논문 2011-48CI-3-10

멀티스레드 어플리케이션을 위한 실시간 성능모니터의 구현

(The Implementation of Real-time Performance Monitor for Multi-thread Application)

김 진 혁*, 신 광 식**, 윤 완 오*, 이 창 호*, 최 상 방***

(JinHyuk Kim, KwangSik Shin, WanOh Yoon, ChangHo Lee, and SangBang Choi)

요 약

마이크로프로세서의 발전과 함께 멀티코어 시스템은 점차 보편화 되어가고 있다. 이러한 하드웨어 성능향상 패러다임의 변화로 인해 소프트웨어의 성능향상을 위해서는 기존의 싱글 스레드 어플리케이션들을 멀티 스레드 어플리케이션들로 교체하는 과정이 필수적이다. 멀티 스레드 어플리케이션 개발의 복잡성 때문에, 성능모니터링 도구는 어플리케이션의 성능 최적화를 지원하기에 유용한 도구로 사용된다. 기존의 성능모니터링 도구는 사용의 편의성이나 실시간성의 지원보다는 성능 측정 자체에 초점이 맞춰져 있다. 실시간 성능 모니터는 멀티 스레드 어플리케이션이 수행하는 동안 나타나는 문제점을 파악하는 것 뿐 아니라 실시간으로 어플리케이션의 동작 상태를 개발자가 확인 할 수 있기 때문에 단순한 성능 지표들만으로 문제점의 원인을 찾아내야하는 비 실시간 성능 모니터에 비해 효과적인 도구로 사용될 수 있다. 본 논문에서는 멀티코어 시스템을 위한 실시간 성능모니터링 도구인 RMPM(Real-time Multi-core Performance Monitor)를 제안하고 성능 측정 주기로 인한 오버헤드와 정확성 사이의 관계를 비교하여 최적의 측정 주기를 결정하였다. 제안한 성능모니터는 전체시스템의 CPU 사용량, 메모리 사용량, 네트워크 사용량 뿐아니라 시스템의 코어별, 어플리케이션의 스레드별 부하 분산상태를 나타낼 수 있다.

Abstract

Multi-core system is becoming more general with development of microprocessors. Due to this change in performance improvement paradigm, switching conventional single thread applications with multi thread applications. Performance monitoring tools are used to optimize application performance because of complexity in development of multi thread applications. Conventional performance monitoring tools are focused on performance itself rather than user friendliness or real-time support. Real-time performance monitor identify the problem while multi-threaded applications should be performed as well as check real-time operating status of the application. So it can be used as an effective tool compared to non-real-time performance monitor that only with simple performance indicators to find the cause of the problem. In this paper, we propose RMPM(Real-time Multi-core Performance Monitor) which is real-time performance monitoring tool for multi-core system. Observation period is optimized by comparing relation between overhead due to performance evaluation period and accuracy. Our performance monitor shows not only amount of CPU usage of whole system, memory usage, network usage but also aspect of overhead distribution per thread of an application.

Keywords: performance monitor, real-time, multi-core, multi-threading, multimedia

Ⅰ. 서 론

1986년부터 2002년까지 마이크로프로세서의 클럭 속도는 매년 52%씩 증가해 왔다. 이는 무어의 법칙처럼 트랜지스터의 가격이 떨어지고, 프로세서 설계 기술이 발달해 왔기 때문에 가능하였다. 소프트웨어 관점에서 이러한 현상은 하드웨어의 발달로 인하여 어플리케이션

* 학생회원, *** 평생회원, 인하대학교 전자공학과
(Dept. of Electronic Engineering Inha University)
** 정회원, 한국전자통신연구원 스마트게임연구팀
(Electronics and Telecommunications Research Institute)
※ 이 논문은 정부(교육과학기술부)의 재원으로 한국연구재단의 중점연구소 지원사업으로 수행된 연구임 (2011-0018394)
접수일자: 2010년9월16일, 수정완료일: 2011년5월12일

의 성능은 자연히 향상된다고 하여 무료점심(free lunch)이라고 하였다. 그러나 최근 들어 하드웨어 성능을 향상을 위한 패러다임이 변화하고 있다. 오늘날에는 예전과 달리 프로세서의 클럭 속도 향상 없이 프로세서 코어의 개수를 증가시킴으로서 전체 프로세서의 성능을 향상시키고 있으며 이미 멀티코어 시스템은 보편화되어 있다. 그러나 멀티코어 시스템은 소프트웨어가 동시에 여러 가지 동작을 수행 할 때에만 비로서 향상된 성능을 사용할 수 있다. 이렇게 하나의 소프트웨어가 여러 프로세서를 사용하는 프로그램을 멀티 스레드 프로그램이라고 하며 기존의 싱글 스레드 프로그램과 개발방법에서 큰 차이를 보인다. 하드웨어 성능 향상 패러다임이 바뀜에 따라 어플리케이션의 성능향상을 위해서는 순차적 연산을 수행하도록 작성된 싱글 스레드 프로그램을 멀티코어 환경에 맞게 다시 작성해야 한다.[1]

일반적으로 멀티미디어 콘텐츠는 시스템 성능에 가장 민감한 어플리케이션으로 꼽을 수 있다. 특히 최근에 출시되고 있는 고화질의 3D게임은 시스템 부하가 가장 큰 콘텐츠이다. 그럼에도 불구하고, 아직까지 대부분의 컴퓨터 게임은 싱글스레드로 동작한다. 멀티코어 시스템의 성능은 해당 어플리케이션이 동시에 얼마나 많은 연산 수행이 가능한가에 따라 그 성능이 결정되기 때문에 만일 어떤 어플리케이션이 멀티스레드로 동작하지 않는다면, 프로세서(코어) 개수와는 상관없이 해당 프로그램은 동일한 성능으로 동작할 것이다.[2]

