공유메모리 다중프로세서 시스템의 다중 프로그래밍 모의실험 기법

(Multi-Programmed Simulation of a Shared Memory Multiprocessor System)

최 효 진 † 전 주 식 ††
(Hyo Jin Choi) (Chu Shik Jhon)

요 약 공유메모리 다중프로세서 시스템의 성능은 하드웨어 구조 뿐 아니라 운영체제의 프로세서 스케줄링 정책 등과 같은 소프트웨어에 의해 큰 영향을 받는다. 하지만, 현재 많이 사용되는 대부분의 모의실험 기들은 하나의 벤치마크 응용프로그램의 수행만을 지원하기 때문에 다중 프로그래밍 환경에 대한 모의실험이 불가능하다. 본 논문은 복수개의 응용프로그램들이 프로세서와 기타 시스템 자원을 공유하며 경쟁하는 다중 프로그래밍 환경에 대한 모의실험을 프로그램 구동형 모의실험 환경 하에서 구현하는 기법을 제안한다. 제안하는 기법은 실제 수행환경에 근접한 모의실험을 가능하게 하며, 이를 통해 제한된 시스템 자원에 대한 공유와 충돌의 영향을 자세하게 분석할 수 있다. 또한, 스케줄링 정책의 구현과 분석을 가능하게 함으로써 시스템 구조에 맞는 최적의 정책을 수립할 수 있도록 한다.

키워드 : 공유메모리 다중프로세서 시스템, 스케줄링 정책, 프로그램 구동형 모의실험기, 다중 프로그래밍 모의실험

Abstract The performance of a shared memory multiprocessor system is dependent on the system software such as scheduling policy as well as hardware system. Most of existing simulators, however, do not support simulation for multi-programmed environment because they can execute only a single benchmark application at a time. We propose a multi-programmed simulation method on a program-driven simulator, which enables the concurrent executions of multiple parallel workloads contending for limited system resources. Using the proposed method, system developers can measure and analyze detailed effects of resource conflicts among the concurrent applications as well as the effects of scheduling policies on a program-driven simulator. As a result, the proposed multi-programmed simulation provides more accurate and realistic performance projection to design a multiprocessor system.

Key words : Shared memory multiprocessor system, scheduling policy, program-driven simulator, multi-programmed simulation

1. 서 론

다중 프로세서 시스템의 개발에 있어 모의실험은 하드웨어의 구현 없이 저비용으로 시스템의 성능을 예측하기 위해 널리 사용된다. 비록 이러한 모의실험의 결과가 실제로 구현된 시스템의 성능과 일치한다고는 할 수 없지만, 모의실험을 통해 시스템의 성능에 영향을 끼치는 여러 요인들을 찾아내고 특히 성능향상에 부정적인 영향을 미치는 잠재적인 병목요인들을 사전에 확인, 개선할 수 있다는 점에서 그 가치가 있다. 다중 프로세서 시스템의 성능평가를 위해서, 그 동안 개발된 여러 종류의 모의실험도구 중에서 프로그램 구동형 모의실험기(program-driven simulator)가 널리 사용되어 왔다. 프로그램 구동형 모의실험기는 다양한 구조의 시스템을 모델링 가능하게 해 줄뿐 아니라, 그 사용 간편성에 비해 상대적으로 실제 시스템에 근접한 성능평가 결과를 유도하도록 도와준다 [1, 2, 3, 4, 5]. 그러나 프로그램

† 비 회 원 : (주)지씨티리써치 책임연구원
bradchoi@gctsemi.com
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
csjhon@riact.snu.ac.kr
논문접수 : 2002년 3월 2일
심사완료 : 2002년 12월 3일

구동형 모의실험기는 한번에 단 하나의 응용프로그램만 실행할 수 있다는 한계 때문에 다중 프로그래밍 환경에 대한 성능평가에는 사용될 수 없다는 단점을 가지고 있다. 또한, 프로그램 구동형 모의실험기는 응용프로그램의 프로세스 개수와 시스템 프로세서 개수가 같다고 가정한다. 이러한 특성들 때문에 사용자는 여러 응용프로그램이 제한된 시스템 자원을 두고 경쟁하는 다중 프로그래밍 환경 하에서의 시스템 성능평가를 할 수 없다.

공유메모리 다중프로세서 시스템의 성능은 프로세서 개수, 캐시 구조와 그에 적용되는 캐시 일관성 규약, 상호연결망의 특성 등 하드웨어 요인들에 의해 크게 영향을 받는다. 또한 현재 대부분의 시스템에 적용되는 다중 프로그래밍 환경에서는, 그러한 하드웨어 요인들에 추가하여 프로세서 자원을 여러 응용프로그램 사이에서 효율적으로 배분하는 운영체제의 스케줄링 정책 또한 시스템의 성능에 큰 영향을 미친다 [6]. 물리적인 프로세서의 개수는 동시에 실행되는 복수개의 응용프로그램이 요구하는 프로세서 만큼을 제공할 수 없는 경우가 일반적이기 때문에 스케줄링 정책은 제한된 프로세서 자원과 그 자원을 할당 받기를 기다리는 여러 프로그램들 사이에서 중재 역할을 수행한다. 프로세서 활용도, 응용프로그램의 병렬성, 캐시 적중률 등과 같은 시스템 성능의 척도들은 운영체제의 스케줄링 정책에 직접적인 영향을 받기 때문에 스케줄링 정책이 시스템의 성능에 중대한 영향을 미치는 것이다 [5]. 따라서 다중프로세서 시스템에 대한 모의실험을 하고자 할 때, 여러 응용프로그램이 제한된 시스템 자원들에 대해 경쟁하며 스케줄링되는 환경에 대한 고려가 없이는 실제 시스템과 비슷한 실용적인 결과를 기대할 수 없게 된다.

현재 사용되는 프로그램 구동형 모의실험기는 스케줄링 정책을 모델링 하는데 필수적인 다중 프로그래밍 환경을 지원하지 않고 있다. SimOS 들과 같이 에뮬레이션에 기반한 도구들은 실제 시스템에 근접한 성능평가를 지원하지만 [7], 상당한 큰 연산기능을 요구한다는 단점과 함께 모의실험에 있어서 스케줄링에 관계된 요인들만을 완전히 구분해 낼 수 없다는 한계가 있다. 만약 프로그램 구동형 모의실험기가 복수개의 응용 프로그램을 동시에 수행할 수 있고, 그와 동시에 스케줄링 정책에 대한 모델링을 가능하게 하는 기능을 지원할 수 있다면, 사용자는 기존의 한 응용 프로그램을 기준으로 성능평가를 수행하는 프로그램 구동형 모의실험기와는 달리, 여러 프로그램 간의 프로세서 공유, 기타 시스템 자원에 대한 충돌 등의 영향을 측정함으로써 실제 수행환경에 근접한 성능평가 결과를 얻을 수 있을 것이다.

