KIPS Tr. Comp. and Comm. Sys.
Vol.4, No.5 pp.147~152 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2015.4.5.147

Checkpoint/Resimulation Overhead Minimization with Sporadic Synchronization in Prediction-Based Parallel Logic Simulation

DoohwanKwak† · Seiyang Yang††

ABSTRACT

In general, there are two synchronization methods in parallel event-driven simulation, pessimistic approach and optimistic approach. In this paper, we propose a new approach, sporadic synchronization combining both for prediction-based parallel event-driven logic simulation. We claim this hybrid solution is pretty effective to minimize both checkpoint overhead and restart overhead, which are related problems with frequent false predictions for improving the performance of the prediction-based parallel event-driven logic simulation. The experiment has clearly shown the advantage of the proposed approach.

Keywords : Event-Driven Simulation, Parallel Simulation

간헐적 동기화를 통한 예측기반 병렬 로직 시뮬레이션에서의 체크포인트/재실행 오버헤드 최소화

곽 두 환† · 양 세 양††

요 약

일반적으로 병렬 이벤트구동 시뮬레이션의 대표적 동기화 방법으로는 비관적 동기화 방식과 낙관적 동기화 방식이 있는데, 본 논문에서는 예측기반 병렬 이벤트구동 로직 시뮬레이션에서 이 두 가지 동기화 방식들을 혼용한 간헐적 동기화를 통한 시뮬레이션 성능 향상 기법을 제시한다. 제안되는 간헐적 동기화 방식은 예측기반 병렬 이벤트구동 로직 시뮬레이션에서 자주 일어나는 틀린 예측과 연관된 체크포인트 오버헤드 및 재실행 오버헤드를 최소화할 수 있어 시뮬레이션 성능 향상에 매우 효과적인데, 이를 다양한 실제 디자인들에 적용한 실험을 통하여 확인할 수 있었다.

키워드 : 이벤트구동 시뮬레이션, 병렬 시뮬레이션

1. 서 론

병렬 이벤트구동 로직 시뮬레이션은 다중 코어 내지는 다중 마이크로프로세서에 다수의 이벤트구동 로직 시뮬레이터들을 병렬적으로 연동시켜서 시뮬레이션의 성능을 높이고자 하는 것으로 이미 오래전부터 많은 연구가 진행되어왔으며[1-5, 11-13], 최근에는 상용 로직 시뮬레이터들도 멀티코어를 통한 병렬 시뮬레이션을 지원하고 있다[6, 7]. 그러나 지금까지의 연구 결과와 상용화 시도는 그리 성공적이라 할 수 없는데, 이의 주요 원인은 병렬 로직 시뮬레이션에 동기 오버헤드와 통신 오버헤드가 매우 과도하게 존재하기 때문이다. 즉, 로직 시뮬레이션을 병렬화하면 할수록 이에 비례하여 동기 오버헤드와 통신 오버헤드가 급격하게 증가하여 병렬화를 통하여 기대하는 시뮬레이션 성능 향상은 쉽게 무력화된다[8].

최근에 이와 같은 병렬 이벤트구동 시뮬레이션의 과도한 동기 오버헤드 및 통신 오버헤드로 인한 문제점을 효과적으로 해결할 수 있는 새로운 병렬 이벤트구동 로직 시뮬레이션 기법이 제안되었다[9]. 그러나 이 예측기반의 병렬 이벤트구동 로직 시뮬레이션에서도 예측이 틀린 경우를 위하여 필요한 체크포인트와 재실행으로 인하여 시뮬레이션의 성능 향상이 제약받을 수 있다는 문제점이 존재한다. 본 논문에

† 비 회 원 : 부산대학교 전자전기컴퓨터공학과 공학석사
†† 정 회 원 : 부산대학교 정보컴퓨터공학부 교수
Manuscript Received : October 22, 2014
First Revision : March 9, 2015
Accepted : March 18, 2015
* Corresponding Author : Seiyang Yang(syyang@pusan.ac.kr)

서는 예측에 기반한 새로운 병렬 이벤트구동 로직 시뮬레이션에서 성능 향상을 제약할 수 있는 체크포인트 오버헤드 및 재실행 오버헤드를 최소화시킬 수 있는 간헐적 동기화(SS; Sporadic Synchronization) 기반의 하이브리드 방식을 새롭게 제안하고, 이의 효능을 실험을 통하여 보인다. 본 논문은 2절의 배경 및 관련 연구, 본론인 3절의 간헐적 동기화를 통한 예측기반 병렬 이벤트구동 로직 시뮬레이션의 성능 향상, 4절의 실험, 마지막 5절의 결론으로 구성된다.

