□ 기술해설 □

병렬분산 이산사건 시뮬레이션

한국과학기술원 성영락* · 김탁곤** · 박규호**

목 차

1. 서 론

최근 들어 다수의 프로세서들로 구성된 병렬 컴퓨터를 이용한 병렬 이산사건 시뮬레이션(Parallel Discrete Event Simulation)에 대한 관심이 크게 증가하고 있다. 이것은 전자공학, 전산학, 산업공학, 경제학 등 많은 시뮬레이션 응용분야에서 규모가 크고 복잡한 시스템을 순차적 컴퓨터를 사용하여 시뮬레이션할 때 소요되는 엄청난 시간의 소모를 줄이기 위해서이다. 그러나 이산사건 시스템의 시뮬레이션은 구조적 특성이 매우 비정형적이며, 입력 데이타에 따른 변화가 크므로 벡터프로세싱 컴퓨터에 의한 병렬처리는 적합하지 않다.

본 논문에서는 이산사건 시스템의 병렬 시뮬레이션 알고리즘에 대해서 살펴본다. 대부분의 병렬 이산사건 시뮬레이션에서는 하나의 시뮬레이션 프로그램을 여러 개의 프로세스로 나눈다. 그리고 그 프로세스들 간에 공유하는 변수들을 제거하여 프로세스들이 각기 다른 프로세서에서 동시에 수행될 수 있도록 한다. 병렬 시뮬레이션 알고리즘은 크게 보수적(conservative) 알고리즘과 낙관적(optimistic) 알고리즘으로 분류된다. 보수적 알고리즘에서는 프로세스들이 전체 시뮬레이터의 시뮬레이션 시계를 조사하여 사건들을 항상 시뮬레이션 시각의 순서대로 처리한다. 반면, 낙관적 알고리즘에서는 프로세스들이 자신의 정보만으로 도착한 사건들의 처리 순서를 정한다. 그러므로 그 프로세스의 정보가 틀릴 경우 오류가 발생한다. 이 경우 그 프로세스는 롤백(rollback)을 통하여 잘못된 사건의 처리를 무효화 시킨 후 시뮬레이션 시각의 순서에 따라 사건들을 다시 처리한다. 그러므로 낙관적 알고리즘에서는 각 프로세스에서의 시뮬레이션 시계가 진행하는 경우도 있지만 롤백에 의해 과거로 후퇴하는 경우도 발생한다. 대표적인 보수적 시뮬레이션 알고리즘으로는 Chandy-Misra 알고리즘[1]이 있고, 낙관적 시뮬레이션 알고리즘으로는 Time-Warp 알고리즘[2]이 있다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 병렬 시뮬레이션이 어려운 이유에 대해 설명하고, 병렬 시뮬레이션의 결과가 순차적 시뮬레이션의 결과와 같기 위한 조건에 대해 언급한다.

*정회원
**종신회원

3장에서는 보수적인 알고리즘에 대해서 Chandy-Misra 알고리즘을 중심으로 설명하고 관련된 연구에 대해서 설명한다. 4장에서는 낙관적인 알고리즘인 Time-Warp 알고리즘의 자료구조와 메모리 관리에 대해서 설명하고 관련된 연구에 대해서 살펴본다. 또, 수학적인 형식론에 기반한 이산사건 모델의 병렬 시뮬레이션의 한 예로서 5장에서는 DEVS(Discrete Event System Specification) 형식론을 위한 병렬 시뮬레이션에 대해서 다룬다. 마지막 6장은 결론이다.

2. 병렬 시뮬레이션에서의 문제점

병렬 시뮬레이션에서는 실제 시각과 시뮬레이션 시각을 나타내는 두 가지 시계가 존재한다. 병렬 시뮬레이션이 어려운 이유는, 각 프로세스에 사건메시지들은 실제 시각의 순서대로 도착하는 데에 비해, 그 사건들은 스케줄된 발생하는 시뮬레이션 시각의 순서대로 처리되어야 하기 때문이다.

일반적으로 사건중심(event based) 방식의 순차적 시뮬레이터는 두 개의 중요한 자료구조를 가진다. 첫째는 시스템의 상태를 기술하는 상태변수들이고, 둘째는 스케줄되었으나 현재까지 실행되지 않은 사건들을 관리하는 사건리스트(event list)이다. 사건은 시뮬레이션되는 시스템의 상태가 바뀌는 것을 의미한다. 각 사건은 그 변화가 일어나는 시각을 표시하는 시간표(timestamp)를 가진다. 순차적 시뮬레이터는 사건리스트에서 시간표의 시각이 가장 작은 사건(제일 먼저 처리되어야 할 사건)을 꺼내어 그것을 처리하는 과정을 반복한다. 이때 사건을 처리하는 과정에서 새로운 사건들이 발생되어 사건리스트에 추가됨으로써 시뮬레이션이 진행된다. 이 방법에서 핵심적인 것은 시뮬레이터는 항상 사건리스트에서 시간표의 시각이 가장 작은 사건(E_{min})을 선택한다는 것이다. 이것은 만약 사건들의 시각표의 시각에 상관없이 사건리스트에 들어온 순서대로 사건들을 처리할 경우, 시간표의 시각이 큰 사건이 먼저 처리되는 경우가 발생하여 E_{min}의 계산에 영향을 주게 되기 때문이다. 즉 미래의 사건이 과거의 사건에 영향을 주는 일이 생기는 것이다. 이는 실제 상황에서는 불가능한 일로서 시뮬레이션 과정에서도 반드시 피해야 한다. 이런 오류를 전역 인과성 오류(global causality error)라고 한다. 또 반드시 그런 오류를 피해야 한다는 조건을 전역 인과성 조건(global causality constraint)라고 한다.

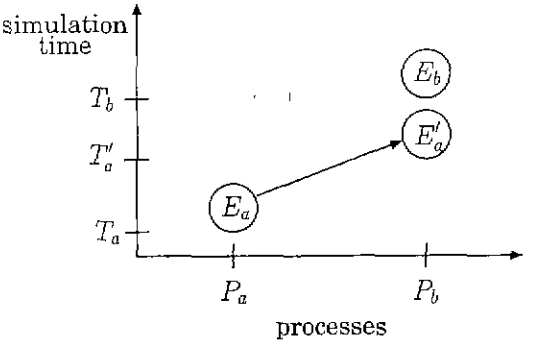

그림 1 전역 인과성 조건

이제 위의 사건중심방식에 기반한 시뮬레이션 프로그램을 병렬화하는 것에 대해 살펴보자. 우선 하나의 시뮬레이션 프로그램을 여러 개의 프로세스로 나눈다. 그리고 그 프로세스들 간에 공유하는 변수들을 제거하여 그 프로세스들이 각기 다른 프로세서에서 동시에 수행될 수 있도록 한다. 그러나 프로세스들이 그냥 사건들을 처리할 경우 인과성 오류가 발생한다. 예를 들어 그림 1의 경우를 살펴보자. 두 프로세스 P_a와 P_b의 사건리스트에는 각각 사건 E_a와 E_b가 들어 있고, E_a와 E_b의 시간표의 시각은 각각 T_a과 $T_b(>T_a)$이다. 그리고 P_a가 E_a를 처리하면 새로운 사건 E'_a이 발생하여 P_b에게 전달된다. 이때 E'_a의 시간표는 $T'_a(<T_b)$이다. 이 경우 E'_a은 반드시 E_b보다 먼저 수행되어야 한다. 그렇지 않을 경우 인과성 오류가 발생한다. 그러므로 전역 인과성 조건이란 사건들이 처리되어야 할 순서를 지정하거나 규정하는 것이다.

