論文 96-21-11-21

고장시뮬레이션의 병렬화 알고리듬에 관한 연구

正會員 송 오 영*

Study on Parallel Algorithm for Fault Simulation

Ohyoung Song* *Regular Member*

※본 논문은 95년도 한국학술진흥재단의 학술연구조성비에 의하여 연구되었음.

요 약

VLSI의 급격한 발달로 매우 큰 회로의 설계를 가능하게 함에 따라서 효과적인 고장시뮬레이션 기법이 필요하다. 고장시뮬레이션은 많은 컴퓨터자원을 필요로 한다. 범용 멀티프로세서가 점점 일상적으로 널리 사용될 수 있게 됨에 따라서 이러한 범용 멀티프로세서는 고장시뮬레이션의 가속화를 위한 효율적인 대안의 하나가 될 수 있다.

동기식 순차회로에 대하여 선형 반복 배열 모델을 사용하고 병렬 고장시뮬레이션 개념과 지선 고장시뮬레이션 개념을 결합하면 효율적인 고장시뮬레이션을 수행할 수 있음이 이미 보고되었다. 우리는 이러한 고장시뮬레이션 기법을 공유기억장치를 갖는 범용 멀티프로세서에서 병렬화 함으로써 고장시뮬레이션을 가속화하는 연구를 하였다. 이러한 실험적 연구를 통하여, 사용되는 프로세서의 개수에 따른 시뮬레이션의 가속화 정도, 프로세서의 이용도, 범용 멀티프로세서의 하드웨어에 의한 성능에 대한 영향 등을 조사할 것이다. 약간의 벤치마크 회로에 대한 실험결과가 보여질 것이다.

ABSTRACT

As design of very large circuits is made possible by rapid development of VLSI technologies, efficient fault simulation is needed. In general, fault simulation requires many computer resources. As general-purpose multiprocessors become more common and affordable, these seem to be an attractive and effective alternative for fault simulation.

Efficient fault simulation of synchronous sequential circuits has been reported to be attainable by using a linear iterative array model for such a circuit, and combining parallel fault simulation with surrogate fault simulation.

*중앙대학교 공과대학 제어계측공학과
Department of Control and Instrumentation Engineering
Chung-Ang University
論文番號:96134-0430
接受日字:1996년 4월 30일

Such fault simulation algorithm is parallelized on a general-purpose multiprocessor with shared memory for acceleration of fault simulation. Through the experimental study, the effect of the number of processors on speed-up of simulation, processor utilization, and the effect of multiprocessor hardware on simulation performance are studied. Some results from experiments with benchmark circuits are shown.

Ⅰ. 서 론

VLSI기술의 발달로 매우 높은 고장검출율(fault coverage)을 갖는 테스트 시퀀스를 얻는 것이 디지탈 집적회로에서 결함여부를 판단하는 데 아주 중요하다. 고장시뮬레이션은 주어진 테스트 시퀀스에 의해 검출되는 고장들을 판별하는 데 사용된다. 이러한 정보는 그 테스트 시퀀스의 질을 평가하기 위해 사용될 뿐만이 아니라 테스트 시퀀스의 자동생성을 가속화하는 데 사용된다.

최근에 생산되는 집적도가 매우 높은 VLSI(수 백만 게이트)들에 대해서 고장시뮬레이션을 수행할 때 많은 컴퓨터자원(cpu수행시간, 메모리 등)을 필요로 한다. 논리시뮬레이션 및 고장시뮬레이션을 가속화하기 위해서 병렬처리 기법에 관한 많은 연구 결과들이 보고되었다. 병렬처리 기법은 크게 두 가지로 분류될 수 있다:(1)하드웨어 가속기를 사용하는 기법과, (2) 범용 멀티프로세서를 사용하는 기법. 하드웨어 가속기를 사용하는 기법들[1, 2, 10, 11, 17, 29]에서는 사용될 알고리듬, 회로 모델, 지연시간 모델 등에 관한 선택이 매우 제한되고, 또한 하드웨어 가속기의 높은 연구개발비와 범용 멀티프로세서의 빠른 저가화를 고려할 때, 하드웨어 가속기의 사용은 범용 멀티프로세서의 사용에 비해 덜 경제적일 것이다.

우리의 연구는 고장시뮬레이션을 위해 공유 기억장치를 갖는 범용 멀티프로세서를 사용하는 기법에 관한 것이다. 본 연구의 목적은 이러한 범용 멀티프로세서의 적용 가능성에 관한 실험적 연구를 통하여 단위 비용에 대한 높은 성능 비율을 갖는 고장시뮬레이션 툴 개발의 가능성을 규명하는 데 있다. 본 연구에서는 저가화되고 있는 중간수준 병렬컴퓨터(medium grained parallel computer)를 사용하여 논리 게이트로 구성된 동기식 순차회로에 대하여 고장시뮬레이션을 수행하는 기법을 보여준다. 본 논문에서는 이산사건 시뮬레이션(discrete event simulation)개념에 국한하여 연구를 수행한다.

본 논문의 나머지 구성은 다음과 같다:제 2절에서는 병렬 시뮬레이션의 주요 이슈 및 기존의 기법들에 관하여 설명한다. 제 3절에서는 새로운 고장시뮬레이션의 병렬화 기법을 제시한다. 제시된 고장시뮬레이션의 병렬화 알고리듬은 후진추적 방식 고장시뮬레이션[33]에 기초를 두고 있으므로 후진추적 방식 고장시뮬레이션에 대한 간단한 소개가 주어질 것이다. 제시된 고장시뮬레이션의 병렬화 알고리듬은 후진추적 방식 고장시뮬레이션에서의 작업을 분석하여 각각의 프로세서에 작업을 골고루 분담시키는 방식을 이용한다. 이에 대한 자세한 설명이 주어진다.

Ⅱ. 병렬 시뮬레이션의 주요 이슈

논리 및 고장시뮬레이션의 성능을 향상시키기 위해 병렬성을 얻는 데는 여러 가지 방식이 있다[25]. 알고리듬 병렬성은 시뮬레이션 알고리듬을 순차적인 과정으로 분할한다. 각각의 과정을 다른 프로세서에 할당하여 파이프라인 방식으로 시뮬레이션을 진행한다. 데이타 병렬성은 다수의 입력벡터를 시뮬레이션할 때 얻어진다. 다수의 입력벡터는 사용된 프로세서의 개수에 따라서 분할되어 각각의 분할이 다른 프로세서에서 시뮬레이션된다. 고장시뮬레이션에서는 다수의 고장들이 분할되어 각각의 분할이 다른 프로세서에서 시뮬레이션될 수 있다. 모델 병렬성은 각각 부분회로에 대한 함수적 계산을 다른 프로세서에 할당하여 수행하게 할 때 얻어진다. 어떤 회로에서 전기적인 신호는 동시에 여러 논리요소들로 전송되므로 서로 다른 논리요소들이 동일한 시각에 활동적이 된다. 이러한 각각의 논리요소들에 대한 함수적 계산이 다른 프로세서에서 병렬적으로 수행될 수 있다.