2. 관련 연구 및 연구 동기

2.1 관련 연구

우선 병렬 이벤트구동 로직 시뮬레이션 기술에서 사용되는 용어들을 설명한다.

- 로컬 시뮬레이션 : 병렬 이벤트구동 시뮬레이션에서 분할된 로컬 설계객체들 각각에 대한 시뮬레이션
- 로컬 시뮬레이터 : 병렬 이벤트구동 시뮬레이션에서 로컬 시뮬레이션을 수행하는 시뮬레이터
- 로컬 시뮬레이션 시간 : 병렬 시뮬레이션을 진행하는 과정에서 각 로컬 시뮬레이션들이 가지게 되는 시뮬레이션 시간

병렬 이벤트구동 로직 시뮬레이션에서 동기(synchronization)란 다수의 로컬 시뮬레이션 시간들이 존재하는 경우에서 이벤트들의 인과(casauality) 관계를 올바르게 유지하는 과정이다. 또한 통신(communication)이란 로컬 시뮬레이션들 사이에 로컬 설계객체들에 의하여 생성되는 이벤트를 실제적으로 주고받는 과정이다. 병렬 시뮬레이션 과정에서 극히 빈번하게 일어나는 동기 및 통신의 양은 전체 디자인을 어떻게 분할하였는지에 크게 좌우된다. 병렬 이벤트구동 시뮬레이션에서 동기방식은 크게 비관적(pessimistic) 방식과 낙관적(optimistic) 방식으로 나누어진다[1-3]. 비관적 동기방식에서는 로컬 시뮬레이션들 간에 절대 인과관계 위반이 일어나지 않도록 모든 로컬 시뮬레이션 시간들을 항상 일치시키면서 시뮬레이션을 진행하게 되므로, 이를 록-스텝(lock-step) 방식이라고도 한다. 이와 같은 비관적 동기방식의 문제점으로는 모든 로컬 시뮬레이션 시간들을 록-스텝으로 항상 일치시키기 위하여 과도한 동기오버헤드가 발생하게 되고 이로 인하여 병렬 시뮬레이션의 성능이 크게 제약되는 것이다. 반면, 낙관적 동기방식에서는 우선 각 로컬 시뮬레이션이 자신의 로컬 시뮬레이션 시간에 맞추어 독립적으로 진행될 수 있도록 허용함으로써 로컬 시뮬레이션들 간에 인과관계 위반이 일시적으로 일어나게 된다. 따라서 낙관적 동기방식에서는 인과관계 위반이 일어난 경우에 이를 바로잡아주는 메커니즘이 필요한데, 이는 주기적인 체크포인트(checkpoint) 생성과 롤백(rollback)을 통하여 가능하다[3]. 즉, 인과관계 위반이 일어난 로컬 시뮬레이션은 다른 로컬 시뮬레이션들에 위반사실과 위반이 일어난 시뮬레이션 시간을 통보하여야 하며 이 경우에 모든 로컬 시뮬레이션들은 위반이 일어난 시뮬레이션 시간에서 제일 가까운 체크포인트로 롤백 하여 이 시점에서부터 인과관계 위반을 바로잡는 시뮬레이션을 재실행(resimulation)하게 된다. 체크포인트를 위해서는 시뮬레이션 프로그램의 실행이미지 전체를 저장하거나 디자인 이미지 전체 내지는 일부를 저장하여야 하므로, 주기적인 체크포인트는 보조기억장치에 저장하게 되며, 따라서 디자인이 큰 경우(최근의 디자인들의 시뮬레이션 프로그램 실행이미지는 수십GB 규모임)에는 시뮬레이션의 성능을 크게 저하시키는 별도의 문제점을 초래한다. 뿐만 아니라 낙관적 동기방식에서 인과관계 위반이 빈번하게 발생하게 되면 빈번한 롤백 후 재실행에 의한 시뮬레이션 시간이 추가되므로 시뮬레이션의 성능은 더욱 저하되게 된다.

