병렬 컴퓨터 시스템 성능 평가를 위한 컴퓨터 시뮬레이션 기술 동향

Simulation Techniques for Parallel Computer System Performance Evaluation

정용화(Y. Chung) 성능평가연구팀 선임연구원
박진원(J.-W. Park) 성능평가연구팀 책임연구원, 팀장
윤석한(S. Yoon) 컴퓨터시스템연구부 책임연구원, 부장

최근 들어 병렬 컴퓨터 시스템의 설계를 평가하기 위한 컴퓨터 시뮬레이션이 활발하게 연구되고 있다. 본 고는 컴퓨터 시뮬레이션 과정에 포함되는 여러 가지 작업 부하 생성 방법에 대해 살펴본다. 각 방법에 대한 정의, 특징을 살펴본 후 대표적인 구현 사례를 요약해본다. 먼저 분포 구동 시뮬레이션 방법과 이를 위하여 사용되는 시뮬레이션 언어에 대해 살펴본 후, 트레이스를 이용하는 추적 구동 시뮬레이션 방법과 병렬 컴퓨터 시스템용 트레이스를 생성하는 방법에 대해서 알아본다. 마지막으로 최근에 각광 받고있는 수행 구동 시뮬레이션 방법과 프로그램 구동 시뮬레이션에 대하여 살펴본다.

I. 서 론

컴퓨터 기술은 매우 빠른 속도로 발전해가고 있으며, 그 중에서도 가장 활발하게 연구가 진행중에 있는 분야는 병렬 처리(parallel processing) 분야이다. 즉, 병렬 컴퓨터 시스템은 대규모 계산이 수반되는 문제를 처리하기 위해 필요한 계산 능력을 효과적으로 제공하고 있으며 이러한 추세는 계속될 전망이다. 따라서 병렬 컴퓨터 시스템의 성능을 예측하기 위한 효과적인 방안을 모색할 필요성이 증대되고 있으며, 이러한 대규모 컴퓨터 시스템을 실제로 구현하기 전에 미리 제안된 아키텍처의 특성을 평가해 보는 것은 매우 중요한 일이다[1, 3]. 이러한 평가에 사용되는 도구는 제안되는 시스템 뿐만 아니라 기존의 시스템 성능 향상에도 매우 유용하게 사용될 수 있다. 그러나, 다중프로세서 아키텍쳐를 갖는 병렬 컴퓨터 시스템은 순차적인 컴퓨터 시스템에 비하여 모델링하기가 훨씬 어려운데, 이는 아키텍쳐와 병렬 프로그램 동작간의 매우 복잡한 상호작용 때문이다.

최근에 설계되고 있는 공유메모리 다중프로세서 시스템에서 전체 시스템 성능에 가장 결정적인 영향을 미치는 부분은 메모리 시스템이다. 즉, 공유메모리 다중프로세서 시스템에서 프로세서의 이용도를 제고시켜 전체 시스템의 성능을 향상시키려면, 메모리 시스템이 프로세서에게 초당 수백만 번의 메모리 접근을 제공할 수 있어야 한다[4]. 프로세서와 메모리 간의 동작 과정은 컴퓨

터 시뮬레이션을 통하여 모델링 과정을 거친 후 그 성능이 평가될 수 있다.

일반적으로 다중프로세서의 메모리 시스템 성능을 평가하기 위한 시뮬레이션 모델은 크게 각 프로세서로부터의 메모리 접근 생성기와 메모리 시스템의 시뮬레이터 두 부분으로 구성된다(그림 1). 메모리 시스템 시뮬레이터는 캐쉬 및 메모리뿐만 아니라 상호연결망까지도 시뮬레이트 한다. 메모리 접근 생성기는 병렬 컴퓨터 시스템에서의 프로세서 역할을 수행한다. 즉, 프로세서의 활동을 시뮬레이트 하고, 메모리 접근이나 명령을 캐쉬나 메모리를 표현하는 메모리 시스템 시뮬레이터로 보낸다. 만약 시뮬레이션이 하나의 프로세서를 갖는 컴퓨터 시스템상에서 수행된다면, 시뮬레이트 되는 복수 개의 프로세스들은 메모리 접근 생성기의 제어에 의하여 하나의 프로세서를 시분할하여 공유한다. 메모리 접근 생성기에 의한 제어의 한 예로는 어떤 프로세스가 메모리 시스템 시뮬레이터에 접근할 때마다 다른 프로세스로 교체되고, 그 교체된 프로세스 또한 다음 메모리 접근 때까지만 수행됨으로써 하나의 프로세서를 공유할 수 있다. 또 다른 제어의 예로는 시뮬레이트되는 매 클럭 싸이클마다 프로세스를 재선택함으로써 프로세서 공유가 가능하다. 메모리 접근을 시뮬레이터에서 생성하는 방법, 즉 구체적인 작업부하 생성 방법에는 다음과 같은 4가지 기법이 있다.