시스템 자원을 최적으로 사용하는 병렬 분산 어플리케이션을 작성하는 것은 어려운 문제이기 때문에 일반적으로 시스템 자원은 덜 사용되거나 비효율적으로 사용되곤 한다. 프로그램의 성능을 결정짓는 요인은 복잡하고, 상호 연관되어 있기 때문에 표면적으로 드러나지 않는다.[3] 이러한 변화와 복잡성 때문에, 성능모니터링 도구는 어플리케이션의 성능 최적화를 지원하기에 유용하다. 성능모니터링 도구는 시스템이 얼마나 많은 자원을 사용하고 있고 성능을 향상시키기 위하여 프로그램의 어느 부분을 개선해야 하는가에 대한 가이드를 개발자에게 제공한다.[4~7]

실시간 모니터링 기술은 프로그램 개발단계에서 개발자에게 효율적인 최적화 도구인 동시에 사용자에게 자신의 시스템을 평가할 수 있는 도구로 활용 될 수 있다. 오늘날의 멀티미디어 콘텐츠는 점점 더 고성능을 요하기 때문에, 이러한 새로운 멀티미디어 콘텐츠를 즐기기 위하여 사용자는 자신이 해당 콘텐츠를 즐기기에 충분한 성능을 갖은 시스템을 보유하고 있는지 확인하고 싶어 하며 충분한 성능을 갖고 있지 않은 시스템을 업그레이드 한다. 이러한 사용자들의 요구에 부응하고자, 최근에 출시되는 일부 3D 게임의 경우 게임을 실행하는 동안 시스템의 성능정보를 화면에 표시해주곤 한다. 비록 대다수의 게임이 FPS (frame per second), 메모리 사용률, 네트워크 트래픽 상태 등과 같은 간단한 정보를 제공하기는 하지만, 아직까지 멀티코어 시스템의 부하 분산 상태 등과 같은 정보를 제공하지는 않는다.

본 논문은 멀티코어 시스템을 위한 실시간 성능모니터링 도구인 RMPM(real-time multi-core performance monitor)를 제안한다. RMPM은 콘텐츠 내부에 적용되어, 해당 콘텐츠를 즐기면서 시스템의 성능과 부하 분산 상태를 확인할 수 있다. 본 논문은 다음과 같이 이루어져 있다. Ⅱ장에서는 성능 모니터링의 필요성과 기존의 성능 모니터링 도구들을 설명하고 Ⅲ장에서는 본 논문에서 제안하는 실시간 성능 모니터의 특징 및 구조를 소개한다. Ⅳ장에서는 성능 측정으로 인해 발생하는 부하를 최소화 하면서 측정 정보의 정확도를 최대화 할 수 있도록 실시간으로 변하는 성능정보 모니터링을 위한 최적의 측정 주기를 결정하고, RMPM이 적용된 어플리케이션의 예를 보여준다. 마지막으로 Ⅴ장에서 결론을 맺는다.

Ⅱ. 배 경

2.1 멀티코어 CPU 구조

빠르고 값싼 프로세싱 능력에 대한 수요로 인해, 프로세서 제조사는 더 조밀하고, 작고, 빠른 회로를 이용하여 CPU의 성능을 높이고자 차세대 프로세서를 만들기 위해서 끊임없이 새로운 방법을 찾아왔다. 최근 들어 멀티코어 프로세서 구조는 하나의 프로세서에 실행 단위인 코어프로세서를 6개 이상 올리는 것을 가능하도록 하였다. 이러한 멀티프로세서는 하나의 프로세서 소켓에 연결되지만, OS는 논리적으로 개별 프로세서로서 각각 독립적으로 실행 가능한 코어로서 인식한다.

멀티코어 구조의 기본 아이디어는 나눔과 정복(divide and conquer) 전략에 바탕을 둔다. 즉, 과거에 단일 코어 마이크로프로세서에 의해 수행되었던 연산 작업이 오늘날에는 여러 코어에 분산되기 때문에, 멀티코어 프로세서는 주어진 시간 안에 더 많은 작업을 수

행할 수 있다. 그러나, 실질적으로 이러한 성능개선을 얻기 위해선 멀티코어 프로세서에서 수행되는 소프트웨어는 다중수행코어가 여러 연산을 분산 실행할 수 있도록 연산로직을 다시 설계해야만 한다. 이러한 기능을 “스레드 레벨 병렬화” 또는 “쓰레딩”이라 부르고, 이러한 기능을 지원하는 어플리케이션을 “스레드 지원 어플리케이션” 또는 “멀티 스레드 지원 어플리케이션”이라고 한다.[8]