2.2 연구 동기

최근에 이와 같은 병렬 이벤트구동 시뮬레이션의 문제점인 과도한 동기 오버헤드 및 통신 오버헤드로 인한 병렬 시뮬레이션 성능 제약을 효과적으로 해결할 수 있는 새로운 예측기반의 병렬 이벤트구동 로직 시뮬레이션 기법이 새롭게 제안되었는데, 제안된 예측기반 병렬 이벤트구동 로직 시뮬레이션은 예상입출력 이용-런 모드와 실제입출력 이용-런 모드의 두 가지 실행모드가 번갈아가면서 다음과 같이 실행된다[9]. 예상입출력 이용-런 모드에서는 로컬 설계객체들 각각이 예상입력을 활용하여서 각 로컬 시뮬레이션을 독립적으로 실행시켜서 각 로컬 설계객체들의 출력에서 실제 출력들을 구하게 되고, 동시에 이들 실제 출력값들 각각을 예상출력들과 시뮬레이션 진행과정 중에 실시간으로 비교하면서 시뮬레이션을 진행한다. 그리고 실제출력과 예상출력의 비교가 일치하게 된다면 로컬 시뮬레이터들 간에 통신 및 동기화를 완전히 생략하고 각 로컬 시뮬레이션을 독립적으로 진행한다. 그러나 만일 실제출력을 예상출력과 비교하여 이들이 일치하지 않으면(이 시뮬레이션 시점을 예상출력/실제출력 불일치시점이라 칭함), 이 예상출력/실제출력 불일치시점에서부터는 모든 로컬 시뮬레이터들 간에 통신 및 동기화과정을 수행하면서 실제입력을 활용하는 실제입출력 이용-런 모드로 분산병렬 시뮬레이션을 진행하게 된다. 즉 이 시뮬레이션 시점에서부터는 로컬 시뮬레이터들 간에 통신 및 동기화과정을 수행하게 되는 실제입출력 이용-런 모드로 실행모드가 전환된다. 그런데 특정한 1 이상의 로컬 시뮬레이션이 상기 예상출력/실제출력 불일치를 일으킨 로컬 시뮬레이션보다 빠르게 진행되어서 이들의 로컬 시뮬레이션 시간이 상기 예상출력/실제출력 불일치 시점을 이미 지나쳐버린 경우에서는, 이들의 로컬 시뮬레이션 시간을 상기예상출력/실제출력 불일치 시점으로 되돌린 후 재실행(restart after roll back) 시킬 필요성이 생긴다. 이를 위하여

[9]에서 각 로컬 시뮬레이션들은 예상입출력 이용-런 모드로 진행하면서 일정 간격마다 체크포인트를 생성한다. 그리고 예상입출력 이용-런 모드 진행 도중에 특정 로컬 시뮬레이션에서 예상출력/실제출력 불일치가 발생하고, 하나 이상의 다른 로컬 시뮬레이션의 로컬 시뮬레이션 시간이 예상출력/실제출력 불일치시점을 넘어선 상황에서는 다음과 같은 과정들을 단계적으로 진행한다. 즉, 첫 번째 단계로는 모든 로컬 시뮬레이션들이 상기예상출력/실제출력 불일치 시점에서 제일 가까운 체크포인트로 롤백을 수행하고 난 후, 두 번째 단계로는 상기예상출력/실제출력 불일치 시점까지 각 로컬 시뮬레이션들은 예상입출력 이용-런 모드로 재실행하여 모든 로컬 시뮬레이션들의 로컬 시뮬레이션 시간을 예상출력/실제출력 불일치 시점으로 맞춘 후(여기서, 상기예상출력/실제출력 불일치가 발생한 특정 로컬 시뮬레이션은 상기예상출력/실제출력 불일치 시점에서 제일 가까운 체크포인트에서부터 예상출력/실제출력 불일치 시점까지 반복되어 재실행되는 재실행 오버헤드가 반드시 존재함을 주목함), 세 번째 단계로는 이 시점부터 실제입력들을 사용하는 실제입출력 이용-런 모드로 모든 로컬 시뮬레이션들을 실행한다. 또한, 실제입출력 이용-런 모드에서 각 로컬 시뮬레이션은 다른 로컬 시뮬레이션들과 통신 및 동기화를 초래하는 시뮬레이션을 진행하면서, 동시에 실제출력과 예상출력을 실시간으로 비교한다. 그리고 이 실제출력이 예상출력과 일정횟수만큼 일치하는 상황이 모든 로컬 시뮬레이션들에서 일어나게 되면, 각 로컬 시뮬레이션들은 다시 예상입출력 이용-런 모드로 재전환하여 예상입력을 사용하여 동기 및 통신과정을 생략한 시뮬레이션을 진행하게 된다. Fig. 1은 예측기반의 병렬 이벤트구동 로직 시뮬레이션의 예상입출력 이용-런 모드 실행과 실제입출력 이용-런 모드 실행상황을 개념적으로 도식한 그림이다. 예상입출력 이용-런 모드 실행에서는 각각의 로컬 시뮬레이션이 다른 로컬 시뮬레이션들과의 동기 및 통신 없이 완전히 독립적으로 실행됨으로써 병렬화를 통한 성능 향상을 극대화할 수 있음이 Fig. 1(a)에 잘 나타나있다.