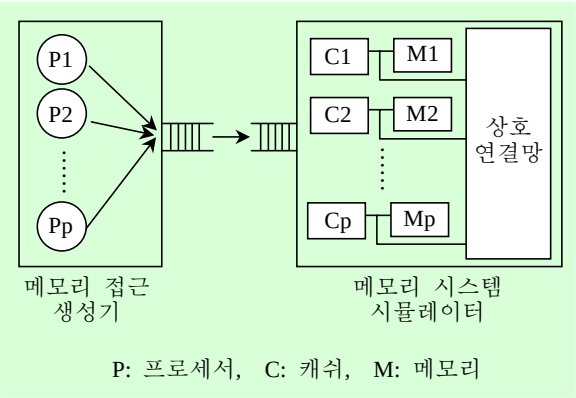
- 분포 구동(distribution-driven) 시뮬레이션: 작업부하가 메모리 접근분포의 통계적 모델로 모델링 된다.
- 추적 구동(trace-driven) 시뮬레이션: 목표 시스템과 유사한 시스템에서 작업부하를 수행시키거나 메모리 접근 생성기를 이용하여 메모리 접근의 트레이스를 한번 생성해낸다.
- 수행 구동(execution-driven) 시뮬레이션: 작업부하가 호스트 컴퓨터에서 수행된다. 공유메모리 접근과 같이 관심 있는 사건이 발생하면, 제어가 시뮬레이션 소프트웨어로 넘어가서 그 메모리 시스템을 시뮬레이트 한다.
- 프로그램 구동(program-driven) 시뮬레이션: 목표 시스템의 메모리 시스템뿐만 아니라 프로세서도 시뮬레이트 된다. 작업부하는 시뮬레이트되는 프로세서에서 수행된다.

본 고에서는 병렬 컴퓨터 시스템의 성능을 평가하기 위한 이러한 시뮬레이션 구동 방법 중 분포 구동, 추적 구동, 수행 구동 등 대표적인 3가지 작업부하 생성 방법에 대해 살펴보기로 한다. 프로그램 구동 시뮬레이션 방법에 대해서는 그 특성이 유사한 수행 구동 시뮬레이션 방법을 설명하면서 간단히 그 차이점만을 언급하겠다.

II. 분포 구동 시뮬레이션

분포 구동(distribution-driven) 시뮬레이션은 시뮬레이션을 구동하기 위하여 프로그램의 통계적 모델을 이용한다. 이 모델은 대개 프로그램의 수행 중에 시스템의 모듈사이를 이동하는 데이터의 생성을 모델링하는 확률(random) 프로세스의 형태를 가지며, 이는 조건부 분기(conditional branch)의 해결을 통계적으로 특성화하는 효과가 있다. 이 방법의 가장 큰 장점은 프로세서 동작의 내부적인 세부사항을 시뮬레이트 하는 방법보다 시뮬레이션이 훨씬 간단하고 빠르다는 점이고,

가장 큰 단점으로는 실제 병렬 프로그램의 적절한 통계적 모델을 구하기 어렵기 때문에 결과에 대한 정확도가 상대적으로 낮다는 점이다. 또한, (그림 1)에 나타낸 것과 같이 메모리 시스템 아키텍쳐와 프로그램 동작간의 피드백이 없다는 문제점이 있다. 이같은 사실은 수행 구동 시뮬레이션과 비교할 때 성능 예측의 심한 불일치를 야기할 수 있는데, 이는 메모리 접근 시각의 변화에 의하여 작업부하가 상이하게 작용할 수도 있기 때문이다.



(그림 1) 공유메모리를 갖는 다중프로세서 시스템용 분포 구동 시뮬레이션

일단 작업부하 모델링이 완료되면 시뮬레이터 구현을 위한 프로그래밍 언어를 결정해야 하는데, 이에 대해서 간단히 살펴보면 다음과 같다. 일반적으로 시뮬레이터 구현이 가능한 언어는 시뮬레이션 언어, 범용 언어, 시뮬레이션을 고려하여 확장된 범용 언어 등의 3가지 부류로 구분될 수 있다. 60년대 이전에는 범용 언어, 특히 FORTRAN을 이용하여 시뮬레이션 하였으나, 시뮬레이션 모델을 수행하는데 필요한 모든 프로그램을 사용자가 직접 개발해야 한다는 부담이 있었다. 따라서 60년대 초에 들어서면서 GPSS[5], SIMSCRIPT[6], SIMULA[7] 등 최초의 시뮬레이션 언어가 발표되었는데, GPSS는 큐잉 시스템을 모델링하는 전용 시뮬레이션 언어의 예이며, SIMSCRIPT는 후에 발표된 GASP[8], SIMAN[9], SLAM[10] 등과 함께 좀 더 광범위한 범용 이산사건 시뮬레이션 언어의 부류에 속한다.