병렬 시뮬레이션 알고리즘은 크게 보수적(conservative) 알고리즘과 낙관적(optimistic) 알고리즘으로 분류된다. 보수적 알고리즘에서는

모든 프로세스가 항상 전역 인과성 조건이 엄격하게 지켜지면서 시뮬레이션을 진행하는 것에 비해, 낙관적 알고리즘에서는 우선 사건을 처리한 뒤에 전역 인과성 조건이 어긋나면 재처리하는 방법을 사용한다.

3. 보수적인 알고리즘

보수적인 알고리즘에서는 항상 전역 인과성 조건이 엄격하게 지켜진다. 그러므로 각 프로세스는 어떤 사건을 처리할 때 추후에 그 사건의 시간표의 시각보다 작은 시각의 시간표를 갖는 사건이 발생되지 않는다는 것이 보장되어야만 그 사건을 처리한다. 전체 병렬 시뮬레이터에 걸쳐 제일 작은 시간표의 시각을 갖는 사건은 언제나 안심하고 처리할 수 있다. 그러나 다른 사건들은 안전하게 처리되는 것을 보장할 수 없다. 이 사건들을 안전하게 처리하기 위해선 시뮬레이션 어플리케이션에 대한 사전지식이 필요하다. 그런 사전지식이 없을 경우에는 제일 작은 시간표의 시각을 갖는 사건을 처리하는 프로세스가 다른 모든 프로세스들에게 그 시각과 같은 시간표의 시각을 갖는 사건을 보내지 않는다는 것이 보장되어야 한다. 즉 시뮬레이션 어플리케이션에 대한 사전지식이 없이는 제일 작은 시간표의 시각을 갖는 사건외에 사건들은 안전하게 처리될 수 없다는 것이다. 그러므로 보수적인 알고리즘에서 각 프로세서는 시뮬레이션 어플리케이션에 대한 사전지식을 이용하여 안전하게 처리될 수 있는 사건들의 집합을 결정하고 처리하는 과정을 반복한다.

3.1 Chandy-Misra 알고리즘

Chandy-Misra 알고리즘은 가장 널리 알려진 보수적인 알고리즘이다[1]. 이 알고리즘에서는 시뮬레이션 대상 시스템이 방향그래프(directed graph)로 표현된다. 그 그래프에서 노드들은 대상 시스템의 각 부분들을 표현하는 프로세스들을 나타내고 유향에지(arc)들은 프로세스들 간의 통신채널들을 나타낸다. 각 프로세스는 사건들을 그 사건들이 처리되어야 할 시간의 순서대로 처리한다. 프로세스 간의 통신은 어떤 프로세스가 다른 프로세스의 사건을 스케줄할 때 발생한다. 이것은 시간표가 붙은 메시지들을 통신함으로써 이루어진다. 또 각 통신채널에는 그 채널을 통하여 최종적으로 전달된 사건의 시간표의 시각이 기록된다. Chandy-Misra 알고리즘이 보수적 알고리즘인 이유는 어떤 프

```
Chandy_Misra()
1  T_i ← 0                    ▷ Initialize simulation clock
2  while simulation completion criterion is not met
       ▷ simulate up to T_i
3      while the simulation clock is smaller than or equal to T_i
4          remove an event message from the event list;
5          process the event;
6          send each output message along the appropriate communication channel;
7      end while
       ▷ receive messages until T_i is changed
8      T_i' ← T_i;
9      while T_i' = T_i
10         wait to receive a message along all incoming channels;
11         insert the received message into the event list;
12         T_i ← the minimum over all incoming channel clock value;
13     end while
14 end while
```

그림 2 Chandy-Misra 알고리즘

로세스에서 시간표의 시각이 t인 메시지는 미래에 도착할 메시지의 시간표가 t보다 작지 않다는 것이 확인될 때까지 처리되지 않기 때문이다. 그러므로 Chandy-Misra 알고리즘에서는 프로세스의 시뮬레이션 시계나 통신채널의 시각이 되돌려 지는 일은 발생하지 않는다.

그림 2는 Chandy-Misra 알고리즘이다. 여기서 T_i는 Chandy-Misra 알고리즘에 따라 시뮬레이션이 안전하게 진행될 수 있는 시각이다. 각 프로세스는 T_i를 정하면 자신의 시뮬레이션 시계가 T_i가 될 때까지 자신의 사건리스트에서 사건들을 꺼내어 처리한다. 이때 발생되는 새로운 사건들은 외부로 향하는 통신채널을 통하여 다른 프로세스들에게 전달된다. 이 과정이 끝나면 각 프로세스는 자신으로 향한 통신채널들의 시각이 진행되기를 기다린다. 그래서 계속해서 자신을 향한 통신채널에 새로운 사건이 도착할 때마다 그 사건을 자신의 사건리스트에 집어넣고 그 통신 채널의 시각도 갱신한다. 그러던 중 자신을 향한 모든 통신채널의 시각 중에서 최소값이 T_i보다 크게 되면 T_i를 그 시각까지 진행시키고 새롭게 사건들을 처리하게 된다.

Chansy-Misra 알고리즘에서는 두가지의 문제가 발생한다. 첫째 메모리의 소모로 인해 버퍼가 오버플로우되는 것이고, 둘째 교착상태(deadlock)이 발생하는 것이다. 만약 Chandy-Misra 알고리즘에서 시뮬레이션 시스템을 표현한 directed 그래프가 사이클을 가지고 있다면 프로세스들이 교착상태에 빠지는 것을 피할 수 없다.