다음은 멀티프로세서와 벡터컴퓨터를 이용한 논리시뮬레이션 및 고장시뮬레이션에 관한 논문들에 관하여 설명한다. 계층화 동시적 고장시뮬레이션(hierarchical concurrent fault simulation)이 loosely coupled machine들로 구성된 분산환경에서 구현되었다[18]. 고

장들의 집합을 나눔으로써 고장시뮬레이션 문제를 보다 작은 부분 문제들로 분할하였다. 분할된 작은 부분 문제들을 각각의 프로세서에 할당하였다.

동시적 고장시뮬레이션이 메시지 전달 기능을 갖는 범용 멀티프로세서에서 구현되었다[25]. 그 방법은 회로를 작은 부분 회로들로 나누고, 각각의 부분 회로를 다른 프로세서에 할당하였다. 각각의 프로세서는 할당된 부분 회로에 대하여 고장시뮬레이션을 수행한다. 그러나 그 방법에서는 부분 회로들이 서로 중복되는 것을 허용하므로, 그 중복된 부분에 대한 시뮬레이션은 낭비적이다.

병렬 고장시뮬레이션이 Cray X-MP 슈퍼컴퓨터에서 벡터연산을 이용하여 구현되었다[29]. 병렬 고장시뮬레이션이 Connection Machine에서 구현되었다[3, 26]. 병렬성을 얻기 위하여 고장 및 입력패턴을 그룹핑하므로써 벡터화하여 병렬 고장시뮬레이션 기법을 벡터 슈퍼컴퓨터에서 구현하는 연구도 있었다[20].

모델 병렬성을 이용한 논리시뮬레이션 기법에 대한 연구 결과가 [8, 14, 31, 32, 34, 35, 36, 37, 39, 42]에 보여진다. 모델 병렬성을 이용한 일반적인 시뮬레이션에 대한 많은 연구들이 있었다[4, 6, 7, 9, 22, 28]. 모델 병렬성을 얻기 위한 기법들은 태스크를 가장 작은 요소들로 분할하여 그러한 요소들의 집합을 프로세서에 균등하게 할당한다. 각각의 프로세서는 자신에게 할당된 요소들에 대하여 함수적 계산을 수행하고 그 결과로 인하여 발생된 사건들은 그 요소들의 fanout으로 전달된다. 이때 그 fanout이 자신의 프로세서에 속하지 않는 요소일 경우는 채널을 통한 정보 전달을 하여야 한다. 각각의 프로세서가 완전히 독립적으로 수행될 때 그 프로세서들의 동기화문제에 대하여 대표적인 두 가지 종류의 연구가 있었다: (1) Chandy-Misra 알고리듬[15], (2) Time Warp 알고리듬[21].

Chandy-Misra알고리듬에서는 어떤 요소의 입력단에 시간 t보다 이른 클럭시간을 갖는 어떠한 사건도 도달될 수 없을 경우에만 각각의 프로세서는 클럭시간 t에 관하여 그러한 요소의 함수적 계산을 수행한다. 어떤 요소들이 그들의 모든 입력단에서 사건을 접수하였을 경우 프로세서들은 서로 다른 클럭시간일 지라도 그러한 요소들의 함수적 계산을 동시에 할 수 있다. 만약 그들의 모든 입력에 적어도 하나 이상의 사건을 접수한 요소들이 하나도 없을 경우, ***deadlock***이 발생했다라고 말한다. 시뮬레이션은 deadlock이 발생할 때까지 비동기식으로 진행된다. Deadlock을 풀기 위해서는 모든 요소들을 검색하여 최소의 클럭시간을 산출하여 그것으로 요소들의 클럭시간을 변경한 후 시뮬레이션을 다시 진행시킨다.

Time Warp알고리듬에서는 적어도 하나의 사건이 어떤 요소의 입력단에 전달되면 프로세서는 그 요소의 함수적 계산을 수행한다. 그 프로세서의 클럭시간을 t라고 하자. 만약 t보다 이른 클럭시간을 갖는 다른 사건이 그 요소의 다른 입력에 전달되면, 그 전의 사건에 대한 함수적 계산의 효과는 취소되어야 한다. 시뮬레이션은 이와 같은 모순되는 경우가 발생할 때까지만 수행된다. 어떤 프로세서가 시간적으로 너무 많이 순방향으로 시뮬레이션을 진행하였을 때 그보다 과거의 사건을 접수한 경우 그 요소의 상태를 과거의 시간으로 ***roll-back***시켜야 하고 그 시간으로부터 다시 시뮬레이션을 진행시켜야 한다. Roll-back의 발생시, 이미 처리된 그러나 모순된 사건들의 효과는 취소되어야 한다.

사건의 숫자가 많지 않아서 충분한 프로세서 이용도(processor utilization)를 얻지 못하는 경우에 Chandy-Misra알고리듬과 Time Warp알고리듬에 근거한 시뮬레이션 기법들은 동시에 많은 사건들을 처리할 수 있는 잠재력을 갖고 있다. 그러나 회로는 일반적으로 귀환루프 혹은 재집결 fanout을 갖기 때문에 이 방법들이 논리시뮬레이션이나 고장시뮬레이션에 적용될 때 deadlock이나 roll-back과 같은 심각한 문제에 직면할 수 있다. 또한 회로들이 함수적 계산량이 작은 요소들을 주로 포함할 경우 병렬계산에 의한 이득은 프로세서간의 메시지 전달에 의한 오버헤드에 의해 압도될 수도 있다.

Ⅲ. 고장시뮬레이션의 병렬화 기법

사용된 멀티프로세서가 medium-grained parallel machine의 범주에 속할 경우, 규모가 큰 회로를 시뮬레이션하면 동일한 시각을 갖는 사건의 숫자는 프로세서의 숫자보다 훨씬 많다. 그런 경우에는 동기식 시뮬레이션[16]이 비동기식 시뮬레이션보다 더 큰 프로세서 이용도를 가질 수 있고 deadlock이나 roll-back이 발생하지 않으므로 훨씬 더 효율적이다. 본 논문에서

는 주요 시뮬레이션 클럭(global simulation clock)을 갖는 동기식 시뮬레이션 기법을 이용한다. 주요 시뮬레이션 클럭의 사용이 뜻하는 바는 시뮬레이션의 매 과정마다 동일한 시각으로 스케줄된 사건들을 모든 프로세서들이 처리하는 방식으로 시뮬레이션의 매 과정들이 동기화 된다는 것이다.