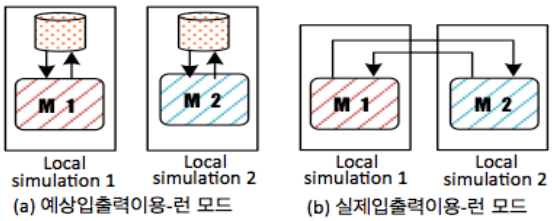

Fig. 1. Two Execution Modes of Proposed Method

이와 같이 예측기반의 병렬 이벤트구동 로직 시뮬레이션의 예측정확도가 높은 경우에서는 전체 시뮬레이션 실행구간의 대부분을 예상입출력 이용-런 모드로 진행하면서 시뮬레이션 시간의 대부분에서 동기 오버헤드와 통신 오버헤드가 완전하게 제거됨으로써 큰 폭의 시뮬레이션 성능 향상을 기대할 수 있다[9]. 그러나 예측 정확도가 떨어지는 경우에서는 예상입출력 이용-런 모드와 동기 오버헤드와 통신 오버헤드가 공히 존재하는 실제입출력 이용-런 모드로의 전환이 빈번하게 일어나게 되고, 특히 예상입출력 이용-런 모드에서 발생하는 체크포인트 오버헤드와 재실행 오버헤드로 인하여 시뮬레이션의 성능 향상에 상당한 제약이 가해지게 되는 문제점을 노출하게 된다. 이와 같은 문제점은 체크포인트가 필요한 낙관적 동기화방식을 채용한 전통적인 병렬 이벤트구동 로직 시뮬레이션에도 동일하게 존재한다.

3. 간헐적 동기화를 통한 예측기반 병렬 이벤트구동 로직 시뮬레이션의 성능 향상

낙관적 동기화방식을 적용한 기존의 병렬 이벤트구동 시뮬레이션 실행과정에 필요한 주기적 체크포인트를 위하여 보조기억장치(예, 하드디스크)를 액세스하는 과정에서 시뮬레이션의 속도가 떨어지는 것과 같은 이유로, [9]에서 제안한 예측기반의 병렬 이벤트구동 로직 시뮬레이션에서도 예상입출력 이용-런 모드로 실행되는 과정에서 체크포인트는 잦은 보조기억장치 액세스가 필요하므로 병렬 시뮬레이션의 속도 향상을 상당한 정도로 제약할 가능성이 있다(이를 본 논문에서는 체크포인트 오버헤드라 칭함). 뿐만 아니라, 예측기반 병렬 이벤트 시뮬레이션에서 낮은 예측정확도 때문에 잦은 예상출력/실제출력 불일치에 의한 재실행이 반복되면 시뮬레이션이 반복적으로 실행되는 시간구간이 증가하게 됨(이를 본 논문에서는 재실행 오버헤드라 칭함)으로써 이 또한 병렬 시뮬레이션의 속도 향상을 제약하는 요소로 작용한다. 일반적으로 예측기반 병렬 이벤트구동 로직 시뮬레이션에서 사용되는 예측데이터는 이전 시뮬레이션 실행과정에서 저장되는 로컬 설계객체들에 존재하는 입력과 출력상의 이벤트값과 해당 시뮬레이션 시간정보이다. 물론 이와 같이 확보된 예측데이터가 높은 예측정확도를 가지고 있다면 이와 같은 문제점이 생기지 않지만, 이전 시뮬레이션 실행 후에 큰 폭의 디자인 설계변경이 이루어진 경우에는 높은 예측정확도를 가지지 못할 확률이 높으며 이 경우에는 높은 빈도의 롤백 및 재실행으로 인한 시뮬레이션 속도 향상 제약이 초래될 가능성이 높아진다.

그런데 더 심각한 문제는, 이와 같은 체크포인트 오버헤드와 재실행 오버헤드가 서로 역으로 엮어져 있다는 것이다. 즉 주기적 보조기억장치 액세스를 초래하는 체크포인트 오버헤드를 줄이기 위하여 체크포인트의 빈도수를 줄이게 되면 체크포인트 오버헤드는 줄일 수 있지만, 재실행 오버헤드는 커지게 된다. 반대로 반복적으로 재실행되는 시뮬레이션의 총 시간구간을 줄이기 위해서는 체크포인트의 간격을 좁히면 되지만 (즉 체크포인트의 빈도수를 증가시키면 되지만) 이는 체크포인트 오버헤드를 증가시키게 된다. 본 논문에서는 이와 같은