2.2 멀티코어 시스템을 위한 태스크 병렬화

멀티코어 시스템은 얼마나 많은 스레드가 동시에 실행될 수 있는가에 따라서 그 성능이 결정된다. 그러므로 멀티코어 시스템은 병렬 컴퓨팅의 수요에 맞춰서 발전한다. 병렬 컴퓨팅은 많은 연산이 동시에 수행되는 연산방법의 한 형태로 큰 문제가 동시에 풀릴 수 있도록 더 작은 문제로 나누는 병렬화 과정이 요구된다[9]. 태스크 병렬화는 일반적으로 하나의 큰 작업이 나눠지는 방법에 따라서 기능적 분리와 데이터 분리로 분류된다. 기능적 분리란 각각의 기능별로 하나의 태스크를 매핑하는 것을 말하고, 데이터 분리란 동일한 기능에 대해 연산되는 데이터에 따라 각기 다른 태스크를 매핑하는 방법을 나타낸다. 이러한 병렬화 기법은 각각 서로 다른 장단점을 갖는다. 기능적 분리는 연산 블록단위로 분리하기 때문에 비교적 스레드로 분리하기가 쉬운 반면, 병렬연산이 가능한 수가 시스템의 코어 수 증가에 관계없이 연산 블럭의 수로 제한되는 단점이 있다. 또한 각 스레드가 수행하는 연산 분석의 복잡도가 다르기 때문에 균등한 부하분산이 어렵다. 데이터 분리 기법은 기능적 분리 기법이 갖는 확장성과 균등한 부하분산에 대한 문제를 비교적 쉽게 해결할 수는 있지만, 하나의 큰 작업을 스레드로 분리하기가 어려운 단점이 있다.

2.3 성능모니터링 도구

병렬처리 구조를 효율적으로 이용할 수 있는 어플리케이션을 만들기 위하여, 프로그래머는 시스템의 성능 데이터를 수집하고 분석하는 효율적인 도구가 필요하다[10]. 또한 일반 사용자 역시 자신이 사용하는 시스템이 해당 어플리케이션을 수행하기에 충분한 성능을 갖추었는지를 보여주는 도구를 필요로 한다. 현재 개발자와 일반 사용자에게 가장 많이 사용되는 성능분석 및 모니터링 도구는 각각 인텔의 VTune[11]과 MS의 Wndows Perfmon[12]이다.

VTune은 개발자를 위한 가장 성능이 좋은 분석도구로 인텔 프로세서 안에 디버깅 목적으로 포함되어있는 레지스터를 이용하여 특정 어플리케이션에 대한 멀티쓰레딩 정보를 비롯한 다양한 시스템 성능을 측정 및 분석하는 것이 가능하다. 프로그램 실행 루틴 단계별 분석이 가능할 정도로 자세한 분석이 가능하지만, 인텔 프로세서에서만 사용이 가능하며 프로그램이 실행되는 동안 실시간으로 성능 변화를 확인 할 수는 없다. 그러므로 프로그램 소스코드의 라인 단위로 자세한 분석이 필요한 개발과정에서는 유용한 분석 도구이지만, 사용방법이 어렵고 바이너리코드를 사용하는 일반사용자에게는 부적절하다. 또한 인텔 프로세서 내의 특수 레지스터를 사용하기 때문에 인텔 프로세서가 아닌 다른 프로세서를 사용하는 시스템에 적용이 불가능하다.

Windows Perfmon은 사용이 쉽고 시스템 성능 정보에 대한 실시간 모니터링이 가능하지만, 어플리케이션 내부 루틴의 정보를 알 수 없기 때문에 멀티 쓰레딩 프로그램의 스레드 정보에 대한 분석이 불가능 하다. 멀티코어 시스템은 기존의 싱글코어 프로세서와 달리 스레드의 부하분산이 시스템 성능을 좌우하기 때문에 시스템 성능평가를 위해선 스레드의 부하 분석이 요구된다.

Ⅲ. 실시간 멀티코어 성능모니터

본 논문에서 제안하는 RMPM은 기존의 성능 모니터들이 제공하지 못했던 실시간성 및 멀티 스레딩 정보 제공을 보완한 멀티코어 시스템을 위한 실시간 성능모니터 도구이다. RMPM은 멀티쓰레딩 프로그램 안에 포함되는 라이브러리의 집합으로 성능 측정 모듈과 모니터링 모듈로 구성된다. 성능 측정모듈은 시스템의 부하정보를 측정하고, 모니터링 모듈은 측정모듈에서 수집된 정보를 해당 콘텐츠 화면위에 그래프와 텍스트로 표시한다. RMPM은 2D/3D 게임은 물론 DirectX를 사용하는 다른 멀티미디어 콘텐츠의 부하 및 시스템 성능모니터링에 사용된다. 주로 해당 프로세서의 CPU 사용률과 메모리 사용률, 각 스레드의 CPU 점유율, 네트워크 트래픽 양, FPS 등을 측정한다. 또한 멀티쓰레딩 어플리케이션 블럭의 각 스레드 정보 추출이 가능하기 때문에 실시간 성능모니터링 뿐만 아니라 멀티코어 시스템과 멀티쓰레딩 프로그램의 부하분산 상태를 보여준다.

3.1 특징

RMPM은 멀티미디어 어플리케이션을 즐기기 위해 멀티코어 시스템을 사용하는 일반 사용자에게 성능모니터링을 위한 다양한 특징을 제공한다. 특히, 기존의 VTune과 Windows Perfmon의 단점으로 지적됐던 부분을 개선하였다.

RMPM은 어플리케이션 내부 모듈로 포함되기 때문에 손쉽게 시스템 성능을 확인할 수 있으며 실시간으로 정보를 제공한다. 또한 사용자가 성능 정보를 쉽게 인식할 수 있는 직관적인 인터페이스를 갖는다. 따라서 멀티 스레드 어플리케이션의 성능 하락이나 시스템 자원의 충족성을 손쉽게 파악할 수 있다.