멀티프로세서에서 하나의 주요 사건목록(global event list)만을 사용할 경우 그 주요 사건목록에 스케줄링을 하고 그 목록을 논리적으로 건전한 상태를 유지하기 위해서는 주요 사건목록에의 상호 배타적(mutual exclusion)인 접근이 필요하다. Semaphore 혹은 lock/unlock 등으로 구현될 수 있는 상호 배타적 접근 방식은 그것의 오버헤드가 무시될 수 있을 정도로 작지 않다면 시뮬레이션의 성능에 나쁜 영향을 줄 수도 있다. 그 이유는 다음과 같다. 어떤 프로세서가 주요 사건목록에서 다음으로 처리해야 할 사건을 획득하기 위해서 그 주요 사건목록을 읽고 있는 동안에, 다른 프로세서는 어떠한 의미 있는 작업도 수행하지 못한 채로 첫 번째 프로세서가 그 사건목록에의 접근을 마칠 때까지 기다려야 한다. 주요 사건목록의 접근이 자주 일어날 경우 상호 배타성에 의해 오버헤드는 클 수가 있다. 이러한 문제를 풀기 위해서 주요 사건목록과 더불어 다수의 보조 사건목록들을 사용할 것이다. 각각의 보조 사건목록은 각각의 다른 프로세서와 관련된다.

동기식 순차회로의 고장시뮬레이션 작업은 고장목록을 분할하여 많은 독립적인 부분작업으로 나눌 수가 있다. 이러한 부분작업들이 상호 의존성이 전혀 없어서 고장시뮬레이션의 병렬화는 쉽게 달성될 수 있다. 각각의 프로세서는 독립적으로 그것에 할당된 고장들을 시뮬레이션한다. 멀티프로세서 환경에서 고장시뮬레이션의 성능효율은 경과시간(turnaround time)에 의하여 결정된다. 경과시간이란 시뮬레이션이 시작되어 모든 프로세서가 그것들에 할당된 부분작업들을 완성할 때까지 걸리는 시간을 말한다. 고장목록 분할에 의한 최적의 병렬화는 경과시간을 최소화할 수 있도록 고장목록을 분할하는 방법을 찾음으로서 가능하다.

고장목록 분할을 하는데 몇 가지 방법이 고려될 수 있다. 먼저 정적 고장목록 분할(static fault list partitioning)을 고려한다. 시뮬레이션을 수행하기 전에 고장목록은 n개로 분할된다. 여기서 n은 사용되는 프로세서의 개수이고 각각의 분할은 동일한 숫자의 고장들을 갖는다. 각각의 프로세서는 그것에 할당된 모든 고장들을 시뮬레이션한다. 이때 시뮬레이션을 수행하는 데 걸리는 시간은 프로세서마다 서로 다르게 되어 불균형이 발생할 수 있다. 고장시뮬레이션의 효율성은 프로세서들의 시뮬레이션 수행시간들 중에서 가장 긴 시간에 의해서 결정되기 때문에, 불균형 수행시간으로 인해서 시뮬레이션 성능효율(throughput)이 저하될 수도 있다.

두 번째 방식은 동적 고장목록 분할(dynamic fault list partitioning)방식으로 시뮬레이션을 수행하면서 필요에 따라서 고장목록을 분할한다. 시뮬레이션을 수행할 대기 상태의 프로세서는 고장목록으로부터 처리할 고장들을 확인한다. 이러한 고장들이 그 프로세서에 할당될 때 고장목록은 상호 배타적 방식으로 접근되어야 한다. 어떤 프로세서가 고장들을 할당받으면 그 프로세서는 그 고장들을 시뮬레이션하고 시뮬레이션 결과, 상태변수에 전달되는 고장들을 다음 패턴의 시뮬레이션을 위해서 새로운 고장목록에 저장한다. 고장들이 새로운 고장목록에 저장되기 전에 그 새로운 고장목록도 상호 배타적으로 접근되어야 한다. 고장목록에 포함된 모든 고장들이 완전히 시뮬레이션될 때까지 매 입력패턴마다 이러한 과정이 반복된다. 동적 고장목록 분할 방법에서 고장목록 접근을 원하는 프로세서들이 고장목록의 접근을 허용받기 위한 권한을 얻을 때까지는 대기상태에서 기다려야 한다. 상호 배타적 접근을 위한 처리시간이 작지 않을 경우 상호 배타적 접근법은 만족스럽지 못한 성능효율을 보여 줄 것이다.

본 연구에서는 프로세서가 시뮬레이션을 수행하는 데 걸리는 시간의 불균형을 감소시키고 상호 배타성에 기인하는 오버헤드를 피하는 새로운 고장목록 분할 기법을 도입한다. 또한 멀티프로세서에서의 효과적인 고장시뮬레이션을 위해서는 충분한 병렬성을 갖고 있으면서 동시에 알고리듬 자체에 효율성을 갖고 있는 고장시뮬레이션 알고리듬의 선택이 중요하다. 본 논문에서는 후진추적 기법[33]을 범용 멀티프로세서에서 병렬화할 것이다. 다음 소절 3.1에서는 후진추적 기법에 대한 소개와 그 기법에 사용되는 고장모델, 회로모델 및 약간의 기술용어에 대한 소개를

하고 다음 소절 3.2에서는 후진추적 기법이 범용 멀티프로세서에서 병렬화 되도록 수정되는 과정을 소개한다.

3.1 고장시뮬레이션

본 연구에서는 게이트 레벨의 동기식 순차회로에 대한 고장시뮬레이션을 고려할 것이다. 이러한 동기식 순차회로는 선형 반복 배열 모델(linear iterative array model)에 의해서 표시된다. 고장모델로서는 단일 고착값 고장(single stuck-at) 모델을 사용한다. 동기식 순차회로와 같은 특별한 종류의 회로에 대하여 고장시뮬레이션을 가속화하는 연구들이 발표되었다[19, 23, 24, 27, 33, 41]. 이러한 기법들은 전형적인 고장시뮬레이션 기법들[5, 30, 38]보다 매우 효율적이다. 그 중에서도 가장 효율적인 기법이 후진추적 방식을 동기식 순차회로에 적용한 것이다[33]. 본 연구에서는 후진추적 방식을 도입한 고장시뮬레이션 알고리듬을 범용 멀티프로세서에서 병렬화한 것이다.

선형 반복 배열 모델은 동기식 순차회로의 조합회로 부분(combinational cell)을 선형적으로 반복하여 배열함으로서 동기식 순차회로를 표현한다. 각각의 조합회로 부분은 하나의 클럭상태 동안의 회로상태를 표현한다. 이러한 회로상태를 그 클럭에 관련된 **시간 후레임**이라고 부른다. 어떤 조합회로 부분으로부터 다음 조합회로 부분으로 전달되는 모든 신호들은 플립플롭 혹은 랫치들을 통해야 한다. 어떠한 시간 후레임에서 그 직전의 시간 후레임으로부터 전달되는 신호들은 **제 2차 입력단**(y_i: pseudo-inputs)이라 불리는 현재 시간 후레임의 입력단에 도달한다. 현재 시간 후레임으로부터 조합회로 부분을 통해서 다음 시간 후레임에 전달되는 신호들은 **제 2차 출력단**(Y_i: pseudo-outputs)이라 불리는 현재 시간 후레임의 출력단을 통과함으로써 가능하다.

부분기선 영역은 분기선을 포함하지 않는 최대의 부분 회로로 정의된다. 어떠한 조합회로 부분도 부분기선 영역(fanout-free region)들의 상호 연결에 의해서 표시될 수 있다. 어떤 부분기선 영역의 입력단들은 주요 입력단, 제 2차 입력단, 혹은 분기선(fanout branch)들이다. 그 부분기선 영역의 출력단은 주요 출력단, 제 2차 출력단, 혹은 지선(fanout stem)이다. 회로의 모든 각각의 신호선은 정확히 하나의 서로 다른 부분기선 영역에 속한다.

