RPSim: Manycore 를 위한 범용 실시간 성능 시뮬레이터

정병관*, 이선우*, 김지민*, 유민수**
*한양대학교 전자컴퓨터통신공학과
**한양대학교 정보통신대학
e-mail : {bkjung, swlee, jmkim, msryu}@rtcc.hanyang.ac.kr

RPSim: A Generic Real-Time Performance Simulator for Manycore

Byung Kwan Jung*, Sunwoo Lee*, Jimin Kim*, Minsoo Ryu**
*Dept. of Electronics and Computer Engineering, Hanyang University
**College of Information and Communications, Hanyang University

요 약

실시간 시스템 개발에 있어서 태스크들의 응답시간을 예측하는 것은 가장 중요한 문제로 인식되고 있다. 그러나 manycore 환경에서는 응답시간을 예측하는 것이 몹시 어려워 만족할 만한 결과를 이끌어내지 못하고 있다. 과거에 스케줄링과 동기화 정책을 고려하여 최악응답시간을 예측하는 방법이 제시되기도 했지만, 상당히 제한적인 태스크 모델을 가정하여 실제로 적용하기에는 어려울 뿐만 아니라 예측한 결과도 시스템의 정확한 응답시간과 상당한 괴리가 있다. 반면, 시뮬레이션 기법은 시스템의 스케줄링 상태를 시뮬레이션해 봄으로써, 상대적으로 정확한 응답시간을 예측하는 것을 가능하게 한다. 따라서 본 논문에서는 범용적이면서도 매우 효과적인 manycore 를 위한 시뮬레이션 기법을 제안한다. 제안하는 기법의 우수성은 시스템 모델의 변화에 따라 소요되는 시뮬레이션 시간을 측정하는 실험을 통해서 확인한다.

1. 서론

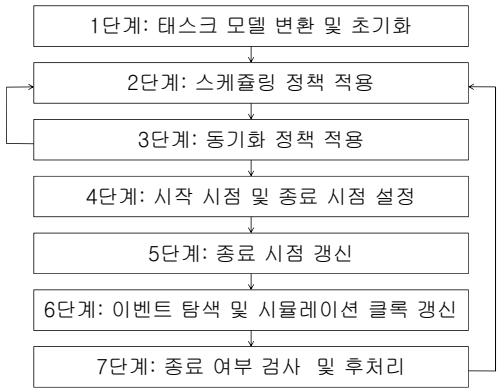
실시간 시스템 개발에 있어서 태스크들의 응답시간을 미리 계산하는 것은 가장 중요한 문제로 인식되고 있다. 이러한 문제를 해결하기 위해 과거 30 여 년간 많은 연구가 수행되었으며, Liu 와 Layland 의 Rate Monotonic Analysis[1], Joseph 과 Pandya 의 Response Time Analysis[2], 멀티프로세서 환경을 고려한 Jun Sun 의 [3]과 Bertogna 등의 [4], 그리고 시뮬레이션을 이용한 FOTISSIMO[5], Cheddar[6] 등은 주목할만한 결과로 꼽을 수 있다. RTA 는 정적우선순위 스케줄링을 독립적인 태스크들에 적용하는 경우에 최악응답시간을 정확하게 계산하는 대표적인 방법이다. 이후, [7, 8, 9]에서는 상호배제와 같은 의존관계가 있는 경우를 고려하여 PIP(priority Inheritance Protocol), PCP(Priority Ceiling Protocol), SRP(Stack Resource Policy) 등을 전제로 최악응답시간을 구하는 방법을 제안하기도 하였다. 또한 FOTISSIMO[5], Cheddar[6] 등은 스케줄링을 시뮬레이션하여 응답시간을 예측 가능하게 하였다. 한편 Design Automation 분야에서는 타깃 소프트웨어와 하드웨어 설계물을 직접 이용하여 명령어 수준 또는 싸이클 수준의 정밀한 시뮬레이션 기법을 사용하기도 한다.

