병렬 분산 환경에서의 DEVS 형식론의 구현

An Implementation of the DEVS Formalism on a Parallel Distributed Environment

성영락*, 정성훈*, 김탁곤*, 박규호*
Yeong-Rak Seong*, Sung-Hoon Jung*, Tag-Gon Kim* and Kyu-Ho Park*

Abstract

The DEVS(discrete event system specification) formalism specifies a discrete event system in a hierarchical, modular form. DEVSIM++ is a C++ based general purpose DEVS abstract simulator which can simulate systems modeled by the DEVS formalism in a sequential environment. This paper describes P-DEVSIM++ which is a parallel version of DEVSIM++. In P-DEVSIM++, the external and internal event of DEVS models can be processed in parallel. For such processing, we propose a parallel, distributed optimistic simulation algorithm based on the Time Warp approach. However, the proposed algorithm localizes the rollback of a model within itself, not possible in the standard Time Warp approach. An advantage of such localization is that the simulation time may be reduced.

To evaluate its performance, we simulate a single bus multiprocessor architecture system with an external common memory. Simulation result shows that significant speedup is made possible with our algorithm in a parallel environment.

Ⅰ. 서론

시뮬레이션은 시스템의 설계나 실시간 처리의 응용분야의 도구로서 많이 이용되어 왔었다. 시뮬레이션은 설계된 시스템에서 성능 등 여러 지식들을 얻고자 하거나 혹은 여러 부수적인 문제로 인해 실제로는 존재하지 않거나 직접적인 실험을 해볼 수 없는 시스템에 대하여서도 그 동작 특성을 파악할 수 있게 해준다. 시뮬레이션을 통하여 우리는 시스템의 병목현상이나 약점들을 알 수 있으며 시스템의 성능도 예측할 수 있고 전력이나 메모리등의 시스

* 한국과학기술원 전기 및 전자공학과 컴퓨터공학연구실

템에 필요한 여러 자원의 사용량도 추측할 수 있다. 그리고 실시간의 응용으로서는 실제로 존재하는 시스템의 한 구성 요소로서 시뮬레이터를 도입하여 시스템의 여러 파라미터들을 자유자재로 변화시켜 가면서 시스템 성능의 변화 추이를 측정할 수 있다.

그러나 불행하게도 시스템의 규모가 커지고 복잡해짐에 따라서 그러한 시스템을 시뮬레이션하는 데에 너무 긴 시간을 소모해야만 한다. 이러한 경우 분산처리 환경에서의 시뮬레이션이 좋은 해결책이 될 수 있다. 분산 시뮬레이션에서의 시스템은 통신을 통하여 서로 정보를 주고 받는 여러 프로세스들의 집합으로 간주하는 것으로서, 많은 분산 시뮬레이션을 위한 알고리즘들이 제안되었다. 분산 시뮬레이션에서 시스템간에 주고 받는 메시지에는 시간표가 붙어 있으며, 그 시간표에는 그 메시지가 처리되어야 하는 시뮬레이션 시간이 기록된다. 즉, 우리가 시간 t에 프로세스 p에서 메시지 m을 처리하게 하고 싶은 경우에는 메시지 m의 시간표에 t를 기록하여 그것을 프로세스 p에게로 전달하면 된다. 일반적인 경우에 각 모델에는 여러 개의 수행해야 할 메시지들이 쌓이게 된다. 이 경우에 시뮬레이션이 올바르게 진행되기 위해서는 그 모델은 자신에게 도착하는 모든 메시지들을 그 메시지의 시간표의 시간 순으로 배열하여서 그 순서에 맞추어 시뮬레이션하여야만 한다. 그러나 일반적인 경우에는 도착하는 메시지들의 시간표의 시간 순서대로 메시지가 도착하지 않으므로 위의 조건을 만족시키기란 매우 어려운 일이 된다. 바로 이 점이 분산 시뮬레이션이 어려운 점이며 연구의 대상이 되고 있다. 앞에서 말한 그러한 조건을 우리는 causality 조건이라고 하는데 지금까지 제안된 여러 분산 시뮬레이션 알고리즘들을 분류해보면 크게 conservative 알고리즘과 optimistic 알고리즘으로 나뉜다.[2] Conservative 알고리즘에서는 causality의 조건은 항상 만족된다. 즉 시뮬레이션을 진행하면서 causality가 만족되지 않으면 그 조건이 만족될 때까지 기다렸다가 시뮬레이션을 계속하는 것이다. 그러므로 교착상태(deadlock)에 빠질 가능성이 존재하게 된다. 대표적인 conservative 알고리즘으로서는 Chandy와 Misra에 의해 제안된 알고리즘이 있다.[7] 반면 optimistic 알고리즘에서는 메시지가 도착하는 즉시 처리하고 causality 조건에 위배되는 것이 발견되면 그것을 수정하게 함으로써 시뮬레이션을 진행한다. 대표적인 알고리즘으로서는 Jefferson과 Sowizral에 의해 제안된 Time Warp 알고리즘이 있다.[8]

DEVS(discrete event system specification)형식론은 Zeigler에 의해 제안되었으며 이산 사건 시스템을 수학적으로 모델링하는 기법이다.[14][13] DEVS 형식론은 계층적이면서 모듈화된 특징을 가진다. Zeigler는 또 DEVS 형식론으로 모델링된 시스템을 시뮬레이션할 수 있는 추상화된 시뮬레이터(abstract simulator)를 제안하였다. 따라서 DEVS의 형식론으로 시스템을 모델링하고 시뮬레이션하고자 할 경우 모델링의 과정과 시뮬레이션의 과정을 나누어 생각할 수 있게 된다. DEVS 형식론에서는 내부사건(internal event)과 외부사건(exteral event)으로 사건을 분류할 수 있다. 여기서 사건이라 함은 그 모델로 어떤 메시지가 전달되어서 그 모델이 수행해야 하는 일이 생기게됨을 의미한다.