클럭 신호선에는 고장이 없는 것으로 가정하며 단일 고착값 고장은 회로의 조합회로 부분에만 존재하는 것으로 가정한다. 플립플롭 혹은 랫치 등을 조합회로 요소인 ***pseudo-flip-flops***[12]으로 모델링함으로서 그것들에 존재하는 고장들에 대한 처리가 가능하다. 회로에 존재하는 어떤 고장의 고장효과가 적어도 하나의 주요 출력단으로 전달되면 그 고장은 검출된다. 단일 고착값 고장은 각각의 조합회로 부분에 존재하는 고장으로써 모델링되어야 하고 그 고장효과는 적어도 하나의 주요 출력단으로 전송될 때까지 여러 개의 시간 후레임을 통과할 수도 있기 때문에 어떤 조합회로 부분에서의 고장효과는 다른 조합회로 부분에 존재하는 동일한 고장의 효과들과 상호 작용을 일으킬 수 있다. 이럴 경우 그러한 고장효과는 자체 소멸(self-masking) 혹은 계속 전송이 될 수 있다. 이와 같이 고장효과가 여러 개의 조합회로 부분을 통과할 때 고장효과들의 복잡한 상호 작용에 관한 연구는 [19, 24]에서 볼 수 있다.

신호값으로는 {0, 1, *X*}를 가정한다. 회로들은 메모리요소로서 D 플립플롭을 포함하며 논리게이트로서는 AND, OR, NAND, NOT만을 포함한다. Wang[40]에 의해 처음으로 정의된 후진추적 방식은 신호선들의 criticality개념을 이용한다. 만약 테스트 t가 고장 l s-a-v를 검출한다면, 신호선 l은 테스트 t에 관하여 ***v-critical***하다라고 말한다. 만약 테스트 t가 고장 l s-a-v의 효과를 그 신호선 l의 지선까지 전달한다면 그 신호선 l은 테스트 t에 관하여 ***v-stem-critical***하다라고 말한다. 모든 지선들은 stem-critical하고 모든 주요 출력단들은 critical하다. 만약 어떤 신호선 l이 테스트 t에 관하여 v-stem-critical하고 그 신호선의 지선이 테스트 t에 관하여 critical하면 신호선 l은 테스트 t에 관하여 v-critical하다.

어떤 고장의 효과도 시간 후레임 t의 상태변수(제 2차 출력단)에 전송되지 않았고 그 고장의 위치에서 고장값이 무고장값과 다르다면 그 고장은 시간 후레임 t에서 **단순-활동적**이다라고 말한다. 어떤 고장의 효과가 시간 후레임 t에서 적어도 하나의 상태변수(제 2차 출력단)에 전달되었다면 그 고장은 **복잡-활동적**이다라고 말한다.

단순-활동적인 고장들에 대한 시뮬레이션은 지선으

로부터의 후진추적과 지선 위의 고장들을 시뮬레이션한 결과로부터 계산될 수 있다. 지선으로부터의 후진추적은 단순-활동적인 고장들이 존재하는 신호선들의 stem-criticality를 결정한다. 지선 위의 고장들에 대한 시뮬레이션은 병렬 고장시뮬레이션 기법(parallel fault simulation)을 이용한다. 사용되는 컴퓨터의 단어크기 만큼의 지선 고장들을 그룹핑하여 bitwise 논리연산을 수행한다. 만약 지선 고장들의 숫자가 컴퓨터의 단어크기보다 클 때는 여러 번의 병렬 고장시뮬레이션의 적용이 필요하다. 복잡-활동적 고장들은 지선 고장을 위한 시뮬레이션 방식과 유사하게 병렬 고장시뮬레이션 방식을 적용한다.

단일 프로세서에서 수행되는 후진추적 알고리듬의 개요는 다음과 같다.

1. 무고장값 시뮬레이션을 수행한다.
2. 단순-활동적 고장을 갖는 신호선들의 stem-criticality를 결정하기 위해 그 신호선들의 지선으로부터 후진추적을 수행한다.
3. 그러한 지선들의 criticality를 결정하기 위해 그 지선들 위의 고장들을 병렬 고장시뮬레이션 개념을 이용하여 시뮬레이션하고 그 지선들과 관련된 stem-critical한 신호선들의 criticality를 결정한다.
4. 복잡-활동적인 고장들은 병렬 고장시뮬레이션 개념을 이용하여 시뮬레이션한다.

다음 소절에서는 이 후진추적 알고리듬을 범용 멀티프로세서를 이용하여 병렬화하는 알고리듬을 소개한다.

3.2 작업분할 기법

멀티프로세서 시스템에 있는 프로세서들 중에서 하나를 선택하여 그것을 주 프로세서라고 하고, 주 프로세서 이외의 다른 프로세서들을 부 프로세서라고 하자. 본 연구에서는 주 프로세서와 부 프로세서들간의 상호 협력을 통해서 작업분할을 수행하도록 할 것이다. 주 프로세서는 부 프로세서들과 자기 자신에게 작업을 분할하여 할당하고 프로세서들은 할당된 부분작업을 수행한다. 또한 주 프로세서는 현재 시간 후레임의 입력 패턴을 읽거나, 직전 시간 후레임으로부터 전달된 상태변수들의 신호값을 현재 시간 후레임의 제 2차 입력단의 신호값으로 변경하는 작업도 수행한다. 이 과정 동안은 다른 프로세서들은 대기상태에 있게 된다.

전 소절 3.1에서 설명한 후진추적 기법에서 제일 처음으로 수행되는 것은 무고장값 시뮬레이션이다. 이 무고장값 시뮬레이션이 어떻게 작업분할 기법에 의해 병렬화되는 지는 다음에 보여진다. 무고장값 시뮬레이션은 어떤 레벨의 계산목록에 등록된 모든 게이트들이 계산된 후에 그 다음 레벨의 계산목록에 등록된 모든 게이트들이 계산되는 동기식 방식으로 진행된다. 무고장값 시뮬레이션 과정에서 어떤 레벨의 계산목록에 등록된 게이트들에 관한 계산은 다음에 보여지는 것처럼 병렬적으로 수행될 수 있다. 각각의 프로세서는 자기 자신만의 부 계산목록을 하나씩 관리한다. 주 프로세서를 제외한 어떤 프로세서도 다른 프로세서의 부 계산목록을 접근할 수 없다. 주 프로세서는 주 계산목록을 관리한다. 주 프로세서는 모든 부 계산목록 및 주 계산목록에 접근할 수 있다. 레벨 l에 있는 게이트들을 계산할 때 주 프로세서는 모든 프로세서와 관련된 레벨 l의 계산목록에 접근하여 그것들을 주 계산목록으로 병합한다. 이때 어떠한 게이트도 주 계산목록에 중복되어 등록되지 않도록 한다. 이것은 동일한 게이트가 매 시간 후레임에서 두 번 이상 계산되지 않는 것을 보장해 준다.