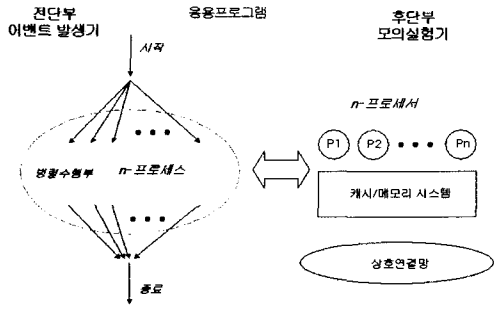
본 논문은 기존의 프로그램 구동형 모의실험기 환경을 기반으로, 여러 응용프로그램의 동시 수행과 스케줄링 정책에 대한 모델링을 지원함으로써 하드웨어 시스템과 스케줄링 정책이 통합된 형태의 다중 프로그래밍 모의실험 방법을 소개한다. 벤치 마크 프로그램 들을 분기-합류(fork-join) 모델에 기반하여 통합을 하고 간단한 스케줄러용 명령어들을 추가함으로써 전단부(front-end)는 다중 프로그램 환경과 같은 이벤트를 생성하게 되고, 후단부(back-end)는 응용 프로그램 간, 프로세스 간의 이벤트를 구분할 수 있게 됨으로써 스케줄링 정책에 대한 모델링이 가능하게 된다. 또한 본 논문은 전단부와 후단부 사이에 프로세서 할당 정책에 관계된 자료구조들을 제공하고 이를 기반으로 스케줄링 정책을 구현하는 구체적인 방법을 설명하고자 한다. 다중 프로그래밍 모의실험에 대한 자세한 구조와 구현방안은 2장에서 설명한다. 3장에서 이를 활용한 모의실험 사례를 제시하며 4장에서 결론을 내린다.

2. 다중 프로그래밍 모의실험

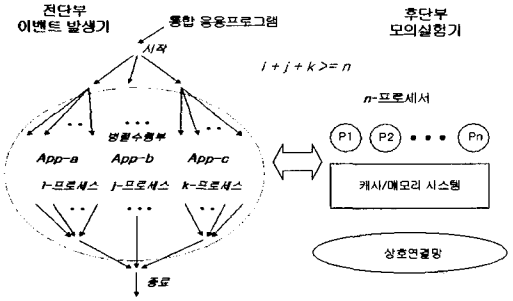
2.1 다중 프로그래밍 모의실험의 구조

기존의 프로그램 구동형 모의실험기는 전단부와 후단부의 두 부분으로 구성된다. 전단부는 수행하고자 하는 실행화일을 해석하거나 바로 수행하는 과정을 통해서 메모리에 대한 참조 이벤트를 발생시킨다. 사용자는 이때 생성되는 메모리 참조 이벤트를 처리하는 부분을 구현함으로써 모의실험 하고자 하는 시스템을 모델링한다. 메모리 참조 이벤트는 후단부에 미리 정의되어 있는 함수들을 호출함으로써 후단부로 전달되며, 이때 이벤트 처리에 필요한 여러 정보들을 함수의 인자를 통해 전달받는다. 전달되는 인자는 프로세스 식별자와 접근하고자 하는 메모리의 주소, 현재 모의실험 시간 등을 포함한다. 후단부로 전해진 이벤트는 사용자가 정의한 시스템의 구조에 따라 메모리시스템, 상호연결망 등을 따라 전달됨으로써 실험이 진행된다. 해당 메모리 참조 이벤트가 종료되었을 때, 즉 후단부에 모델링된 시스템을 따라 이벤트에 대한 처리가 완료되었을 때 후단부는 전단부로 수행결과를 전달하고, 전단부는 이 결과에 따라 다음 명령들을 처리하며 새로운 이벤트를 발생시킨다[1, 2, 3, 4, 5].

프로그램 구동형 모의실험기는 다른 종류의 모의실험 기법에 비해 몇 가지 중요한 장점을 갖는다. 프로그램 구동형 모의실험기는 실행프로그램을 직접 해석, 수행하고 또한 실행프로그램의 소스코드에 이벤트 생성을 위한 변경이 필요 없기 때문에 실제 상황에 근접한 실험



(가) 기존 프로그램 구동형 모의실험기



(나) 다중 프로그래밍 프로그램 구동형 모의실험기

그림 1. 프로그램 구동형 모의실험기의 구조

결과를 얻을 수 있다. 또한 모의실험기가 실행되는 환경의 운영체제 등에 대한 어떠한 변경도 필요없기 때문에 사용하기 편리하다. 프로그램 구동형 모의실험기에서 사용되는 모델링 기법 및 인터페이스가 널리 알려져 있다는 점도 실험의 편의성을 증대시키는 요인이다.

이러한 여러 장점에도 불구하고, 프로그램 구동형 모의실험기는 운영체제의 스케줄링 정책과 같은 시스템 소프트웨어에 대한 모델링은 지원하지 않는다는 단점이 있다. 단 하나의 응용프로그램을 수행하면서 얻는 모의실험 결과는 실제 다중 프로세서 시스템의 수행 하에서 발생할 수 있는 여러 프로그램들 간의 경쟁 상황이 반영되지 않기 때문에 정확한 결과라고 볼 수 없다. 그림 1 (가) 는 기존의 프로그램 구동형 모의실험기에서 응용프로그램과 모의실험기에 구현된 시스템 프로세서와의 관계를 보여준다. 응용프로그램이 수행 중에 병렬적인 처리 단계로 진입하면 시스템 상에 정의된 프로세서의 개수만큼 자식 프로세스를 생성하며, 각각의 자식 프로세스는 전용으로 프로세서를 할당받아 실행되게 된다. 이러한 경우, 시스템의 자원을 두고 프로세스 간, 혹은 응용프로그램 간에 경쟁하는 환경을 모델링할 수 없기 때문에, 시스템 자원에 대한 접근 충돌의 영향과 프로세스와 시스템 자원 사이의 분배전략 등에 대한 성능평가가 불가능해진다. 이는 바로 정확한 모의실험을 제한하는 결과를 가져온다.