LISP 언어의 한 종류인 Scheme 언어의 객체 지향형 환경에서 개발된 DEVS-Scheme[11]에서 계층적 분산 시뮬레이션의 방법이 실현되고 분산 DEVS 시뮬레이션의 성능 분석[2] 등이 연구된 이래 많은 연구에서 외부 사건을 병렬화 하는 방법을 통하여 DEVS abstract simulator가 병렬 구현되었다.[4] Concepcion은 DEVS abstract simulator에 적합한 병렬 컴퓨터의 구조를 제안하였다.[3] 그리고 DEVS abstract simulator의 성능 평가에 대한 연구가 있었다.[5] Christensen은 Ada 언어 환경에서의 DEVS abstract simulator를 개발하였고 거기에 Time Warp을 결합하여 병렬화 시켰다.[6] 다른 한편으로 객체 지향형 환경하에서 DEVS 형식론의 구현에 관한 연구로서 C ++ 환경에서의 DEVS 형식론인 DEVSIM ++의 환경이 개발되었다.[1]

본 논문에서는 DEVSIM ++의 병렬화를 구현한 P-DEVSIM++을 다룬다. P-DEVSIM++은 abstract simulator 외부 사건이외에도 내부 사건에 대해서도 병렬로 처리할 수 있는 기능을 가진다. 그것을 위하여 optimistic 분산 시뮬레이션 알고리즘들을 이용한다. 즉 P-DEVSIM++에서는 Time Warp의 경우와는 달리 어떤 모델이 rollback을 하는 경우 다른 모델들에게는 영향을 주지 않는다. 그러므로 반대 메시지(anti-message)는 발생될 필요가 없다.

2장에서는 P-DEVSIM++의 이론적인 기초가 되는 DEVS 형식론과 abstract simulator의 이론에 대하여 기술하였다. 3장에서는 모델링된 시스템의 계층적인 구조와 그들간의 연결 상태를 표현하는 configuration 파일에 대하여

고찰하겠다. 4장에서는 개발된 P-DEVSIM++의 알고리즘을 제시하였다. 또한 분산 시뮬레이션을 위한 데이타 구조와, rollback이 한 모델에서만 일어나고 다른 모델들에게는 영향을 주지 않는 이유에 대해 살펴보겠다. 5장에서는 P-DEVSIM++의 성능평가를 위해 하나의 버스에 여러 프로세서와 하나의 공통 메모리가 연결되어 있는 병렬 컴퓨터 시스템의 시뮬레이션 모델링과 그 결과를 분석하였다. 6장은 결론이다.

2. DEVS 형식론

DEVS 형식론은 이산 사건 시스템을 계층적이고 모듈화된 형태로 표현하는 수학적인 도구이다. DEVS 형식론으로 시스템을 표현하기 위해서는 전체의 시스템을 모듈화된 여러 작은 모델들로 나누고 그것들을 계층적으로 구성하여야 한다. 그 기본이 되는 모델들을 atomic 모델이라고 하며 다음과 같이 표현된다.[14] [13]

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

여기서,

X 입력 사건들의 집합.
S 일련의 상태들의 집합.
Y 출력 사건들의 집합.
δ_{int} $S \rightarrow S$: 내부 (상태)전의 함수.
δ_{ext} $Q \times X \rightarrow S$: 외부 (상태)전의 함수.
λ $S \rightarrow Y$: 출력함수.
ta $S \rightarrow R_0^+, \infty$: 시간 진행 함수.

여기서 Q는 M의 전체 상태를 나타내는 것으로서 다음과 같이 주어진다.

$$Q = \{(s,e) | s \subset S \text{ and } 0 \leq e \leq ta(s)\}$$

다음의 형태는 coupled 모델이라는 것으로 구성요소가 되는 몇개의 모델들이 결합되어 새로운 모델을 만드는 것을 표현한다. 그런데 이 coupled 모델 또한 큰 모델의 구성요소 모델이 될 수 있다. 이것을 이용하여 복잡한 모델을 계층적으로 구성할 수 있게 된다. Coupled 모델 DN은 다음과 같이 정의된다.[14] [13]

$$DN = \langle D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, select \rangle$$

여기서,

D 구성요소들의 이름의 집합
for each i in D
M_i D의 i번째 구성요소의 DEVS.
I_i i번째 모델의 influencee 모델들의 집합.
for each j in I_i
$Z_{i,j}$ $Y_i \rightarrow X_j$: i번째 모델의 출력을 j번째 모델로의 입력으로 연결하는 함수.
select subsets of $D \rightarrow D$: 여러 구성 요소 모델들이 같은 시간에 스케쥴을 원할때 그것들을 순서화 시키는 함수.

Atomic DEVS와 coupled DEVS에 대한 보다 상세한 것은 14)와 13)을 참조하여라.

DEVS의 abstract simulator는 DEVS로 표현된 모델들의 시뮬레이션되는 동안의 동작을 해석하기 위한 분산 알고리즘이다.[13] Abstract simulator는 atomic 모델들을 위한 simulator와 coupled 모델을 위한 coordinator의 두 종류로 나뉜다.