그러나 이러한 시뮬레이션 언어들이 개발되고 있는 것과는 별도로 아직도 많은 시뮬레이션 모델이 범용 언어를 이용하여 코딩되고 있는데, 이는 시뮬레이션 언어가 특정 컴퓨터에 동작하지 않을 수 있다는 제약과 새로운 언어 습득에 대한 부담 등이 그 이유이다. 따라서 이러한 어려움을 극복하기 위하여 시뮬레이션 언어를 개발하는 또 다른 방법으로 기존의 프로그래밍 언어에 시뮬레이션 프리미티브를 추가하는 방법이 있다. 즉, 사용자는 범용 언어를 이용하여 모델의 기본적인 부분(예를 들어, 사건, 활동, 프로세스)을 작성하고, 추가적으로 제공되는 시뮬레이션용 프리미티브는 단순히 이용하기만 하면 된다. 이러한 부류에 속하는 언어로 범용 언어 C를 확장한 CSIM[11]과 SMPL[12] 등이 있다.

마지막으로 시뮬레이션 연구의 새로운 추세로 객체 지향 개념을 시뮬레이션과 접목하는 것이 있는데, 객체 지향 프로그래밍에서는 모델을 절차적인 시뮬레이션 언어가 제공하는 것보다 한 차원 높은 수준의 추상화로 표현 가능하게 하는 장점이 있다. 이러한 시뮬레이션 도구로 SES[13]가 있는데, 특히 SES는 GUI에 의한 신속하고 계층적인 모델 생성을 가능케 하며 모델 수행의 애니메이션도 지원한다.

이상 언급한 시뮬레이션 언어를 사건 중심 및 프로세스 중심의 모델 분류에 의해서 정리하면 〈표 1〉과 같다.

〈표 1〉 시뮬레이션 언어 비교

사건 중심	프로세스 중심
SIMSCRIPT, GASP, SIMAN, SLAM, SMPL	GPSS, SIMSCRIPT, SIMULA, SIMAN, SLAM, CSIM

III. 추적 구동 시뮬레이션

추적 구동(trace-driven) 시뮬레이션은 아키텍처의 시뮬레이션 모델을 구동하기 위하여 트레이스(프로그램의 수행 기록)를 이용한다. 먼저 시뮬레이트 할 컴퓨터와 유사한 실제 컴퓨터에서 프로그램을 수행하여 트레이스를 얻는다. 즉, 실제 프로그램이 수행되면서 수행의 특징적인 면(예를 들어, 수행된 인스트럭션이나 접근된 메모리 위치의 시간별 리스트)을 기록함으로써 트레이스가 만들어진다. 트레이스가 이렇게 얻어지면, 이 트레이스는 시뮬레이트 되는 아키텍처에 대한 작업부하로 사용된다.

그러나, 분포 구동 시뮬레이션 방법에서와 같이 추적 구동 시뮬레이션에서도 메모리 시스템 아키텍처와 프로그램 동작간의 피드백이 없어 정확도가 떨어질 수 있다는 문제점을 갖고 있다. 이 외에도 시뮬레이션을 구동하기 위해 인스트럭션 트레이스를 이용할 때 어려운 점은 시뮬레이트 되는 인스트럭션 세트가 그 트레이스를 얻은 프로세서의 인스트럭션 세트와 논리적으로 반드시 일치해야 한다는 점이다. 또한, 하나의 트레이스 파일이 일반적으로 수백만 내지 수억 개의 트레이스를 포함하므로, 추적 구동 시뮬레이션을 하기 위해서는 트레이스 파일을 저장하기 위한 대용량의 디스크가 필요하고 시뮬레이션 시간 또한 길다. 더욱이, 순차적인 컴퓨터 시스템에서 추출된 트레이스로 병렬 컴퓨터 시스템을 시뮬레이트 하는 데에는 많은 어려움이 따르는데, 그 이유는 병렬 컴퓨터 시스템에서의 인스트럭션 수행 순서가 대개 병렬 프로그램의 여러 부분이 수행을 완료하는 순서에 의존적이라는 점이다. 또한, 병렬 컴퓨터 시스템에서 조건부 분기 인스트럭션의 결과는 여러 프로세스 상호작용의 시점에 따라 달라질 수 있으며, 이는 결국 시스템 아키텍처의 특성에 영향을 받을 수 있다[14]. 이러한 병렬 컴퓨터 시스템용 어드레스 트레이스 수집에 대해서 간단히 살펴보면 다음과 같다.

병렬 컴퓨터 시스템에서는 프로세서간 효과적인 메모리 접근과 데이터 공유와 관련하여 많은 기술적인 이슈가 있으며, 추적 구동 시뮬레이션에서 정확도를 보장하기 위하여 트레이스는 사용자와 커널의 메모리 접근 모두를 포함해야 하고 (기록할 때의) 트레이싱 메커니즘이나 (활용할 때의) 시뮬레이터에 의한 시간 왜곡(time distortion)이 최소로 되어야 한다[15, 16].