그동안의 연구에서 교착상태를 회피하거나 복구하기 위한 많은 알고리즘들이 제안되었다. 대표적인 알고리즘으로 Misra의 Null 메시지를 이용한 알고리즘이 있다[3, 4]. Null 메시지란 단순히 시스템내의 시뮬레이션 시간을 진행시키기 위해 사용되는, 내용을 포함하지 않은 메시지를 말하는 것으로 시뮬레이션 대상이 되는 시스템에서는 존재하지 않는 것이다. 어떤 프로세스가 교착상태에 빠져 있다고 간주될 경우, 그 프로세스는 Null 메시지를 보내어 전체 시뮬레이터의 시뮬레이션 시간을 진행시켜 교착상태를 해제하고 실제의 사건메시지들을 처

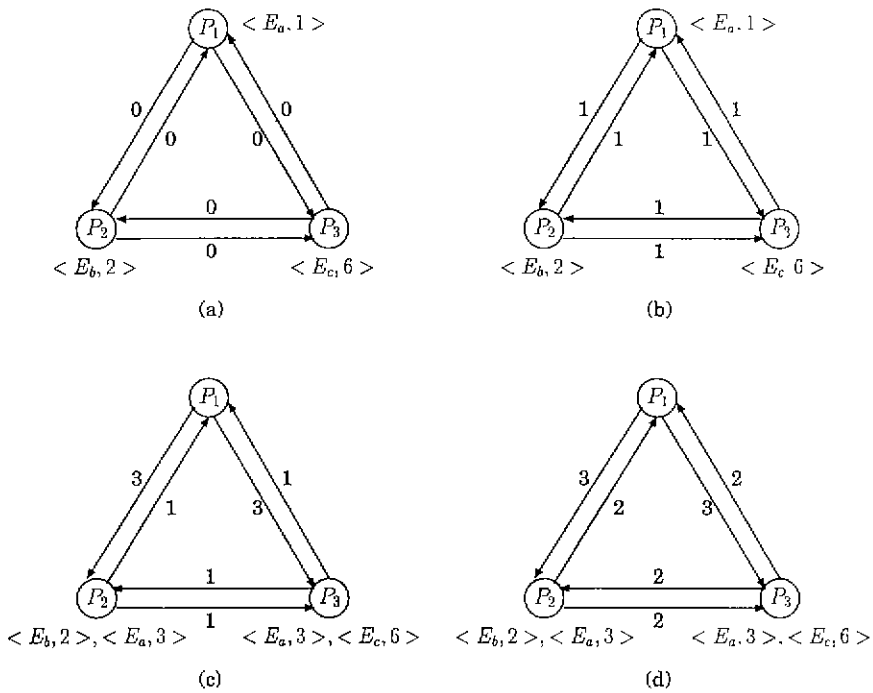

그림 3 Chandy-Misra 알고리즘의 예

리하게 한다.

그림 3(a)와 같이 세 개의 프로세스로 구성된 시스템의 경우를 살펴보자. 이 시스템에서 각 프로세스는 어떤 사건을 처리하면 다른 두 프로세스에게 메시지를 보내어 새로운 사건을 스케쥴하게 한다. 그러므로 통신형태는 단순히 양방향성 링구조이다. 각 프로세스는 사건을 관리하는 리스트를 가진다. 그림에서 $\langle E_a, 1\rangle$로 표시한 것은 시뮬레이션 시각이 1 일때 사건 E_a가 처리되도록 스케쥴되었음을 말한다. 그리고 유향에지(arc)에 표시된 값은 해당 통신채널을 통해서 최종적으로 전송된 메시지의 시간표의 시각을 의미한다. 이 시스템에서 각 프로세스는 하나의 사건을 처리하는 데에 최소한 1 만큼의 시간을 소모하는 것으로 가정한다. 또 통신채널의 시각은 0으로 초기화된 것으로 가정한다. 이제 그림 2의 알고리즘으로 시뮬레이션해 보자. 각 프로세스는 T_i를 0으로 초기화한 뒤에 자신의 사건리스트에서 시각표의 시각이 0보다 크지 않은 사건이 존재하는 지를 살펴본다. 하지만 그런 프로세스는 존재하지 않으므로 자신을 향한 통신채널의 시각이 증가하기를 기다린다. 모든 프로세스들이 서로 상대방이 채널의 시각을 바꿔주기를 기다리므로 전체 시뮬레이터는 교착상태에 빠지게 된다. 그런데 자세히 살펴보면 각 프로세스는 최악의 경우에 시간표의 시각이 0인 사건메시지를 받더라도 최소한 1 만큼의 시간 동안은 처리해야 하므로 시간표의 시각이 1보다 작은 메시지는 절대로 발생할 수 없다. 이 정보를 이용하여 각 프로세스는 시간표의 시각이 1인 Null 메시지를 만들어 통신채널로 보낸다. 그 결과 그림 3(b)처럼 모든 통신채널의 시각은 1이 된다. 이 경우에도 P_2와 P_3는 여전히 사건리스트에 있는 사건들의 시간표의 시각이 통신채널의 시각보다 크므로 사건을 처리할 수 없지만, P_1의 경우에는 사건 E_1을 처리할 수 있다. 만약 P_1이 E_1을 처리하는 데에 걸리는 시뮬레이션 시간이 2로 가정하면, P_1이 사건처리를 끝마친 후의 전체 시뮬레이터는 그림 3(c)처럼, P_1에서 외부로 향하는 통신채널의 시각은 3이 되고, P_2와 P_3의 사건리스트에는 $\langle E_a, 3\rangle$이라는 사건이 추가된다. 이제 P_1은 사건리스트가 비어서 시뮬레이션을 진행할 수 없고, P_2와 P_3는 서로 간의 통신채널의 시각이 여전히 1로 남아 있으므로 새로운 사건메시지가 도착하였지만 여전히 아무 사건도 처리할 수 없다. 결과적으로 전체 시뮬레이터는 새로운 교착상태에 빠진다. 그것을 해결하기 위하여 P_2와 P_3는 새로운 Null 메시지를 만들어서 보내게 된다. 현재 통신채널의 시각의 최소치가 1이므로 새로 발생된 Null 메시지의 시간표의 시각은 2가 된다. 그 결과 전체 시뮬레이터는 그림 3(d)처럼 되고 P_2는 사건 $\langle E_b, 2\rangle$를 처리할 수 있게 된다.

