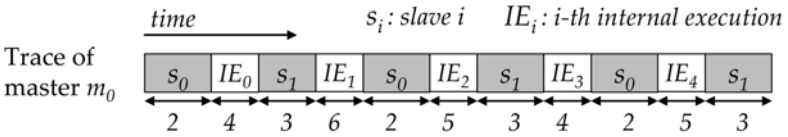
그림 1(b)는 issue capability가 1과 3인 경우에 3개의 연속된 트랜잭션들이 버스에서 처리되는 과정을 보여줌으로써 multiple-outstanding transaction이 버스의 성능을 어떻게 향상시킬 수 있는가를 보여준다. Issue capability가 1일 경우, 주소 'A21'부터 시작하는 두 번째 트랜잭션은 현재 진행중인 첫 번째 트랜잭션이 마지막 데이타 블록 'D13'을 받은 후에야 버스로부터 응답을 받을 수 있다. 이 때 응답에 필요한 1 사이클은 이전에 처리된 첫 번째 트랜잭션의 수행 시간과 중첩될 수 없다. 반면 그림 1(b)의 아래와 같이 issue capability가 3일 경우, 3개의 트랜잭션들이 동시에 버스에서 처리될 수 있다. 이는 두 번째와 세 번째 트랜잭션들이 즉시 버스로부터 응답을 받았다는 것을 의미하며, 이에 필요한 추가의 2 사이클은 이미 처리되고 있는 첫 번째 트랜잭션에 대한 데이타가 버스에 실려 있는 시간과 중첩된다. 따라서 issue capability가 1인 경우보다 응답에 필요한 시간을 2 사이클만큼의 지연 시간을 줄일 수 있으며 데이타 전송 처리량 또한 동시에 향상되는 효과를 가져온다.

Out-of-order completion을 이용할 경우, 3개의 트랜잭션에 대한 데이타들은 응답을 받은 순서와 관계 없이 임의의 순서로 도착할 수 있기 때문에 버스 사용 효율을 높일 수 있다. 가령 그림 1(b)의 첫 번째 트랜잭션이 느린 슬레이브를 접근할 경우 빠른 슬레이브를 접근하는 두 번째와 세 번째 트랜잭션들의 데이타가 먼저 도착할 수 있어 데이타 버스의 사용 효율을 높일 수 있다. 본 논문에서는 이러한 out-of-order completion은 고려하지 않고 응답을 받은 순서대로 데이타가 도착하는 in-order completion을 가정한다.

3.2 문제 정의 및 가정

본 논문에서 제안하는 성능 예측 기법은 다음과 같은 입력들을 가진다. 첫 번째로, 버스 매트릭스 통신 구조는 마스터들의 집합 $\Phi_M=\{m_i\}(i=0,..,N_M-1)$과 슬레이브들의 집합 $\Phi_S=\{s_i\}$ $(i=0,..,N_S-1)$을 갖는다. 설명의 편의를 위해 그림 1(a)과 같이 하나의 버스에는 하나의 슬레이브만 연결되어 있다. 가령, 그림 1(a)에서 두 개의 슬레이브 'DRAM'과 'ROM'은 하나의 버스에 연결되지 않는다. 또한 모든 마스터들은 모든 슬레이브들에 접근할 수 있는 버스 매트릭스를 고려하기로 한다.

두 번째 필요한 입력은 응용 예제가 실행되는 동안 발생하는 메모리 트레이스에 대한 통계적인 정보이다. 이는 주어진 응용 예제가 프로세서에서 실행될 때 발생하는 메모리 트레이스가 통신 구조에 의해 어떠한 지연도 발생하지 않는 것을 가정하여 얻어진다. 그림 2와 같이 트레이스는 하나의 마스터가 슬레이브들에 대한 일련의 접근들을 시간에 따라 나타낸 것이며 슬레이브를 접근하지 않는 시간(그림 2의 IE_i)들은 마스터 내부에서의 실행을 나타낸다. 마스터 m_i이 슬레이브 s_j에 접근하는 트랜잭션들의 간격을 나타내는 확률 변수를 $V_{i,j}$로 나타내며, 이는 $fv_{i,j}(v)=P\{V_{i,j}=v\}$로 정의되는 확률 밀도 함수를 갖는다. 또한 마스터 m_i이 슬레이브 s_j에 접근할 때 이에 대한 서비스 시간을 나타내는 확률 변수는 $L_{i,j}$이며, $fl_{i,j}(l)=P\{L_{i,j}=l\}$로 정의되는 확률 밀도 함수를 갖는다. 임의의 확률 변수 A에 대해서 $E[A]$와 $E[A^2]$는 A의 평균(또는 기대값)과 분산으로 정의한다. 예를 들어 $E[L_{i,j}]$는 마스터 m_i가 슬레이브 s_j를 접근하기 위해 필요한 시간(또는 서비스 시간)에 대한 확률 변수 $L_{i,j}$의 평균을 나타내며, 마찬가지로 $E[V_{i,j}]$는 $V_{i,j}$의 평균을 나타낸다.



Random variable	Probability density function
$V_{0,0}=\{IE_0+IE_1, IE_2+IE_3\}=\{10,9\}$	$fv_{0,0}(v)=0.5$ if $v=10$ or 9, 0 otherwise
$V_{0,1}=\{IE_1+IE_2, IE_3+IE_4\}=\{11,9\}$	$fv_{0,1}(v)=0.5$ if $v=11$ or 9,0 otherwise
$L_{0,0}=\{2\}$	$fl_{0,0}(l)=1$ if $l=2$, 0 otherwise
$L_{0,1}=\{3\}$	$fl_{0,1}(l)=1$ if $l=3$, 0 otherwise

그림 2 메모리 트레이스로부터 $fv_{i,j}(\cdot)$와 $fl_{i,j}(\cdot)$를 얻어내기 위한 과정

관련 연구에서 살펴본 것처럼 여러 개의 마스터들이 하나의 버스에 동시에 접근할 때 중재(bus arbitration)에 의해 발생하는 동적 지연을 예측하는 것이 정확한 성능 분석을 위해 매우 중요하다. 따라서 본 논문에서 풀고자 하는 문제는 주어진 응용 예제에 대해 각 마스터들의 메모리 트레이스로부터의 통계 정보를 얻은 후, 이를 이용하여 주어진 버스 매트릭스 구조에서 응용 예제의 실행 시간을 예측하는 것이다. 이를 위해 다음의 2가지를 가정한다.