Coordinator는 (*,t), (x,t), (y,t), (done,tN)의 네 가지의 메시지로서 상위의 coordinator와 시뮬레이션에 필요한 메시지를 주고 받는다. 위에서 t는 전역(global)시간이고 tN은 시뮬레이션되는 DEVS 모델의 내부 전이가 스케쥴된 시간이다. (*,t) 메시지는 t=tN을 알리는 메시지이다. 만약 어떤 한 coordinator가 (*,t)를 받게 되면 그 coordinator는 자신의 subordinate 모델들 중에서 가장 먼저 내부 전이를 수행해야 하는 모델, 즉 tN이 최소인 모델로 그것을 전달한다. 물론 이때 그 subordinate 모델이 내부 전이를 하게 스케쥴된 시간은 (*,t)의 t와 같아야 한다. 그렇게 해서 그 메시지가 simulator에게 까지 전달되면, simulator는 자신이 시뮬레이션하는 atomic 모델의 내부 전이 함수를 수행하게 된다. Abstract 시뮬레이터에서는 내부 전이가 일어날때에만 출력을 발생시키며 시간 t에서의 출력되는 메시지는 (y,t)로 표시된다.

어떤 simularor가 (y,t)를 발생시면 그것은 상위의 coordinator에게 전달된다. 그 coordinator는 그 메시지가 자신이 시뮬레이션하고 있는 그 coupled 모델의 내부에서 사용되는 것인지 아니면 외부로 전달할 필요가 있는지를 판단한다. 만약 내부적으로 사용되어야 하는 것이면 그것을 (x,t)의 형태로 바꾸어서 그 메시지를 보낸 모델의 influencee 모델들에게 전달한다. 그래서 그 메시지가 simulator에게 도달하게 되면 해당 모델의 외부 전이 함수를 수행하게 된다. 내부 혹은 외부 사건은 그 모델의 상태를 변화시키기도 한다.

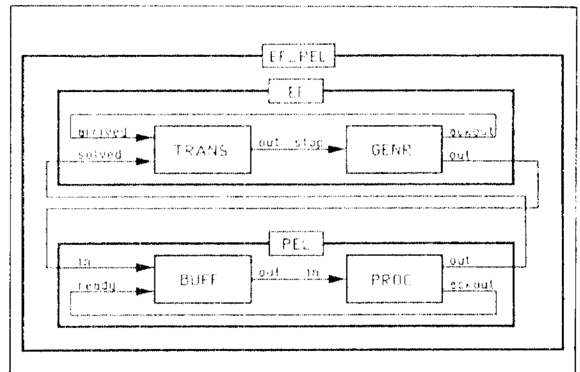
Simulator에서 내부 혹은 외부 사건의 처리가 끝나면 반드시 동기신호인 (done, tN)메시지를 상위의 coordinator에게 보내게 된다. 그 메시지는 해당 모델의 내부 전이가 스케쥴되어야 하는 시간 tN을 상위의 coordinator에게 전달하는 역할을 한다. 그리고 coordinator에서는 모든 subordinate들의 tN을 알게되면 그 중 최소값 자신의 tN으로 정하고 그것을 (done, tN)의 메시지로 상위의 coordinator에게 전달한다. 결국 그 메시지들은 최상위의 coordinator (root coordinator)에게 전달될 것이고 그 최상위 coordinator는 (*,t)를 발생하게 되는데 이때 t는 최상위 coordinator의 tN과 같다. 좀 더 상세한 설명은 [3]에서 찾을 수 있다.

3. Configuration 파일

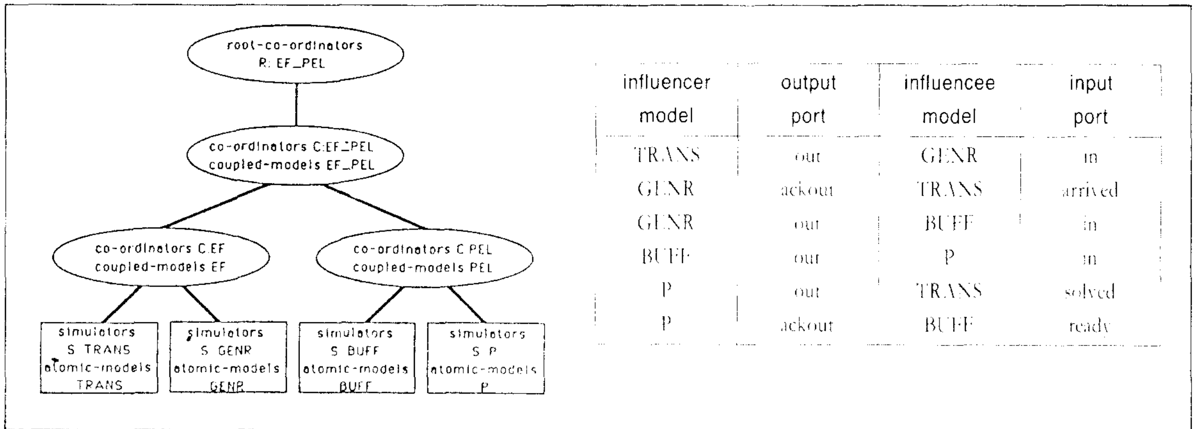
DEVS 형식론으로 어떤 시스템을 모델링하려면 시스템을 작은 시스템들로 분리를 하고 그것들을 모델링한 후에 그 모델들을 이용하여 전체 시스템의 모델을 구성하여야 한다. 이 과정은 계속 재귀적으로 반복되며 결국은 전체 시스템이 atomic 모델들로 나뉘고 coupled 모델들이 서브모델들을 세층적으로 구성하면서 모델들의 입출력포트들을 연결한 형태로 표현된다. 그러나 만약 어떤 시스템을 이루는 모든 atomic 모델에 대한 베이스가 미리 구축되어 있었다면 단지 coupled 모델을 이용하여 구조와 연결 상태들을 기술하는 것만으로도 그 시스템에 대한 모델링을 할 수 있다. 시스템의 계층적인 구조와 모델들 사이의 입출력 포트의 연결 정보는 모델링하고자 하는 시스템의 composition tree로 부터 구할 수 있다.[13]