위에서 언급한 방법들은 상당히 제한적인 태스크 모델을 가정하여 실제로 적용하기에는 어려운 한계점을 가진다. 상당수의 실시간 시스템들은 다수의 태스크들이 서로 협력적으로 실행하면서 통신 및 선후행 관계를 가지는 것이 일반적이나, 과거의 방법에서는 이러한 의존관계를 고려하지 않는다. 또한 스케줄링 정책에 있어서도 단일 프로세서 상에서 우선순위 기반의 알고리즘 부류만을 고려하는 것이 대부분이다. 하지만, 최근 멀티프로세서 기술이 발전하면서 PFair 와 같이 전혀 새로운 부류의 스케줄링 알고리즘이 등장하였으며, 멀티프로세서 환경을 고려한 다양한 동기화 방법들이 새롭게 등장함에 따라 기존의 방법을 적용하는 것은 거의 불가능한 상황이다.

또한 응답시간 예측 관점에서 본다면 RTA 를 포함한 [7, 8, 9]들은 최악의 경우를 고려한 응답시간만을 계산하기 때문에 실제 시스템의 정확한 응답시간과는 상당한 괴리가 발생하는 문제점이 있다. 하지만 시스템의 스케줄링 상태를 시뮬레이션 한다면 상대적으로 정확한 응답시간을 예측하는 것이 가능해 진다. 그러나 하드웨어 명령어 수준 또는 싸이클 수준의 시뮬레이션 기법은 상당한 비용과 시간을 요구하여 실시간 시스템의 응답시간만을 분석하는 용도로 사용하기에는 부적합하다.

본 논문에서는 범용적이면서도 다양한 태스크 모델이 적용 가능한 매우 효과적인 manycore 를 위한 실시간 성능 시뮬레이터 RPSim 을 제안한다. 제안하는 RPSim 은 임의의 의존관계를 가지는 태스크 모델은 물론 현재까지 알려진 다양한 스케줄링 알고리즘과 다양한 동기화 정책을 모두 지원할 수 있는 개방적 프레임워크를 가지고 있다. RPSim 은 (그림 1)과 같이

7 단계의 시뮬레이션 과정을 수행하며, 각각의 단계는 약속된 인터페이스를 통해 상호간의 의존성이 최소화된 컴포넌트로 구현되어 있다. 따라서 태스크 모델, 스케줄링 정책, 동기화 정책이 변경될 경우 해당 컴포넌트만을 수정하거나 교체하는 것으로 충분하다. 이러한 범용성에도 불구하고 RPSim 은 매우 효율적으로 시뮬레이션 하도록 설계되었다. 우선 태스크간의 다양한 의존관계를 단순한 선후행관계로 변환하여 동기화 및 통신행위를 간단하게 시뮬레이션하는 것이 가능하도록 하였다. 또한, 기존의 시뮬레이션 방법과는 전혀 다른 방법으로 태스크들의 시작시점과 종료시점을 계산하여 시뮬레이터 구현의 복잡도를 크게 줄였을 뿐만 아니라 시뮬레이션에 소요되는 시간도 상당히 개선하였다.



(그림 1) RPSim 의 시뮬레이션 단계

RPSim 의 또 다른 특징은 태스크의 내부 행위 모델을 분석하여 동기화와 통신에 의한 블로킹 시간을 정확하게 고려할 수 있다는 점이다. 전통적인 방법에서는 대부분 최소한의 태스크 속성만을 가지고 응답시간을 계산함에 따라 블로킹의 발생시점과 그 크기를 정확하게 파악할 수 없었다. 예를 들면 [8]에서는 태스크가 접근하는 세마포어와 임계영역에서 소비하는 시간만을 고려하여 최악의 블로킹 시간만을 계산하고 있다. 하지만, RPSim 에서는 태스크 속성과 함께 그 내부의 상세한 행위모델을 가정하여, 이로부터 블로킹이 발생하는 정확한 시점과 크기를 파악하는 것이 가능하다.

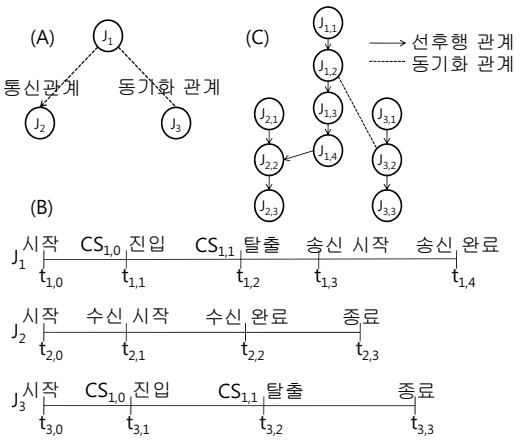
본 논문은 다음과 같이 구성된다. 2 장에서는 RPSim 에서 사용하는 시뮬레이션 기법을 설명하고 3 장에서는 RPSim 의 구조를 알아본다. 그리고 4 장에서는 실험을 통해 제시하는 시뮬레이션 기법의 효율성을 보인다. 마지막으로 5 장에서는 앞으로의 계획을 언급하고 논문을 결론짓는다.