첫째, 버스 마스터는 한번에 하나의 트랜잭션만을 생성할 수 있다. 실제로 하나의 마스터가 여러 개의 트랜잭션을 동시에 생성할 수 있지만 이런 마스터는 여러 개의 트랜잭션들을 발생하는 마스터들이 내부에 여러 개 있는 것으로 생각할 수 있으며, 이는 한번에 하나의 트랜잭션을 생성하는 여러 개의 마스터들로 구성되어 있는 것으로 고려하며 내부의 각 마스터들은 개별적인 메모리 트레이스를 갖는다고 가정한다.

두 번째, 버스는 최대의 성능을 내기 위해 버스에 연결된 마스터 개수만큼의 issue capability를 갖고 있다. 직관적으로 보다 많은 issue capability를 제공할수록 높은 성능을 얻을 수 있을 것으로 예상되지만, 주어진 응용의 메모리 접근 패턴 및 빈도 등의 통신 특성에 따라서 특정 issue capability 값 이상에서는 더 이상의 성능 향상이 없을 수도 있다. 가령 10개의 마스터가 연결된 버스는 issue capability는 최대 10이 될 것이지만, 응용 예제의 특성상 동시에 5개 이상의 마스터들이 동시에 하나의 버스에 접근하는 일이 없다면 이 버스에서는 5 이상의 issue capability가 필요하지 않을 것이다. 본 논문에서는 제안하는 성능 예측 기법을 이용하여 버스에서 평균적으로 처리되고 있는 트랜잭션의 개수를 구하여 이를 버스 매트릭스가 최대의 성능을 내기 위한 issue capability의 하한 값으로 사용할 수 있음을 보일 것이다.

마지막으로 동시에 하나의 슬레이브에 여러 개의 마스터들이 접근하려고 할 때 다양한 중재 방법이 가능하지만 본 논문에서는 특정 중재 방법을 가정하지 않는다. 이는 issue capability가 높아질수록 동시에 접근을 시도하는 마스터들이 중재 방법에 의한 영향은 크지 않을 것으로 예상되기 때문이다.

3.3 기본 버스 구조의 성능 예측 기법

하나의 버스로 이루어져 있는 기본 버스 구조에 대한 성능 예측 기법을 먼저 설명한다. 가정에 따라 기본 버스 구조에서는 하나의 슬레이브 s_0만 존재하므로, 편의상 위해 마스터 m_i의 슬레이브 s_0에 대한 트랜잭션이 발생하는 간격과 슬레이브에 접근 시의 서비스 시간(메모리 접근 시간)들에 대한 확률 변수 $V_{i,0}$ 와 $L_{i,0}$ 대신 V_i 와 L_i를 사용하기로 한다. 마찬가지로 확률 밀도 함수들도 $fv_{i,0}(\cdot)$와 $fl_{i,0}(\cdot)$ 대신 $fv_i(\cdot)$와 $fl_i(\cdot)$로 표현한다.

본 논문에서는 기본 버스 구조를 모델하기 위하여 M/G/1 큐잉 시스템을 이용한다. 'M'은 마스터들에 의해 트랜잭션이 발생하는 간격이 지수 분포(exponential distribution)라는 것을 나타내며, 'G'는 서버(메모리 시스템)의 서비스 시간에 대한 확률 분포가 일반 분포임을 나타낸다. 즉 $fl_i(\cdot)$는 일반 분포(general distribution)를 갖는다. 마지막의 '1'은 서버의 개수가 한 개임을 나타낸다.

그림 3은 3개의 마스터들을 가진 기본 버스 구조를 보여준다. 따라서 버스의 issue capability는 3이고, 마스터들의 트랜잭션들은 생성되는 즉시 버스로부터 응답을 받으면서 큐잉 모델의 서버 앞에 있는 FIFO에 저장된다. 실제 시스템의 구현에서는 FIFO는 메모리 컨트롤러 내부에 위치한다. FIFO에 저장된 트랜잭션들은 저장된 순서대로 처리되므로 슬레이브에 대한 동시 접근들의 중재 방법에 의한 영향은 작아지게 된다. 또한 중재에는 1 사이클이 필요하지만 이는 현재 처리되고 있는 트랜잭션의 수행 시간과 중첩되므로 여기에서는 고려하지 않는다.

마스터 m_i이 트랜잭션을 생성하는 빈도는 λ_i로 나타내며 트랜잭션을 발생하는 평균 간격의 역수와 같다.

$$\lambda_i = \frac{1}{E[V_i]} \tag{1}$$

이제 m_i이 트랜잭션을 생성한 후 서버로부터 이에 대한 서비스를 받기 전까지 기다려야 하는 대기 시간(즉

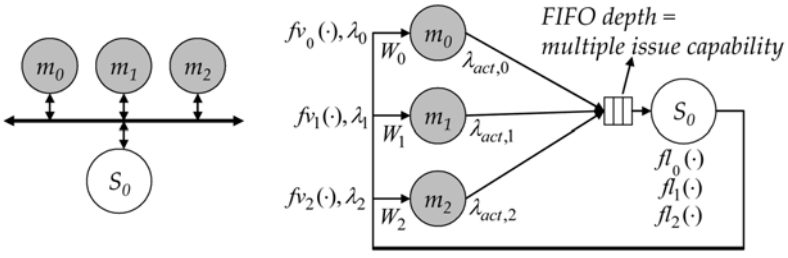


그림 3 3개의 마스터로 구성된 기본 버스 구조의 M/G/1 큐잉 모델

메모리에 대한 접근을 시작할 때까지 동적 충돌에 의해 지연되는 시간)에 대한 확률 변수를 W_i라고 하자. m_i은 현재 이미 수행 중인 트랜잭션이 완료될 때까지 기다려야 되기 때문에 m_i의 트랜잭션이 발생하는 간격은 다른 마스터의 트랜잭션들이 완료될 때까지 기다려야 하는 시간을 반영해야 한다. 이를 고려한 트랜잭션의 발생 빈도를 $\lambda_{act,i}$로 정의하고 다음과 같이 나타낼 수 있다.

$$\lambda_{act,i} = \frac{1}{E[V_i] + E[W_i]} = \frac{1}{1/\lambda_i + E[W_i]} \tag{2}$$

이제 마스터 m_i의 평균 대기시간 $E[W_i]$을 구한다. 마스터 m_i가 트랜잭션을 생성할 때 이전에 발생한 마스터들로부터의 트랜잭션들은 서비스를 받고 있거나 FIFO에서 대기하는 두 가지 경우 중의 하나이다. 현재 마스터 m_i가 서비스를 받고 있는 확률을 $P(B_i)$라고 정의하자. 마스터 m_i가 서비스를 받고 있는 임의의 순간에 시스템을 관찰했을 때 남아 있는 서비스 시간에 대한 확률 변수를 R_i로 정의하기로 한다. $E[R_i]$은 잔여 서비스 시간의 평균이 될 것이다. 그리고 N_i은 m_i으로부터 평균적으로 발생한 트랜잭션 개수에 대한 확률 변수를 나타낸다. 마스터들은 하나의 트랜잭션만을 생성하므로 평균적으로 발생한 트랜잭션의 개수 N_i는 트랜잭션이 발생되었는지 아닌지의 확률로도 생각할 수 있다. 따라서 마스터 m_i가 기다려야 하는 시간을 나타내는 확률 변수 W_i의 평균 $E[W_i]$은 다음과 같다.