이러한 트레이싱 기법은 하드웨어나 소프트웨어적으로 해결될 수 있는데, 그 중 하드웨어 모니터링 방법[17, 18]은 매우 높은 정확도를 제공할 수 있다. 예를 들어, [17]에서는 동일한 시스템을 이용하여 트레이스되는 시스템에 의해 생성된 메모리 접근을 트레이스 메모리가 초과되기 전까지 트레이스하는 시스템의 메모리에 저장한다. 이 때 트

레이스하는 시스템은 그 트레이스의 저장을 시작하는데, 수집된 트레이스의 정확한 재생을 위하여 동기점(synchronization point)에서 시간 스탬프를 끼워 넣는다. 그러나 이러한 하드웨어적인 방법은 시간 왜곡이 없고 정확하지만, 구현 비용이 비싸고 저장공간의 제약을 받는다는 문제가 있다. 또한 최근의 프로세서는 상당량의 캐쉬를 내장하고 있는데, 이러한 프로세서 내부의 메모리 접근에 대해서는 하드웨어 트레이싱 방법으로 포착이 불가능하다는 단점이 있다. 마지막으로 하드웨어적인 방법으로 다중프로세서 시스템에서 입수된 트레이스로는 여러 가지 경우의 성능 연구가 어려운데, 이는 다양한 프로세서 수에 대한 각각의 트레이스를 생성해내는 것이 현실적으로 불가능하기 때문이다.

소프트웨어 트레이싱 기법으로는 프로그램 조작(program instrumentation), 단일 스텝 수행(single-step execution), 마이크로코드 수정(microcode modification) 등이 있는데, 트레이싱 메커니즘의 매우 높은 오버헤드에 의하여 시간 팽창(time dilation)이 일어난다는 문제를 가지고 있다. 즉, 이러한 오버헤드는 비동기적인 사건들의 상대적인 시간을 심각히 변화시켜, 하드웨어적인 방법에 비하여 정확도가 떨어진다. 세 가지 소프트웨어 트레이싱 기법을 자세히 살펴보면 다음과 같다.

먼저, 프로그램 조작(program instrumentation) 기법은 런타임시에 각각의 기본 블록(분기를 포함하지 않는 머신 인스트럭션의 순서)에 대한 상대적인 트레이스의 부분을 생성하기 위하여 몇 개의 인스트럭션을 추가한다. 조작 단계는 소스나 엑제큐터블 수준에서 활성화되는데, 후자는 사용자가 취급하기에는 간단하지만 구현하기 힘들고 모든 프로그램의 조작이 가능하지 않을 수도 있다. 반면, 전자인 어셈블리 수준에서의 조작은 쉽지만 사용자가 전체 소스 코드를 접근해야만 한다. 또한, 커널 루틴에 대한 접근을 포착하기가 어렵기 때문에, 이러한 트레이스의 불완전함이 모델의 정확도를 떨어뜨리는 요인으로 작용한다. 이러한 프로그램 조작 기법의 예로 Mptrace[19], Trapeds[20, 21], Tangolite[22] 등이 있는데, Trapeds와 Tangolite는 다음 장에서 설명할 수행 구동 시뮬레이션도 지원한다.

단일 스텝 수행(single-step execution) 기법은 각 인스트럭션 후에 인터럽트에 의해 프로그램의 수행이 해석될 수 있는 프로세서에 한해서 적용된다. 그러나 대부분 커널 루틴은 인터럽트가 불가능하기 때문에, 이 기법은 대개 커널 루틴의 수행에서 생성된 메모리 접근을 포착하지 못한다는 문제가 있다. 예를 들어, [23]에서의 트레이싱 메커니즘은 각각의 인스트럭션에서 정지하고 트레이스 정보를 저장하기 위하여 트레이스 트랩 기능을 이용한다.

마지막으로 마이크로코드 수정(microcode modification)기법의 전형적인 예인 ATUM (Address Tracing Using Microcode)[24, 25]은 통상적인 수행의 부수적인 효과로서 메모리 접근을 메인 메모리의 특정 장소에 기록하기 위하여 프로세서의 마이크로 코드를 이용한다. 다른 기법과 비교할 때, 적은 시간 왜곡과 매우 빠른 기록을 갖는 장점이 있다. 또한 추가 하드웨어 없이 모든 시스템 활동이 관찰될 수 있다는 특징이 있다. 그러나, 유연성이 부족하고 트레이스 길이가 트레

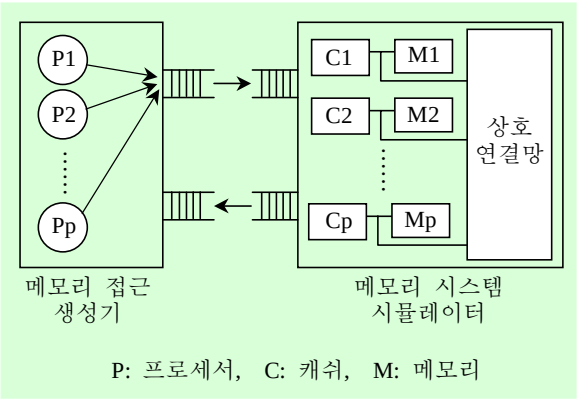
이스 저장장소로 지정된 메모리의 크기로 제한된다는 문제가 있다. 이상 언급한 트레이싱 수집 기법을 요약하면 〈표 2〉와 같다.