P-DESIM++에서는 composition tree에 대한 정보를 configuration 파일의 형태로 표현하였다. configuration 파일에는 우선 composition tree의 각 노드에 해당하는 모델의 이름과 모델의 id가 주어지며 만약 atomic 모델의 경우에는 그 모델의 출력 포트가 어떤 모델의 입력 포트에 연결되는지에 대한 정보를 기록하게 된다. 원래 composition tree에서는 coupled 모델에 대한 입출력 포트의 연결 정보도 주어지지만 configuration 파일에서는 그것을 이용하지 않았는데 이것은 모델간의 coupling이 일부만 변하는 경우에 그것을 수정하기가 어려워지기 때문이다. 그리고 configuration 파일에는 각 모델이 시뮬레이션 환경의 어느 프로세서에 mapping될 것인지에 대한 정보도 기록된다. 사용자가 만든 configuration 파일을 시뮬레이션 환경에 다운로드하기 위해선 그것을 다운로드가 용이한 형태로 바꾸어야 한다. 이때 모델 베이스의 각 모델의 이름과 그 모델의 입출력 포트의 이름은 파일의 형태로 주어지며 이를 이용하여 모델링하는 과정에서 생긴 에러를 검출하게 된다. 바뀌어진 configuration 파일에서는 atomic 모델들의 입출력 포트들의 연결 정보 대신에 각 coupled 모델들의 연결 정보가 들어있다.

〈그림 1〉은 간단한 CPU 시스템의 구조이며 〈그림 2〉는 그 CPU 시스템의 composition tree와 각 모델의 출력포트의 influencee 모델들과 입력 포트들을 나타낸다. 〈그림 3〉은 모델 베이스에 대한 정보를 담은 파일이다. 처음의 두 칸에는 모델의 이름과 그 모델 종류에 대한 코드를 다음의 두 칸에는 입출력 포트들의 수를 나타낸다. 나머지 칸에는 그 입출력 포트들의 이름을 나타낸다. 〈그림 4〉는 위의 composition tree와 atomic 모델들의 입출력 포트간의 연결 정보 그리고 모델 베이스의 정보들을 이용하여 만든



〈그림 1〉 CPU 시스템의 단순화된 구조



influencer model	output port	influencee model	input port
TRANS	out	GENR	in
GENR	ackout	TRANS	arrived
GENR	out	BUFF	in
BUFF	out	P	in
P	out	TRANS	solved
P	ackout	BUFF	ready

〈그림 2〉 CPU 시스템의 composition tree와 입출력 포트 연결 정보

ROOT	0	0	0				
COOR	1	0	0				
TRANS	2	2	1	arrived	solved	out	
GEN	3	1	2	stop		out	ackout
BUF	4	2	1	ready	in	out	
PROC	5	1	2	in		out	ackout

〈그림 3〉 모델 베이스에 대한 정보 파일

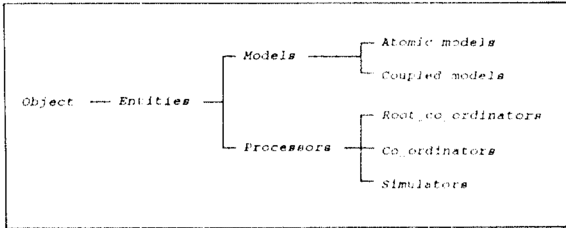
CPU 시스템의 configuration 파일이다. Configuration 파일에서 '#'는 각 모델들의 정보들을 나누기 위한 분리 표시이다. 처음의 '#'까지의 모든 내용은 주석으로 취급된다. 처음의 세 칸에는 각 모델들에게 사용자가 부여하는 id와 모델의 종류에 대한 코드 그리고 시뮬레이션 환경에서 mapping되는 프로세서의 번호를 표시한다. 그리고 만약 그 모델이 coupled 모델일 경우에는 나머지 칸에서는 subordinate 모델의 수와 종류를 표현하고 아닐 경우에는 출력 포트의 수와 이름 그리고 그 포트의 influencee 모델의 id와 연결된 입력포트를 표시한다.

```
Sample Model Configuration Data
 Atomic  Models
         TRANS  : Transducer
         GEN    : Generator
         BUF    : Buffer
         PROC   : Processor
#
0        ROOT     0     1       10
#
10       COOR     0     2       20     50
#
20       COOR     0     2       30     40
#
30       TRANS    1     1
         out      40    stop
#
40       GEN      2     2
         ackout   30    arrived
         out      60    in
#
50       COOR     1     2       60     70
#
60       BUF      1     1
         out      70    in
#
70       PROC     2     2
         ackout   60    ready
         out      30    solved
```

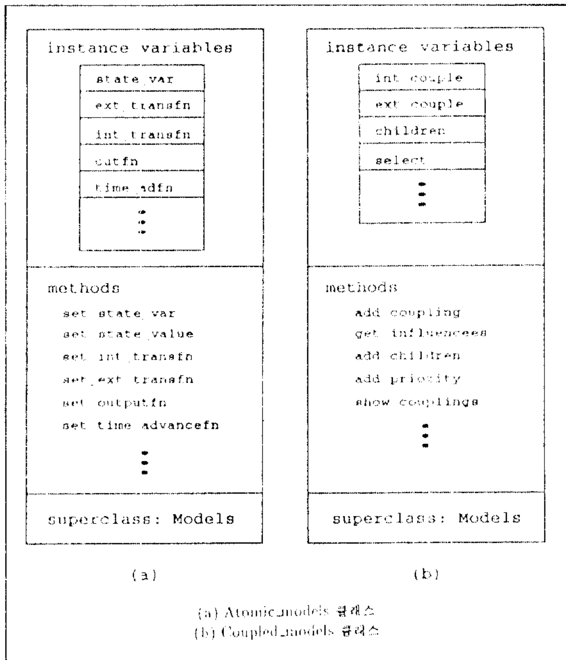
〈그림 4〉 CPU 시스템의 configuration 파일

4. DEVS 시뮬레이션의 병렬처리

DEVS 형식론에 대한 시뮬레이션을 병렬화하는 것은 DEVS abstract simulator의 외부 혹은 내부 사건을 병렬화 함으로써 이루어진다. 만약 시스템에 대한 모델링이 계층적이지 않을 경우에는 내부 혹은 외부 사건이 다른 내부 사건을 야기시키므로 병렬성을 찾기가 쉽지 않다.