부하 분산은 멀티 쓰레딩 프로그램 및 멀티코어 시스템의 성능을 나타내는 가장 중요한 척도이다. RMPM은 시스템의 프로세서 또는 스레드별 CPU점유율을 나타냄으로써 멀티코어 시스템의 부하 분산 정도를 사용자에게 제공한다. 또한 다른 실행중인 스레드와의 부하 비교를 통하여 개발 중인 멀티미디어 컨텐츠의 시스템 부하도를 측정할 수 있기 때문에 개발자는 좀 더 최적화된 어플리케이션을 개발할 수 있다.

3.2 구조

RMPM은 실시간 성능모니터링 API로 RMPMCore와 RMPMGraph 두 개의 C++ 클래스로 구성된다. RMPM은 프로세서의 제약없이 Microsoft Windows OS를 사용하는 모든 플랫폼에서 사용이 가능하며 멀티미디어 어플리케이션 내부에 라이브러리 형태로 삽입되어 동작한다. RMPMCore와 RMPMGraph 클래스는 각각 성능 측정과 성능모니터링 기능을 수행한다. RMPMCore는 PDH (Performance Data Helper) 라이브러리와 kernel32 라이브러리를 사용하여 시스템의 성능 정보를 수집한다. PDH는 용어 그대로 성능 데이터를 수집하고 계산하는 과정을 도와주는 Win32 라이브러리 이다. PDH에 성능측정을 요구하기 위해서는 Counter name string형식을 사용해야 한다. Counter name string이란 얻어내고자 하는 카운터를 지정하는 형식화된 문자열로서 다음과 같은 형식으로 이루어져 있다.

\PerfObject(ObjectInstance)\Counter

또한 얻어내고자 하는 카운터는 반드시 쿼리(Query)에 넣어야 한다. 그리고 나서 PDH에 쿼리를 보내어 정보를 얻는다. 일반적인 데이터 베이스(DataBase)의 스킴(Scheme) 이라고 볼 수 있다. 이를 통해 PDH를 통해

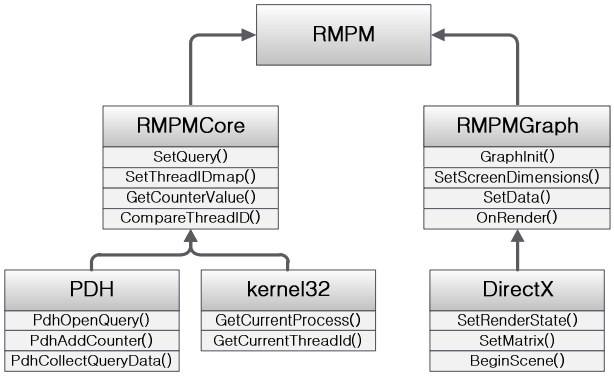

그림 1 RMPM 구조
Fig. 1. RMPM architecture.

한 번에 하나 이상의 서로 다른 카운터들에 대한 정보를 얻을 수 있다. 필요한 모든 카운터를 쿼리에 넣고 PDH에 보내주면 PDH는 쿼리 된 모든 카운터들의 자료를 획득하여 제공한다.

PDH를 통하여 주로 CPU와 메모리 사용률, FPS, 네트워크 트래픽 양 등의 정보를 수집할 뿐만 아니라 kernel32 라이브러리로부터 취득한 프로세스와 스레드 정보와의 관계를 이용하여 해당 어플리케이션에 의해 발생하는 프로세스 및 개개 스레드의 부하량을 측정한다. 한편 RMPMGraph는 DirectX를 이용하여 그래프, 텍스트 등으로 수집된 정보를 현시한다. 그림 1은 RMPM의 구조를 보여준다.

3.2.1 RMPMCore

RMPMCore는 초기화 모듈, 쿼리 셋팅, 그리고 성능정보 수집의 세 단계로 구성된다. 먼저 다른 초기화 루틴과 함께 시스템의 프로세서 개수를 확인한 후 측정할 항목을 선택하기 위해 RMPMCore의 SetQuery() 함수를 이용하여 성능 카운터 쿼리를 설정한다. 일단 쿼리가 생성되면 지정된 측정 주기에 따라 GetCounterValue() 함수를 이용하여 성능 정보를 얻을 수 있다. 이러한 과정을 통하여 각 코어의 사용량 측정 대상 프로세스의 CPU 점유율 및 메모리 점유율, 네트워크 송/수신 패킷의 전송률 등 스레드 정보를 제외한 대부분의 성능 데이터를 측정할 수 있다.

스레드는 해당 프로그램의 구조에 종속되어 동작하기 때문에 특정 스레드의 부하량을 측정하기 위해서는 추가적인 작업이 요구된다.

특정 스레드의 정보를 수집하여 초기화 단계에서 프로그램의 각 루틴을 수행한 뒤, 생성한 스레드에 대한 정보를 저장하기 위해 CreateThreadInfo(),

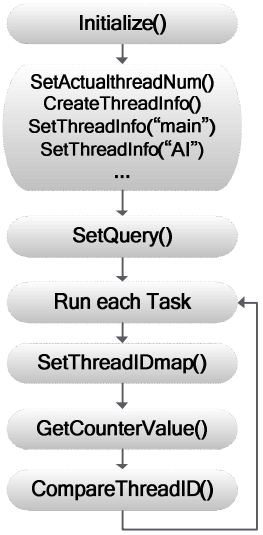

그림 2. RMPMCore의 순서도
Fig. 2. Flow chart of RMPMCore.

SetActualThreadNum(), SetThreadInfo()의 함수를 이용하여 ThreadInfo 변수를 할당한 후 스레드 ID와 해당 태스크 정보를 저장한다. 만일 현재 어플리케이션이 태스크를 수행할 때 마다 다른 스레드가 지정되는 스레드 풀을 사용한다면, SetThreadIDmap() 함수와 같이 ThreadInfo와 해당 스레드 사이의 관계를 연결하는 루틴이 추가되어야 한다. 그림 2는 RMPMCore의 순서도를 보여준다.