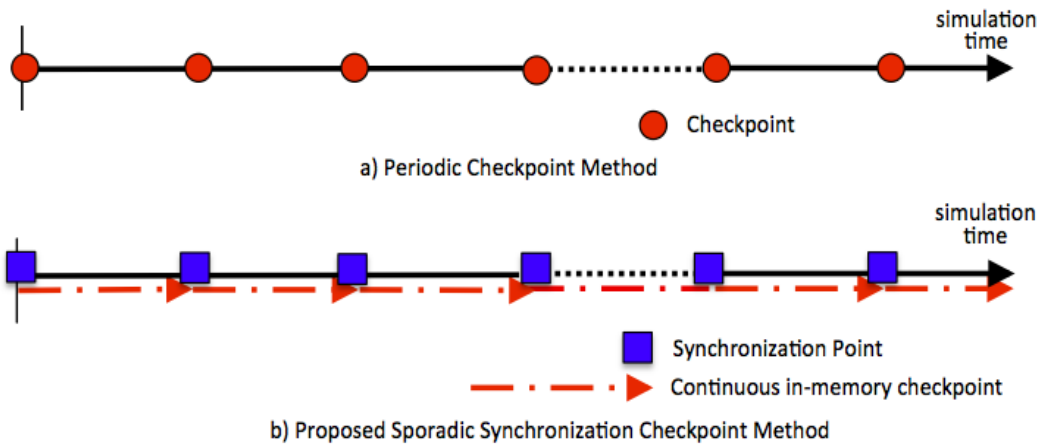

Fig. 2. Conceptual Drawing of Two Different Checkpoint Methods

체크포인트 오버헤드 및 재실행 오버헤드를 함께 최소화시킬 수 있는 하이브리드 동기화 방식을 새롭게 제안한다. 새롭게 제안되는 하이브리드 동기화 방식은 낙관적 동기화 방식과 비관적 동기화 방식을 혼용한 방식으로 임의로 정한 일정한 시간간격마다 모든 로컬 시뮬레이션들 간에 동기화를 진행동기화를 진행하는 시뮬레이션시점을 동기점(SP; Synchronization Point)이라 칭함] 하는데, 일반적으로 인접한 SP들 간의 간격은 간헐적 동기화에 의한 시뮬레이션 성능 저하가 무시될 수 있을 만큼 충분히 넓게 설정한다. 이와 같이 모든 로컬 시뮬레이션들을 위한 동기점을 시뮬레이션 시간에 간헐적으로 도입하는 것을 본 논문에서는 간헐적 동기화(SS; Sporadic Synchronization)라 칭하기로 한다.

이와 같은 간헐적 동기화를 통하여 예측기반 병렬 이벤트구동 로직 시뮬레이션에서 성능 향상을 제약할 수 있는 체크포인트 오버헤드와 재실행 오버헤드를 최소화하는 것이 가능하다. 이를 구체적으로 설명하면 다음과 같다. 임의의 동기점 SPj(SPj는 여러 동기점들 중에서 j번째 동기점을 말함)에서 모든 로컬 시뮬레이션들의 동기화가 이루어지고 나면 이후의 시뮬레이션 실행과정에서 예측 틀림에 의하여 롤백이 설혹 일어나더라도 이 롤백은 상기 임의의 동기점 SPj보다 더 이전으로는 일어날 수 없다. 따라서 임의의 동기점 SPj에서 모든 로컬 시뮬레이션들의 동기화가 이루어지고 나면 SPj 이전의 체크포인트 정보는 더 이상 필요하지 않으며, 이로부터 체크포인트를 더 이상 보조기억장치에 저장시키지 않고 주기억장치에 저장시킬 수 있으므로 체크포인트 오버헤드를 최소화시키는 것이 가능하다. 뿐만 아니라, 예측 틀림이 반복되면서 초래되는 반복적 재실행의 전체 실행시간을 최소화시키는 것도 아래 체크포인트의 연속성에서 설명한 이유로 가능한데, 이와 같은 인메모리 체크포인트(In- memory checkpoint)에서는 다음 두 가지를 고려하여야 한다.