〈그림 5〉 P-DEVSIM++의 클래스의 계층구조



〈그림 6〉 Atomic-models과 Coupled-models 클래스의 구조

DEVSIM++은 C++ 환경에서 계층적인 이산 사건 시스템의 모델들을 위한 범용의 DEVS abstract simulator이다.[1] 본 논문에서는 P-DEVSIM++을 구현하였다. P-DEVSIM++은 병렬 분산 환경에서 DEVSIM++의 내부 혹은 외부 사건들을 병렬처리 하였다. 특히 내부 사건을 병렬처리하기 위하여 optimistic 시뮬레이션의 알고리즘이 도입되었다. 그러나 이 알고리즘에서는, Time Warp의 알고리즘과는 달리 어떤 모델이 rollback할때 그것이 그 모델에만 국한된다.

〈그림 5〉는 P-DEVSIM++의 클래스들의 계층구조이다. 클래스 Object는 NIHCL의 최상위 클래스로서 클래스의 이름이나 비교등의 오브젝트에 대한 일반적인 기능들을 제공한다. P-DEVSIM++의 대표 클래스 Entities는 클래스 Object의 서브클래스이다. C++의 inheritance 기능에 의하여 모든 파생클래스들이 위의 두 클래스에서 제공하는 일반적인 기능을 세습받게 된다. 〈그림 6〉는 P-DEVSIM++의 Atomic_models와 Coupled_models 클래스의 구조를 나타낸 것이다.

4.1 외부 사건의 병렬처리

DEVS 시뮬레이션을 가장 쉽게 병렬화하는 방법은 외부 사건의 처리를 병렬화하는 것이다. DEVS 형식론에 의하면 모델은 내부 사건을 수행할 경우에만 출력을 발생한다. 그리고 그 출력은 influencee 모델들에게 외부 사건의 메시지로 전달된다. 예를 들어 〈그림 1〉에서 모델 GENR이 지금 내부 사건을 수행하고 출력 메시지를 그 모델의 influencee 모델들인 모델 TRANS와 모델 PROC의 외부 사건 메시지로 보내고자 할때 만약 모델 TRANS와 모델 PROC이 다른 프로세서에 mapping된다면 그 influencee 모델들은 외부 사건을 병렬로 수행할 수 있다. 어떤 한 순간에 전체 시스템에서 내부 사건 메시지를 받아서 처리하는 모델은 내부 사건에 대해 병렬 처리를 하지 않을 경우 하나만 존재하며, 앞서 언급한 바와 같이 외부 사건을 수행한 뒤에는 출력이 발생하지 않으므로 다른 모델들에게 영향을 주지 않으며 병렬처리로 인한 시뮬레이션의 에러는 발생하지 않는다.

DEVS 시뮬레이션에서 abstract simulator 사이의 메시지의 전달(message passing)이란 메세지를 보내는 abstract simulator에서 메시지를 받는 abstract simulator가 해당 메시지에 대한 method를 호출(invokation)하도록 요청하는 것이다. 즉〈그림 1〉과 〈그림 2〉에서 만약 모델 PEL의 abstract simulator가 하위의 모델 PROC의 abstract simulator에게 (x,t)메시지를 송신하고자 하면 모델 PEL의 abstract simulator는 모델 PROC의 abstract simulator에게 모델 PROC의 외부 사건 처리 함수에 대한 method를 호출하도록 요청하게 된다. 일반적인 순차처리 환경에서는 두 abstract simulator간의 method 호출 요청은 매우 간단하게 표현된다. 예를들어 모델 PROC의 abstract simulator를 S:PROC이라 하고 외부 사건 처리의 method가 when-rcv-x()라면, 위의 예는 단순히 S:PROC->when-rcv-x()라는 표현으로 구현된다. 그러나 분산의 환경하에서는 일반적으

```
Function monitor()

/* initialization */
read the input configuration file
initialize ModelID by the above data

/* start simulation */
if this processor executes the root coordinator
    send initialization messages to subordinates

/* the main simulation routine */
while {
    get a message from message-queue or input channels
    check the destination processor of the message
    MODEL := the ModelID of the destination model of the message

    /* interprete message type */
    switch(type of the message) {
        case X:    MODEL->when_rcv_x();
        case Y:    MODEL->when_rcv_y();
        case STAR: MODEL->when_rcv_star();
        case DONE: MODEL->when_rcv_done();
```

〈그림 7〉 모니터의 알고리즘

instance variables	methods	
Model	send_x_msg	when_rcv_x
id	send_y_msg	when_rcv_y
pid	send_star_msg	when_rcv_star
type	send_done_msg	when_rcv_done
tN	⋮	⋮
priority		
⋮		

〈그림 8〉 ModelID 클래스의 구조

로 위의 두 모델은 다른 프로세서에 mapping되므로 위의 방법을 사용할 수 없으므로, 실제의 메시지들이 프로세서 간에 통신되어야 한다. 즉 위에서 직접적인 method의 호출은 실제의 통신 메시지로 전달되고 모델 PEL에서는 그 메시지를 해석하여 해당하는 method를 호출하게 된다. 즉, 메시지의 전달(message passing)로 바뀌어져야 한다. 우리는 메시지의 통신을 위해 하이퍼큐브의 각 노드에 모니터 프로그램을 두고 모니터가 그 노드내의 통신과 노드간의 통신을 담당하도록 하였다. 이렇게 함으로써 모델들은 다른 모델이 어떤 프로세서에 mapping되었는지에 관계없이 자신의 메시지 처리 함수들을 수행할 수 있다. 〈그림 7〉은 모니터의 알고리즘이다.