〈표 2〉 트레이싱 기법의 비교[20]

	하드웨어 모니터링	프로그램 조작	단일 스텝 수행	마이크로 코드 수정
정확도	높음	보통	보통	보통
속도	1×	10×	100×	10×
적용성	낮음	높음	보통	낮음
트레이스 완전성	낮음	높음	높음	높음
커널/멀티 프로그램	포함	가능	불포함	포함

IV. 수행 구동 시뮬레이션

수행 구동(execution-driven) 시뮬레이션의 가장 큰 특징은 시뮬레이트 되는 작업부하가 프로그램의 수행에 의하여 동적으로 생성된다는 점이다. 따라서, 수행 구동 시뮬레이션에서는 프로그램의 수행과 아키텍쳐 시뮬레이션 모델의 수행이 인터리빙 된다고 말할 수 있다. 즉, 먼저 하나의 프로세스가 선택되어 프로세스간 상호작용(interaction)이 발생할 때까지 수행된 후, 두 번째로 상호작용에 의한 프로세서 모듈간 데이터 이동이 시뮬레이트 되고 시뮬레이션 시간이 진행된다. 이 두 가지 단계는 여러 프로세스를 처리하면서 모든 프로세스 혹은 시뮬레이션이 종료할 때까지 반복된다. 이 방법의 가장 큰 장점은 프로세서 내부의 활동을 시뮬레이트 하기 위한 오버헤드는 거의 없이 단지 프로세서간 상호작용만이 자세하게 시뮬레이트 된다는 점이다. 즉, 하나의 프로세서 내부에 국한되는 프로세스 활동은 단순히 그 부분을 호스트 컴퓨터에서 수행시킴으로써 처리되고, 사건 큐 유지나 데이터 이동을 시뮬레이트 하기 위한 오버헤드는 다른 프로세서상의 프로세스간 상호작용이 필요한 경우에만 부과된다.

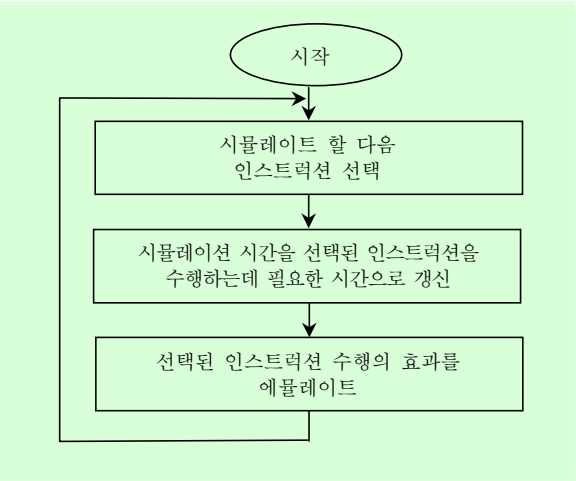


(그림 2) 공유메모리를 갖는 다중프로세서 시스템용 수행 구동 시뮬레이션[4]

(그림 2)에 수행 구동 시뮬레이션을 나타내었는데, (그림 1)에 나타낸 분포 구동이나 추적 구동과 달리 메모리 접근기와 메모리 시스템 시뮬레이터간의 정보의 흐름이 양방향이다. 즉, 메모리 시스템 시뮬레이터가 메모리 접근 생성기로부터 명령을 받으면 확장된 메모리 계층을 통하여 메모리 접근 패스를 시뮬레이트 한 후, 그 메모리 접근이 만족되는 시간을 메모리 접근 생성기에게 돌려준다. 이 시간 정보는 메모리 접근 생성기에 의하여 다음 스케쥴의 시뮬레이트 될 프로세스를 결정하는데 사용된다. 따라서 메모리 시스템 시뮬레이터에서 메모리 접근 생성기로 피드백이 존재하며, 이로 인해서 추적 구동에 비하여 보다 정확한 시뮬레이션을 할 수 있다.

수행 구동 시뮬레이션과 같이, 아키텍쳐 시뮬

레이션을 구동하기 위하여 프로그램을 사용하는 것은 아주 일반적인 접근 방법이다. 그러나, 각각의 목표 프로세서 인스트럭션의 수행은 대개 시뮬레이션 호스트상에서 여러 개의 인스트럭션을 수행함으로써 시뮬레이트 된다. 이러한 방법을 대개 프로그램 구동(program-driven) 또는 인스트럭션 구동(instruction-driven) 시뮬레이션이라 부르는데, 병렬 프로그램의 시뮬레이션을 수행하는데 필요한 시간은 프로그램을 직접 수행하는 시간에 비하여 300배까지 느리다는 단점을 가지고 있다.