3.1 체크포인트의 방식

로직 시뮬레이션에서 체크포인트 방식은 크게 두 가지가 있는데, 시뮬레이션 체크포인트 방식과 디자인 체크포인트 방식이 그것이다. 시뮬레이션 체크포인트 방식은 시뮬레이션의 실행이미지 전체를 저장하는 방식으로 일반적으로 해당 시뮬레이터들에서 제공하는 해당 시뮬레이션 명령어 내지는 시스템 테스크를 실행함으로써 이루어진다. 반면 디자인 체크포인트 방식은 디자인의 동적 정보 전체 또는 일부분만을 저장하는 방식이다. 디자인의 동적 정보란 디자인에 존재하는 모든 신호선들(Verilog 디자인의 경우에는 register와 wire)의 신호값들을 의미한다. 디자인 체크포인크가 가능한 이유는 로직 시뮬레이션 대상이 되는 디지털 하드웨어가 동시성(concurrency)을 가지고 있기 때문이다. 만일 디자인의 동적 정보 일부분만을 저장하는 체크포인트를 수행하는 경우에는 디자인에 존재하는 모든 기억소자(플립플롭, 래치, 메모리)들의 값들과 닫힌 궤환루프(closed feedback loop)를 형상하는 조합회로가 존재하는 경우에는 닫힌 궤환루프를 끊을 수 있는 닫힌 궤환루프상의 모든 신호선값들을 포함시켜야만 한다. 일반적으로 시뮬레이션 체크포인트 방식으로 생성되는 체크포인트의 크기가 디자인 체크포인트 방식으로 생성되는 체크포인트의 크기보다 훨씬 크기 때문에 인메모리 체크포인트를 위해서는 디자인 체크포인트 방식이 절대적으로 유리하다.

3.2 체크포인트의 연속성

인메모리 체크포인트를 위하여 체크포인트 방식과 함께 고려되어야 하는 것이 체크포인트의 연속성인데, 이는 체크포인트를 연속적으로 수행하여야 할 것인지 아니면 불연속적으로 수행하여야 할 것인지에 관한 것이다. 우선 연속적으로 체크포인트를 수행하는 경우에서는 롤백이 예측 틀림이 발생한 시뮬레이션 시간으로 직접 되돌아갈 수 있다는

장점이 있는 반면에, 연속적인 체크포인트에서의 한 메모리 사용이 늘어난다는 단점이 있다. 불연속적으로 체크포인트를 수행하는 경우에서는 메모리 사용을 최소화할 수 있는 것이 장점인 반면, 롤백이 예측 틀림이 발생한 시뮬레이션 시간으로 되돌아갈 수 없고 그곳에서 제일 근접한 체크포인트로만 돌아갈 수 있기 때문에 재실행시간이 늘어난다는 단점이 있다. 그런데 연속적인 인메모리 체크포인트 방식에서 메모리 사용을 최소화시킬 수 있는 방법으로 인크리멘탈 체크포인트[10]를 활용할 수 있다. 따라서 본 논문에서는 연속적 인크리멘탈 인메모리 체크포인트를 사용하여 메모리 사용을 최소화하면서 재실행 오버헤드도 함께 효과적으로 줄일 수 있도록 한다. Fig. 2는 기존의 주기적 체크포인트 방법과 본 논문에서 간헐적 동기화를 활용한 인메모리 체크포인트 방법의 실행과정을 개념적으로 도시한 것이다.

4. 실험 및 분석

본 논문에서 제안한 간헐적 동기화를 통하여 예측기반 이벤트구동 로직 시뮬레이션의 잠재적 문제점인 체크포인트 오버헤드 및 재실행 오버헤드가 얼마나 최소화되는지를 실험적으로 확인하기 위하여 다수의 디자인들을 대상으로 실험을 진행하였다. 실험을 위하여 사용된 이벤트구동 로직 시뮬레이터는 Cadence 사의 Verilog 시뮬레이터인 IUS이다[6]. IUS는 이미 다양한 최적화 기술들이 적용된 Synopsys 사의 VCS와 더불어 현존하는 제일 빠른 로직 시뮬레이터이다.

Table 1. Experimental Result for Measuring Checkpoint Overhead

디자인명	MSSICP (단위: 초)	HDCP (단위: 초)	HCICP (단위: 초)	MSSICP/HDCP (단위 : %)	MSSICP/HCICP (단위 : %)
AES	7,748	9,140	14,371	85	54
AC97	138	241	29,702	57	0.46
JPEG	4,804	5,822	338,368	83	1.42
PCI	716	929	65,871	77	1.09
PIC	461	474	4,138	97	11
VGA	5,028	12,031	19,934	42	25