2. RPSim 의 시뮬레이션 기법

Manycore 를 위한 범용 실시간 성능 시뮬레이터 RPSim 은 태스크의 응답시간을 예측하기 위하여 독립적인 7 단계로 구성된 시뮬레이션 알고리즘을 사용한다. 제안하는 시뮬레이션 알고리즘은 종전과는 비교되는 두 가지의 특징이 있다. 첫 번째는 시뮬레이션에 입력되는 태스크 모델을 서브태스크로 분할하는 것이다. 이것은 태스크 응답시간을 산출하는데 있어 블로킹시간 계산을 쉽고 정확하게 한다. 두 번째는 태스크의 종료시점을 우선적으로 설정한 후 갱신하는 방식이다. 이러한 기법은 시뮬레이션에 소요되는 시간을 상당히 감소시키는 효과가 있다.

본 논문에서 제안하는 시뮬레이션 알고리즘의 각 단계를 좀 더 상세히 살펴보면 다음과 같다.

1) 태스크 모델 변환 및 초기화 단계: RPSim 은 일반적인 의존관계의 태스크에서 태스크 내부의 행위 모델을 가정하여 각 태스크를 서브태스크로 분할한다. 예를 들어 (그림 2)(A)에서 통신과 동기화 관계에 있는 J_1, J_2, J_3 의 내부 행위 모델을 (그림 2) (B)와 같은 타이밍도로 표현 가능하다면, (그림 2)(C)와 같이 태스크 J_i 를 $t_{n,m}$, $t_{n,m+1}$ 단위의 서브태스크로 분할하고 각각을 선후행 관계로 연결한다. 단, 동기화 관계는 서브태스크간에 점선으로 표시하고 통신관계는 서브태스크간의 선후행관계로 변환한다.



(그림 2) 태스크 모델 변환

2) 스케줄링 정책 적용 단계: 시스템의 스케줄링 정책이 적용되는 단계로 서브태스크들을 프로세서에 배정하고, 해당 프로세서에서 실행할 서브태스크를 선정하며, 선정된 서브태스크들의 상태를 RUNNING 으로 변경한다. 이때, 선점이 발생하여 서브태스크의 상태가 RUNNING 에서 READY 로 변경되는 경우는 현재의 시뮬레이션 클록(clock)의 값을 기록한다.

$$clock\_preempt_{i,j} = clock$$

3) 동기화 정책 적용 단계: 시스템의 공유자원 동기화 정책이 적용되는 단계로 RUNNING 상태의 서브태스크들 중 태스크 종류가 동기화인 경우는 정책에 따라 실행 또는 블로킹 여부를 결정한다. 만약 블로킹으로 결정된 서브태스크가 존재한다면 2 단계로 복귀하여 스케줄링 정책을 재적용한다.

4) 시작 시점 및 종료 시점 설정 단계: 서브태스크의 상태가 최초로 RUNNING 이 된 경우 그 시작 시점($S_{i,j}$)을 시뮬레이션 클록(clock)의 현재 값으로 설정하고 종료 시점을 시작 시점과 실행 요구 시간($E_{i,j}$)의 합으로 설정한다.

$$S_{i,j} = clock,\ F_{i,j} = S_{i,j} + E_{i,j}$$

5) 종료 시점 갱신 단계: 현재 RUNNING 상태인 서브태스크들 중 $clock\_preempt_{i,j}$ 의 값이 0 이 아닌 서브태스크들에 대해 과거에 선점 당했던 시점으로부터 현 시점까지 경과된 시간을 종료시점에 추가한다.