주 프로세서는 주 계산목록에 등록된 게이트들을 n개의 집합으로 분할한다. 여기서 n은 멀티프로세서 시스템에 존재하는 모든 프로세서의 개수이다. 각각의 집합은 근사적으로 거의 동일한 숫자의 게이트들을 포함한다. 주 프로세서는 각각의 집합을 각각의 프로세서에 할당한다. 각각의 프로세서는 하나의 집합을 할당받자마자 그 집합에 포함된 게이트들의 출력을 계산한다. 또한 주 프로세서는 하나의 집합을 자기 자신에게 할당하고 모든 다른 프로세서들에 작업분담을 끝낸 후에 자기 자신에게 할당된 집합에 포함된 게이트들의 출력을 계산한다. 어떤 프로세서가 게이트 g의 출력을 계산했을 때 그 출력 신호값이 변할 경우 그 프로세서는 게이트 g의 fanout 게이트들에 새로운 신호값을 전달하고 그 fanout 게이트들을 적절한 레벨의 부 계산목록에 등록한다. 그 프로세서에 할당된 모든 게이트들의 출력을 계산한 후 그 프로세서는 ***barrier***라고 불리는 동기화 상태에 도달해서 모

든 다른 프로세서가 그 barrier에 도달할 때까지 대기한다. 모든 프로세서들이 그 barrier에 도달하면 다음 레벨의 게이트들에 대한 시뮬레이션이 수행되어져야 한다. 주 프로세서는 다음 레벨에 대해서도 유사한 작업분할을 반복한다. 시뮬레이션 과정은 그 레벨에 대해서도 동일한 방식으로 반복된다. 모든 프로세서와 관련된 부 계산목록에 포함된 모든 게이트들이 주어진 시간 후레임에 대해서 계산되었을 때 그 시간 후레임에 대한 무고장값 시뮬레이션은 끝이 난다.

동기식 순차회로를 위한 후진추적 방식에서 다음으로 수행되어야 할 것들은 (가) 단순-활동적 고장을 갖는 신호선들의 stem-criticality를 결정하기 위한 그 신호선들의 지선으로부터의 후진추적과 (나) 그러한 지선들의 criticality결정을 위한 그 지선들 위의 고장들과 복잡-활동적인 고장들에 대한 병렬 고장시뮬레이션의 수행이다. 이러한 작업들이 동적 작업분할 방식에 의해서 보다 잘 균형 잡힌 부분 작업들로 나뉠 수 있을 지라도, 동적 작업분할 방식은 상호 배타성의 구현을 필요로 하기 때문에 이 방식이 항상 효율적인 성능을 보장할 수는 없다. 정적 작업분할 방식을 사용하여 후진추적을 수행한다면 다음과 같을 것이다. 대기 상태의 프로세서는 후진추적할 지선들에 관한 정보를 갖고 있는 무분기선 영역 목록에 대한 상호 배타적 접근 권한을 부여받고 후진추적할 하나의 무분기선 영역을 선택한다. 그 목록의 상호 배타적 접근에 대한 권한을 내 놓는다. 그 다음, 그 무분기선 영역에서 후진추적을 수행한다. 그 프로세서가 무분기선 영역 목록에 상호 배타적 접근 권한을 부여받고 있는 동안에 다른 프로세서들은 그 프로세서가 부여받은 권한을 내놓을 때까지 기다려야 한다. 상호 배타적 접근 권한을 부여받거나 내놓을 때 걸리는 시간이 무시할 정도로 작지 않는 한 이러한 기다림은 비효율적인 성능을 보여줄 것이다. 본 논문에서는 이러한 상호 배타적 접근 방법 대신에 다음과 같은 작업 분할 기법을 사용한다. 주 프로세서는 작업을 n개의 무중복 부분 작업들(disjoint subtasks)로 분할하여 각각의 부분 작업을 자기 자신을 포함한 모든 프로세서들에 할당한다. 이러한 과정이 아주 신속하고 각각의 부분 작업이 잘 균형잡힐 경우 이 방법은 상호 배타성을 이용하여 동적 작업 분할하는 방식보다 더 효율적이다.

주 프로세서는 고장 목록을 스캔하여 단순-활동적 고장들과 복잡-활동적 고장들을 판별해 낸다. 단순-활동적 고장들에 대해서는 그 고장들의 지선들을 확인하여 그 지선들을 무분기선 영역 목록에 등록한다. 이때 일단 그 목록에 한 번 등록된 지선은 다시 등록되지 않도록 한다. 주 프로세서는 무분기선 영역 목록에 등록된 모든 지선들을 n개의 집합으로 나눈다. 각각의 집합은 동일한 숫자의 지선들을 포함하도록 한다. 주 프로세서는 각각의 집합을 모든 부 프로세서에 하나씩 할당한다. 모든 프로세서는 할당받은 집합에 포함된 지선들에 대하여 후진추적을 수행한다. 어떠한 프로세서도 다른 프로세서들과는 다른 무분기선 영역을 후진추적하기 때문에 프로세서들 간에 동일한 영역에 접근할 수 없다.

프로세서들이 할당받은 무분기선 영역의 후진추적 수행을 끝마쳤을 때, 어떤 무분기선 영역내의 적어도 하나의 고장이 그 무분기선 영역의 지선까지 전달되면 그 지선 위의 고장은 시뮬레이션되어야 한다. 이와 같은 방식에 의해 시뮬레이션되어야 할 지선 위의 고장들이 판별된다. 주 프로세서는 이런 모든 지선 위의 고장들을 n개의 집합으로 분할한다. 주 프로세서는 각각의 분할된 집합을 모든 프로세서들에 할당한다. 각각의 프로세서는 할당된 지선 위의 고장들을 병렬 고장시뮬레이션(parallel fault simulation)기법을 이용하여 시뮬레이션한다. 사용하는 멀티프로세서의 단어 크기가 w라고 하자. 어떤 프로세서에 할당된 지선 위의 고장들의 개수가 w개보다 많을 때에는 한 번 이상의 패스에 의해서 시뮬레이션하여야 한다. 어느 지선 위의 고장들이 어느 주요 출력단으로 전달되고 그 주요 출력단에서의 고장값과 무고장값이 X가 아니라면 그 지선까지 도달했던 단순-활동적 고장들은 검출된다. 만약 어느 지선 위의 고장이 어느 주요 출력단으로도 전달되지 않고 어느 제 2차 주요 출력단으로 전달된 경우는 그 지선까지 도달했던 단순-활동적 고장들은 그 제 2차 주요 출력단으로 전달되는 것이다. 이때의 고장효과와 제 2차 주요 출력단 번호는 고장목록에 저장된다. 각각의 프로세서는 서로 다른 지선 위의 고장들을 시뮬레이션하며 어떠한 지선 위의 고장들도 두 개의 집합에 중복되어 포함되지 않으며 각각의 프로세서는 자기 자신만의 부 계산목록과 모든 신호선의 고장값 저장을 위한 데이터구조를 가

지므로 서로 다른 프로세서가 동일한 기억장소를 접근하는 경우는 발생하지 않는다.