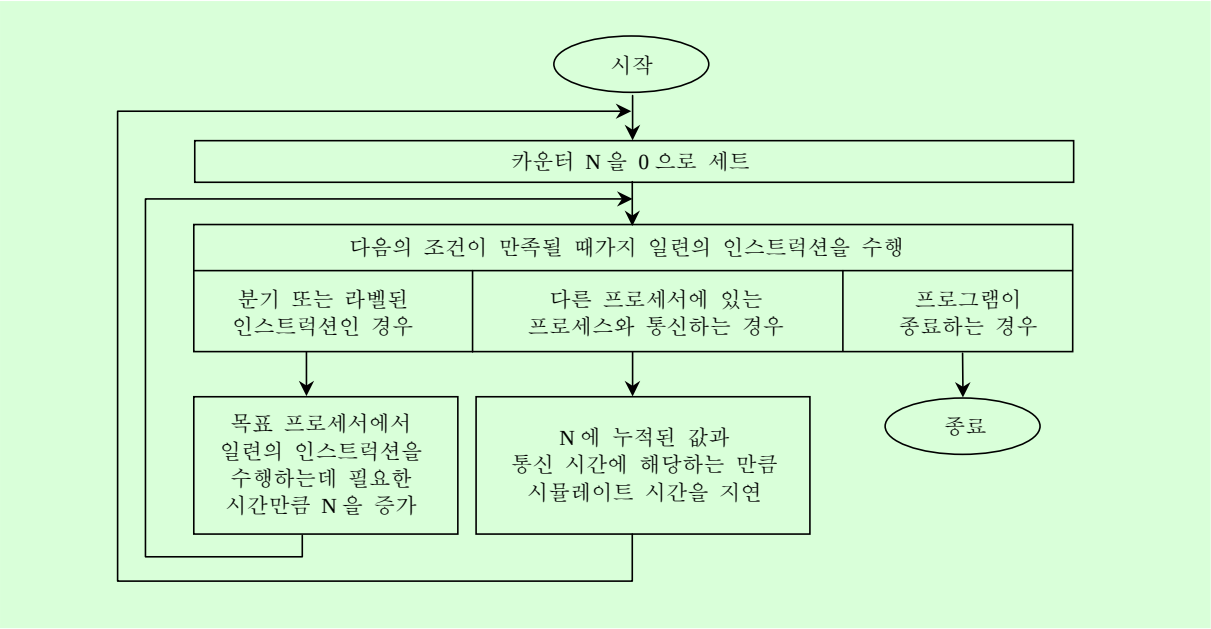


(그림 3) 프로그램 구동 시뮬레이션의 수행 싸이클 [26, 27]

수행 구동과 프로그램 구동 시뮬레이션의 차이를 나타내기 위하여, (그림 3)에 프로그램 구동 시뮬레이션의 기본 싸이클을 나타내었다. 이 싸이클은 하나의 프로세서상에서의 활동을 시뮬레이트 하며, 시뮬레이트 되는 시스템에서 각 프로세서는 그림과 같은 각각의 독립적인 싸이클에 의해서 시뮬레이트 된다. 이러한 방법의 시뮬레이션에서 가장 큰 오버헤드를 차지하는 부분은 인스트럭션의 동작을 에뮬레이트 하는 부분(그림에서 세 번째 블록)인데, 이는 목표 프로세서에서 단지 하나의 인스트럭션 수행을 시뮬레이트 하기 위하여 호스트 컴퓨터상에서 여러 인스트럭션이 수행되어야 하기 때문이다. 인스트럭션의 수행 시간과 관련된 부분은 그림에서 두 번째 블록으로, 시뮬레이트 되는 각각의 인스트럭션에 대해서 한 번만 수행된다. 이 동작 역시 사건 큐 조정을 위하여 여러 개의 인스트럭션 수행을 필요로 한다.

(그림 4)에 병렬 컴퓨터 시스템 중 하나의 프로세서에 대한 수행 구동 시뮬레이션의 기본적인 싸이클을 나타내었다. 이 싸이클은 병렬 컴퓨터 시스템의 프로그램 구동 시뮬레이션에 대응하는 싸이클에 비하여 상당히 빠른데, 이는 시뮬레이트 되는 프로세서의 동작을 시뮬레이트 하는 대신에 수행 구동 시뮬레이션에서 직접 그 인스트럭션을 수행하며 수행하는 사건 큐 동작의 수도 월등히 적기 때문이다. 즉, 라벨되지 않은 인스트럭션이나 분기 인스트럭션이 아니거나 다른 프로세서와 상호작용을 필요로 하지 않는 인스트럭션에 대해서는 오버헤드가 없다. 분기나 라벨된 인스트럭션은 카운터 N을 증가시키기 위하여 한 인스트럭션 만큼의 오버헤드가 발생한다. 시뮬레이트 되는 프로그램이 다른 프로세스와의 상호작용이 필요한 경우에만 상당한 양의 오버헤드가 발생한다. 이 때 사건 큐 삽입이 처리되며 상호연결망 구조를 정의하는 사용자 정의 프로그램이 수행된다.