$$E[W_i] = \sum_{j=0,\forall j\neq i}^{N_M-1} \Big(E[N_j]\cdot E[L_j] + P(B_j)\cdot E[R_j] \Big), \quad \forall i \in \{0,\ldots,N_M-1\} \tag{3}$$

Little의 법칙에 의해 다음과 같은 식이 성립한다[19].

$$E[N_i] = \lambda_{act,j}\cdot E[W_i] \tag{4}$$

또한 $P(B_i)$는 마스터 m_i에 의해 서버가 사용되고 있을 확률, 즉 서버 사용률을 나타내므로 아래의 식이 성립한다[19].

$$P(B_i) = \lambda_{act,i}\cdot E[L_i] \tag{5}$$

M/G/1 큐잉 시스템에서, 잔여 서비스 시간은 서비스 시간의 첫 번째 모멘텀과 두 번째 모멘텀인 $E[L_i]$과 $E[L_i^2]$으로 다음과 같이 표현할 수 있다[19].

$$E(R_i) = \frac{E[L_i^2]}{2E[L_i]} \tag{6}$$

식 (4), (5), (6)을 식 (3)에 대입하여 전개하면 다음과 같은 식을 얻는다.

$$E[W_i] = \sum_{j=0,\forall j\neq i}^{N_M-1} \lambda_{act,j}\cdot\left(E[W_j]\cdot E[L_j] + \frac{E[L_j^2]}{2} \right), \quad \forall i \in \{0,\ldots,N_M-1\} \tag{7}$$

$E[L_i^2]$는 $fl_i(\cdot)$를 이용하여 계산할 수 있다. 마스터 m_i의 $E[W_i]$는 다른 마스터들 m_j의 평균 지연 시간 $E[W_i]$에 의해 계산되기 때문에 각 버스 마스터들은 식 (7)을 가지며 모든 $E[W_i]$들이 수렴할 때까지 반복하여 식 (7)을 계산하여 구할 수 있다.

다음으로, FIFO에 저장되어 서비스를 기다리는 모든 트랜잭션의 개수를 나타내는 확률 변수를 N이라고 할 때 $N=\Sigma N_i$이고 $i=0,\cdots,N_M-1$이다. Little의 법칙을 다시 적용하여, $E[N]$는 다음과 같이 표현된다.

$$E[N] = \sum_{i=0}^{N_M-1} E[N_i] = \sum_{i=0}^{N_M-1} \lambda_{act,i}\cdot E[W_i] \tag{8}$$

현재 서비스를 받고 있는 트랜잭션까지 고려한다면, 시스템에는 $E[N]+1$의 트랜잭션들이 존재하고, 버스가 최대의 성능을 얻기 위해 필요한 issue capability의 하한 값 B_L은 아래와 같다.

$$B_L = \lceil E[N]+1 \rceil \tag{9}$$

엄밀하게 말해 식 (9)는 정확한 하한 값은 아니다. 만일 마스터들의 트랜잭션 생성 간격에 대한 편차가 심해, 평균 트랜잭션 생성 비율보다 훨씬 높은 비율로 메모리를 접근하려는 구간에서는 B_L보다 큰 값을 가져야 그 때의 버스 접근 요구들을 최대의 성능으로 처리할 수 있다. 그러나 평균적으로 처리되는 버스에서의 통신 요구량을 고려한다면, 식 (9)는 issue capability의 하한 값을 결정하기 위한 하나의 기준이 될 수 있다. 제시한 하한 값을 버스 매트릭스에서 최대의 성능을 얻기 위한 최소의 issue capability로 사용할 수 있고, 이에 대한 구체적인 결과를 실험 단락에서 보일 것이다.

3.4 버스 매트릭스의 성능 예측 기법

앞에서 설명한 기본 버스 구조에 대한 성능 예측 기법은 N_M개의 마스터와 N_S개의 슬레이브를 갖는 버스 매트릭스 구조로 확장할 수 있다. 그림 4는 3개의 마스터와 2개의 슬레이브를 갖는 버스 매트릭스를 나타낸다. 기본 버스 구조에 대한 분석 방법과 마찬가지로 몇 가지 항을 먼저 정의한다. 주어진 버스 매트릭스에서 마스터 m_i가 슬레이브 s_j를 접근하기 위해 기다려야 하는 대기 시간을 나타내는 확률 변수를 $W_{i,j}$로 정의하자. $\lambda_{i,j}$는 마스터 m_i가 슬레이브 s_j에 접근하는 트랜잭션을 생성하는 빈도를 나타낸다.

또한 $C_{i,j,k}$를 마스터 m_i가 슬레이브 s_k에 접근하는 연속된 두 개의 트랜잭션들 사이에 나타나는 다른 슬레이브 s_j를 접근하는 트랜잭션의 개수를 나타내는 확률 변수로 정의하자. $C_{i,j,k}$의 확률 밀도 함수 $fc_{i,j,k}(\cdot)$는 $fc_{i,j,k}(c) = P\{C_{i,j,k}=c\}$로 표현된다.

$\lambda_{i,j}$를 계산하기 위해서는 슬레이브 s_j가 아닌 다른 슬레이브들을 접근할 때 발생하는 대기 지연 시간들을 고

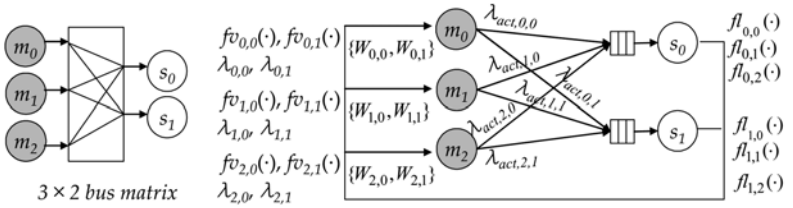


그림 4 3개의 마스터와 2개의 슬레이브로 이루어진 버스 매트릭스에 대한 큐잉 모델

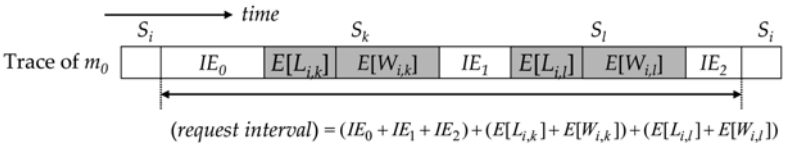


그림 5 m_i가 s_j를 접근하는 트랜잭션들 사이의 접근 간격은 이들 사이에 나타나는 다른 슬레이브들 s_k와 s_l에 대한 트랜잭션을 고려해야 한다.