본 논문에서는, 프로그램 구동형 모의실험기에서 하드웨어 시스템과 함께 스케줄링 정책을 동시에 모의실험하기 위해 기존의 프로그램 구동형 모의실험 환경에 추가적으로 필요한 두가지 기능을 제안한다. 우선 첫째로, 스케줄링 정책을 모델링 하기 위해서는 여러 개의 벤치마크 응용프로그램을 동시에 수행할 수 있는 기능이 필요하다. 앞에서도 언급했듯이 기존의 프로그램 구동형 모의실험기는 한번에 단 하나의 프로그램만을 수행 가능하다. 또한 수행되는 응용프로그램의 총 프로세스 개수와 일치하는 시스템 프로세서를 가정한다. 본 논문은 복수의 벤치마크 응용프로그램을 하나의 응용프로그램으로 통합하고 소스코드에 간단한 이벤트 발생 코드를 추가하는 방법을 통해 전단부로 하여금 여러 벤치마크 응용프로그램을 동시에 수행시키는 방법을 제공한다. 통합된 응용프로그램은 여러 벤치마크 프로그램을 하위 프로세스로서 동시에 수행함으로써 다중화된 메모리 접근 이벤트를 발생시킨다. 즉, 서로 다른 응용프로그램에 소속된 프로세스들이 경쟁적으로 이벤트를 발생시키며 후단부 상에 모델링된 시스템은 이들 이벤트 사이에 프로세서를 할당하는 스케줄러를 구현할 수 있는 것이다. 소스코드에 추가되는 이벤트 발생코드는 스케줄러가 알고 있어야 할 응용프로그램의 정보를 전달하는 이벤트를 발생시키는 역할을 한다. 이들 이벤트는 모의실험이 시작될 때와 종료될 때 발생하므로 실제 모의실험 결과를 추출하는데 있어서 어떤 부정적인 영향도 끼치지 않는다. 두번째로, 후단부 상에서 스케줄링 정책을 구현할 수 있는 방안을 제안한다. 전단부에서 통합된 벤치마크 프로그램을 수행하면서 생성되는 프로세스의 수는 후단부에 모델링된 프로세서의 수보다 많을 수 있기 때문에, 프로세스에 대한 제한된 프로세서 자원의 할당을 스케줄링 할 스케줄러가 필요하다. 우리는 전단부와 후단부 사이에 스케줄러를 구현하고, 전단부로 부터 발생되는 모든 메모리 접근 이벤트를 스케줄러를 경유하도록 하였다. 스케줄러는 동시에 수행되는 응용프로그램 간, 또한 한 응용프로그램 내의 서로 다른 프로세스들 간에 프로세서 할당 정책을 관리한다. 전단부로부터 발생된 이벤트는 이 스케줄러를 통해 실제로 수행될 프로세서를 할당 받은 후 나머지 하드웨어 시스템을 통해 처리된다. 또한, 현재 상용되거나 연구 중인 스케줄링 정책에 대한 모델링을 지원하기 위해서 스케줄러는 특정 응

용프로그램이나 특정 프로세스에 대한 보류와 재시작을 제어할 수 있어야 한다.

그림 1 (나) 는 다중 프로그래밍 모의실험기의 구조를 보여준다. 통합된 벤치마크 응용프로그램은 전단부로 하여금 복수개의 프로그램을 동시에 활성화 시키며, 각각 프로그램의 하위 프로세스들은 제한된 시스템 프로세서를 서로 점유하기 위해 경쟁한다. 이러한 다중 프로그래밍 상황이 실제 시스템이 동작하는 상황이므로, 제안하는 다중 프로그래밍 모의실험을 통해 좀더 실질적인 성능평가가 가능하게 된다. 또한 다중 프로그래밍 모의실험을 통해 스케줄링 정책의 성능 자체를 평가할 수 있으므로, 시스템의 구조에 적합한 스케줄링 정책의 개발에도 유용하게 사용될 수 있다.

본 논문에서 제안하는 다중 프로그래밍 모의실험 방법을 구현하는 플랫폼으로 AugMint[3] 프로그램 구동형 모의실험기를 채택하였다. AugMint는 익숙한 인터페이스와 사용 편이성, 정확한 성능평가로 가장 널리 사용되는 모의실험기의 하나인 Mint[4] 계열의 모의실험기로서 전단부는 Intel x86 계열의 명령어를 해석하여 메모리 접근 이벤트를 발생시킨다.

2.2 다중 프로그램 이벤트 생성

이 장에서는 복수의 프로그램이 동시에 경쟁적으로 수행되는 환경을 구현하기 위해 벤치마크 응용프로그램을 통합하는 방안을 제시한다. 전단부는 프로세스 정보와 함께 메모리 시스템에 대한 접근 이벤트를 발생시킨다. 이때 포함되는 프로세스 정보는 모의실험 시계, 프로세스 식별자, 주소 등이고, 이들 정보는 후단부에서 메모리 읽기/쓰기, 잠금 변수(lock)와 배리어(barrier)에 대한 접근 시도 등의 동작을 실험하고자 할 때 사용된다. 기존의 프로그램 구동형 모의실험기에서 후단부는 전단부로부터의 프로세스 식별자를 해당 프로세스가 수행되는 실제 프로세서와 동일하게 간주한다. 즉, 전단부로부터 발생되는 이벤트는 그 이벤트를 발생시킨 프로세스와 동일한 식별자를 갖는 프로세서에서 수행된다는 것을 의미한다. 이는 모의실험을 하고자 하는 시스템에 정의된 프로세서의 총 개수는 모의실험의 입력으로 사용되는 응용프로그램의 프로세스 개수와 같다는 것을 의미하며, 결국, 하나의 프로세서를 여러 프로세스가 경쟁하며 공유하는 환경에 대한 모델링이 불가능하다.

우리는 전단부로 하여금 복수의 응용프로그램을 동시에 수행할 수 있도록 사용자 정의 이벤트를 발생시키는 방안을 고안했다. 통합된 벤치마크 응용프로그램을 수행하면서 전단부는 사전에 정의된 사용자 정의 이벤트를 발생시키고, 후단부는 이러한 사용자 정의 이벤트를 받아 스케줄러를 구동시켜 다중프로그램의 실행과 스케줄링 정책을 구현한다. 사용자 정의 이벤트는 모의실험의 시작과 종료, 각 응용프로그램의 시작과 종료, 각 하위 프로세스의 시작과 종료 등을 알려주며, 이는 스케줄러에게 스케줄링에 필요한 핵심적인 정보가 된다. 이들 응용프로그램의 정보를 발생시키기 위해서 응용프로그램 통합 단계에서 약간의 추가적인 변경이 필요하다. 그림 2 는 복수개의 응용프로그램 통합과 사용자 정의 이벤트를 발생시키는 부가적인 코드 변경에 대해 설명한다. 이들 사용자 정의 이벤트를 발생시키기 위해 AugMint가 매크로로 제공하는 *GEN_USER_EVENT()* 함수가 사용되었다. 그림 2 는 3개의 응용프로그램을 통합하는 예를 보여주고 있다. 각각의 응용프로그램은 통합된 하나의 메인 함수에서 분기-합류 모델에 따라 하위 프로세스로서 분기된다. 또한 각각의 응용프로그램은 내부에서 그 하위 프로세스들을 자식 프로세스로서 생성한다. 분기된 프로세스들은 서로 종료되기를 기다리며, 각 하위 프로세스들이 모두 종료되었을 때 하나의 응용프로그램이, 모든 응용프로그램들이 종료되었을 때 전체 모의실험이 종료되게 된다. 라인 3의 *NumAPP*는 동시에 수행되는 응용프로그램의 개수를 나타낸다. 스케줄러는 이 이벤트를 통해 자신이 스케줄링 해야 할 응용프로그램의 개수를 알게 되고, 그에 따라 필요한 자료구조를 초기화한다.