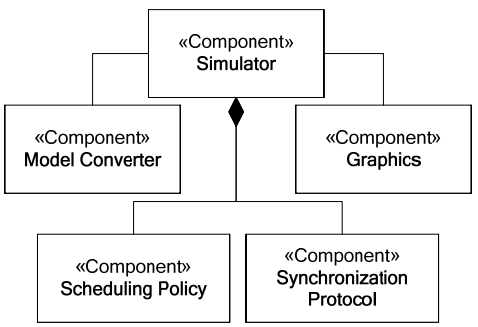
$$F_{i,j} = F_{i,j} + (clock - clock\_preempt_{i,j})$$

6) 이벤트 탐색 및 시뮬레이션 클록 갱신 단계: 새로운 서브태스크들의 가장 이른 릴리스시점($\min\{R_{a,b}\}$), 퀀텀 만료시점(t_q), 그리고 RUNNING 중인 서브태스크의 가장 이른 종료시점($\min\{F_{x,y}\}$) 가운데 최소값으로 시뮬레이션 클록을 갱신한다.

$$clock = min\{min\{R_{a,b}\},\ t_q,\ min\{F_{x,y}\}\}$$

7) 종료 여부 검사 및 후처리 단계: 6 단계에서 시뮬레이션 클록이 서브태스크의 종료시점으로 갱신된 경우에는 해당 서브태스크의 상태를 TERMINATED 로 변경한다. 그리고 해당 서브태스크를 제외한 종료된 서브태스크의 후행서브태스크의 모든 선행태스크가 TERMINATED 인지 확인한 후에, 종료된 서브태스크의 후행서브태스크들의 상태를 READY 로 변경해준다. 그리고 시뮬레이션 알고리즘의 종료여부는 판별하게 된다. 만약 미리 설정한 종료시점에 도달하였거나 모든 태스크가 완전종료상태라면 알고리즘은 종료하고, 그렇지 않다면 2 단계로 돌아가 알고리즘은 종료 시까지 반복 수행한다.

3. RPSim 의 구조



(그림 3) RPSim 컴포넌트 다이어그램

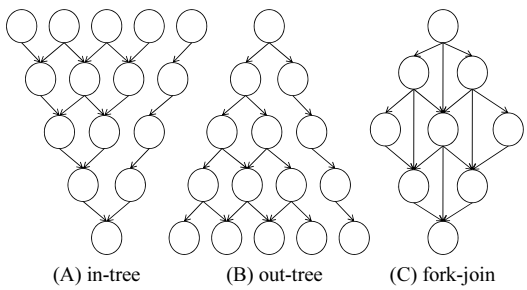
RPSim 은 다양한 스케줄링 정책과 동기화 정책을 적용할 수 있는 범용적인 특성과 단순한 구조를 위해 주요 기능들을 컴포넌트화 했다. (그림 3)은 RPSim 소프트웨어의 컴포넌트 다이어그램을 도시한 그림으로, Model Converter, Simulator, Graphics, Scheduling Policy 와 Synchronization Protocol 등 총 다섯 개의 컴포넌트가 있다. 특히, 시뮬레이션 알고리즘 2, 3 단계에 해당하는 스케줄링과 동기화 정책은 시뮬레이터 부분에서 별도의 컴포넌트로 구성되어있기 때문에 다른 단계와 독립적이다. 따라서 다양한 스케줄링과 동기화 정책을 시험해 보는 것이 각각의 컴포넌트 교체만으로 충분하다.

4. 실험

본 장에서는 RPSim 에서 사용하는 시뮬레이션 알고리즘의 효율성을 실험을 통해서 보인다.

4.1 실험환경

본 논문에서 제안하는 시뮬레이션 알고리즘의 실험을 위하여 (그림 4)와 같이 3 가지 태스크 그래프 유형[10]을 고려하였다. 태스크는 (그림 4)(A), (B), (C) 모델의 조합으로 80 개를 생성하였으며, 각 태스크의 주기는 1000 으로 설정하였다. 또한, 태스크의 실행요구시간은 10~50 사이의 값을 무작위로 선정하였으며 10,000 번 반복하도록 설정하였다. 스케줄링 정책은 partitioned fixed priority 로 설정하였으며 동기화 정책은 실험에서 고려치 않았다. 또한, 본 논문에서 제안하는 방법과는 달리 실행요구시간을 시스템 클록이 증가한 만큼 감소시키는 방법(Traditional Approach)을 구현하여 실험을 통해서 알고리즘의 복잡도를 비교하였다.