려해야 하며 이로 인한 s_j에 대한 트랜잭션의 생성 빈도를 $\lambda_{act,i,j}$라고 정의하기로 한다. 예를 들어 그림 5와 같이, 슬레이브 s_i에 대한 연속적인 두 개의 트랜잭션들 사이에 다른 슬레이브 s_k와 s_l에 대한 트랜잭션들이 있다고 가정하자. s_i에 대한 두 트랜잭션의 간격은 슬레이브 s_k와 s_l를 접근하는 트랜잭션에 의해 발생하는 서비스 시간(메모리 접근 시간)과 대기 지연 시간에 의해 늘어난다. 따라서 접근 간격은 마스터 내부 실행 시간(IE_0+IE_1+IE_2)에 s_k와 s_l를 접근할 때 소요되는 시간들(메모리 접근 시간과 FIFO에서의 대기 시간)까지 합하여 고려해야 한다. 그림 5의 예제를 일반적으로 확장하면 다음과 같은 식을 얻는다.

$$\lambda_{act,i,j} = \frac{1}{E[V_{i,j}] + \sum_{s=0,\forall s\neq j}^{N_S-1} E[C_{i,j,s}]\cdot\left(E[L_{i,s}] + E[W_{i,s}]\right)}, \quad \forall i \in \{0,\ldots,N_M-1\} \tag{10}$$

$\lambda_{act,i,j}$를 계산하면, 버스 매트릭스를 구성하는 기본 버스들에 대해 식 (3)을 적용하여 버스 매트릭스에서 마스터들이 각 기본 버스의 슬레이브를 접근할 때 필요한 평균 대기 시간을 구할 수 있다.

3.5 태스크들의 실행 순서를 이용한 응용 예제의 성능 분석 방법

본 논문에서 제안한 성능 예측 기법을 이용하여 응용 예제의 성능을 평가하기 위한 가장 단순한 방법은 각 프로세서들의 버스 트랜잭션 생성 비율이 전체 실행 구간에 걸쳐 동일한 지수 분포를 갖는다고 가정하는 것이다. 그러나 실제 응용 예제에서는 프로세서들의 버스 트랜잭션들은 특정 수행 구간에서 집중되는 경향을 보인다. 영상 압축 알고리즘에서의 움직임 추정(motion estimation)이 그 대표적인 예라고 할 수 있다. 따라서 제안하는 성능 예측 기법으로는 이러한 경향을 반영할 수 없으므로 성능 예측의 결과가 정확하지 않을 가능성이 커진다.

이러한 문제를 해결하기 위한 한가지 방법은 그림 6

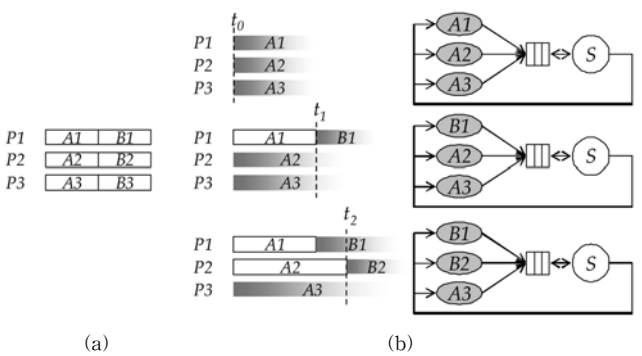


그림 6 (a) 2개의 블록으로 나누어진 각 프로세서들에서 실행 구간, (b) 프로세서들에서 실행되는 구간에 따른 큐잉 시스템의 구성

과 같이 응용 예제의 전체 실행 구간을 앞서 언급한 버스 트랜잭션들의 특성을 고려하여 몇 개의 실행 구간들로 나누어 해당 구간이 각 프로세서에서 실행될 때를 고려하여 개별적으로 큐잉 시스템을 구성하는 것이다. 전체 수행 구간을 나누기 위해 본 논문에서는 주어진 응용 예제가 여러 개의 기능 블록들로 나누어져 있고 블록들의 실행 순서는 정적으로 정해져 있다고 가정하였다. 이에 대한 성능 예측 방법은 스케줄에 따라 각 프로세서들에서 수행되는 기능 블록들의 단위로 큐잉 시스템을 구성하여 해당하는 큐잉 시스템을 계산해 나가는 것이다[6]. 가령, 그림 6(a)와 같이 세 개의 프로세서들이 각각 두 개의 실행 구간들을 갖고 있다고 할 때, 그림 6(b)의 윗 그림처럼 시간 t_0 에서는 성능 예측 기법을 실행 구간 A1, A2, A3에서의 메모리 트레이스들의 통계 정보를 이용한 큐잉 시스템에 대하여 적용한다. 그 후, 그림 6(b)의 가운데 그림과 같이 시간 t_1 이후에 A1이 가장 먼저 끝났다고 가정할 때 프로세서 P1은 새로운 실행 구간 B1을 수행하게 되므로 B1, A2, A3로 이루어진 새로운 큐잉 시스템을 고려해야 한다. 이러한 과정을 각 프로세서들의 모든 실행 구간이 수행될 때까지 반복한다.

4. 실험 및 분석

제안한 성능 예측 기법의 정확성과 실행 속도를 검증하기 위하여 SystemC[20]로 구현된 사이클 단위 정확도를 갖는 버스 매트릭스 시뮬레이터와 비교한 결과를 제시한다. 사용된 버스 매트릭스 시뮬레이터는 사용자가 지정 가능한 issue capability에 의한 multiple-outstanding transaction과 out-of-order completion 프로토콜을 지원할 수 있다. 그러나 앞서 가정한 바와 같이 out-of-order completion을 고려하지 않으므로 시뮬레이터도 in-order completion으로 동작하도록 설정하였으며 메모리는 워드 읽기/쓰기에 각각 1 사이클이 필요한 온-칩 SRAM을 가정하였다. 대부분의 온-칩 SRAM은 이러한 타이밍 특성을 갖는다. 여러 마스터들이 동시에 특정 버스를 요구할 때 고정된 우선 순위를 이용한 중재 방법을 이용한다. 또한 모든 버스는 32-bit의 데이타 폭을 가지며, 메모리들과 버스들은 하나의 클럭으로 동작한다고 가정하였다. 모든 실험은 3.0-GHz Xeon 프로세서와 4.0-GB의 메인 메모리로 구성되었고, 리눅스를 운영체제로 하는 워크스테이션에서 수행하였다.