현재 공유메모리 다중프로세서 시스템에 대한 연구에 널리 이용되는 메모리 접근 생성기로는, 수행 구동 시뮬레이션에서 MIPS 머신용의 Proteus[28], Tango[29], Tangolite[22], MINT[30]와 In-



(그림 4) 수행 구동 시뮬레이션의 수행 싸이클[26, 27]

tel 머신용의 Augmint[31] 등이 있으며, 프로그램 구동 시뮬레이션에서 Sparc 머신용의 Cache-Mire[32] 등이 있다. 특히, 이러한 시뮬레이션이 시뮬레이트 되는 프로세서의 수에 비례하여 매우 많은 계산 양을 필요로 한다는 점에서, 메모리 접근 생성기 자체가 병렬 컴퓨터 시스템에서 병렬로 수행되는 방법도 제안되었다[33]. 또한, 이러한 메모리 접근 생성기를 이용하여 공유메모리 다중프로세서 시스템에 대한 기존의 연구는 대부분 Splash[34] 등 과학계산용 응용프로그램을 벤치마크로 사용하였다. 그러나 이것은 현실과 거리가 있는데, 대부분의 고성능 시스템이 데이터베이스 엔진이나 웹 서버와 같이 상용 응용에 적용되기 때문이다. 상용 응용이 기존의 과학계산 응용과 다른 점은 운영체제가 수행시간의 많은 부분을 차지하고 입출력 또한 무시할 수 없다는 점이다. 최근 이러한 상용 작업부하를 이용할 수 있도록 해주는 몇몇의 소프트웨어[35, 36]가 개발되어, 향후 이러한 상용 작업부하를 이용한 시뮬레이션이 주류를 이룰 것으로 기대된다[37].

마지막으로 본 고에서 언급한 세 가지 시뮬레이션 구동 방법을 간단히 요약하면 〈표 3〉과 같다.

V. 맺음말

지금까지 설계서로 존재하는 병렬 컴퓨터 시스템의 성능을 평가하기 위한 컴퓨터 시뮬레이션 과정 중 핵심이 되는 여러 가지 작업부하 생성 방법에 대한 정의 및 장단점 등에 대해 알아보았다. 또한 시뮬레이터에 부과되는 응용 프로그램의 작업부하 생성 방법에 대해 각각의 대표적인 구현 사례 및 기술적인 이슈에 대해서도 알아보았다.

〈표 3〉 시뮬레이션 구동 방법의 비교

	복잡도	유연성	저장공간		속도	정확도
			작업부하	시뮬레이터		
분포 구동	낮음	높음	없음	적음	빠름	낮음
추적 구동	보통	없음	많음	보통	보통	보통
수행 구동	높음	높음	보통	많음	느림	높음

응용 프로그램의 작업부하 생성 방법은 각각의 방법이 지원하는 정확도, 속도, 복잡도 등 각기 다른 특징을 지니고 있기 때문에, 어떤 시뮬레이션 환경에 적합한 방법이 다른 환경에서는 적합치 않을 수가 있다. 따라서, 주어진 시뮬레이션 환경에 가장 적합한 방법으로 작업부하를 모델링하여 최적의 성능을 제공하는 병렬 컴퓨터 시스템을 설계해야 할 것이다.

참 고 문 헌

[1] L. Kleinrock, *Queueing Systems*, Vol. I, II, Prentice-Hall, Inc., 1975.

[2] P. Heidelberger and S. Lavenberg, "Computer Performance Evaluation Methodology," *IEEE Tr. on Computers*, Vol. 33, No. 12, 1984, pp. 1195–1220.

[3] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc., 1991.

[4] D. Culler, J. Singh, and A. Gupta, *Parallel Computer Architecture–A Hardware/Software Approach*, Morgan Kaufmann, Pub., 1998.

[5] J. Schriber, *Simulation Using GPSS*, John Wiley & Sons, Inc., 1974.

[6] P. Kiviat, R. Villanueva, and H. Markowitz, *SIMSCRIPT II.5 Programming Language*, CACI, 1973.

[7] O. Dahl and K. Nygaard, "SIMULA: An ALGOL Based Simulation Language," *Communication of ACM*, Vol. 9, 1966, pp. 671–678.

[8] A. Pritsker, *The GASP IV Simulation Language*, John Wiley & Sons, Inc., 1974.

[9] C. Pegden, "Introduction to SIMAN," *Proc. of Winter Simulation Conference*, 1986, pp. 95–103.