모니터는 통신 라우터(router)의 기능외에도 앞서 언급한 configuration 파일을 읽어서 각 노드 프로세서에 mapping되는 모델들을 생성하고 초기화하는 역할을 한다. 〈그림 8〉에는 이러한 모니터의 기능을 위하여 도입된 ModelID 클래스의 구조이다. ModelID에는 전체 모델에 대해서 mapping 정보를 포함하고 있으며 그외에도 coupled model DEVS의 select 함수를 위한 모델의 우선 순위와 내부 사건을 스케쥴하는 시간인 tN을 가진다. 그 두 데이타는 DEVS 형식론에 의하면 각 모델이 가지는 데이타이지만 그 정보를 이용할 경우에 분산된 모델간의 상호 정보 교환의 수가 증가하게 된다. 그러나 ModelID 클래스를 이용할 경우 분산된 모든 모델들이 자신이 속한 노드 프로세서의 로칼 데이타에 의해 필요한 정보를 구할 수 있으므로 실제적인 메시지의 전달이 필요없게 된다. 우선 순위는 시뮬레이션의 과정에서 일정한 데이타이므로 처음 ModelID가 초기화될 때 한번만 정해지면 되지만 tN은 시뮬레이션이 진행되면서 바뀌는 동적인 데이타이다. 우리는 tN을 done 메시지를 통하여 전달되도록 하였으며, 모니터는 done 메시지가 도착할 때마다 ModelID의 데이타를 바꾸어 주었다.

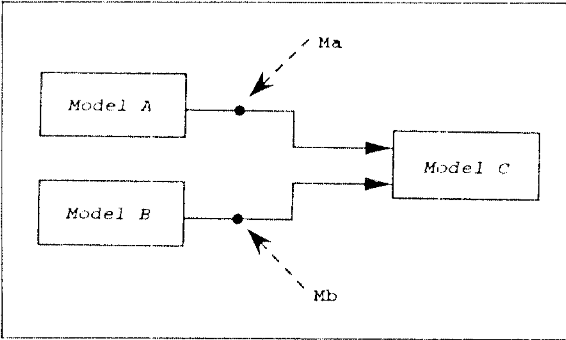
같은 프로세서 내의 모델들간의 통신을 위해서 메시지 큐가 구현되었다. 모니터에게 어떤 모델이 메시지의 송신을 요구할 경우, 모니터는 우선 그 메시지를 받을 모델이 어떤 프로세서에 mapping되어 있는지를 ModelID 클래스를 이용하여 찾는다. 이때 만약 그 모델이 같은 프로세서 내에 mapping이 되어 있다면, 그 메시지는 메시지큐에 추가되고, 그렇지 않을 경우에는 프로세스간의 통신을 위한 함수들을 이용하여 전달된다.

어떤 프로세서에서 현재 메시지를 처리하고 있는 모델이 없을 경우 모니터가 그 프로세서의 동작 제어권을 가진다. 이때 모니터는 메시지 큐에 메시지가 있는지를 확인하여 있을 경우 하나의 메시지를 꺼내어 그 메시지가 전달되어야 하는 모델에게로 보낸다. 이때 메시지를 꺼내는 방식은 FCFS(first come first serve)의 방식이 이용된다. 그리고 만약 메시지 큐가 비어있으면 그 프로세서의 통신 채널에서 메시지를 꺼내게 된다. 이때 교착 상태(deadlock)는 결코 발생하지 않는다.

4.2 내부사건의 병렬처리

내부 사건을 처리한 뒤에는 출력을 발생시켜 다른 모델에게 영향을 주게 되므로 외부 사건을 병렬화하는데에 비하여 내부 사건을 병렬화하는 것은 매우 힘들다. 예를 들어 〈그림 1〉 처럼 지금 두개의 모델 PROC와 모델 GENR

의 tN이 같아서 두 모델의 내부 사건을 병렬처리하고자 한다. 이 경우에 모델 PROC와 모델 GENR는 각각 출력 메시지를 모델 BUFF와 TRANS에게 전송한다면 모델 BUFF와 TRANS에서는 어떻게 두 모델로 부터 전송되어 오는 메시지들을 순서지을 것인가? DEVS에서 모델들은 전체 시스템에 대한 정보를 가지지 않으므로 일반적인 상황에서 어떤 모델이 어떤 시뮬레이션 시간에 몇 개의 메시지를 입력받게 되는지는 전혀 알 수 없다. 그러므로 앞의 예에서, 모델 BUFF와 TRANS는 모델 PROC나 GENR에서 메시지가 전송되어 왔을 경우 즉각 그 메시지를 처리해야 한다. 그러나 그 메시지들을 수행하는 것은 일정



〈그림 9〉 내부 사건의 병렬처리

한 순서를 따라야 한다. 그러므로 메시지들의 도착 순서에 관계없이 우리가 원하는 순서대로 메시지들을 순서지울 수 있어야만 내부 사건 처리를 병렬화시킬 수 있다. 기존의 순차처리 환경에서의 수행을 할 경우에는 tN이 같아도 coupled DEVS의 select 함수에 의해 내부 사건의 처리가 순서화되므로 이러한 문제가 발생하지 않는다. 〈그림 9〉에서 모델 A와B 가 같은 tN을 가지고 모델 A가 모델 B에 비해 우선순위가 앞선다면 select 함수는 모델 A의 내부 사건이 먼저 수행되게 된다.