3.2.2 RMPMGraph

RMPMGraph는 그래프 초기화, 측정정보 로드, 결과 현시과정으로 구성된다. 초기화 단계에서는 GraphInit(), SetLocation(), SetSize()의 함수를 이용하여 현시 타입, 위치, 크기 등을 설정한다. 뿐만 아니라 OnCreateDevice(), OnResetDevice()와 같은 DirectX API 사용을 위한 DirectX 디바이스 설정과정도 필요하다. 초기화 루틴이 끝나면 HxPerfThread 객체로부터 성능 측정정보를 받아서 화면에 출력하고자 하는 데이터 값을 설정하는 SetData() 함수를 실행한다. 이와 같은 성능모니터링을 위한 준비가 끝나면 Updategraph(), Render() 등을 이용하여 표시할 데이터 값을 갱신한 후 그래프 혹은 텍스트로 화면에 그려준다. 그림 3은 RMPMGraph의 플로우 차트를 보여준다.

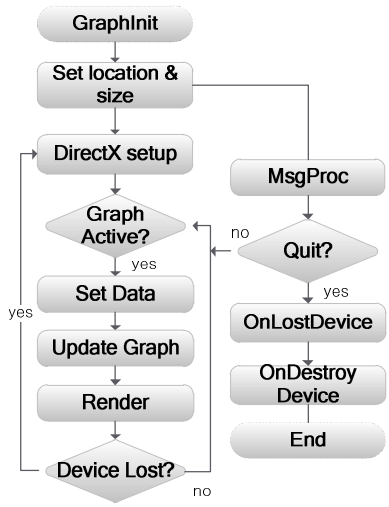

그림 3. RMPMGraph의 순서도
Fig. 3. Flow chart of RMPMGraph.

Ⅳ. 성능 측정

이번 장에서는 성능 측정으로 인해 발생하는 오버헤드와 측정값의 정확도 사이의 관계에 대해 설명하고, 성능정보의 정확성을 개선할 수 있는 적절한 측정 주기를 선택한다. 또한 최적화된 측정 주기를 바탕으로 멀티코어 시스템에서의 성능모니터링 결과를 보여준다.

4.1 측정 주기

이번 절에서는 측정 오버헤드와 측정값의 정확도 사이의 관계에 대해 설명한다. 성능 데이터를 모으는 일은 측정 빈도에 따라 추가적으로 발생하는 부하가 요구된다[13].

오버헤드를 줄이기 위해서는 측정 빈도를 줄여야 하지만, 이는 동적으로 변하는 성능정보에 대한 측정값의 정확성에 악영향을 끼친다. 이와 같이 측정 주기에 따른 오버헤드와 정확성은 트레이드오프 관계가 있다. 그러므로, 최적의 측정 주기를 결정하기에 앞서서, 측정 주기에 따른 오버헤드와 정확성에 대한 에러율을 비교한다.

4.1.1 오버헤드

성능 측정 과정은 시스템에 추가적인 부하를 야기한다. 성능 측정으로 인한 오버헤드는 피할 수 없지만, 적절한 측정 주기를 설정함으로써 시스템 성능에 무리를 주지 않는 범위 내에서 오버헤드를 제한할 수 있다. 본 논문에서는 성능 측정 주기에 따라 발생하는 오버헤드를 분석하기 위하여 독립적인 스레드를 추가하여 시스템 성능을 측정하고, 추가된 스레드의 CPU 사용량을 측정하여 오버헤드 값을 구한다. 이를 위하여 8개의 코

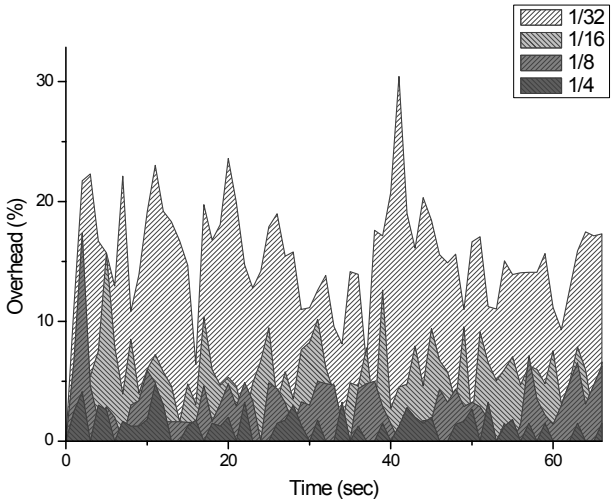

그림 4. 측정 주기에 따른 시스템 오버헤드 비교
Fig. 4. Overhead comparison according to sampling period.

표 1. 측정 주기에 따른 평균 시스템 오버헤드
Table 1. Average overhead according to sampling interval.

Interval (sec.)	1/32	1/16	1/8	1/4
AVG Overhead (%)	15.43	5.84	3.19	1.19

어를 갖고 있는 시스템에서 4개의 스레드로 동작하는 어플리케이션을 사용하여 성능 측정 주기에 따른 오버헤드를 측정하였다. 시스템의 코어 수가 측정하는 어플리케이션의 필요 스레드 수보다 크므로 오버헤드 분석을 위해 추가된 스레드는 시스템 성능에 영향을 주지 않는다. 그림 4는 다양한 측정 주기로 시스템 성능을 측정하였을 때의 오버헤드 변화를 보여준다.