라인 4~5에서 응용프로그램 1과 2가 하위 프로세스로서 분기되어 실행되며, 분기된 응용프로그램들은 각각의 수행을 마친 후 라인 7에서 합류를 기다린다. 라인 9~24는 각 응용프로그램 내부에서의 사용자 정의 이벤트 생성기법을 나타낸다. 라인 11의 *START_APP* 는 응용프로그램-0 의 본격적인 실행 시작을 나타낸다. 라인 12는 응용프로그램-0 의 병렬성, 즉 총 하위 프로세스의 개수를 스케줄러에게 알려준다. 이들 프로세스 개수 정보는 스케줄러가 제한된 프로세서를 응용프로그램간에 분할하기 위해서 각 응용프로그램의 병렬성을 알아야 하기 때문에 필요하다. *Slave_Start()* 함수 내의 라인 21은 각 하위 프로세스가 시작됨을 알려준다. 주어진 예의 경우, *START_PROC*는 프로세스의 시작을, ID0는 응용프로그램-0 에 속한 프로세스임을 알린다. 프로세스가 종료될 때도 마찬가지로 라인 23에서 보듯이 END_PROC 정보를 통해 해당 프로세스의 종료를 스케줄러에게 보고한다.

이들 사용자 정의 이벤트는 전단부에 의해 생성되어 후단부의 사용자 정의 이벤트 처리 함수인 *sim_user()* 함수를 부른다. 스케줄러는 모의실험의 시작단계에서 응용프

```
1  main()
2  {
3      GEN_USER_EVENT(START_SIM, NumAPP);

       /* 응용프로그램 시작 */
       /* 하위 응용프로그램 생성 */
4      CREATE(app1_main);
5      CREATE(app2_main);

6      app0_main();

7      WAIT_FOR_END(2);    /* 합류 */
8  }

9  app0_main()
10 {
11     GEN_USER_EVENT(START_APP, ID0);
12     GEN_USER_EVENT(START_PAR, ID0, NumProc);

13     for(i = 1; i < NumProc;i++)
14         CREATE(SlaveStart); /* 하위 프로세스 생성 */

15     SlaveStart();

16     WAIT_FOR_END(NumProc - 1);  /* 합류 */
17     GEN_USER_EVENT(END_APP, ID0);
18 }

19 SlaveStart()
20 {
21     GEN_USER_EVENT(START_PROC, ID0);

22     /* 병렬 작업 */

23     GEN_USER_EVENT(END_PROC, ID0);
24 }
```

그림 2 SPLASH-II 벤치마크 응용프로그램의 통합

로그램의 개수와 각 응용프로그램의 하위 프로세스 개수를 이 함수를 통해 전달받아 그에 필요한 자료구조들을 초기화한다. 스케줄러가 관리하는 자료구조는 프로세스-프로세서 할당 테이블(process-processor mapping table, **PPMT**)과 각 응용프로그램에 대한 프로세서 할당 리스트, 그리고 대기중인 프로세스들을 관리하는 대기 큐(waiting queue) 등이다. 이들 자료구조에 대해서는 2.3 절에서 자세히 언급될 것이다.

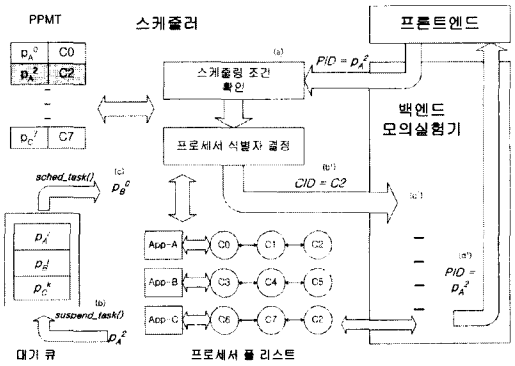
다중프로그래밍 환경에 대한 모의실험은 복수개의 병렬 응용프로그램이 경쟁적으로 수행될 때의 시스템 성능과 각 시스템 인자의 영향을 조사하는 것이 목적이다. 각 응용프로그램의 초기화 부분은 아직 모든 하위 프로세스들을 생성하기 이전이므로 경쟁상황이라고 보기 어렵고, 결국 이러한 초기화 부분은 경쟁상황에 대한 정확한 관찰을 방해하는 요인이 된다. 따라서, 다중프로그래밍 환경에서의 정확한 모의실험을 위해 각 응용프로그램의 초기화 부분을 제외한 병렬수행 부분에 대해서 실험이 이루어져야 한다. 각 응용프로그램의 초기화부분에서는 아직 병렬수행 프로세스들이 생성되기 전이므로 각각 수행된 이후, 병렬수행부분에서 생성된 하위 프로세스들은 생성과 동시에 대기 상태로 전이하게 된다. 처음에 정의된 응용프로그램의 총 수와 각 응용프로그램의 병렬성을 만족시키도록 모든 하위 프로세스들이 생성 완료되는 시점에서 대기 상태에 보류되어 있던 모든 프로세스들이 재시작되고, 이후 경쟁적인 실행단계로 진입한다. 병렬단계의 다중프로그램 수행은 전적으로 스케줄링 정책에 달려있다. 수행 중인 프로세스를 보류시키고 다른 프로세스에 대해 프로세서를 할당하여 수행시키는 방법은 스케줄링 정책에 따라 여러가지 전략을 따른다. 타이머 만료에 의한 방법이 될 수도 있고, 혹은 잠금변수 획득 실패의 경우, 프로세서의 활용도를 높이기 위해 잠금변수가 해제될 때까지 다른 프로세스를 수행하기도 한다. 한 프로세스의 수행이 완료되면 그 프로세스는 점유하고 있던 프로세서를 놓아주게 되고, 스케줄링 정책에 따라 PPMT 와 프로세서 할당 리스트의 변경이 필요하게 된다.

2.3 스케줄링 정책의 구현

이 장에서는 다중 프로그래밍 환경에서의 프로세서 공유 및 할당 정책을 모델링하기 위한 제어구조 및 자료구조를 제시한다. Gang 스케줄링 정책[8, 9] 과 같은 일반적인 시간분할형(time-multiplexing) 스케줄링 정책은 할당된 시간 만료와 함께 모든 프로세서에 대한 재배치가 필요하므로, 프로세스 대기 큐와 함께 PPMT가 필요하다. 이 경우 재배치는 스케줄링 초기에 정적으로 결정된 할당 정보를 따르므로 대기 큐로부터 꺼낸 프로세스를 PPMT에 기록된 프로세서로 바로 할당하는 방법을 따른다. 2-단계 프로세서 집합(2-level processor set) 스케줄링 정책과 같은 공간공유형(space-sharing) 스케줄링 정책의 경우, 스케줄러는 각각의 응용프로그램에 할당된 프로세서 풀(processor pool) 정보를 관리해야 한다[10, 11]. 제안하는 다중 프로그래밍 모의실험 방법은 시간분할형과 공간공유형의 두가지 스케줄링 정