4.1 무작위로 생성된 트레이스들에 대한 성능 예측 기법의 정확도 분석

첫 번째 실험에서는 다양한 마스터 개수(2, 4, 8, 16)와 트랜잭션 생성 비율(0.1, 0.2, 0.3)을 갖는 기본 버스 구조에 대해 제안한 성능 예측 기법의 정확도를 분석하였다. 대표적인 멀티미디어 응용 예제인 H.263 영상 압축기나 MP3 음성 압축기의 경우 0.15 이하의 트랜잭션 생성 빈도를 갖는다[6]. 0.2 이상의 값은 버스와 메모리 시스템에 매우 큰 부하를 주는 트랜잭션 생성 빈도라고 생각할 수 있다. 마스터의 개수와 트랜잭션 생성 비율의 조합에 대해, 각 마스터들이 100,000번의 버스 트랜잭션을 갖도록 메모리 트레이스를 포아송 분포(Poisson distribution)에 따라 100번씩 생성하였다. 하나의 트랜잭션은 2, 4, 또는 8번의 워드 읽기나 쓰기들이며 이는 균일 분포(uniform distribution)의 확률에 의해 선택된다. 따라서 트랜잭션들에 대한 서비스 시간은 2, 4, 8 사이클 중의 하나가 된다. 각 마스터들 사이의 실행 의존성은 존재하지 않는 것으로 가정하여 병렬적으로 트랜잭션들을 수행할 수 있으며 모든 트랜잭션들이 종료될 때까지의 시간을 성능 예측 기법으로 얻은 후 시뮬레이션과의 상대적인 정확도를 그림 7(a)에 제시하였다. 시뮬레이션의 정확도를 100%라고 했을 때 제시한 성능 예측 기법의 정확도는 평균 94% 이상임을 보여주고 있다.

또한 제안한 성능 예측 기법이 다양한 트레이스들에 대해 얼마나 일관된 예측 정확도를 보이는지 확인하기 위하여 100번의 메모리 트레이스 생성에 대한 예측 오차들의 표준 편차들을 그림 7(b)에 제시하였다. 성능 예측 기법은 다양한 메모리 트레이스에 대해 3% 이하의 표준 편차를 보여준다. 이로부터 버스 트랜잭션 생성 비율이나 마스터 개수의 변화에 관계없이 일정한 성능 예측의 정확도가 유지되고 있음을 알 수 있다.

제안하는 성능 예측 기법을 다양한 개수의 마스터(16, 24, 32)와 슬레이브(8, 16)를 갖는 버스 매트릭스에 적용하여 시뮬레이션과 비교한 정확도를 그림 8에 나타내었다. 모든 마스터들은 모든 슬레이브에 접근할 수 있으므로, 16개의 마스터와 8개의 슬레이브를 갖는 버스 매트릭스(그림 8의 그래프에서 x축의 16×8에 해당)라면 이는 16개의 마스터를 갖는 기본 버스를 8개 갖고 있는 셈이 된다. 그림 7에서의 실험과 마찬가지로 3가지의 트랜잭션 발생 비율을 적용하였다. 따라서 모든 파라메터들을 조합하여 18가지의 버스 매트릭스 구조들을 고려할 수 있으며 각 구조들에 대해 100번의 메모리 트레이스들이 생성되었다. 그림 8에 나타난 바와 같이, 기본 버스 구조에 대한 실험 결과와 비슷한 경향이 관찰된다. 많은 버스를 갖고 있는 복잡한 구조의 버스 매트릭스에 대해서도 제안한 성능 예측 기법은 시뮬레이션과 비교해 모든 경우에서 94% 이상의 정확도를 보여주었다.

제안하는 예측 기법의 실행 시간을 표 1에 제시하였다. 16개의 슬레이브를 사용하는 세 가지의 버스 매트릭스 구조를 대상으로 하였다. 예측 기법의 수행 속도는 시뮬레이션보다 1000배 이상 빠른 것으로 나타났다. 시

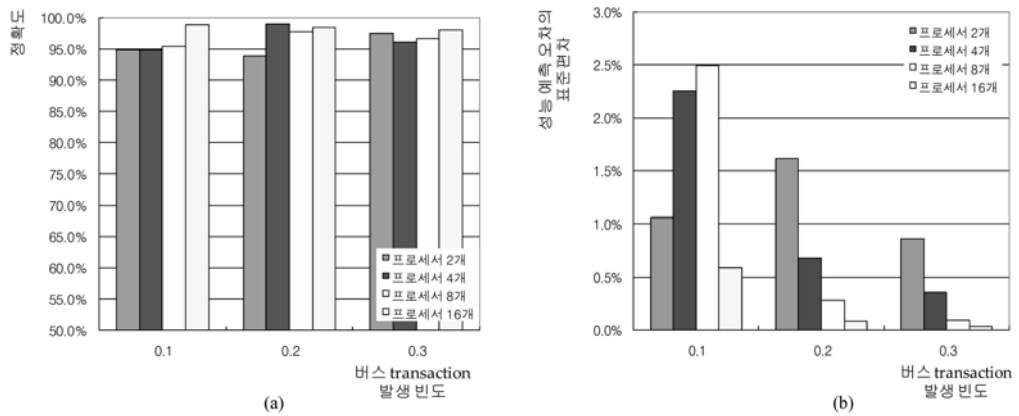


그림 7 기본 버스 구조에 대해 제안하는 성능 예측 기법을 시뮬레이션과 비교한 정확도

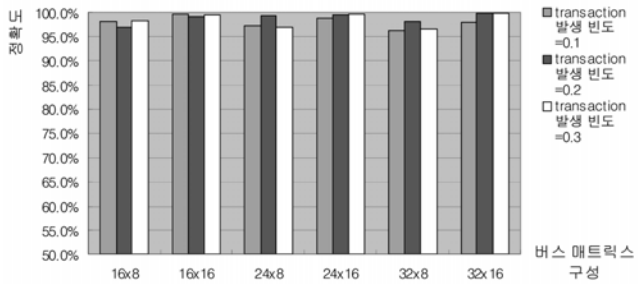


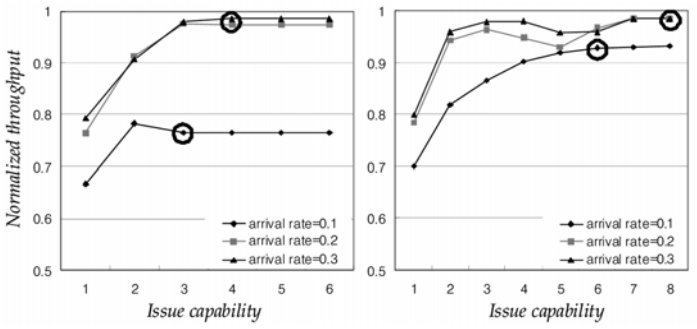
그림 8 버스 매트릭스에 대해 제안하는 예측 기법으로 얻어진 결과의 정확도