그림 4에서 초당 32번 측정할 때를 제외한 나머지 오버헤드는 측정 주기에 비례한다. 1/32 초의 주기로 측정할 때, 오버헤드는 폭발적으로 증가함을 알 수 있다. 다음 표 1은 측정 주기에 따른 평균 오버헤드를 보여준다. 그림 4와 표 1의 결과로부터 시스템의 성능에 크게 무리를 주지 않는 적절한 측정 주기를 선택할 수 있다.

4.1.2 측정값의 정확성

시스템 성능 측정에 있어서 측정값의 정확성은 가장 중요한 요소이다. 이론적으로는 시스템의 부하가 바뀔 때 마다 측정하는 것이 가장 정확하겠지만, 앞서 그림 4에서 본 바와 같이 이는 오버헤드를 증가시키기 때문에 허용 가능한 오차범위 내에서 정확성을 보장하는 적절한 측정 주기를 찾아야 한다. 측정값의 정확성을 시험해 보기 위해서 시스템 성능지표 중에서 변화량이 가장 심한 CPU의 전체 사용률을 측정하고 그 결과를 비교하였다. 그림 5는 측정 주기에 따른 시스템 성능 측정값을 보여준다. 측정 주기가 커질수록 측정값의 분포가 단순해지는 것을 확인 할 수 있다. 그림 5의 (a)는 1/32 초 주기로 측정한 결과이며 가장 정확한 측정값을 갖는다. 또한 1/32초 이하의 측정 주기를 가지더라도 측정값의 정확도가 더 이상 증가하지 않음을 확인하였다. 따라서 본 논문에서는 1/32 주기로 측정한 값을 기준으로 각각의 측정 주기 별로 오차율을 비교하였다. 표 2는 각 주기별 측정값의 평균 오차율을 보여준다. 그림 5와 표 2로부터 측정값의 정확성을 고려하여 적정 주기를 선택하는 것이 가능하다.

본 논문에서는 표 1, 2로부터 오버헤드와 측정값의

표 2. 측정 주기에 따른 오차율
Table 2. Error rate according to sampling period.

Interval (sec.)	1/16	1/8	1/4
Error Rate (%)	6.36	9.02	10.24

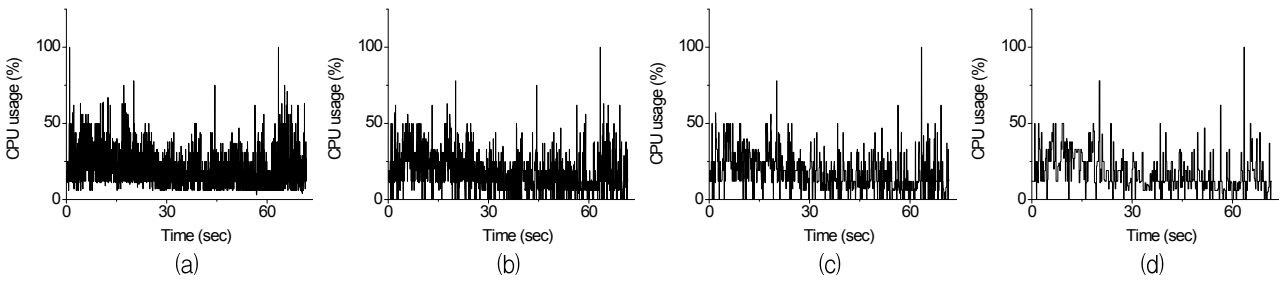

그림 5. 측정 주기에 따른 CPU 사용량
(a) 1/32 초, (b) 1/16 초, (c) 1/8 초, (d) 1/4 초
Fig. 5. Measured total CPU usage according to sampling period.
(a) 1/32 sec, (b) 1/16 sec, (c) 1/8 sec, (d) 1/4 sec

정확성을 고려하여 시스템 성능 측정 주기로 1/16 초를 설정하였다.

4.2 성능모니터 적용

이번 절에서는 제안된 성능모니터를 적용한 예를 보여준다. 본 논문에서는 성능모니터를 적용할 어플리케이션으로 2006년 GDC에서 멀티코어 플랫폼을 위한 시범콘텐츠로 인텔에서 선보였던 DtC (Destroy the Castle)[14]을 사용하였다. DtC는 복잡한 연산을 수행하는 FPS (First Person Shooter) 게임으로 주 스레드와 물리, AI, 파편 연산 스레드의 총 4개의 스레드로 실행된다.

그림 6은 본 논문에서 제안한 성능모니터의 적용 예를 보여준다. 그림 6의 (a)로부터 메모리 사용량에 비해 CPU 사용량이 좀 더 많은 변동이 있는 것을 확인 할 수 있다. 이는 메모리는 프로그램이 시작할 때 할당되는 것과 달리 CPU 사용률은 실시간으로 변하기 때문이다. 그림 6의 (b)와 (c)는 코어별, 스레드별 프로세서 사용량을 보여준다. 그림 6의 (b)와 (c)를 통해 DtC의 네 개의 스레드가 각기 다른 네 개의 코어에 할당되어 작동하는 것을 확인 할 수 있으며 그 중 부하가 심한 코어 또는 스레드를 구분 할 수 있다. 그림 6의 (d)는 현재 시스템의 네트워크 사용량을 보여준다. 네트워크 사용량은 초당 송/수신 하는 IP 데이터그램의 크기로 보여준다.