3.2 Chandy-Misra 알고리즘에 관련된 연구

그 동안 Chandy-Misra 알고리즘의 성능을 향상시키기 위해서 여러 알고리즘들이 제안되었다. Chandy-Misra 알고리즘의 성능에 가장 큰 영향을 주는 것은 Null 메시지의 수이다. 그림 3의 경우에서 각 프로세스에서 하나의 사건을 처리하는 데에 걸리는 시간의 최소값이 매우 작은 경우를 생각해 보자. 예를 들어 만약 그 시간이 0.1이라면 통신채널의 시각을 1 만큼 진행시키기 위해서는 10 개의 Null 메시지가 필요하다. 만약 그 시간이 더 작다면 훨씬 더 많은 Null 메시지들이 필요할 것이다. 이런 경우에 Null 메시지의 수를 줄이기 위하여 Carrier Null 메시지가 도입되었다[5]. Carrier Null 메시지에는 기존의 Null 메시지의 정보외에 메시지가 전달된 프로세스들의 이름과 사건리스트의 최초 사건의 시간표의 시각들이 기록된다. 그래서 각 프로세스들은 추가된 정보를 이용하여 시뮬레이션 시계를 빠르게 진행시킬 수 있다.

또 De Vries[6]는 시뮬레이션 시스템을 나타내는 directed 그래프를 feedforward 네트워크와 feedback 네트워크들로 분리하였다. 그리고 각각의 경우에 대해서 Chandy-Misra 알고리즘에서의 Null 메시지의 수를 줄이기 위한 알고리즘을 제시하였다.

4. 낙관적 알고리즘

낙관적인 알고리즘은 검출 및 회복(detection and recovery) 방식을 기반으로 한다. 즉 우선 낙관적으로 사건들을 처리하고 나중에 인과성 오류가 검출되면, 롤백(rollback)을 이용하여 회복하는 방식이다. 낙관적인 알고리즘은 시뮬레이션 어플리케이션에 대한 아무런 사전지식을 필요로 하지 않기 때문에, 순차적인 시뮬레이션에서 제공하는 일반화되고 어플리케이션에 구애받지 않는 동기화 알고리즘처럼 일반화된 동기화 알고리즘(synchronization algorithm)을 제공한다.

가장 널리 알려진 낙관적인 시뮬레이션 알고리즘으로 가상시간이론[7]에 기반을 둔 Time Warp 알고리즘이 있다[2]. 여기서 가상시간이란 시뮬레이션 사건을 의미한다. 모든 낙관적인 방법은 오류를 발견하고 정정하는 체계를 가지고 있다. Time Warp에서 각 프로세스는 외부에서 사건 메시지가 도착할 때마다 인과성 오류가 발생하는 지를 검출한다. 만약 새로 도착한 사건 메시지의 시간표의 시각이 그 프로세스의 시뮬레이션 시계보다 빠른 경우 인과성 오류가 발생한다. 이때 그 메시지를 지각메시지(straggler message)라고 한다. 지각메시지가 발생한 경우 프로세스는 롤백을 통하여 자신이 처리한 사건 중에서 지각메시지보다 시간표의 값이 큰 사건들로 인해 야기된 변화들을 원상복구시켜야 한다.

어떤 사건을 처리하게 되면 두가지의 변화가 생긴다. 첫째 그 사건을 처리한 프로세스의 상태가 변화하게 되며, 둘째 사건처리 중에 다른 프로세스들에게 새로운 사건 메시지들을 보내게 된다. 프로세스 상태변화에 대한 롤백은 프로세스가 주기적으로 프로세스의 상태를 저장해 두었다가 롤백이 필요할 때 저장된 상태를 복구하는 것으로 이루어진다. 또 발생된 메시지들을 제거하기 위해서는 발생된 메시지와 반대가 되는 메시지를 발생시켜 그들을 상쇄시키는 방법을 사용한다. 이 반대가 되는 메시지를 반대메시지(anti-message) 혹은 제거메시지(annihilation message)라고 한다. 반대메시지는 원래의 사건메시지와 동일한 내용을 가지지만 극성이 반대인 메시지를 말하는 것으로 원래의 사건메시지와 만나면 두 메시지는 상쇄된다. 이때 만약 그 원래의 사건메시지가 처리되기 전이라면 단순히 그 메시지들을 제거하고, 처리된 후라면 그 사건의 처리를 롤백을 통하여 무효화시킨다. 그 롤백과정에서 또 새로운 롤백이 발생할 수 있다. 그 일련의 롤백과정을 통하여 인과성 오류를 발생시킨 사건은 완전히 무효화된다.

4.1 구 조

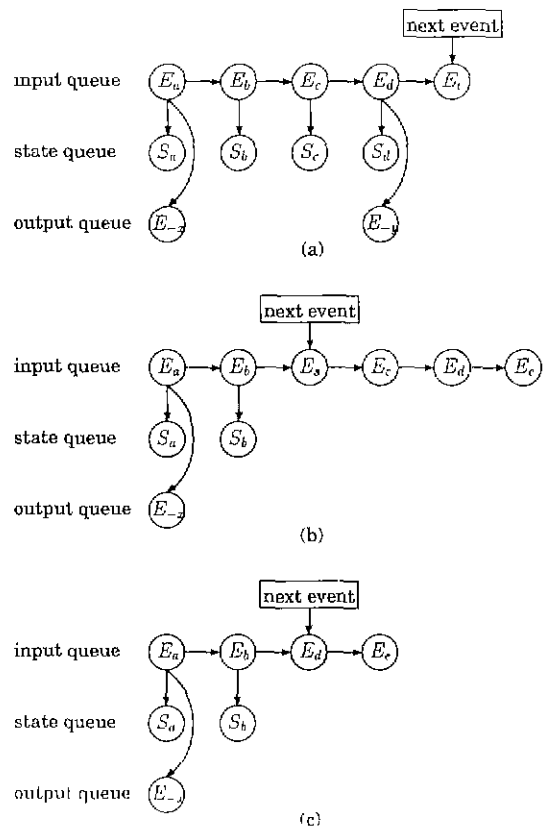

그림 4 Time Warp에서의 자료구조

Time Warp의 각 프로세스에는 그림 4처럼 입력큐(input queue), 상태큐(state queue), 출력큐(output queue)가 존재한다. 입력큐는 자신에게 도착한 사건메시지들을 시간표의 시각의 순서대로 정리한 것이다. 이것은 순차적

```
Time_Warp()
1  while simulation completion criterion is not met
2      while there is no unprocessed events in the input queue
3          wait until such an event exist;
4      end while
5      read the event referenced by the next event pointer from the input queue;
6      save the variables associated with the event;
7      save the state of the process into the state queue;
8      process the event;
9      for each output message generated during the processing
10         save the negative copy of the message into the output queue;
11     end for
12     advance the next event pointer;
13 end while
```