책 유형을 모두 지원하기 위해 세가지 자료구조를 제시한다. 우선 스케줄 되기 위해 대기하는 프로세스들에 대한 정보를 저장하기 위해 대기 큐가 필요하다. 스케줄링 시점에서 스케줄러는 대기 큐로부터 가장 높은 우선순위를 갖는 프로세스를 선택해서 비어있는 프로세서를 할당한다. 대기 큐에 저장되는 프로세스 정보는 프로세스 식별자, 가장 최근에 프로세스가 수행되었던 시간 정보, 최근에 대기 상태로 보류될 당시 수행 중이던 트랜잭션 정보 등을 포함한다. 두번째 자료구조로 프로세스-프로세서 할당 테이블이 있다. 이 테이블은 각 프로세스가 현재, 혹은 가장 최근에 할당되어 수행된 프로세서 식별자를 관리한다. 어떤 프로세서가 활용 가능해졌을 때, 스케줄러는 이 테이블 정보를 기반으로 해당 프로세서에 할당할 프로세스 후보 군을 추출한 후, 우선순위에 따라 최종 프로세스를 결정, 수행한다. 세번째 자료구조는 각 응용프로그램에 할당된 프로세서 리스트를 관리한다. 공간공유형 스케줄링 정책의 경우, 각 응용프로그램은 자신에게 할당된 프로세서 풀 내에서 하위 프로세스들을 수행하기 때문에, 할당된 프로세서 풀을 관리하기 위한 자료구조가 이 프로세서 할당 리스트이다. 이 프로세서 할당 리스트는 수행되는 모든 응용프로그램마다 각각 관리된다. 제시한 자료구조들을 이용해 사용자는 다중 프로그래밍 환경에서의 스케줄링 정책과 자원 경쟁이 유발하는 영향을 하드웨어 시스템과 함께 모의실험 할 수 있다.

프로그램 구동형 모의실험기의 실행 구조 상, 스케줄러를 활성화 시키는 방안 역시 논의되어야 한다. 프로그램 구동형 모의실험기는 스케줄러 제어 등을 위한 사용자 쓰레드를 구현할 방법을 제공하지 않으므로, 스케줄러를 활성화시키기 위한 조건 등을 확인하는 유일한 시점은 전단부로부터 메모리 접근 이벤트가 발생할 때이다. 본 논문에서 제안하는 모의실험 방안에서 스케줄러를 활성화 시키는 이벤트는 메모리 읽기/쓰기, 잠금변수와 배리어에 대한 접근 등으로 요약된다. 이들 이벤트가 전단부로부터 발생할 때마다 우선 스케줄러의 스케줄링 조건을 확인하는 루틴을 통해 스케줄러의 활성화 여부를 결정한다. 스케줄 조건은 스케줄링 정책에 따라 달라지지만, 일반적으로 할당된 시간을 다 소모한 경우, 혹은 잠금변수와 배리어에 대한 접근 실패의 경우를 재스케줄을 위한 스케줄러 활성화로 간주한다. 스케줄링 조건 확인 이후, 메모리 읽기/쓰기 이벤트를 발생시킨 프로세스 식별자는 실제로 수행될 프로세서 식별자로의 전환이 필요하다. 이는 해당 이벤트가 실제로 모의실험하는 대상은 프로세스가 아닌 프로세서와 그에 해당되는 메모리 시스템이기 때문이다. 이 프로세서 식별자는 PPMT로부터 추출되며, 이후 후단부에서의 메모리 시스템 모의실험은 이 프로세서 상에서 이루어진다.



-PPMT : 프로세스-프로세서 할당 테이블 (Process-to-Processor Mapping Table)
-PID : 프로세스 식별자, CID : 프로세서 식별자

그림 3 스케줄러 내부 구조와 동작

그림 3은 스케줄러의 내부 구조와 동작을 보여준다. 전단부로부터 발생한 이벤트는 우선 스케줄러의 스케줄링 조건 확인 루틴을 참조하며, 이 때 이벤트를 발생시킨 프로세스를 계속 수행할 것인지, 이 프로세스를 쫓아내어 보류시키고 다른 대기 프로세스를 재실행 할 지 여부를 결정한다. 그림 3에서 보듯이, 응용프로그램-A의 하위 프로세스인 pA2 가 메모리 읽기 이벤트를 발생시켰을 때 스케줄러는 우선 스케줄링 조건을 확인한다(a). 만약 할당된 시간이 만료되어 해당 프로세스가 쫓겨나야 되는 상황이라면 스케줄러는 대기 큐로부터 C2 프로세서로 재할당되어 실행될 프로세스 p_B^0 를 선택하여 재실행시킨다(c). 대기 중인 프로세스를 재실행시키는 방법으로 AugMint 모의실험기에 정의된 sched_task() 함수를 사용한다. 그와 동시에 현재 수행중인 pA^2 프로세스는 대기 큐에 저장되며 실행이 보류된다(b). 대기 큐에 저장될 때는 추후 재실행을 위해 해당 프로세스의 현재 메모리 읽기 트랜잭션에 관계된 정보들이 함께 저장된다. 프로세스의 보류는 스케줄러가 전단부에게 해당 이벤트의 보류를 알려줌으로써 이루어진다. 우리가 제공하는 **suspend_task()** 함수는 해당 프로세스 정보를 대기 큐에 저장함과 동시에 전단부로 해당 프로세스의 보류를 의미하는 *T_YIELD*를 반환함으로써 프로세스에 대한 보류작업을 수행한다. 보류신호

를 받은 전단부는 스케줄러에 의해 다시 재시작될 때까지 보류된 프로세스를 수행하지 않는다.

스케줄링 조건을 만족시키지 않을 경우에는 기존 프로세스의 수행이 계속된다(b^1). 우선 PPMT로부터 프로세스가 할당된 프로세서 식별자를 가져온 후 남은 모의실험 과정을 거친다(c^1). pA^2 프로세스가 *C2* 프로세서에 할당되어 있으므로, 후단부는 이 *C2* 프로세서 상에서 메모리 시스템과 상호연결망을 모의실험한다. 후단부에서 이벤트에 대한 모의실험이 완료되었을 경우, 전단부로 이벤트의 수행 종료를 알려주게 되는데 이때는 처음에 이벤트 발생시킨 프로세스 정보와 함께 알려준다(dI). 주어진 예의 경우 전단부는 시스템 프로세서에 대한 정보를 해석할 수 없으므로 반드시 pA^2라는 프로세스 식별자와 함께 이벤트 종료를 알려줘야 한다.