본 논문에서는 RMPM의 정확성을 증명하기 위해 다양한 어플리케이션의 평균 프로세서 사용율을 RMPM과 인텔의 VTune을 이용하여 측정하고 비교하였다. 인텔의 VTune은 프로세서 안에 디버깅 목적으로 포함되어있는 레지스터를 이용하기 때문에 인텔 프로세서를 사용한 시스템에서 매우 정확한 측정값을 구할 수 있다. 표 3은 RMPM과 VTune을 각각 적용했을 때 평균 프로세서 사용율의 비교를 보여준다.

표 3에서 DtC를 제외한 어플리케이션은 DirectX 9.0

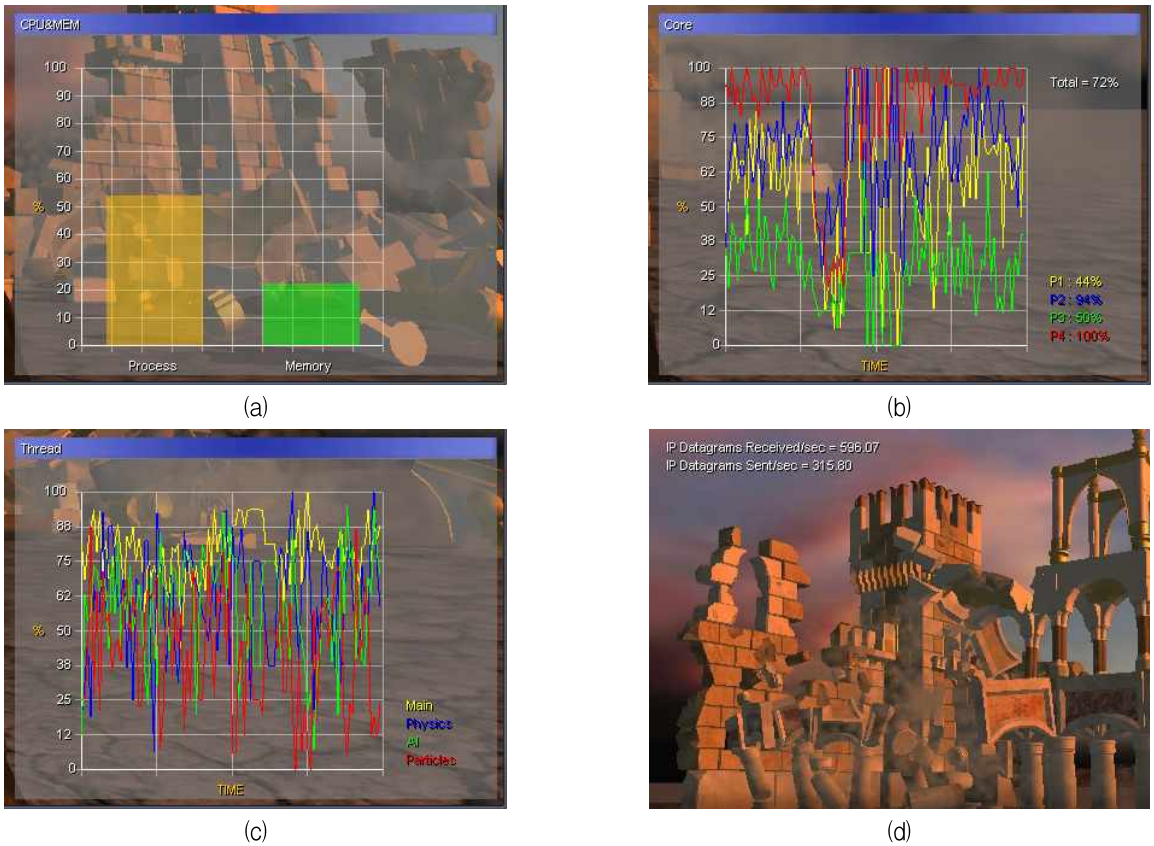

그림 6. 제안된 성능모니터 적용 예
(a) CPU 와 Memory 사용량, (b) 코어별 사용량, (c) 스레드별 사용량, (d) 네트워크 사용량

Fig. 6. Application of proposed performance monitor.
(a) CPU and Memory usage, (b) Core usage, (c) Thread usage, (d) Network usage

표 3. RMPM과 VTune 측정결과 비교
Table 3. Comparison of measure result between RMPM with VTune.

Application	RMPM	VTune
DtC	16.500 %	16.479 %
BasicHLSL	5.875 %	5.875 %
HDRLighting	5.250 %	5.249 %
HDRPipeline	5.125 %	5.125 %
MultiAnimation	7.750 %	7.750 %

SDK에 포함된 예제 어플리케이션이다. VTune의 측정 주기는 RMPM의 측정주기와 같은 1/16 초로 설정하였다. DirectX 9.0 SDK 예제 어플리케이션은 비교적 DtC 보다 낮은 프로세서 사용율을 갖는 것을 확인 할 수 있다. 이는 DtC가 복잡한 물리연산 및 인공지능 연산을 수행하기 때문인 것으로 보인다.

표 3에서 각각의 어플리케이션을 RMPM과 VTune으로 측정하였을 때 평균 프로세서 사용율의 오차가 최대 0.021인 것을 확인할 수 있으며 BasicHLSL등과 같은 경우 측정값이 같게 나오는 것을 확인 할 수 있다. 프로세서에 내장된 정보를 이용한 VTune과 비교하였을 때 RMPM은 매우 정확한 측정값을 갖는 것을 확인할 수 있다.