그림 5 Time Warp 알고리즘

```
Interrupt_Handler(Message msg)

1  switch the type of the incoming message
2      case positive message in the future:
3          insert msg into the input queue;
4      case anti-message in the future:
5          remove the corresponding positive message from the input queue;
6          annihilate the two messages;
7      case positive message in the past(straggler):
           ▷ initiate a rollback operation
8          restore the state associated with the message following msg in the input queue
9          for each processed event message following msg in the input queue
10             discard the state associated with the event;
11             send the corresponding anti-message in the output queue;
12         end for
13         set the "next event" pointer to msg;
14     case anti-message in the past.
15         remove the corresponding positive message from the input queue;
16         annihilate the two messages;
17         restore the state associated with the message following msg in the input queue
18         for each processed event message following msg in the input queue
19             discard the state associated with the event;
20             send the corresponding anti-message in the output queue;
21         end for
22         set the "next event" pointer to the message immediately following the moved positive
           message;
23 end switch
```

그림 6 메시지의 도착에 의한 인터럽트의 처리

시뮬레이터의 사건리스트와 유사하지만 전체 시뮬레이터에서 쓰이는 사건들이 아니라 하나의 프로세스에 관한 사건만을 저장하고, 이미 처리한 사건들도 저장한다는 점에서 다르다. 그림 4(a)에서 E_a, E_b, E_c, E_d는 이미 처리가 끝난 사건들이고 E_e는 아직 처리되지 못한 사건이다. 상태큐는 프로세스가 자신의 상태를 기록하여 저장하기 위한 자료구조이다. 설명을 단순화하기 위하여 각 프로세스는 한 사건을 처리하기 시작할 때마다 자신의 상태를 저장하는 것으로 가정하자. 그림 4(a)처럼 이미 처리가 끝난 사건들(E_a, E_b, E_c, E_d)에는 상태저장 정보에 대한 포인터가 연결된다. 출력큐는 그 프로세스에서 발생한 사건메시지들에 대한 반대메시지들을 저장한다. 출력을 발생시킨 사건들은 그 출력 메시지의 반대메시지에 대한 포인터가 연결된다. 그림 4(a)의 경우 사건 E_a와 E_d는 각각 E_x와 E_y를 발생하였으므로 반대메시지 E_{-x}와 E_{-y}에 대한 포인터를 가지고 E_b와 E_c 출력을 발생하지 않았으므로 포인터를 가지지 않는다.

각 프로세스에서의 알고리즘은 그림 5와 같다. 우선 입력큐에서 아직 처리되지 않은 새로운 사건이 있는지를 확인한다. 이때 만약 처리되지 않은 사건이 없을 경우에는 새로운 사건이 도착할 때까지 기다린다. 다음으로 추후에 롤백이 일어날 경우에 대비하여 그 사건에 관련되는 변수들과 프로세스의 상태를 상태큐에 저장한다. 그리고 그 사건을 처리한다. 이때 발생되는 출력메시지에 대해서는 반대메시지를 만들어 출력큐에 저장한다. 그리하여 모든 처리가 끝나면 입력큐의 "next event" 포인터를 진행시키고 처음부터 다시 시작한다. 만약 현재 처리한 사건이 입력큐의 마지막 사건이었으면 입력큐의 "next event" 포인터를 시간표의 값이 무한대인 가상메시지를 가리키도록 해서 시뮬레이션이 진행되는 것을 막는다.

메시지가 도착할 때마다 그림 5의 알고리즘은 인터럽트된다. 어떤 프로세스에 도착하는 메시지는 메시지의 극성과 시간표에 따라 4 종류로 분류된다. 각 메시지가 도착했을 때 Time Warp 프로세스는 그림 6의 알고리즘에 의해서 처리한다.

예를 들어 그림 4의 경우를 살펴보자. 그림 4(a)의 상황에서 지각메시지 E_s가 도착하였다. 이때 만약 $E_b.time_stamp < E_s.time_stamp < E_c.time_stamp$라면 이미 처리된 메시지 E_c, E_d는 롤백된다. 그 결과 프로세서의 상태는 S_c로 복구되고, E_{-y}가 다른 프로세스들에게 전달되어 E_y에 의한 영향을 상쇄시킨다. 이때 그 프로세스의 자료구조는 그림 4(b)처럼 된다. 그리고 새로이 E_{-s}가 도착하면 이젠 E_s와 상쇄된다. 최종적으로 그 프로세서의 자료구조는 그림 4(c)처럼 된다.

4.2 메모리 및 입출력 관리

앞서 설명한 알고리즘에는 크게 두 가지 문제점을 가진다. 첫째, 메모리의 소모이다. 각 프로세스에서는 미래에 발생할 지도 모르는 롤백에 대비하여 앞으로 처리할 메시지 외에 이미 처리가 끝난 메시지도 보관한다. 또 수시로 프로세스의 상태와 출력메시지의 반대메시지도 보관한다. 그러므로 만약 발생된 모든 사건들에 대한 정보들을 그냥 둔다면 메모리는 곧 고갈될 것이다. 둘째, 롤백이 불가능한 동작들 예를 들어 입출력을 처리하는 문제이다. 그러므로 그런 동작들이 수행되고 나면 그에 관련된 사건은 롤백되어서는 안된다. 이 문제들을 해결하기 위하여 Time Warp에서는 화석제거(Fossil Collection) 체계를 가지고 있다. 미래에 있을 롤백에 의해 어떤 프로세스의 시계가 어느 시각 이전으로는 되돌려지지 않는다고 가정하자. 이 경우에 그 시각 이전의 시간표를 가진 사건들에 관련된 자료구조들은 지울 수 있으며 그 시각까지의 입출력 동작들도 수행할 수 있다. 그러므로 앞의 문제들을 해결할 수 있다.

이제 그 시각을 구해보자. 롤백은 과거의 시각을 가지는 메시지를 받았을 때 일어난다. Time Warp에서 어떤 프로세스에서 발생하는 메시지의 시간표의 시각은 그 시점에서 그 프로세스의 시뮬레이션 시계보다 항상 느리다. 그러므로 전체 시뮬레이터에서 프로세스들의 시뮬레이션 시계의 시각 중에서 제일 늦은 시각을 구하면 그 시각 이전의 시간표를 가지는 메시지는 발생하지 않는다. 그러므로 그 시각 이

전의 사건에 대한 롤백은 발생하지 않는다. 이 때 그 시각을 전역가상시각(global virtual time)이라 한다.

전역가상시각을 구하는 과정은 다음과 같다. 시뮬레이션이 진행되는 도중 어떤 프로세스에서 메모리 오버플로우가 예상되면 그 프로세스는 전역가상시각의 계산을 요구하는 메시지를 다른 프로세스들에게 보낸다. 그러면 각 프로세스들은 자신의 시뮬레이션 시계의 시각을 전역가상시각의 계산을 요구한 프로세스에게 보낸다. 이때 전역가상시각은 전체 시뮬레이터의 프로세스들의 시뮬레이션 시계의 시각들 중에서 최소값이다. 전역가상시각이 구해지면 그것은 다시 모든 프로세스에게 전달되어 각 프로세스들은 자신의 입력큐에서 전역가상시각보다 작은 시각을 갖는 사건들을 제거하고 관련된 입출력 동작을 수행한다.