우선 본 논문에서 제안한 간헐적 동기화를 통한 인메모리 체크포인트 방식(MSSICP)을 채용하는 경우에 체크포인트 오버헤드의 감소 정도를 측정한 실험 결과가 Table 1에 나와있다. 비교대상은 기존의 하드디스크를 사용하는 불연속적 체크포인트 방식(HDCP)과 하드디스크를 사용하는 연속적인 크리멘탈 체크포인트 방식(HCICP)이며, 이들 세 가지 방식들을 채용한 예측기반 이벤트구동 로직 시뮬레이션에서 예상입출력 이용-런 모드에서의 시뮬레이션 시간을 측정하여 비교하였다. Table 1에서 보이는 것과 같이 모든 디자인들에서 본 논문에서 제안한 간헐적 동기화를 채용한 인메모리 체크포인트 방식에서 체크포인트 오버헤드가 제일 작음을 알 수 있다. 또한 기존의 방식들 중에서는 하드디스크를 사용하는 연속적인 크리멘탈 체크포인트 방식이 하드디스크를 사용하는 불연속적 체크포인트 방식에 비하여 훨씬 큰 체크포인트 오버헤드를 가지게 되는 사실 또한 Table 1을 통하여 알 수 있다.

두 번째로, 간헐적 동기화를 통한 인메모리 체크포인트 방식을 통하여 예측기반 병렬 시뮬레이션 실행에서 재실행 오버헤드가 어느 정도 효과적으로 감소되는지를 분석하여 보았다. 앞서 설명한 것과 같이 본 논문에서 제안한 간헐적 동기화를 통한 인메모리 체크포인트 방식은 두 개의 인접한 동기화 시점 사이에서 인크리멘탈 방식의 연속적 체크포인트를 메모리 덮어쓰기 방식으로 수행한다. 따라서 실제출력이 예상출력과 다름이 관찰된 해당 로컬 시뮬레이션은 재실행을 수행할 필요 없이 예상출력/실제출력 불일치 시점에서부터 실제입출력 이용-런 모드로 전환할 수 있다. 그러나 기존의 체크포인트 방식인 하드디스크를 사용하는 불연속적 체크포인트 방식은 실제출력이 예상출력과 다름이 관찰된 해당 로컬 시뮬레이션이 반드시 예상출력/실제출력 불일치 시점으로부터 제일 근접한 체크포인트로 롤백 후에 이 체크포인트에서부터 예상출력/실제출력 불일치 시점까지 재실행되어야만 하는 피할 수 없는 재실행 오버헤드가 존재함을 알 수 있다. 물론 기존의 하드디스크를 사용하는 연속적인 크리멘탈 체크포인트 방식도 실제출력이 예상출력과 다름이 관찰된 해당 로컬 시뮬레이션은 재실행을 수행할 필요 없이 예상출력/실제출력 불일치 시점에서부터 실제입출력 이용-런 모드로 전환할 수 있다. 그러나 이를 위해서 하드디스크를 사용하는 연속적인 크리멘탈 체크포인트 방식은 Table 1에서 보이는 것과 같이 매우 과도한 체크포인트 오버헤드를 감수하여야만 할 뿐만 아니라, 매우 큰 하드디스크 저장공간도 필요한 추가적인 단점이 존재한다. 또한 재실행 오버헤드는 예측정확도에 따라서 달라지게 된다. 즉, 100%의 예측정확도에서는 재실행에 소요되는 시뮬레이션 시간이 0이지만, 예측정확도가 떨어질수록 체크포인트를 연속적으로 진행하지 않는다면 재실행에 의하여 소요되는 시뮬레이션 시간도 증가될 수 있다. 그런데 본 논문에서 제안한 간헐적 동기화를 통한 인메모리 체크포인트 방식을 사용하게 되면 예측정확도가 떨어지더라도 이와 같은 전체 시뮬레이션 시간에서 상당한 부분을 차지하게 되는 재실행 오버헤드를 효과적으로 제거하는 것도 가능하다.

정리하면, 본 논문에서 제안한 비관적 동기화와 낙관적 동기화를 혼용한 하이브리드 동기화 방식은 예측기반 병렬 이벤

트구동 로직 시뮬레이션에 간헐적 동기화를 채용함으로써, 연속적 인메모리 인크리멘탈 체크포인트를 가능하게 할 뿐만 아니라 예측 틀림과 연관된 체크포인트 오버헤드와 재실행 오버헤드를 함께 최소화시킬 수 있는 방안임을 알 수 있다.