표 1 제안하는 성능 예측 기법과 시뮬레이션의 실행 시간 비교

버스 매트릭스	16×16	24×16	32×16
제안하는 성능 예측 기법	< 0.1 초	< 0.1 초	< 0.1 초
시뮬레이션	7.64 초	14.5 초	25.7 초

뮬레이션의 실행 시간은 메모리 트레이스의 길이에 비례하여 증가하지만, 제안하는 기법은 통계 정보만을 이용하므로 트레이스의 길이에 영향을 받지 않는다.

다음으로는 식 (9)에서 제시한 issue capability의 하한 값의 정확도를 알아보기 위해 시뮬레이션을 통해 issue capability에 따른 데이타 전송 처리량의 변화를 그림 9에 제시하였다. 각 그래프에서 굵은 원으로 표시된 부분은 식 (9)에서 제시된 버스의 최대 성능을 얻기 위한 issue capability의 하한 값을 나타낸다. 그래프에서 나타난 바와 같이 굵은 원으로 표시된 이후의 issue



(a) 마스터 4개 (b) 마스터 8개

그림 9 Issue capability에 따른 데이타 전송 처리량(throughput)의 변화

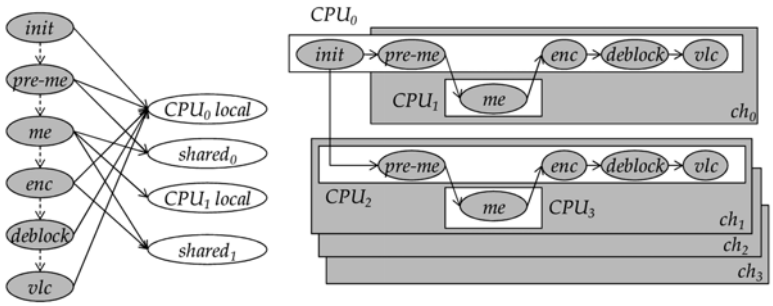
capability에서는 버스의 성능 향상이 더 이상 일어나지 않는 것을 볼 수 있으며, 이로부터 식 (9)의 정확성을 확인할 수 있다.

4.2 4-채널 DVR 응용 예제를 이용한 성능 예측 기법의 정확도 분석

두 번째 실험에서는 4-채널 DVR(Digital Video Recorder) 응용 예제를 이용하여 제안한 성능 예측 기법의 정확성 및 효율성을 검증한다. 4-채널 DVR의 각 채널은 외부에서 들어오는 영상 입력을 받아 H.264 알고리즘을 이용하여 압축, 저장한다. 그림 10(a)는 H.264 알고리즘을 구성하는 태스크들과 접근하는 이들이 사용하는 메모리 블록들을 나타낸다. 각 메모리 블록들은 프로세서들 사이의 통신을 위해 사용되는 공유 메모리나, 하나의 프로세서에 의해 독점적으로 사용되는 전용 메모리 블록들을 나타낸다. 그림 10(b)은 각 태스크들의 프로세서에 대한 할당을 나타낸다. 각 채널들에서 '*me*' (motion estimation) 블록은 빠른 수행을 위해 전용 프로세서에 할당되며, 나머지 태스크들은 범용 프로세서에 할당되어 실행된다고 가정하였다. 첫 번째 채널(ch_0)은 외부에서 영상 입력이 들어오면 각 채널들을 제어하여 영상 압축을 시작할 수 있도록 하는 역할을 추가로 수행한다. 그림 10(b)의 메모리 블록들이 물리적 메모리에 할당되어 그림 11과 같이 3개의 버스 매트릭스 구조를 생성할 수 있다. 최적화된 메모리 할당에 대한 논의는 본 논문의 범위를 벗어나므로, 여기에서는 각 버스 매트릭스들이 성능의 차이를 나타낼 수 있도록 메모리 블록들을 직관적으로 분할하여 할당하였다.

4-채널 DVR을 이용한 실험은 다음과 같이 진행되었다. 먼저 H.264 알고리즘의 메모리 접근 특성을 파악하기 위해 50 프레임의 시뮬레이션에 대한 트레이스로부터, H.264 알고리즘을 구성하는 각 태스크들의 메모리 접근에 대해 메모리 접근 빈도 및 횟수의 2가지 인자들의 최소, 최대값을 찾아내었다. 그리고 이 값들 사이에서 메모리 접근 빈도와 횟수를 갖도록 하여 각 태스크들의 트레이스를 무작위로 1000개씩 생성하였다. 이들을 이용하여 4.1장에서의 실험과 마찬가지로 트레이스 기반 시뮬레이션과 제안한 성능 분석 기법을 비교하여 그림 12에 나타내었다. 첫 번째 실험과 달리 여러 개의 태스크들이 정해진 스케줄에 의해 실행되므로 3.5장에서 설명한 태스크들의 스케줄을 이용한 성능 예측 기법을 사용하였다. 그림 12의 각 그래프는 그림 11의 물리적 메모리 개수에 의해 구분되는 버스 매트릭스들에 해당한다. 1000번의 시뮬레이션 결과들에 대해 실행 시간 순으로 정렬하였고 이에 대한 성능 예측 결과를 함께 나타내었다. 평균 성능 예측 정확도는 그림 12(a)부터 (c)까지 각각 97.7%, 95.3%, 97.0%이다.