4.3 Time Warp 알고리즘에 관련된 연구

그 동안에 많은 연구에서 Time Warp 알고리즘의 성능을 개선하기 위한 방법들이 제안되었다. Time Warp 알고리즘의 성능에 가장 큰 영향을 주는 것은 롤백에 의한 오버헤드이다. 지금까지 제안된 롤백의 오버헤드를 줄이는 방법은 크게 네 가지로 구분된다.

첫째, Time Warp 알고리즘의 낙관성을 더욱 높이는 방법이다. Gafni[8]는 프로세스가 롤백을 할 때 반대메시지를 보내는 시점을 미루는 방법을 제안하였다. 이 알고리즘에서는 어떤 프로세스에서 롤백이 일어나면 반대메시지를 보내는 것을 유보하고 그 프로세스의 상태변수들만 롤백시켜 다시 계산한다. 그래서 만약 그 계산 결과가 롤백되기 이전과 같을 경우에는 반대메시지를 보내지 않고, 그렇지 않을 경우에는 반대메시지와 새로운 사건메시지를 보내게 된다. 이 알고리즘은 잘못된 사건이라고 할지라도 맞는 결과를 발생할 수 있다는 점을 이용한 것으로서, 시뮬레이션 대상 시스템의 특성에 따라서 시뮬레이션 성능을 크게 향상시킬 수 있다. 그러나 어떤 잘못된 사건이 롤백되는 데에 너무 긴 시간이 소요될 수 있다는 단점이 있다.

둘째, Time Warp 알고리즘의 낙관성을 제한하여 롤백을 줄여서 시뮬레이션 성능을 향상시키는 방법이다. 이 방법들은 자칫 정상적인 계산을 방해할 수 있으므로 논란의 대상이 되고 있다. 우선 time window를 설정하는 방법이 고려되었다. Time window란 어떤 시뮬레이션 시각의 범위를 말하는 것으로 프로세스의 시뮬레이션 시계가 그 범위 이상으로 진행하는 것을 방지하는 것이다. 이 방법을 사용할 경우 잘못된 계산이 너무 빨리 진행되는 것을 방지할 수 있다. Time window의 크기를 정하는 것에 대해 여러 방법들이 제안되었다[9, 10]. 극단적으로 window의 크기가 무한대이면 원래의 Time Warp 알고리즘과 같고 window의 크기가 0이면 보수적인 알고리즘이 된다. Wolf 알고리즘[11]은 한 프로세스에서 롤백이 일어날 때 다른 프로세스에게 특별한 메시지를 보내어 롤백될 사건에 의해서 영향받는 다른 프로세스들이 잘못된 계산을 진행하는 것을 막는 방법이다. 이 알고리즘을 위해서는 어떤 사건이 각 프로세스에게 영향을 주는 관계와 그 사건이 현재 어디까지 전파되었는지에 대한 정보가 필요하다.

세째, 낙관성에 영향을 주지 않고 롤백이 연속적으로 일어나는 것을 막기 위한 방법이 있다. 여러 프로세스가 사이클을 형성하는 경우를 생각해 보자. 설명을 단순화하기 위하여 모든 프로세스에서 하나의 사건을 처리하는 데에 걸리는 시뮬레이션 시간과 실제 계산시간은 균일하다고 가정하자. 이 경우에 만약 어떤 프로세스에서 발생된 메시지가 롤백될 경우 그 메시지에 의해 전체 프로세스들이 끊임없이 롤백해야 할 경우가 발생한다. Prakash와 Subramanian[12]은 이렇게 연속적으로 발생되는 롤백을 방지하기 위하여 Filter 알고리즘을 개발하였다. 이 알고리즘에서 사건메시지에는 자신이 거쳐간 프로세스들의 주소들이 기억된다. 그 정보를 이용하여 프로세스들은 롤백이 계속해서 반복되는 것을 방지한다.

마지막으로 하드웨어적 방법으로 해결하는 것이다. Time Warp 알고리즘에서 프로세스는 주기적으로 자신의 상태를 저장한다. 이런 상태저장의 과정은 Time Warp 알고리즘의 성능에

상당히 큰 영향을 준다[13]. 이것을 해결하는 방법으로 상태저장하는 횟수를 줄일 수 있지만 그럴 경우 롤백되는 시점이 늦추어져서 오히려 성능을 악화시킬 수도 있다. 그래서 Fujimoto [14] 등은 롤백을 위한 칩을 만들어 상태변수를 저장하고 롤백하는 시간을 줄이는 방법을 제안하였다. 그래서 프로세스의 요구가 있을 때마다 상태변수가 저장된 메모리 영역을 저장하고 복구시킨다.

한편 Time Warp 알고리즘의 최적성에 대한 연구가 있었다. Lin과 Lazowska[15]은 롤백 오버헤드가 없을 경우에 Time Warp 알고리즘이 최적이 되는 조건에 대해 연구하였다. 여기서는 시뮬레이션이 진행되기 위한 사건처리의 최적의 임계경로(critical path)를 정하고 만약 그 경로상에서 롤백이 일어나면 최적이 아닌 것으로 간주하였다. 또 Lipton과 Mizell[16]은 롤백에 소요되는 시간이 상수라고 가정할 때 Time Warp 알고리즘과 Chandy-Misra 알고리즘의 성능을 비교하였다. 그 결과 Time Warp 알고리즘의 성능은 Chandy-Misra 알고리즘의 성능보다 얼마든지 좋은 경우가 있지만, Chandy-Misra 알고리즘의 성능은 Time Warp 알고리즘의 성능보다 최상의 경우에도 상수배 만큼만 좋다는 것을 보였다.

또 Time Warp 알고리즘의 롤백에 대해서도 연구되었다. Lin과 Lazowska[17]는 롤백의 방법에 따라 성능에 미치는 영향을 조사하고 사건메시지에 의해서 프로세스의 동작이 중단(preemption)되는 경우에 성능의 변화에 대해서도 연구하였다. 또 롤백이 시뮬레이션 결과에 미치는 영향에 대한 연구가 있었다. Leivant와 Watro[18]는 Time Warp 알고리즘에 대한 정형화된 모델을 만들었다. 또 그 모델을 이용하여 롤백의 발생 횟수와 시기에 상관없이 항상 같은 결과를 가지고, 시뮬레이션 시계는 언젠가는 반드시 진행되며, 그리고 언젠가는 알고리즘이 종료된다는 것을 증명하였다.