지선 위의 고장들의 분할과 시뮬레이션을 수행하는 방식과 마찬가지로, 주 프로세서는 복잡-활동적 고장들을 n개의 집합으로 분할한다. 모든 프로세서는 한 패스당 w개의 복잡-활동적 고장들을 그룹핑하여 병렬 고장시뮬레이션을 수행한다. 이 과정은 할당된 모든 복잡-활동적 고장들에 대한 시뮬레이션이 끝날 때까지 반복된다. 만약 어느 복잡-활동적 고장이 어느 주요 출력단에 도달되고 그 주요 출력단에서의 고장값과 무고장값이 X가 아니면 그 복잡-활동적 고장은 검출된다. 어느 복잡-활동적 고장이 어느 주요 출력단으로도 전달되지 않고 제 2차 주요 출력단으로 전달될 때는 그 고장효과와 제 2차 주요출력단 번호는 고장목록에 저장된다. 지선 위의 고장들의 시뮬레이션과 복잡-활동적 고장들의 시뮬레이션 사이에는 어느 동기화를 위한 barrier도 필요하지 않다. 어느 프로세서도 그것에 할당된 지선 위의 고장에 대한 시뮬레이션을 끝낸 후에 곧바로 그것에 할당된 복잡-활동적 고장들을 시뮬레이션한다.

Ⅳ. 실험 및 결과

본 논문에서 제안된 방법은 32bit 단어크기를 갖으며 공유기억 장치를 갖는 Sequent Symmetry 컴퓨터에서 C언어로 구현되었다. 그 컴퓨터는 8개의 프로세서로 구성되어 있으며, 각각의 프로세서는 자신만의 기억장치(local memory)로서 64KB의 RAM을 갖고 있다. ISCAS'89 벤치마크 순차회로[13]중에서 6개의 큰 회로들에 대해서 200개의 임의 입력 패턴(random input pattern)을 사용하여 시뮬레이션을 하였다. 실험결과로서 무고장값 시뮬레이션의 경과시간 및 무고장값 시뮬레이션 수행할 때의 프로세서 이용도, 고장 시뮬레이션의 총 경과시간, 작업분할을 위한 수행시간을 제외한 고장시뮬레이션의 총 경과시간 등을 사용하는 프로세서의 개수를 변화시키면서 측정하였다.

표 1은 사용된 프로세서의 숫자에 따른 무고장값 시뮬레이션의 경과시간을 보여준다. s38417과 s38584 회로에 대해서 8개의 프로세서가 사용되었을 때 가속화 비율은 약 3정도이며, 그 외의 다른 회로들에 대해서 가속화 비율은 미미한 수준이었다. 무고장값 시뮬레이션에서 각각의 논리게이트의 출력값을 계산하는 작업에 비해서 사건(event)들을 등록하고 처리하는 작업은 상대적으로 크고, 작업분할은 논리게이트의 출력값을 계산하는 작업에 대해서 주로 이루어지므로 가속화 비율은 크지 않을 것이다.

표 2는 무고장값 시뮬레이션을 수행할 때의 프로세서 이용도를 보여준다. 프로세서 이용도는 다음과 같이 계산될 수 있다. n개의 프로세서가 이용된다고 가

표 1. 무고장값 시뮬레이션의 경과시간

Circuit Name	Processors				
	1 [sec.]	2 [sec.]	4 [sec.]	6 [sec.]	8 [sec.]
s5378	17.9	13.1	9.7	13.2	9.0
s9234	9.8	8.1	7.8	8.2	9.1
s13207	14.4	11.0	9.7	10.1	11.0
s15850	37.1	27.1	22.2	21.4	21.8
s38417	82.1	53.5	37.5	35.1	30.9
s38584	156.5	99.4	68.1	79.1	51.8

표 2. 무고장값 시뮬레이션 수행동안의 프로세서 이용도

Circuit Name	Processors				
	1	2	4	6	8
s5378	100.00	98.22	94.63	91.17	88.09
s9234	100.00	94.43	84.46	76.71	70.20
s13207	100.00	96.98	91.12	85.80	81.06
s15850	100.00	97.81	94.03	89.48	86.12
s38417	100.00	99.16	97.36	95.76	94.06
s38584	100.00	99.64	98.94	98.18	97.41

표 3. 고장시뮬레이션의 총 경과시간

Circuit Name	Processors				
	1 [sec.]	2 [sec.]	4 [sec.]	6 [sec.]	8 [sec.]
s5378	553.5	340.6	225.1	223.6	164.7
s9234	1942.3	1230.6	830.7	765.1	530.8
s13207	2895.3	2100.8	1701.9	1571.1	1103.1
s15850	3350.3	1940.1	1150.3	1042.3	823.0
s38417	7625.2	4472.8	2831.2	2283.1	1768.7
s38584	10852.2	6703.5	5057.5	4760.5	4280.1

표 4. 가속화 비율

Circuit Name	Processors			
	2	4	6	8
s5378	1.63	2.46	2.48	3.36
s9234	1.58	2.34	2.54	3.66
s13207	1.38	1.70	1.84	2.62
s15850	1.70	2.69	3.34	4.31
s38417	1.70	2.69	3.34	4.31
s38584	1.62	2.15	2.28	2.54

정하자. 시간후레임 t에서 레벨 l에서 출력계산될 논리게이트의 숫자를 $e(t, l)$이라 하자. 어떤 특정한 하나의 프로세서를 제외한 모든 프로세서는 $[e(t, l)/n]$개의 논리게이트를 할당받는다.

그 특정한 프로세서는 $e(t, l)-(n-1)[e(t, l)/n]$개의 논리게이트만을 할당받게 된다. 다른 말로 말하면, 다른 프로세서들이 $n[e(t, l)/n]-e(t, l)$개의 논리게이트들의 출력계산을 하는 동안 그 특정한 프로세서는 대기상태에 있게 된다. 그러므로 무고장값 시뮬레이션 수행중의 프로세서 이용도 $\rho(n)$는 다음의 식에 의해서 표현된다.

$$\rho(n)=\frac{\Sigma_t\Sigma_l e(t, l)/n}{\Sigma_t\Sigma_l [e(t, l)/n]} \quad (1)$$

프로세서 이용도 $\rho(n)$은 이용 가능한 프로세서의 숫자가 증가할 수록 감소한다. 표 2에서 이와 같은 현상을 관찰할 수 있었다. s38417과 s38584회로에 대해서 8개의 프로세서가 사용되었을 때, 다른 회로들에 비해서 프로세서 이용도가 컸다.

표 3에서는 사용된 프로세서의 개수를 변경시키면서 고장시뮬레이션 수행의 총 경과시간을 측정한 결과를 보여준다. 표 4는 표 3의 결과를 이용하여 프로세서 1개만을 사용했을 때를 기준으로 하여 산출한 가속화 비율을 보여준다. 8개의 프로세서가 사용되었을 때 가속화 비율은 2.54~4.13 정도였다.