[10] A. Pritsker, *Introduction to Simulation and SLAM II*, 2nd Ed., John Wiley & Sons, Inc., 1984.

[11] H. Schwetman, "Using CSIM to Model Complex Systems," *Proc. of Winter Simulation Conference*, 1988, pp. 246–253.

[12] M. MacDougall, *Simulating Computer Systems–Techniques and Tools*, MIT Press, 1987.

[13] *SES/Workbench Reference Manual*, Scientific and Engineering Software, Inc., 1992.

[14] M. Holliday and C. Ellis, "Accuracy of Memory Reference Traces of Parallel Computations in Trace-Driven Simulation," *IEEE Tr. on Parallel and Distributed Systems*, Vol. 3, No. 1, 1992, pp. 97–109.

[15] R. Giorgi *et al.*, "Trace Factory: Generating Workloads for Trace-Driven Simulation of Shared Bus Multiprocessors," *IEEE Concurrency*, Vol. 5, No. 4, 1997, pp. 54–68.

[16] R. Uhlig and T. Mudge, "Trace-Driven Memory Simulation: A Survey," *ACM Computing Surveys*, Vol. 29, No. 2, 1997, pp. 128–170.

[17] B. Vashaw, *Address Tracing Collection and Trace-Driven Simulation of Bus-Based, Shared-Memory Multiprocessors*, Technical Report, CMU, 1993.

[18] J. Flanagan *et al.*, "BACH: BYU Address Collection Hardware, the Collection of Complete Traces," *Proc. of International Conference on Modeling Techniques and Tools for Computer Performance Evaluation*, 1992, pp. 128–137.

[19] S. Eggers *et al.*, "Techniques for Efficient Inline Tracing

on a Shared-Memory Multiprocessor," *Proc. of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1989, pp. 70−78.

[20] C. Stunkel, B. Janssens, and W. Fuchs, "Address Tracing for Parallel Machines," *IEEE Computer*, Vol. 24, No. 1, 1991, pp. 31−38.

[21] C. Stunkel and W. Fuchs, "TRAPEDS: Producing Traces for Multicomputers via Execution-Driven Simulation," *Proc. of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1989, pp. 70−78.

[22] S. Herrod, *Tangolite: A Multiprocessor Simulation Environment,* Technical Report, Stanford University, 1993.

[23] S. Eggers and R. Katz, "A Characterization of Sharing in Parallel Programs and Its Application to Coherency Protocol Evaluation," *Proc. of International Symposium on Computer Architecture*, 1988, pp. 373−382.

[24] A. Agarwal, R. Sites, and M. Horowitz, "ATUM: A New Technique for Capturing Address Traces Using Microcode," *Proc. of International Symposium on Computer Architecture*, 1986, pp. 119−127.

[25] R. Sites and A. Agarwal, "Multiprocessor Cache Analysis using ATUM," *Proc. of International Symposium on Computer Architecture*, 1990, pp. 37−46.

[26] R. Covigton *et al.*, "The Rice Parallel Processing Testbed," *Proc. of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1988, pp. 4−11.

[27] S. Dwarkadas, J. Jump, and J. Sinclair, "Execution-Driven Simulation of Multiprocessors: Address and Timing Analysis," *ACM Tr. on Modeling and Computer Simulation*, Vol. 4, No. 4, 1994, pp. 314−338.

[28] E. Brewer *et al.*, "Proteus: A High-Performance Parallel Architecture Simulator," *Performance Evaluation Review*, Vol. 20, No. 1, 1992, pp. 247−248.

[29] H. Davis, S. Goldschmidt, and J. Hennessy, *Tango: A Multiprocessor Simulation and Tracing System,* Technical Report, Stanford University, 1990.

[30] J. Veenstra, "MINT: A Frontend for Efficient Simulation of Shared-Memory Multiprocessors," *Proc. of International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 1994, pp. 201−207.

[31] A. Nguyen *et al.*, "The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures," *Proc. of International Conference on Computer Design*, 1996.

[32] M. Brorsson *et al.*, "The CacheMire Testbench: A Flexible and Effective Approach for Simulation of Multiprocessors," *Proc. of Simulation Symposium*, 1993, pp. 41−49.

[33] S. Reinhardt *et al.*, "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers," *Proc. of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1993, pp. 48−60.

[34] J. Pal Singh, W. Weber, and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory," *Computer Architecture News*, Vol. 20, No. 1, 1992.

[35] M. Rosenblum *et al.*, "Complete Computer System Simulation: The SimOS Approach," *IEEE Parallel and Distributed Technology*, Vol. 3, No. 4, 1995, pp. 34−43.

[36] P. Magnusson and B. Werner, "Efficient Memory Simulation in SimICS," *Proc. of Simulation Symposium*, 1995, pp. 62−73.

[37] First Workshop on Computer Architecture Evaluation Using Commercial Workloads, *IEEE Computer Society*, 1998.