위의 문제를 해결하기 위하여 대표적인 optimistic 시뮬레이션 알고리즘인 Time Warp의 알고리즘이 도입되었다. Time Warp에서는 모든 메시지들은 시간표를 가지며 그 시간표의 값이 메시지들을 순서화시키기 위한 열쇠로 작용한다. 그러므로 각 모델에서는 도착하는 메시지들을 시간표내의 시간의 순서대로 정열시킬 수 있다. 이때 어떤 메시지의 시간표의 시간이 그 모델의 시뮬레이션 시간에 비해 작을 경우 그 메시지를 straggler 메시지라고 한다. 그리고 만약 straggler 메시지가 발생할 경우 그 모델은 메시지의 수행 순서를 맞추기 위하여 rollback을 하여야 한다. Rollback은 우선 첫째로 그 모델이 수행했던 메시지 중에서 straggler 메시지에 비해 시간표의 시간이 더 큰 메시지들에 의해 수행된 것을 무효와 시키고, 둘째 그렇게 무효화된 메시지들을 처리하면서 발생시켰던 출력 메시지들을 없애기 위한 반대 메시지(anti-message)를 보내고, 세째 straggler 메시지부터 시작하여 다시 모델의 정상적인 메시지 처리 함수들을 수행하여야 한다. 일련의 rollback을 수행하기 위해서는 Time Warp의 모든 모델들은 입력 메시지를 저장하는 큐와 상태를 저장하는 큐 그리고 출력 메시지들의 반대 메시지를 저장하는 큐가 필요하다. 여기에 대한 보다 상세한 것은[8][9][10]에서 참조할 수 있다.

우리가 제안한 알고리즘에서는 메시지의 우선 순위를 coupled DEVS의 select 함수에 의해 정하여야 한다. 이를 위하여 각각의 메시지에 우선 순위와 시뮬레이션의 순번을 두어야 한다. 메시지의 우선순위는 (*,t)일 경우에는 0이고 (y,t)의 경우에는 그 메시지를 발생시킨 abstract simulator의 모델의 우선순위이며 (x,t)의 경우에는 그 메시지를 야기한 (y,t) 메시지의 우선순위이다. 우리는 모델의 우선순위를 그 모델에 사용자가 정의한 id로 하였다. 사용자가 정의한 id는 configuration 파일로 전해지며 초기화 단계에서 정해진다. 시뮬레이션 순번은 최상위 coordinator에서 (*,t) 메시지를 발생할때 마다 하나씩 증가시키고, 그외의 메시지들은 자신이 발생한 원인이 되는 (*,t) 메시지의 시뮬레이션 순번과 같다. 어떤 순간 P-DEVSIM++의 모든 메시지들은 같은 시뮬레이션 순번을 가진다.

관찰 1 전체 시스템의 coordinator인 최상위 coordinator는 항상 전체 구성요소 모델들의 tN 중에서 최소값을 자신의 tN으로 한다. 그러므로 최상위 coordinator에서 생성하는 모든 메시지들은 항상 causality의 조건을 만족한다. 시뮬레이션 순번은 최상위 coordinator에서 새로운 메시지들을 생성할 때마다 증가되는 것이므로 causality의 순서를 의미한다.

```
Algorithm  PAR_INT(message)

/* flush the input message queue and the state queue */
if (last_sim_count != message.sim_count) {
    last_sim_count = message.sim_count;
    inputQ.flush();
    stateQ.flush();
}

/* insert input message to the queue */
rank = inputQ.insert(message);

/* rollback */
for (i = rank+1; i < inputQ.length(); i++)
    restore(stateQ[i], inputQ[i]);

/* save previous state and process input message */
for (i = rank; i < inputQ.length()-1; i++) {
    savestate(stateQ[i]);
    call_model(message[i], NO_DONE_MESSAGE);
}
savestate(stateQ[inputQ.length()-1]);
call_model(message[inputQ.length()-1], DONE_MESSAGE);
```

〈그림 10〉 PAR-INT 알고리즘

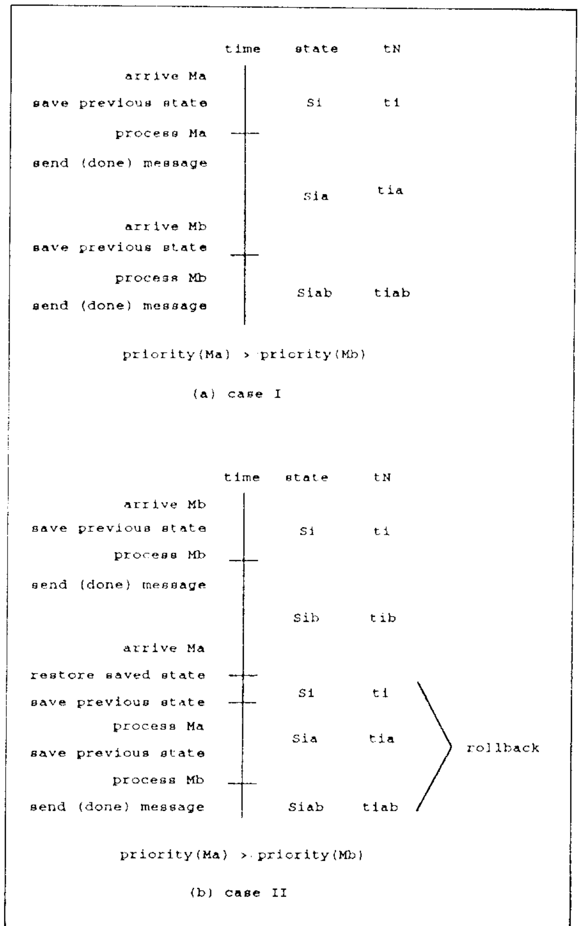
관찰 2 Root coordinator가 새로운 (*,t) 메시지를 발생할 때에는 다른 모델에는 메시지가 존재하지 않는다. 그리고 그 다음 (*,t)가 발생되기 전까지의 모든 메시지는 앞의 (*,t)와 같은 시뮬레이션 순번을 가지므로 어떤 한 순간에 abstract simulator 내의 모든 메시지들은 같은 시뮬레이션 순번을 가진다.