일반적으로 가속화 비율에 나쁜 영향을 줄 수 있는 중요한 요소로는 2가지가 있다. 알고리듬에 의한 영향과 하드웨어에 의한 영향이 그것이다. 알고리듬에 의한 영향으로서는 작업분할을 수행함으로써 기인되는 오버헤드와 균형이 잡히지 않은 분할된 작업량이

표 5. 작업분할을 위한 수행시간을 제외한 총 경과시간

Circuit Name	Processors				
	1 [sec.]	2 [sec.]	4 [sec.]	6 [sec.]	8 [sec.]
s5378	502.4	298.8	181.2	176.4	122.6
s9234	1846.1	1135.5	734.6	671.9	436.6
s13207	2773.9	1978.4	1580.3	1461.8	1019.8
s15850	3153.5	1787.6	1006.4	894.7	674.0
s38417	7271.7	4127.3	2481.7	1946.2	1418.2
s38584	10449.3	6359.3	4699.3	4405.3	3950.3

가속화 비율에 나쁜 영향을 줄 수 있다. 하드웨어에 의한 영향으로는 사용된 컴퓨터인 Sequent Symmetry의 버스 구조를 들 수 있다. Sequent Symmetry는 단일버스를 통해서 각각의 프로세서에 의해서 접근할 수 있는 공유 기억장치를 갖는다. 따라서 버스 contention이 가속화 비율에 나쁜 영향을 줄 수 있다. 작업분할에 의한 오버헤드가 가속화 비율에 얼마나 나쁜 영향을 주는 지를 조사하기 위해 고장시뮬레이션의 총 경과시간으로부터 작업분할 수행의 경과시간을 제외한 결과가 표 5에 주어진다.

표 6은 표 5를 근거로 계산한 가속화 비율을 보여준다. 8개의 프로세서가 사용되었을 때 가속화 비율은 2.54~4.13 정도이다. 작업분 오버헤드를 제외했을 때의 가속화 비율은 포함시켰을 때보다 0.10~0.86정도 증가하였다. 이것은 약 5~10%정도의 가속화 비율의 증가이다. 이 결과로부터 작업분할 오버헤드의 총 가속화비율에 대한 나쁜 영향은 5~10%정도라는 것을 알 수 있다.

표 6. 작업분할을 위한 수행시간을 제외했을 때의 가속화 비율

Circuit Name	Processors			
	2	4	6	8
s5378	1.68	2.77	2.85	4.10
s9234	1.63	2.51	2.75	4.23
s13207	1.40	1.76	1.90	2.72
s15850	1.76	3.13	3.52	4.68
s38417	1.76	2.93	3.74	5.13
s38584	1.64	2.22	2.37	2.65

V. 결 론

본 논문에서는 공유 기억장치를 갖는 범용멀티프로세서에서 고장시뮬레이션에 관한 알고리듬이 제안되었다. 고장시뮬레이션 자체가 갖고 있는 데이타 병렬성과 알고리듬 병렬성을 최대한 이용하기 위하여 작업분할 기법을 사용하였고 비동기식 시뮬레이션이 갖고 있는 문제점인 roll-back이나 deadlock이 없는 동기식 시뮬레이션 기법을 사용하였다. 중간단위 멀티프로세서(medium-grained multiprocessors)시스템에서는 수개에서 수 십개 정도의 프로세서를 갖는다. 이러한 시스템에서의 고장시뮬레이션은 프로세서의 숫자보다는 많은 데이타 병렬성을 가지므로 동기식 시뮬레이션 기법이 비동기식 시뮬레이션보다 좋은 성능을 보여 줄 수 있다.

ISCAS'89 벤치마크 순차회로들을 사용하여 실험을 하여 무고장값 시뮬레이션의 경과시간과 고장시뮬레이션의 총경과시간을 사용하는 프로세서의 개수를 변화시키면서 측정하였다. 무고장값 시뮬레이션 동안의 프로세서 이용도와 사용된 프로세서의 개수와의 관계도 연구되었다.

제안된 고장시뮬레이션 알고리듬은 작업분할기법의 오버헤드, 불균형적으로 분할된 작업량, 멀티프로세서 하드웨어의 단점 등에 의해 알고리듬의 가속화에 나쁜 영향을 받을 수 있다. 작업분할 기법의 오버헤드는 약 5~10%정도이다. 본 실험에서 사용된 Sequent Symmetry시스템을 64KB의 작은 로컬메모리를 갖기 때문에 많은 정보를 공유 기억장치로부터 얻어온다. 이 경우 시스템버스에서 많은 버스 contention이 발생하여 시스템 성능에 많은 나쁜 영향을 준다. 이러한 악 영향을 줄 수 있는 여러장애에도 불구하고 8개의 프로세서가 사용되었을 경우 평균적으로 3이상의 가속화 비율을 보여 준 것은 고장시뮬레이션 알고리듬의 병렬화를 위해서 범용 멀티프로세서의 사용은 희망적이라 할 수 있다. 로컬메모리가 비교적 큰 범용멀티프로세서를 사용하여 로컬메모리의 크기와 가속화 비율간의 관계를 연구하는 것은 차후의 연구 계획으로 남겨둔다.

참 고 문 헌

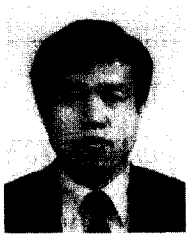
1. Abramovici, M., Levendel, Y. M. and Menon, P. R., "A logic simulation engine," *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 2, pp. 82-94, Apr. 1983.
2. Agrawal, P., Agrawal, V. D., Cheng, K-T., and Tutundjian, R., "Fault Simulation in a Pipelined Multiprocessor Environment," *Proc. 1989 Intl. Test Conf.*, pp. 727-734, August 1989.
3. Agrawal, A., and Bhattacharya, D., "CMP3F:A High Speed Fault Simulator for The Connection Machine," *Proc. 1990 Intl. Test Conf.*, pp. 410-416, September 1990.
4. Abrams, M. and Reynold, P. F., *Proc. of the 6th Workshop on Parallel and Distributed Simulation*, SCS, 1992.
5. Armstrong, D. B., "A Deductive Method of Simulating Faults in Logic Circuits," *IEEE Trans. Comput.*, vol. C-21, pp. 464-471, May 1972.
6. Arvind, D.K., Bagrodia, R., and Lin, Y.-B., *Proc. of the 8-th Workshop on Parallel and Distributed Simulation*, SCS, 1994.
7. Bagrodia, R., Jefferson, D., eds. *Proc. of the 7th Workshop on Parallel and Distributed Simulation*, SCS, 1993.
8. Bailey, M. L., Briner, J. V., jr., and Chamberlain, R. D., "Parallel Logic Simulation of VLSI Systems." *ACM Computing surveys*, 26(3):pp. 255-294, September 1994.
9. Bailey, M. L., Lin, Y.-B., eds., *Proc. of the 9th Workshop on Parallel and Distributed Simulation*, IEEE, 1995.
10. Beece, D. K., Papp, G., and Villante, F., "The IBM Verification Engine," *Proc. 25th ACM/IEEE Design Automation Conf.*, 1988.
11. Blank, T., "A Survey of Hardware Accelerators Used in Computer-Aided Design," *IEEE Design & Test of Computers*, vol. 1, pp. 21-39, August 1984.
12. Breuer, M. A. and Friedman, A. D., *Diagnosis & Reliable Design of Digital Systems*, Computer Sci-

ence Press, 1976.