V. 결 론

본 논문에서는 최근 보편화되고 있는 멀티코어 프레임워크의 최적화를 위한 성능모니터를 설계하고 구현하였다. 본 논문에서 제안한 성능모니터는 기존의 성능모니터들이 지원하지 못했던 실시간성과 사용자 편의성을 지원한다. 또한 멀티코어 시스템 환경을 고려하여 코어별 스레드별 분석을 가능하게 하여 멀티 쓰레딩 지원 어플리케이션 개발의 효율성을 증대하였다. 제안하는 모니터링 기술을 적용한 멀티미디어 콘텐츠는 시스템의 전체 부하상태는 물론 코어간의 부하분산 및 어플리케이션 내의 스레드간 부하 분산 상태를 실시간으로 보여주기 때문에 사용 중인 시스템은 물론 해당 어플리케이션의 성능을 실시간으로 확인할 수 있다.

또한 성능 측정 주기의 변화로 인해 발생하는 오버헤드와 측정값의 정확도 사이의 관계를 제안한 성능모니터를 통하여 확인하였으며 오버헤드(5.84%)와 측정값의 정확성(6.36%)을 고려하여 최적의 측정 주기(1/16초)를 결정하였다.

참 고 문 헌

[1] D. Callahan, “Paradigm Shift: Design considerations for parallel programming,” *MSDN Magazine*, Microsoft, Oct. 2008.

[2] “Measuring Application Performance on Multi-core Hardware,” http://software.intel.com/en-us/articles/measuring-application-performance-on-multi-core-hardware, Feb. 2009.

[3] Training materials, “Performance Analysis Tools and Topics for LC'S IBM ASC Systems-Performance Analysis Tools,“ Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/performance_tools/

[4] J. Reinders, “Intel Threading Building Blocks,” O'Reilly Media, 2007.

[5] Reza Azimi, David K. Tam, Livio Soares, and Michael Stumm, “Enhancing operating system support for multicore processors by using hardware performance monitoring,” SIGOPS Oper. Syst. Vol.. 43, No. 2, pp. 56-65, Apr. 2009.

[6] Brinkley Sprunt, “2002. Pentium 4 Performance-Monitoring Features,” IEEE Micro Vol. 22, No. 4, pp. 72-82 Jul. 2002.

[7] M. Prvulovic and J. Torrellas. Reenact, “Using thread-level speculation mechanisms to debug data races in multithreaded codes,” In Proc. IEEE/ACM International Symposium on Computer Architecture, pp. 110 - 121, Jun. 2003.

[8] Lin, K, Liao, S., “Service Monitoring and Management on Multicore Platforms,” *In Proc. of the IEEE int'l Conf. on E-Business Engineering*, Oct. 2006, pp.623-630

[9] G. S. Almasi and A. Gottlieb, “Highly Parallel Computing 2ed,” *Benjamin-Cummings publishers*, 1994.

[10] Moore, S., Cronk, D., London, K. S., and Dongarra, J. “Review of Performance Analysis Tools for MPI Parallel Programs,” In Proc. of the 8th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing interface, Sep 2001, pp.241-248

[11] Intel Inc, “Intel VTune Performance Analyzers,” http://www.intel.com/software/products/vtune/

[12] Mark Friedman and Odysseas Pentakalos, “Windows 2000 Performance Guide,” O'REILLY, 2002.

[13] Kwang Sik Shin, Jin Ha Jung, Jin Young Cheon, and Sang Bang Choi, “Real-time network monitoring scheme based on SNMP for dynamic information,” Journal of Network and Computer Applications, vol. 30, issue 1, pp. 331-353, Jan. 2007.

[14] Intel Inc, “Destroy the Castle,” http://software.intel.com/en-us/articles/destroy-the-castle-demo/

저 자 소 개

김 진 혁(학생회원)

2009년 인하대학교 전자공학과 학사 졸업.
2011년 인하대학교 전자공학과 석사 졸업.
2011년～현재 인하대학교 전자공학과 박사과정

<주관심분야 : 멀티미디어 통신, 무선 통신, 컴퓨터 네트워크, 병렬 및 분산 컴퓨팅>

윤 완 오(학생회원)

2000년 경기대학교 전자공학과 학사 졸업.
2002년 인하대학교 전자공학과 석사 졸업.
2010년 인하대학교 전자공학과 박사 졸업.
2010년～현재 인하대학교 전자공학과 연구교수

<주관심분야 : 분산 처리 시스템, 병렬프로그램밍, 컴퓨터 구조, 무선 통신, 컴퓨터 네트워크>

최 상 방(평생회원)

1981년 한양대학교 전자공학과 학사 졸업.
1981년～1986년 LG 정보통신(주).
1988년 University of washinton 석사 졸업.
1990년 University of washinton 박사 졸업.
1991년～현재 인하대학교 전자공학과 교수

<주관심분야 : 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템>

신 광 식(정회원)

2001년 인하대학교 전자공학과 학사 졸업.
2003년 인하대학교 전자공학과 석사 졸업.
2008년 인하대학교 전자공학과 박사 졸업.
2008년～2010년 9월 한국전자통신연구원 HD 게임연구팀 Post doc.
2010년 9월～현재 한국전자통신연구원 HD 게임연구팀 선임연구원

<주관심분야 : 멀티미디어 통신, 무선 통신, 컴퓨터 네트워크, 병렬 및 분산 컴퓨팅>

이 창 호(학생회원)

2008년 청주대학교 전자공학과 학사 졸업.
2010년 인하대학교 전자공학과 석사졸업.
2010년～현재 인하대학교 전자공학과 박사과정

<주관심분야 : 컴퓨터 구조, 컴퓨터 네트워크, 무선 통신, 병렬 및 분산 처리 시스템>