5. 결 론

병렬 이벤트구동 로직 시뮬레이션에서 제일 큰 문제점은 병렬화를 통하여 기대할 수 있는 성능 향상의 많은 부분을, 로컬 시뮬레이션들 간의 과도한 동기 오버헤드 및 통신 오버헤드로 인하여 실제로는 얻을 수 없다는 것이다. 최근에 이를 해결할 수 있는 효과적인 방법으로 예측에 기반한 병렬 이벤트구동 로직 시뮬레이션 기법이 제안되었다. 제안된 기법은 예측이 맞는 상황에서는 각 로컬 시뮬레이션이 다른 로컬 시뮬레이션들과 동기 및 통신을 완전히 생략하고 독립적으로 실행될 수 있게 함으로써 성능 향상을 큰 폭으로 가능하게 한다. 그러나 예측이 빈번히 틀려지는 상황에서는 체크포인트 오버헤드와 재실행 오버헤드로 인하여 성능 향상에 제약을 초래하게 된다.

본 논문에서는 비관적 동기화와 낙관적 동기화의 하이브리드 방식인 간헐적 동기화 및 이를 통한 인메모리 체크포인트를 통하여 체크포인트 오버헤드와 재실행 오버헤드를 최소화시킬 수 있는 새로운 기법과 이를 활용하여 예측기반 병렬 이벤트구동 로직 시뮬레이션의 성능 향상을 극대화할 수 있는 방법을 제안하였다. 제안된 기법은 다수의 디자인들을 대상으로 한 실험을 통하여 예측기반 병렬 이벤트구동 로직 시뮬레이션에서 문제가 될 수 있는 체크포인트 오버헤드 및 재실행 오버헤드를 최소화시킴으로써 시뮬레이션의 성능 향상에 효과적임이 확인되었다.

References

[1] R. M. Fujimoto, “Parallel Discrete Event Simulation,” *Communication of the ACM*, Vol.33, No.10, pp.30-53, Oct., 1990.

[2] D. M. Nicol, “Principles of Conservative Parallel Simulation,” Proceedings of the 28th Winter Simulation Conference, pp.128-135, 1996.

[3] R. M. Fujimoto, “Time Warp on a Shared Memory Multiprocessor,” *Transactions of the Society for Computer Simulation*, Vol.6, No.3, pp.211-239, Jul., 1989.

[4] L. Li, C. Tropper, “A design-driven partitioning algorithm for distributed Verilog simulation,” in Proc. 20th International Workshop on Principles of Advanced and Distributed Simulation (PADS), pp.211-218, 2007.

[5] D. Chatterjee, A. DeOrio, and V. Bertacco, “Event-driven gate-level simulation with general purpose GPUs,” Proceedings of Design Automation Conference (DAC09), pp.557-562, Jun., 2009.

[6] IUS Simulator Usermanual, Cadence Design Systems [Internet], http://www.cadence.com

[7] VCS Simulator Usermanual, Synopsys [Internet], http://www.synopsys.com

[8] K. Chang, C. Browy, “Parallel Logic Simulation: Myth or Reality?”, *Computer*, Vol.45, No.4, pp.67-73, Apr., 2012.

[9] Jaehoon Han et al., “Predictive parallel event-driven HDL simulation with a new powerful prediction strategy,” Proc. of Design, Automation and Test in Europe Conference and Exhibition (DATE), pp.1-3, Mar., 2014.

[10] H. Bauer, C. Sporrer, “Reducing Rollback Overhead in Time Warp Based Distributed Simulation with Optimized Incremental State Saving,” Proc. 26th Annual Simulation Symposium, pp.12-20, Mar., 1993.

[11] James Gross et al., “Multi-Level Parallelism for Time- and Cost-efficient Parallel Discrete-Event Simulation on GPUs,” Proc. of 26th ACM/IEEE Workshop on Principles of Advanced and Distributed Simulation 2012 (PADS 2012), Jun., 2012.

[12] Zhang Yuxuan et al, “Logic simulation acceleration based on GPU,” Proc. of the 18th International Conference on Mixed Design of Integrated Circuits and Systems (MIXDES 2011), pp.608-613, Jun., 2011.

[13] Wenjie Tang, Yiping Yao, “A GPU-based discrete event simulation kernel,” *Journal of Simulation*, Vol.89, No.11, pp.1335-1354, Nov., 2013.

곽 두 환

e-mail : taoist29@gmail.com

2013년 부산대학교 정보컴퓨터공학부(학사)

2015년～현 재 부산대학교 전자전기컴퓨터공학과 공학석사

관심분야 : 설계자동화, 특히 SoC검증

양 세 양

e-mail : syyang@pusan.ac.kr

1981년 고려대학교 전자공학과(학사)

1985년 고려대학교 컴퓨터공학과(공학석사)

1990년 University of Massachusetts, Amherst 컴퓨터공학과(공학박사)

1992년～현 재 부산대학교 정보컴퓨터공학부 교수

관심분야 : 설계자동화, 특히 SoC검증