2.4 동기화 처리

다중프로세서 시스템에서의 스케줄링 정책을 고려함에 있어 동기화 정책에 대한 처리는 큰 중요성을 갖는다. 이는 동기화 실패로 발생하는 프로세스의 봉쇄(blocking) 상태로 인해 시스템 프로세서가 유효한 작업을 수행하지 못함으로써 전체 시스템의 성능을 저하시킬 수 있기 때문이다. 만약 시스템이 스핀잠금(spin lock) 방식을 채택하고 있다면 잠금변수 획득 실패 시 프로세서 자원은 반복되는 잠금변수 획득 시도로 인해 낭비된다. 반면에 봉쇄잠금(blocking lock) 방식을 채택하는 시스템에서는 봉쇄된 프로세스가 자신의 프로세서를 다른 대기 프로세스에게 양보함으로써 프로세서의 활용도를 높이는 스케줄링 정책 수립이 가능하다. 본 논문에서 다중프로그래밍 모의실험을 구현하는 기반 도구인 AugMint 모의실험기는 기본적으로 스핀잠금을 채택하고 있지만, 다양한 스케줄링 정책을 실험하기 위해 스케줄러에서 봉쇄잠금을 지원하는 방안을 제시한다. 스케줄러는 잠금변수에 대한 접근 시도 이벤트 시 잠금변수에 대한 대기 프로세스가 존재하는지 확인한다. 만약에 현재 다른 프로세스에 의해 점유되고 있는 잠금 변수에 대해 어떤 또 다른 프로세스가 먼저 접근을 시도하고 있다면, 이는 곧 현재 수행중인 프로세스가 바로 잠금변수를 획득할 수 없음을 의미한다. 이 경우 스케줄링 정책에 따라 현재 프로세스를 보류시키고 다른 프로세스를 수행 시킴으로써 프로세서의 활용도를 높일 수 있다. 잠금변수에 대한 사전 대기 프로세스 정보를 알기 위해서 우리는 간단한 전단부 내부 잠금변수 관리 루틴으로의 접근을 통한 확인 함수를 제공한다. 배리어의 경우, 마찬가지로 스케줄링 정책의 봉쇄 상황 처리 방안에 따른다. 만약 스케줄링 정책이 봉쇄 상황에 대해 동적인 스케줄링 전략을 채택한다면, 배리어에 대한 접근이 발생했을 경우, 스케줄링 조건을 만족하는 것으로 간주하고 스케줄러를 활성화 시키게 된다.

스케줄러는 각 프로세스에 대해서 획득하고 있는 잠금변수 개수를 유지한다. 잠금변수에 대한 획득이 성공한 경우, 스케줄러는 현재 수행 중인 프로세스가 획득한 잠금변수의 개수를 증가시키고 유지하여야 한다. 반대로 잠금변수를 해제 할 경우 잠금변수 개수를 감소시킨다. 만약 획득한 잠금 변수가 존재하는 상황, 즉 잠금 변수 개수가 0보다 큰 경우 이 프로세스는 보류되어서는 안된다. 만약 이러한 프로세스가 보류된다면, 이 프로세스가 보유하고 있는 잠금변수가 풀리기를 기다리는 다른 프로세스들까지 활성화되지 못하는 무한교착(deadlock) 상태로 전이될 수 있다.

3. 구현 사례

본 장에서는 본 논문을 통해 제안한 다중 프로그래밍 프로그램 구동형 모의실험 기법을 실제 구현하여 다중프로세서 시스템의 성능평가 및 개선에 응용한 사례를 제시한다. 제안한 방법을 통해서 개발 중인 하드웨어 시스템과 운영체제의 스케줄링 정책을 함께 적용한 성능평가를 수행하고, 주어진 시스템 구조의 장단점을 활용해 최적의 성능을 낼 수 있도록 하는 새로운 스케줄링 정책을 제안한다. 이 구현 사례는 본 논문이 제안한 다중 프로그래밍 모의실험 기법이 스케줄링 정책과 자원 충돌 현상에 대해 고려한 성능평가를 가능하게 함으로써 시스템의 성능을 좀더 최적화 시키는데 유용하게 사용될 수 있음을 보여준다.

3.1 구현 시스템 및 스케줄링 정책

복수의 응용프로그램이 동시에 수행되는 환경에서 그들 간에 한정된 프로세서를 할당하는 스케줄링 전략은 크게 공간공유형과 시간분할형의 두 범주로 나눈다. 공간공유형 전략에서는 각각의 응용프로그램이 일정한 양의 프로세서를 할당받아 그 프로세서 풀 내에서 자신의 작업을 수행한다. 이 방법에서는 응용프로그램 간에 상호 배타적인 방식으로 프로세서를 할당받아 수행하기 때문에 시스템 자원을 공유하는 부분이 적어지게 됨으로써 자원 접근 충돌 및 공유로 인한 간섭현상이 크게 줄어든다[10, 11]. 이 방법의 단점은 응용프로그램의 병렬성을 최대한 활용할 수 없다는 점과 효율적인 동기화가 어렵다는 점이다. 시간분할형 전략은 응용프로그램의 병렬성을 최대한 활용해 한 순간에 하나의 프로그램만을 수행하는 것이다. 수행되는 응용프로그램 간의 전환은 할당된 시간이 만료된 시점에 이루어 진다. 응용프로

그램이 병렬수행 상태의 하위 프로세스 간에 활발한 통신이 필요한 특성을 갖는다면, 이 방법에서는 모든 프로세스가 한순간에 함께 수행되기 때문에 효율적인 프로세스 간 통신과 동기화가 이루어진다[8, 9, 12]. 또한 정적인 스케줄링 특성상 예측이 쉽고 스케줄링 전략 수립이 용이하다는 장점을 갖는다. 그러나 이 방법에서는 복수의 응용프로그램이 같은 프로세서를 서로 공유하며 수행하는 과정에서 캐시 적중 확률을 낮추는 등의 간섭현상이 발생하면서 심각한 시스템 성능 저하를 유도할 수 있다는 단점이 있다.

본 실험에서 구현할 세 종류의 스케줄링 정책은 다음과 같다. 우선, Gang 스케줄링[8] 정책은 시간분할형 전략을 채택한 전형적인 기법으로서, 응용프로그램의 병렬성이 요구하는 만큼 시스템 프로세서를 최대한 할당한 후 정해진 시간량에 따라 프로그램 간의 전격적인 전환을 통해 다중프로그램을 실행한다. 프로세서 집합 정책은 시스템 프로세서를 각 응용프로그램이 요구하는 프로세서 개수 비율에 따라 분할하여 서로 배타적인 프로세서 공간을 할당한다. 이 정책은 대표적인 공간공유형 전략을 채택하여 응용프로그램 간의 부정적인 간섭을 최소화 하는 장점을 갖는다. 프로세스 제어(process control)[6, 11] 정책은 기본적으로 프로세서 집합 정책을 따르지만, 사용가능한 프로세서가 새로 생길 때, 즉 어떤 응용프로그램이나 프로세스가 종료되었을 때 기존에 수행되던 응용프로그램의 병렬성을 증대시켜 프로세서를 최대한 활용하도록 하였다. 이 방법은 소개한 세 가지 스케줄링 정책 중 가장 높은 성능을 보이는 것으로 알려져 있다[6].