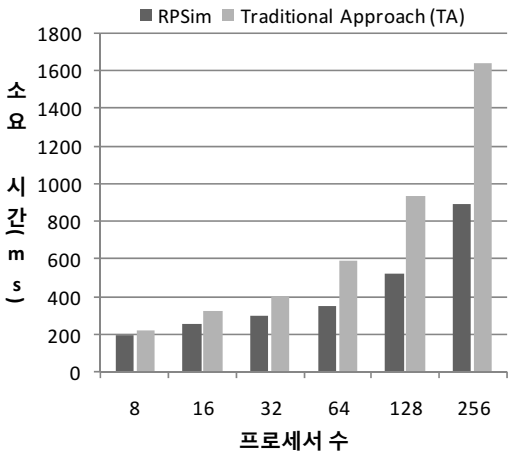


(그림 4) 태스크 그래프 유형

4.2 실험결과

실험은 시뮬레이션의 복잡도의 변화에 따른 알고리즘의 성능 변화를 보기 위해 프로세서 수를 늘려가며 진행하였다. (그림 5)는 그 실험의 결과인데 프로세서 수가 증가할수록 알고리즘의 성능향상 비율이 증가함을 알 수 있다. 특히, 프로세서 수가 256 개 일 때는 약 80% 이상의 성능향상 비율을 보인다. 그 이유는 Traditional Approach 에서는 시스템 클록이 증가할 때

마다 프로세서에 할당된 모든 태스크의 실행요구시간을 감소시키는 연산이 필요한 반면, RPSim 시뮬레이션 알고리즘에서는 자신의 종료시점이 갱신되는 시점에서만 연산이 필요하기 때문이다. 즉, 종료시점은 태스크가 READY 에서 RUNNING 상태로 변경된 경우에만 갱신되므로 5 단계의 연산은 모든 프로세서에 할당된 태스크에서 필요한 것이 아니라, 선점되었던 태스크만이 필요하다. 따라서 프로세서 수가 늘어 날수록, 태스크가 선점하는 횟수가 적을수록 본 논문에서 제안하는 시뮬레이션 알고리즘은 효과적이다.



(그림 5) 프로세서 수에 따른 RPSim 과 TA 의 성능비교

5. 결론 및 향후 계획

본 논문에서는 manycore 시스템에서 태스크 응답시간 산출을 위한 시뮬레이션 기법을 제시하였다. 제시하는 기법은 범용적일 뿐만 아니라 시뮬레이션에 소요되는 시간도 상당히 개선하였다.

향후에는 시뮬레이션을 통해 다양한 실험을 할 수 있도록 사용자 수준에서 스케줄링과 동기화 정책을 작성할 수 있도록 RPSim 을 확장해 나갈 것이다. 또한, 알고리즘이 더욱 효율적으로 동작하도록 개선할 것이다.

참고문헌

[1] C. Liu and J. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment,” Journal of the ACM, 20(1):46-61, Jan. 1973.

[2] M. Joseph and P. Pandya, “Finding Response Times in a Real-Time System,” The Computer Journal, Vol. 29, No. 5, 1986.

[3] Jun Sun and Jane W. S. Liu, “Bounding the End-to-End Response Time in Multiprocessor Real-Time Systems,” Parallel and Distributed Real-Time Systems, pp. 91-98, April 1995.

[4] M. Bertogna and M. Cirinei, “Response-Time Analysis for globally scheduled Symmetric Multiprocessor Platforms,” IEEE Real-Time Systems Symp., pp. 146-160, Dec. 2007.

[5] T. Kramp, M. Adrian and R. Koster, “An Open Framework for Real-Time Scheduling Simulation,” IPDPS Workshops, 2000.

[6] F, Singhoff, J. legrand, L. Nana and L. Marce, “Cheddar: a Flexible Real Time Scheduling Framework,” Annual International Conference on Ada (SIGAda), pp. 1-8, 2004.

[7] L. Sha, R. Rajkumar and J. Lehoczky, “Priority Inheritance Protocols: An Approach to Real-Time Synchronization,” IEEE Transactions on Software Engineering, 29(3), pp. 1175-1185, Sep. 1990.

[8] R. Rajkumar, L. Sha, and J. P. Lehoczky, “Real-time Synchronization for Multiprocessors,” IEEE Real-Time Systems Symp., pp. 259-269, Dec. 1988.

[9] T.P. Baker, “A Stack-Based Resource Allocation Policy for Realtime Processes,” IEEE Real-Time Systems Symp., pp. 191-200, Dec. 1990.

[10] Minsoo Ryu and Seongsoo Hong, "A Period Assignment Algorithm for Real-Time System Design," Lecture Notes in Computer Science, Vol. 1579, pp. 34-43, March 1999.