그림 13은 제안하는 성능 예측 기법이 메모리 접근 시간을 일반 분포로 다루었던 본 논문의 방법과 달리



(a) 각 채널의 태스크 모델 (b) 태스크들의 프로세서에 대한 분할

그림 10 4-채널 DVR 응용 예제

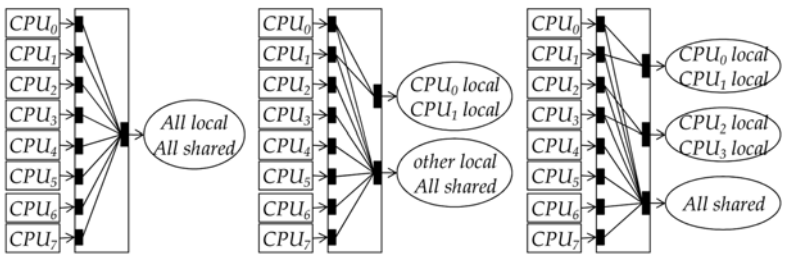


그림 11 논리적 메모리 블록의 물리적 메모리 블록 할당에 의한 3개의 버스 매트릭스 구조

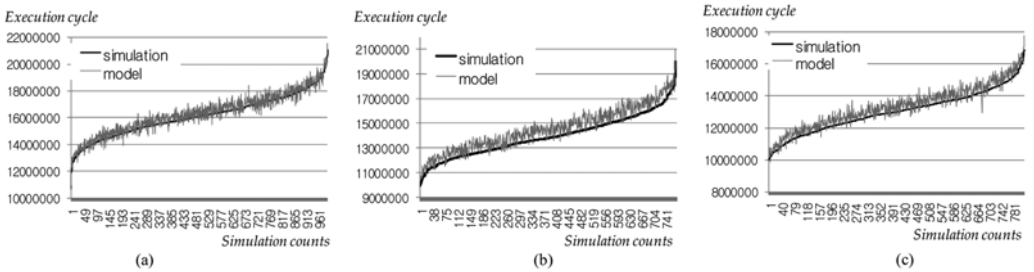


그림 12 4-채널 DVR에 대해 시뮬레이션과 비교한 제안한 성능 예측 기법의 정확도
(물리적 메모리 개수: (a) 1개, (b) 2개, (c) 3개)

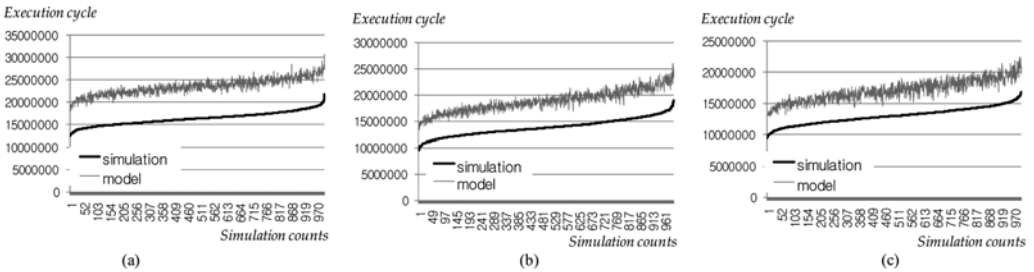


그림 13 지수 분포 기반의 메모리 접근 시간을 이용할 때 성능 예측 기법의 정확도
(물리적 메모리 개수 (a) 1개, (b) 2개, (c) 3개)

이를 지수 분포로 가정하였을 때의 정확도를 나타낸 것이다. 그림 12의 실험과 동일한 과정을 거쳤으며, 지수 분포에 의한 잔여 서비스 시간은 평균 서비스 시간과 동일하기 때문에 식 (6)의 잔여 서비스 시간은 $E(R_i)=E[L_i]$로 바뀌어야 한다. 3개의 그래프에 나타난 바와 같이 메모리 접근 시간에 대한 지수 분포의 서비스 모델을 사용하였을 때는 시뮬레이션과 비교하여 성능 예측 오차가 급격하게 커지는 것을 볼 수 있다. 그림 13의 실험에서의 평균 정확도는 63.0%로서, 이는 그림 12에 나타난 바와 같이 일반 분포를 이용한 메모리 접근 시간 모델의 정확성을 보여준다.

반면 표 2에 나타난 바와 같이 제안한 성능 예측 기법은 시뮬레이션보다 훨씬 빠르게 주어진 버스 매트릭스 구조에 대해 응용 예제의 성능을 예측할 수 있다. 4-채널 DVR과 같은 실제 응용 예제의 트레이스를 이용한 시뮬레이션 시간은 4.1장에서의 예제보다 평균 10배 이상 길다. 반면 제안한 성능 예측 기법이 필요로 하는 계산 시간은 이전 실험과 비슷하다. 이는 본 논문에서 제안한 방법과 같은 정적인 분석 기법의 효율성을 보여주는 좋은 예로서, 시뮬레이션은 주어진 응용이 크고 복잡해지거나 고려하는 통신 구조가 복잡해질수록 성능 예측에 필요한 시간이 기하급수적으로 증가하는 문제가 있다. 반면 제안하는 성능 예측 기법은 통계적 분석에 필요한 몇 개의 인자들만으로 주어진 응용 예제나 통신 구조를 모델하므로 시뮬레이션과 달리 응용 예제나 시스템의 복잡도 증가에 훨씬 덜 민감하다.

표 2 4-채널 DVR에 대한 성능 예측 기법과 시뮬레이션의 실행 시간 비교

버스 매트릭스의 물리적 메모리 개수	1	2	3
제안하는 성능 예측 기법	< 0.1 초	< 0.1 초	< 0.1 초
시뮬레이션	297 초	259 초	245 초

4.3 프로세서의 메모리 접근 모델

본 논문에서 사용한 프로세서의 메모리 접근 모델은 메모리 접근들의 발생 간격이 지수 분포인 프아송 프로세스를 가정하고 있다. 이러한 트래픽 모델에 대한 가정이 맞지 않는 응용 예제도 존재할 것이다. 그러나 관련 연구[18]에서 논의된 바와 같이 프아송 프로세스는 하나의 변수(3.3장의 λ_i 및 3.4장의 $\lambda_{i,j}$이 이에 해당)만을 이용하여 정확한 시스템의 성능 예측을 가능하게 해주는 유일한 수학적인 모델이다. 이러한 이유로 프아송 모델은 다양한 분야에서 시스템의 성능을 예측하기 위한 수학적인 도구로 널리 사용된다. 더욱 정확한 성능 분석을 위해 “self-similar”나 “long-range dependent”의 특성을 갖는 확률 프로세스를 프로세서의 트래픽 모델로 사

용할 수 있다[21]. 그러나 이러한 확률 프로세스들은 본 논문에서 제안한 바와 같이 '시스템 전체의 성능'을 분석할 수 있는 수학적 모델을 만들기에는 너무 복잡하다. 그러므로 시스템의 전체 구조가 결정된 후 특정 부분에 대한 세부 조정 등에 주로 이용된다[2][18]. 따라서 복잡한 SoC 설계를 위해 개발 초기 단계에서 다양한 통신 구조를 빠르게 탐색하기 위한 목적으로는 지수 분포를 가정한 프로세서의 트래픽 모델이 충분히 의미가 있을 것이라고 생각된다.