본 논문에서는 현재 본 연구팀이 개발 중인 PANDA-II 시스템에[13], 앞에서 언급한 세 종류의 스케줄링 정책을 적용하여 성능평가를 한다. 또한, PANDA-II 시스템에서의 성능을 최적화 하기 위해 노드 친화성(node affinity)을 각 스케줄링 정책에 적용하여 좀 더 높은 성능을 보이는 스케줄링 정책을 제안하고 성능을 평가한다. PANDA-II 시스템은 스누핑 캐시 일관성 프로토콜을 유지하는 캐시일관성 유지-비대칭형 메모리 접근(CC-NUMA) 시스템으로서 4개의 프로세서가 하나의 대칭형 다중프로세서(Symmetric Multiprocessor, SMP) 노드를 이루며 각 노드는 SCI(Scalable Coherent Interface)를 채택한 양방향 링 구조의 상호연결망으로 연결되어 있다. 시스템 메모리는 각 노드에 분산되어 있으며, 원격 메모리 접근을 줄이기 위한 원격캐시(remote cache)를 채택한다.

3.2 노드 친화성을 갖는 스케줄링 정책

제안한 PANDA-II 시스템은 확장성을 주된 목표로 설계되는 시스템이다. 분산된 메모리 구조와 SCI로 이루어진 링 구조의 상호연결망은 시스템의 규모가 커지더라도 대역폭이 함께 증가하기 때문에 상호연결망의 병목현상 없이 확장 가능하다. 이 구조의 단점은 점대점 방식의 링으로 이루어진 상호연결망의 높은 지연시간 때문에, 노드 간 메모리 접근, 즉 원격 노드의 메모리 접근 지연 시간이 매우 크다는 점이다. 원격 메모리에 대한 접근이 많아질 수록 시스템 성능이 저하되기 때문에 원격 메모리 영역에 대해서만 캐싱을 하는 원격캐시를 채택하지만, 다중 프로그래밍 환경에서 복수의 응용프로그램이 경쟁적으로 수행되는 상황에서는 몇 가지 문제점이 있다. 우선 서로 다른 응용프로그램 간에 캐시를 공유하게 되면 캐시 적중률을 낮춰 시스템 성능을 저하시키게 된다[6]. 한 응용프로그램이 활성화되면 캐시에 존재하는 이전 프로그램 데이타를 쫓아내게 되며, 이는 곧 캐시의 적중률을 저하시킴과 동시에 원격메모리로의 접근을 빈번하게 발생시키기 때문에 효율적인 시스템 운용이 어렵다. 또한, 만약에 한 프로그램의 하위 프로세스들이 여러 노드에 분산되어 실행된다면 그들 간의 데이타 교환 역시 원격메모리 접근을 통해 이루어지므로 높은 성능을 기대할 수 없다.

우리는 PANDA-II 시스템의 구조적 장점을 극대화 시키고자 노드친화성 스케줄링 정책을 제안하고 이를 본 논문에서 제안하는 다중 프로그래밍 프로그램 구동형 모의실험을 통해 성능평가한다. 프로세스가 프로세서를 할당 받을 때, 최대한 동일 응용프로그램의 프로세스들이 속한 노드 내에 할당될 수 있도록 우선순위를 조정한다. 동일 응용프로그램의 하위 작업들이 같은 노드 내에서 실행되면, 공유되는 데이타 접근이나 데이타 교환 시에 원격 메모리로의 접근을 피할 수 있기 때문에, 상호연결망의 높은 지연시간을 피할 수 있다. 또한 보류되었던 프로세스가 재실행 될 경우, 이전에 수행되던 프로세서 상에서 재 실행되는 것이 최우선이겠지만, 이 프로세서가 이미 다른 작업을 수행 중인 경우, 최대한 같은 노드 내의 프로세서로 이동하게 한다. 이 경우, 프로세서 위치는 달라지지만 원격 캐시를 공유하기 때문에 이전에 사용하던 데이타가 원격캐시에 남아 있을 확률이 다른 노드로 이동하는 것보다 높아진다. 이는 원격메모리 접근이 줄어든다는 것을 의미하기 때문에 시스템의 성능향상을 가져올 수 있다.

3.3 다중 프로그래밍 모의실험 결과

모의실험을 위한 다중프로그램 입력으로 3개의 SPLASH-II 벤치마크 응용프로그램을 사용한다. FFT,

LU, Radix 프로그램이 통합되었으며, 각각 16개의 하위 프로세스를 생성시켜 총 48개의 프로세스가 시스템 자원을 두고 경쟁하면서 동시에 수행된다. 각 응용프로그램의 작업량은 표 1에, 모의실험 인자는 표 2에 설명되어 있다. 스케줄링 정책 간의 성능 차이를 정확히 측정하기 위해, 본 실험에서는 각 응용프로그램의 병렬 수행 부분만을 실험 결과로서 분석하였다. 표 3은 각각의 스케줄링 정책에 따른 원격캐시 적중률과 동기화 오버헤드를 보여준다. 노드 친화성을 채택한 프로세서 집합 정책과 프로세스 제어 정책은 3.5 ~ 3.6 % 정도의 캐시 적중률 증가를 보였다. Gang 스케줄링 정책은 시간분할형 전략의 장점인 가장 좋은 동기화 성능을 보인다. 다른 정책에서의 동기화는 Gang 보다는 좋지 않은 성능을 보이지만, 노드친화성 정책을 채택한 경우 보다 나아진 동기화 성능을 보임을 알 수 있다.

표 1 응용프로그램의 작업크기

Workload	Size	Processes
FFT	65536 complex doubles, 256 × 256 matrix	16
LU	16 × 16 element blocks	16
Radix	262144 keys, radix = 1024	16

표 2 모의실험 인자

총 프로세서 수	16
노드당 프로세서 수	4
프로세서 캐시 크기	64KB
원격 캐시 크기	512KB
캐시 라인 크기	32B
프로세서 캐시 조합수 (set associativity)	1
원격 캐시 조합수	4
프로세서 클럭	500MHz
지역버스 클럭	100MHz
데이타버스 너비	64 bits
링 클럭	500MHz
링 데이타 너비	16 bits
프로세서 캐시 접근 시간	1 프로세서 클럭
원격 캐시 접근 시간	10 지역버스 클럭
지역버스 명령 지연시간	5 지역버스 클럭
지역버스 데이타 지연시간	10 지역버스 클럭
인접 링 간의 명령 지연시간	25 링 클럭
인접 링 간의 데이타 지연시간	50 링 클럭
프로세스 교체 지연시간	0.1 ms
Gang 스케줄링 정책의 시간할당량	100 ms

표 3 원격 캐시 적중률과 동기화 지연시간

스케줄링 정책	원격 캐시 적중률 (%)	동기화 지연시간[a]
Gang	36.9	100
Ps [b]	39.5	136.7
Ps-aff [c]	43.1	133.5
Pc [b]	39.1	127.2
Pc-aff [c]	42.6	122.3