5. DEVS 모델의 병렬시뮬레이션

이산사건 시스템을 모델링하기 위한 수학적인 형식론들 중의 하나로 Zeigler에 의해 제안된 DEVS(Discrete Event System Specification) 형식론이 있다[19]. DEVS 형식론에서는 일반적인 이산사건 모델과는 달리 시스템이 계층적으로 모듈화된 형태로 기술된다. 이런 특징으로 인하여 DEVS 모델의 시뮬레이션은 일반적인 이산사건의 모델의 시뮬레이션과는 달리 외부로부터의 입력을 나타내는 외부사건과 시뮬레이션 시각의 진행을 나타내는 내부사건의 두 종류의 사건이 있다. 그 동안의 많은 연구에서 DEVS 시뮬레이션의 병렬처리 알고리즘들이 제안되었다. 이들 알고리즘들은 크게 세 종류로 분류된다.

우선 첫째, 같은 시각으로 스케줄된 DEVS의 외부사건만을 병렬처리하는 방법이 있다. Wang [20]은 인텔의 iPSC 컴퓨터에서 DEVS의 외부사건만을 병렬처리하는 병렬 DEVS 시뮬레이터를 구현하였다. Concepcion[21]은 DEVS 시뮬레이션을 위한 병렬 컴퓨터의 구조를 제안하였다. 또 Zeigler와 Zhang[22]은 계층적 시뮬레이션(hierarchical simulation) 알고리즘을 제안하였다. 이 연구에서는 병렬 시뮬레이션 알고리즘외에 매핑 알고리즘도 제안하였다. 또, 성능 분석을 통하여 외부사건만을 병렬처리하는 알고리즘에서의 시뮬레이션 속도의 향상의 상한치(upper bound)에 대해서도 연구하였다.

둘째, 같은 시각으로 스케줄된 DEVS의 내부사건만을 병렬처리하는 연구가 있다. Wang[23]은 SIMD 환경에서 비계층적인 broadcast 모델의 내부사건을 병렬처리하였다. 일반적인 DEVS 모델의 내부사건을 병렬처리한 연구로 P-DEVSim++가 있다[24, 25]. 여기서는 PAR_INT 알고리즘이 제안되었다. 이 알고리즘에서는 DEVS의 시뮬레이션 시계 제어방식을 그대로 두면서 내부사건에 의해 발생된 외부사건에 우선순위를 두어 그 외부사건들의 처리 순서를 낙관적 시뮬레이션 알고리즘으로 정렬시켰다. 이 경우 일반적인 낙관적 시뮬레이션 알고리즘과는 달리 롤백이 그것을 일으킨 모델내에서만 국한되는 성질을 가진다. 또, PAR_INT 알고리즘을 위한 매핑 알고리즘도 연구되었다 [26, 27]. 일반적인 이산사건의 시뮬레이션에서

는 프로세스들 간의 통신형태가 매우 유동적으로 변화하므로 통신형태를 고려한 일반화된 매핑 알고리즘을 구하기는 어렵다. 그러나 DEVS에서는 내부사건에 의해 발생되는 출력에 의해 외부사건이 스케쥴되므로 프로세스들 간의 통신이 어떤 규칙성을 가진다. 이 연구에서는 하이퍼큐브 환경에 적합한 병렬 DEVS 시뮬레이션 매핑 알고리즘을 개발하였다.

세째, 일반적인 이산사건 시뮬레이션의 경우처럼 다른 시뮬레이션 시각의 사건들을 병렬처리하는 연구가 있다. Christensen[28]은 Time Warp 알고리즘을 이용하여 DEVS 시뮬레이션을 병렬화시켰다. 여기에서는 분산 시스템의 각 프로세서에 병렬 시뮬레이션을 위해 분산 coordinator를 두어, 각 프로세서 내에서의 동기는 Zeigler의 계층적 시뮬레이션 알고리즘을 이용하고, 다른 프로세서와의 동기(synchronization)는 Time Warp 알고리즘을 이용하였다.

6. 결 론

시뮬레이션은 시스템 설계나 실시간 처리의 응용 등의 분야에서 널리 이용되고 있다. 시스템 설계의 분야에서 시뮬레이션은 설계된 시스템의 성능을 분석하거나, 실제로는 존재하지 않거나 직접적인 실험을 하기 어려운 시스템의 동작 특성을 파악할 수 있게 해준다. 또 실시간 처리의 응용으로서는 실제로 존재하는 시스템의 한 구성 요소로서 시뮬레이터를 도입하여 시스템의 파라미터들을 자유자재로 변화시킬 때의 시스템의 성능 변화를 측정할 수 있게 해 준다.

이산사건 시스템의 병렬 시뮬레이션은 급속하게 발전하고 있는 연구분야이다. 병렬 이산사건 시뮬레이션 알고리즘은 전역 인과성 조건을 해결하는 방법에 따라 크게 보수적인 알고리즘과 낙관적인 알고리즘으로 분류된다. 본 논문에서는 현재까지 가장 널리 알려진 보수적인 알고리즘인 Chandy-Misra 알고리즘과 낙관적인 알고리즘인 Time Warp 알고리즘에 대해 다루었다. 그리고 DEVS 형식론에 기반을 둔 이산사건 시스템의 병렬 시뮬레이션에 대해서도 살펴보았다.

병렬 시뮬레이션은 시뮬레이션되는 시스템의 구조가 복잡하고 규모가 방대할수록 필수적으로 요구된다. 이미 미국과 유럽의 여러 나라들에서는 오래 전부터 병렬 시뮬레이션에 대한 인식이 확산되어 많은 연구가 진행되어 왔다. 우리나라에서도 차츰 시뮬레이션의 필요성이 인식되는 점을 감안하면 병렬 시뮬레이션에 대한 많은 연구가 필요하다.

참고문헌

[1] K. M. Chandy and Jayadev Misra, "Distributed simulation : A case study in design and verification of distributed programs," *IEEE Transactions on Software Engineering*, vol. 5, no. 5, pp. 440-452, Sept., 1979.

[2] Richard M. Fujimoto, "Optimistic approaches to parallel discrete event simulation," *Trans. of The Society for Computer Simulation*, vol. 7, no. 2, pp. 153-191, 1990.

[3] Jayadev Misra, "Distributed discrete-event simulation," *ACM Computing Surveys*, vol. 18, no. 1, pp. 39-65, Mar., 1986.

[4] Nathaniel J. Davis IV, David L. Mannix, Wade H. Shaw, and Thomas C. Hartrum, "Distributed discrete-event simulation using null message algorithms on hypercube architectures," *J. Parallel and Distributed Computing*, vol. 8, pp. 349-357, 1990.

[5] W. Cai and S. J. Turner, "An algorithm for distributed discrete-event simulation-the "carrier null message approach", in *Proceedings of the SCS Multiconference on Distributed Simulation*, pp. 3-8, 1990.