5. 결론 및 향후 과제

본 논문에서는 큐잉 모델에 기반한 온-칩 버스 매트릭스의 성능 예측 기법을 제안하였다. 이는 빠르고 정확하게 버스 매트릭스의 성능을 예측할 수 있어 MPSoC의 통신 구조에 대해 효율적인 설계 공간 탐색을 가능하게 해준다. 제안하는 예측 기법은 시뮬레이션과 비교해 94% 이상의 정확도를 보여주며 동시에 1000배 이상 빠른 실행 시간을 가진다. 본 논문의 향후 과제는 다음과 같다. 제안한 성능 예측 기법에서는 버스의 특정 중재 방법을 가정하지 않았다. 실제로 우선 순위 중재 방법을 이용한 시뮬레이션과의 비교에서 정확한 성능 예측이 가능함을 보여주었지만 round-robin, TDMA 등과 같은 다양한 중재 방법에 대해서 얼마나 정확한 성능 예측을 보여주는지에 대한 정량적인 평가가 필요하다. 다단계 버스 매트릭스 구조와 NoC에 적용할 수 있도록 제안한 성능 예측 기법을 확장하는 것도 향후 연구 과제 중의 하나로 고려하고 있다.

참 고 문 헌

[1] M. Loghiy, F. Angiolini, D. Bertozzi, and L. Benini, "Analyzing on-chip communication in a MPSoC Environment," in *Proc. Design Automation and Test in Europe*, pp. 752-757, Feb. 2004.

[2] Niagara 2 Opens the Floodgates, Microprocessor Report, Nov. 2006.

[3] Advanced AMBA 3 Interconnect IP (PL301), ARM, http://www.arm.com/products/solutions/PL301_AMBA3AXI.html

[4] K. Lahiri, A. Raghunathan, and S. Dey, "Design space exploration for optimizing on-chip communication architectures," *IEEE Transactions on Computer-Aided Design of integrated circuits and systems*, Vol.23, No.6, Jun. 2004.

[5] S. Kim and S. Ha, "Efficient exploration of bus-based System-on-Chip architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, Vol.14, No.7, pp. 681-692, Jul. 2006.

[6] AXI, ARM, http://www.arm.com/products/solutions/AMBA3AXI.html

[7] J. Yoo, S. Yoo, and K. Choi, "Communication architecture synthesis of cascaded bus matrix," in *Proc. ASP-DAC*, pp. 171-177, Jan. 2007.

[8] S. Pasricha, N. Dutt, and M. Ben-Romdhane, "Constraint-driven bus matrix synthesis for MPSoC," in *Proc. Asia and South Pacific Design Automation Conference*, pp. 30-35, Jan. 2006.

[9] O. Ogawa, S. Bayon de Noyer, P. Chauvet, K. Shinohara, Y. Watanabe, H. Niizuma, T. Sasaki, and Y. Takai, "A practical approach for bus architecture optimization at transaction level," in *Proc. Design Automation and Test in Europe*, pp. 176-181, Mar. 2003.

[10] S. Kim, C. Im, and S. Ha, "Schedule-aware performance estimation of communication architecture for efficient design space exploration," *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, Vol.13, No.5, pp. 539-552, May 2005.

[11] K. Lahiri, A. Raghunathan, and S. Dey, "System-level performance analysis for designing system-on-chip communication architecture," *IEEE Transactions on Computer-Aided Design of integrated circuits and systems*, Vol.20, pp. 768-783, Jun. 2001.

[12] P. V. Knudsen and J. Madsen, "Communication estimation for hardware/software codesign," in *Proc. International Workshop on Hardware/Software Codesign*, pp. 55-59, Mar. 1998.

[13] P. Knudsen and J. Madsen, "Integrating communication protocol selection with partitioning in hardware/software codesign," in Proc. *International Symposium on System Level Synthesis*, pp. 111-116, Dec. 1998.

[14] S. Pasricha, N. Dutt, E. Bozorgzadeh, and M. Ben-Romdhane, "FABSYN: Floorplan-aware bus architecture synthesis," *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, Vol.14, No.3, pp. 241-253, Mar. 2006.

[15] S. Murali, L. Benini, and G. De Micheli, "An application-specific design methodology for on-chip crossbar generation," *IEEE Transactions on Computer-Aided Design of integrated circuits and systems*, Vol.26, No.7, pp. 1283-1296, Jul. 2007.

[16] E.-G. Jung, J.-G. Lee, S.-H. Kwak, K.-S. Jhang, J.-A Lee, and D.-S. Har, "High performance asynchronous on-chip bus with multiple issue and out-of-order/In-order completion," in *Proc. ACM Great Lake Symposium on VLSI*, pp. 152-155, Apr. 2005.

[17] S. Lee and S.-C. Park, "Transaction analysis of multiprocessor based platform with bus matrix," in *Proc. Workshop on System-on-Chip for Real-Time Applications*, pp. 552-556, Jul. 2005.

[18] J. Hu, U. Y. Ogras, and R. Marculescu, "System-level buffer allocation for application-specific Net-

2) Network-on-Chip의 라우터 내부의 버퍼 크기가 좋은 예이다.

works-on-Chip router design," *IEEE Transactions on Computer-Aided Design of integrated circuits and systems*, Vol.25, No.12, pp. 2919-2933, Dec. 2006.
[19] L. Kleinrock, "Queueing systems, Volume I: Theory," Wiley Interscience, New York, 1975.
[20] SystemC Language Reference Manual, ver 2.1. (2005, May). http://www.systemc.org/web/sitedocs/lrm_2_1.html
[21] G. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for mpeg-2 video applications," *IEEE Transactions on Very Large Scale Integration (VLSI) systems*, Vol.12, No.1, pp. 108-119, Jan. 2004.

김 성 찬

1998년 서울대학교 재료공학부 학사. 2000년 서울대학교 컴퓨터공학부 석사. 2005년 서울대학교 전기컴퓨터공학부 박사 2005년~2006년 삼성전자 반도체총괄 SystemLSI 사업부 책임연구원. 2006년~현재 서울대학교 전기컴퓨터공학부 박사 후 과정. 관심분야는 하드웨어-소프트웨어 통합 설계, MPSoC의 성능 분석 및 구조 최적화

하 순 회

1985년 서울대학교 전자공학과 학사. 1987년 서울대학교 전자공학과 석사. 1992년 미국 University of California, Berkeley, 전기컴퓨터공학과 박사. 1993년~1994년 현대전자 근무. 1994년~현재 서울대학교 전기컴퓨터공학부 교수. 관심분야는 임베디드 시스템의 하드웨어-소프트웨어 통합 설계 및 설계 방법론, MPSoC의 임베디드 소프트웨어 설계