13. Brglez, F., Bryan, D., and Kozminski, K., "Combinational Profiles of Sequential Benchmark Circuits," *Proc. IEEE Intl. Symp. Cir. & Sys.*, pp. 1929-1934, May 1989.
14. Chamberlain, R.D., "Parallel Logic Simulation of VLSI Systems," *Proc. 32nd Des. Autom. Conf.*, pp. 139-143, June 1995.
15. Chandy, K. M. and Misra, J., "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Comm. of the ACM*, vol. 24, no. 11, pp. 198-206, April 1981.
16. Comfort, J.C., "The Design of a Multi-microprocessor Based Simulation Computer-I," *Proc. 15th Annual Simulation Symposium*, pp. 45-53, March 1982.
17. Dennau, M. M., "The Yorktown Simulation Engine," *Proc. 19th Des. Autom. Conf.*, pp. 55-59, June 1982.
18. Duba, P.A., Roy, R.K., Abraham, J.A., and Rogers, W.A., "Fault Simulation in a Distributed Environment," *Proc. 25th Des. Autom. Conf.*, pp. 686-691, June 1988.
19. Hill, F.J., Abuelyamen, E., Huang, W.-K., and Shen, G.-Q., "A New Two Task Algorithm for Clock Mode Fault Simulation in Sequential Circuits," *Proc. 25th Des. Autom. Conf.*, pp. 583-586, June 1988.
20. Ishiura, N., Ito, M., and Yajima, S., "Dynamic Two-Dimensional Parallel Simulation Technique for High-Speed Fault Simulation on A Vector Processor," *IEEE Trans. CAD*, vol. 9, pp. 868-875, August 1990.
21. Jefferson, D., "Fast Concurrent Simulation Using the Time Warp Mechanism," *1985 Society for Computer Simulation Multiconferences*, San Diego, California, January, 1985.
22. Madisetti, V., Nicol, D., Fujimoto, R., eds. *Proc. of the SCS Multiconf. on Advances in Parallel and Distributed Simulation*, SCS, 1991.
23. Menon, P.R., Levendel, Y., and Abramovici, M., "Critical Path Tracing in Sequential Circuits," *Proc. Intl. Conf. CAD*, pp. 162-165, November 1988.
24. Menon, P.R., Levendel, Y., and Abramovici, M., "SCRIPT: A Critical Path Tracing Algorithm for Synchronous Sequential Circuits," *IEEE Trans. CAD*, vol. 10, pp. 738-747, June 1991.
25. Mueller-Thuns, R. B., Saab, D. G., Damiano, R. F. and Abraham, J. A., "VLSI Logic And Fault Simulation on General-purpose Parallel Computers," *IEEE Trans. CAD*, vol. 12, no. 3, pp. 446-460, March, 1993.
26. Narayanan, V. and Pitchumani, V., "A Parallel Algorithm for Fault Simulation on The Connection Machine," *Proc. 1988 Intl. Test Conf.*, pp. 89-93, September 1988.
27. Niermann, T. M., Cheng, W-T. and Patel, J. H., "PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator," *Proc. 27th Des. Autom. Conf.*, pp. 53-540, June 1990.
28. Noble, B. L., Peterson, G. D., Chamberlain, R. D., "Performance of Synchronous Parallel Discrete-event Simulation." *Proc. of 28th Hawaii Int'l Conf. on System Sciences*, vol. II, pp. 185-186, IEEE, 1995.
29. Özgüner, F. and Daoud, R., "Vectorized Fault Simulation on The Cray X-MP Supercomputer," *Proc. Intl. Conf. CAD*, pp. 198-201, November 1988.
30. Seshu, S., "On An Improved Diagnosis Program," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 76-79, 1965.
31. Smith, R. J., "Fundamentals of Parallel Logic Simulation," *Proc. 23rd ACM/IEEE Design Automation Conf.*, 1986.
32. Smith, E. J., Underwood, B. and Mercer, M. R., "An Analysis of Several Approaches to Circuit Partitioning for Parallel Logic Simulation." *Proc. Int'l Conf. on Computer Design*, pp. 664-667, IEEE, 1987.
33. Song, O and Menon, P. R., "3-Valued Trace-

Based Fault Simulation for Synchronous Sequential Circuits," *IEEE Trans CAD*, vol. 12, no. 8, pp. 1418-1424, September 1993.

34. Soule, L. and Blank, T., "Statistics for Parallelism And Abstraction Level in Digital Simulation," *Proc. 24th ACM./IEEE Des. Autom. Conf.*, 1987.
35. Soule, L., and Blank, T., "Parallel Logic Simulation on General Purpose Machines," *Proc. 25th ACM/IEEE Des. Autom. Conf.*, pp. 166-171, 1988.
36. Soule, L. and Gupia, A., "An Evaluation of The Chandy-Mirsa-Bryant algorithm for Digital Logic Simulation." *Proc. 6th Workshop on Parallel and Distributed Simulation*, pp. 129-138, SCS, 1992.
37. Sporrer, C. and Bauer, H., "Corolla Partitioning for Distributed Logic Simulation of VLSI Circuits" *Proc. 7th Workshop on Parallel and Distributed Simulation*, pp. 86-92, SCS, 1993.
38. Ulrich, E.G. and Baker, T., "Concurrent Simulation of Nearly Identical Digital Networks," *Computer*, vol. 7, pp. 39-44, April 1974.
39. Walker, P. A. and Ghosh, S., "Asynchronous, Distributed Event Driven Simulation Algorithm for Execution of VHDL on Parallel Processors," *Proc. 32nd Des, Autom. Conf.*, pp. 144-150, June 1995.
40. Wang, D. T., "Properties of Faults and Criticalities of Values Under Tests for Combinational Networks," *IEEE Trans. Computers*, vol. C-24, no. 7, pp. 746-750, July 1975.
41. Wang, X., Hill, F. J., and Mi, Z., "A Sequential Circuit Fault Simulation by Surrogate Fault Simulation," *Proc. 1989 Intl. Test Conf.*, pp. 9-18, 1989.
42. Wong, K. F., Franklin, M. A., Chamberlain, R. D., Shing, B. L., "Statistics in Logic Simulation," *Proc. 23rd ACM/IEEE Des. Autom. Conf.*, 1986.



송 오 영(Ohyoung Song) 정회원
서울대학교 전기공학과 공학사
한국과학기술원 전기 및 전자공학과 공학석사
미국 Massachusetts주립대 공학박사
국방부기술연구 사무관
Intergraph, Electronics 수석연구원
IBM, Microelectronics 수석연구원
삼성전자 수석연구원
현재 : 중앙대학교 전기전자제어공학부 조교수