[a]Gang 스케줄링 정책의 전체 동기화 지연시간을 100으로 간주

[b]Ps : 프로세서 집합 정책,
Pc : 프로세스 제어 정책

[c]aff : 노드친화성 적용 시

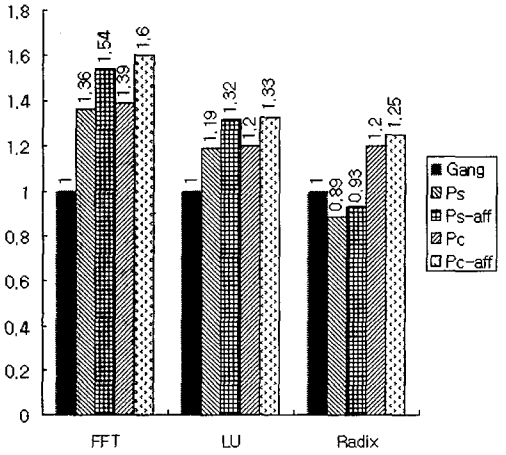


그림 4 Gang 정책과 비교한 각 스케줄링 정책의 성능향상

그림 4는 Gang 스케줄링 정책에서의 시스템 성능을 기준으로 한 각 스케줄링 정책의 성능향상을 나타낸다. 스케줄링 정책을 적용한 시스템의 성능은 프로세서 활용도, 캐시 적중률, 동기화에 따른 지연시간 등이 복합적으로 작용하는 결과이다. FFT 에서는 노드친화성이 적용된 프로세스 제어 정책이 Gang 정책에 비해 1.60 배의 성능을 보인다. 노드친화성을 적용하지 않은 기존의 프로세스 제어 정책의 경우 성능향상은 1.39 배에 머무른다. LU의 경우 동기화 작업이 많은 부분을 차지하기 때문에[14], 공간공유형 정책에서의 성능향상은 한계를 보일 수 밖에 없다. 따라서 FFT에 비해 비교적 적은 성능향상을 보임을 확인할 수 있다. Radix는 정렬연산이 주된 작업인 프로그램으로 원격메모리 쓰기가 불규칙하게 분산되어 자주 발생한다[5]. 이러한 특성은 높

은 동기화 성능을 요구하기 때문에 Gang 스케줄링 정책이 프로세서 집합 정책보다 높은 성능을 보인다. 프로세스 제어 방식이 Gang 정책보다 높은 성능을 보이는 이유는, 동기화로 인해 봉쇄된 프로세스 대신 다른 작업을 실행시킴으로써 향상된 프로세서 활용도가 동기화 지연에 따른 오버헤드를 상쇄시키기 때문이다. 프로세서의 활용도를 높이는 특성으로 인해 프로세스 제어 정책이 다른 스케줄링 정책에 비해 가장 좋은 시스템 성능을 보이며, 노드 친화성을 채택했을 경우 시스템 성능을 더욱 향상시킴을 본 실험을 통해 알 수 있다.

4. 결 론

본 논문은 공유메모리 다중프로세서 시스템의 성능평가를 수행함에 있어, 하드웨어 시스템만이 아닌 운영체제의 스케줄링 정책까지 통합 실험함으로써, 다중 프로그래밍 환경을 모의실험하는 방법을 제안했다. 프로그램 구동형 모의실험기를 기반으로 복수의 벤치마크 응용프로그램을 통합하여 다중 프로그래밍 이벤트를 발생시키고, 프로세스 간의 프로세서 할당 알고리즘을 운용하는 스케줄러 구현기법을 제시하였다. 제안된 방안을 통해 시스템 개발자는 시스템의 구현없이 저비용으로 실제 환경과 유사한 다중 프로그래밍 환경에서의 성능에 대한 예측과 분석이 가능하다. 복수의 응용프로그램을 동시에 수행하는 과정을 통해, 제한된 시스템 프로세서의 공유와 충돌에 따르는 영향을 측정할 수 있으며, 또한 시스템의 구조에 따른 스케줄링 정책의 영향을 파악할 수 있다. 개발 중인 다중프로세서 시스템의 성능평가에 이 기법을 도입함으로써 보다 실제적인 모의실험을 통해 시스템의 성능을 극대화시키는 스케줄링 정책 수립이 가능함을 개발사례를 통해 제시하였다.

참 고 문 헌

[1] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl. Proteus: A high-performance parallel-architecture simulator, Technical Report LCS/TR-516, MIT, Sept. 1991.

[2] H. Davis, S. R. Goldschmidt, and J. Hennessey. Multiprocessor simulation and tracing using tango., In Proc. of International conference on Parallel Processing, 1991.

[3] A-T. Nguyen, M. Michael, A. Sharma, and J. Torrellaz, The Augmint multiprocessor simulation toolkit for Intel x86 architecture, In Proceedings of the IEEE International Conference on Computer Design, Oct. 1996.

[4] J. E. Veenstra and R. J. Fowler.. Mint tutorial and user manual, Technical Report TR452, The university of Rochester, June 1993.

[5] B. Verghese, A. Gupta, and M. Rosenblum. Performance isolation: Sharing and isolation in shared-memory multiprocessors, In Proc. of ASPLOS VIII, Oct. 1998.

[6] A. Gupta, A. Tucker, and S. Urushibara. The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications, In Proc. of SIGMETRICS, 1991.

[7] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer simulation: The simos approach, In IEEE Parallel Distrib. Technol. Winter, 1995.

[8] J. K. Ousterhout. Scheduling techniques for concurrent systems, In Proc. of International Conference on Distributed Computing Systems, 1982.

[9] Y. Zhang, H. Franke, J. E. Moreira, and A. Sivasubramaniam. Improving parallel job scheduling by combining gang scheduling and backfilling techniques, In Proc. of International Parallel and Distributed Processing Symposium, May. 2000.

[10] D. L. Black. Scheduling support for concurrency and parallelism in the mach operating system, In IEEE Transaction on Computer, May 1990.

[11] A. Tucker and A. Gupta. Process control and scheduling issues for multiprogrammed shared-memory multiprocessors, In 12th ACM Symposium on Operating Systems Principles, 1989

[12] R. Chandra, S. Devine, and B. Verghese. Scheduling and page migration for multiprocessor compute servers, In Proc. of ASPLOS-VI, Oct. 1994.

[13] 윤주범, 장성태, 전주식, 이중 링 CC-NUMA 시스템에서 링 구조 변화에 따른 시스템 성능 분석, 정보과학회 논문지, 시스템 및 이론, 29권 pp. 105 115, 2002년 2월

[14] S. C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. Methodological considerations and characterization of the splash-2 parallel application suite, In Proc. of 22th Annual International Symposium on Computer Architecture, 1995.

최 효 진

1994년 2월 서울대학교 컴퓨터공학과 학사. 1996년 2월 서울대학교 컴퓨터공학과 석사. 1998년 2월 서울대학교 컴퓨터공학과 박사과정 수료. 2001년~현재 (주)지씨티리써치 책임연구원. 관심분야는 컴퓨터구조, 병렬처리, 임베디드 시스템

전 주 식

1975년 2월 서울대학교 응용수학과 학사. 1977년 2월 한국과학기술원 전산학과 석사. 1983년 2월 미국 Univ. of Utah 박사. 1983년~1985년 Univ. of Iowa 조교수. 1985년~현재 서울대학교 컴퓨터공학과 교수. 1996년~2002년 컴퓨터신기술공동연구소 소장. 2001년~현재 Korea bluetooth forum committee chariman. 관심분야는 컴퓨터 구조, 병렬처리, VLSI/CAD