[6] Ronald C. De Vries, "Reducing null messages in misra's distributed discrete event simulation method," *IEEE Transactions on Software Engineering*, vol. 16, no. 1, pp. 82-91, Jan., 1990.

[7] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, pp. 77-93, July, 1985.

[8] A. Gafni, "Rollback mechanisms for optimistic distributed simulation systems," in

Proceedings of the SCS Multiconference on Distributed Simulation, pp. 61-67, 1988.

[9] L. M. Sokol, D. P. Briscoe, and A. P. Wieland, "MTW : A strategy for scheduling discrete simulation events for concurrent execution," in *Proceedings of the SCS Multiconference on Distributed Simulation*, pp. 34-42, 1988.

[10] L. M. Sokol and B. K. Stucky, "MTW : Experimental results for a constrainted optimistic simulation paradigm," *Proceedings of the SCS Multiconference on Distributed Simulation*, pp. 169-173, 1990.

[11] V. Madisetti, J. Walrand, and D. Messerschmitt, "Wolf : A rollback algorithm for optimistic distributed simulation systems," in *1988 Winter Simulation Conference Proceedings*, pp. 296--305.

[12] Atul Prakash and Rajalakshmi Subramanian, "Filter : An algorithm for reducing cascaded rollbacks in optimistic distributed simulations," in *Proceedings of the 24th Annual Simulation Symposium*, pp. 123-132, 1991.

[13] Richard M. Fujimoto, "Time warp on a shared memory multiprocessor," *Trans. of The Society for Computer Simulation*, vol. 6, no. 3, pp. 211-239, July, 1989.

[14] Richard M. Fujimoto, J. Tsai, and G. Gopalakrishnan, "Design and evaluation of the rollback chip : Special purpose hardware for Time Warp," *IEEE Transactions on Computers*, vol. 41, no. 1, pp. 68-82, Jan., 1992.

[15] Y. B. Lin and E. D. Lazowska, "Optimality considerations of "Time Warp" parallel simulation," in *Distributed Simulation*, vol. 22, 1990.

[16] R. Lipton and D. Mizell, "Time warp vs. chandy-misra : A worst-case comparison," in *Distributed Simulation*, vol. 22, 1990.

[17] Y. B. Lin and E. D. Lazowska, "A study of Time Warp rollback mechanisms," *ACM Transactions on Modelling and Computer Simulations*, vol. 1, no. 1, Jan., 1991.

[18] Jonathan I. Leivent and Ronald J. Watro, "Mathematical foundations for Time Warp systems," *ACM Transactions on Programming Languages and Systems*, vol. 15, no. 5, pp. 771-794, Nov., 1993.

[19] Bernard P. Zeigler, *Theory of Modelling and Simulation*, John Wiley, 1976.

[20] Y. H. Wang, "The implementation of the hierarchical abstract simulator on the iPSC computer," Master's thesis, University of Arizona, 1987.

[21] A. I. Concepcion, *Distributed Simulation on Multiprocessor : Specification, Design and Architecture*, PhD thesis, Wayne State University, 1985.

[22] B. P. Zeigler and G. Zhang, "Mapping hierarchical discrete event models to multiprocessor systems : Concepts, algorithm, and simulation," *J. Parallel and Distributed Computing*, vol. 10, no. 3, pp. 271-281, July, 1990.

[23] Y. H. Wang, *Discrete-Event Simulation on a Massively Parallel Computer*, PhD thesis, University of Arizona, 1992.

[24] 성영락, 정성훈, 김탁곤, 박규호, "병렬 분산 환경에서의 DEVS 형식론의 구현," 한국시뮬레이션학회논문지, vol. 1, no. 1, pp. 64-76, 1992.

[25] Yeong Rak Seong, Sung Hoon Jung, Tag Gon Kim, and Kyu Ho Park, "Parallel simulation of hierarchical modular DEVS models : A modified Time Warp approach," Accepted for Publication in International J. in Computer Simulation.

[26] Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park, "Mapping modular, hierarchical discrete event models in a hypercube multicomputer," in *Proceedings of International Conference on Massively Parallel Processing*, 1994.

[27] Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park, "Mapping hierarchical, modular discrete event models in a hypercube multicomputer," Accepted for Publication in J. Simulation Practice and Theory.

[28] E. R. Christensen, *Hierarchical Optimistic Distributed Simulation : Combining DEVS and*

Time Warp, PhD thesis, Uiversiy of Arizona, 1990.

성 영 락

1989 2 한양대학교 공과대학 전자공학과(공학사)
1991 2 한국과학기술원 전기 및 전자공학과(공학석사)
1994 2 한국과학기술원 전기 및 전자공학과(공학박사)
1995 3~현재 한국과학기술원 전기 및 전자공학과 위촉연구원
관심분야 : 시뮬레이션 형식론, 병렬 시뮬레이션, 병렬처리

박 규 호

1973년 서울대학교 전자공학과 졸업(공학사)
1975년 한국과학기술원 전기 및 전자공학과 졸업(공학석사)
1983년 프랑스 파리 11대학 졸업(공학박사)
1975년~1978년 동양정밀 개발과장
1983년~1988년 한국과학기술원 전기 및 전자공학과 조교수
1988년~1992년 한국과학기술원 전기 및 전자공학과 부교수
1992년~현재 한국과학기술원 전기 및 전자공학과 교수
주관심분야 : 병렬처리, 컴퓨터 구조, 컴퓨터 비젼

김 탁 곤

1975년 2월 부산대학교 전자공학과 졸업(공학사)
1980년 2월 경북대학교 전자공학과 졸업(공학석사)
1988년 5월 아리조나대학교 전기 및 전산공학과 졸업(공학박사)
1975년 2월~1977년 6월 육군 통신장교
1980년 9월~1983년 1월 부산수산대학교 전자통신공학과 전임강사
1987년 8월~1989년 7월 아리조나 환경연구소 연구엔지니어
1989년 8월~1991년 8월 캔사스대학교 전기 및 컴퓨터공학과 조교수
1991년 9월~1993년 8월 한국과학기술원 전기 및 전자공학과 조교수
1993년 9월~현재 한국과학기술원 전기 및 전자공학과 부교수
1991년~현재 한국시뮬레이션학회 편집위원장
1990년~현재 Associate Editor in *IEEE/ACM Simulation Digest*
1994년~현재 Associate Editor in *Int Journal in Computer Simulation*
1994년~현재 Editor in Textbooks on Simulation (Monographs) published by The Society for Computer Simulation
1995년~현재 Associate Editor in *Simulation*
주관심분야 : 모델링이론, 병렬/지능형 시뮬레이션 환경, 컴퓨터 시스템 해석