〈그림 10〉에는 우리가 제안하는 P-DEVSIM++의 내부 사건을 병렬화시키는 알고리즘 PAR-INT이다. 이 알고리즘에서는 각 atomic 모델은 입력 메시지 큐와 상태 저장 큐를 가진다. 반대 메시지는 결코 발생되지 않으므로 출력 메시지 큐는 필요가 없다. 전역 변수인 last-sim-count는 최근에 수행된 메시지의 시뮬레이션 순번이며 0으로 초기화된다.

정리 1 PAR-INT 알고리즘에서 입력 메시지 큐와 상태 큐에 저장된 데이타는 새로 도착한 메시지의 시뮬레이션 순번이 바뀔때마다 제거할 수 있다.

증명 : 관찰 1과 2에서 현재의 시뮬레이션 순번보다 작은 방향으로의 rollback은 결코 발생하지 않는다. 그러므로 새로 도착한 메시지의 시뮬레이션 순번이 last-sim-count보다 클 경우 입력 메시지 큐와 상태 큐의 모든 데이타는 더 이상 쓸모가 없다.

그러므로 이 알고리즘에서 시뮬레이션 순번은 Time Warp GVT와 같은 역할을 한다.[8] Coupled 모델은 시스템의 구성요소들을 어떻게 연결하는가에 대해서만 정의하므로 크게 수정하지 않고서도 PAR-INT에서 사용할 수 있다. 각 모델들은 외부로부터 메시지를 받을때 마다 그것을 입력 메시지 큐에 저장한다. 입력 메시지 큐는 메시지의 우선순위에 의해 정렬되어 있다. 이때 straggler 메시지는 메시지의 우선순위가 최근에 처리한 메시지의 우선순위보다 큰 메시지로 정의된다. 그리고 각 모델들은 rollback에 대비하여 입력 메시지들을 처리할 때마다 자신의 상태를 저장하기 위한 함수인 savestate()를 호출한 후에 call-model()함수를 써서 자신의 메시지 처리 함수를 호



〈그림 11〉 그림 8의 rollback

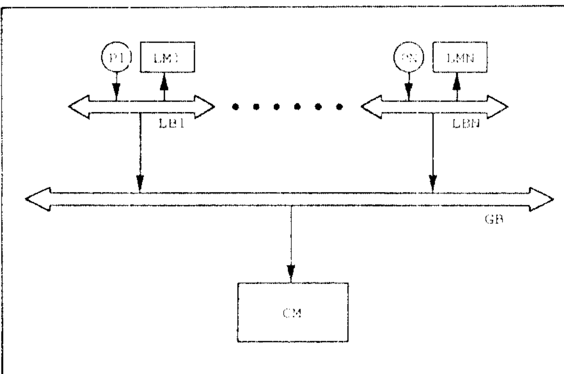
출하여야 한다. 그리고 만약 straggler 메시지가 도착했을 경우에는 restore() 함수를 이용하여 rollback을 하여야 한다. Restore() 함수는 두개의 매개변수를 가진다. 첫번째 매개 변수는 처리되어야 할 메시지이고 두번째 매개 변수는 함수를 수행한 후에 done 메시지를 발생시킬지 않을지에 대한 플래그이다.

정리 2 모든 rollback은 그 rollback을 발생시킨 모델에 국소화(localization)된다.

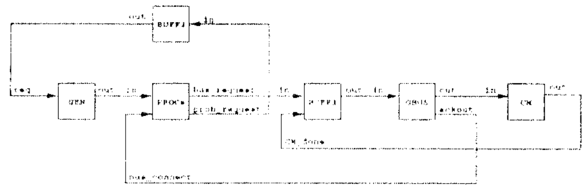
증명 : coordinator는 subordinate 모델들로 부터 자신이 보낸 내부 혹은 외부 메시지의 수 만큼의 done 메시지를 받을때 까지 done 메시지를 발생시키지 않는다. 그러므로 어떤 atomic 모델이 M개의 메시지를 받게 되었을 경우, 그 모델의 simulator의 상위의 coordinator는 M개의 done 메시지를 받은 후에야 자신의 done 메시지를 발생시킨다. 만약 그 simulator의 atomic 모델이 coordinator의 coupled 모델로 부터 하나의 메시지를 받을때마다 하나의 done 메시지를 발생시키고 그 시뮬레이션의 결과가 옳다면 전체적인 시뮬레이션의 결과 또한 옳다.

〈그림11〉에는 위의 과정의 예가 나타나 있다. I의 경우에는 모델 A, B에서 발생한 메시지가 모델 C에 우선순위의 순서대로 도착하였기 때문에 rollback이 발생하지 않는다. 그러나 II의 경우에는 그 순서가 반대로 되었기 때문에 rollback이 필요하게 된다. 그러나 rollback의 과정에서 반대 메시지는 발생치 않고 있다.

5. 실험 및 결과분석



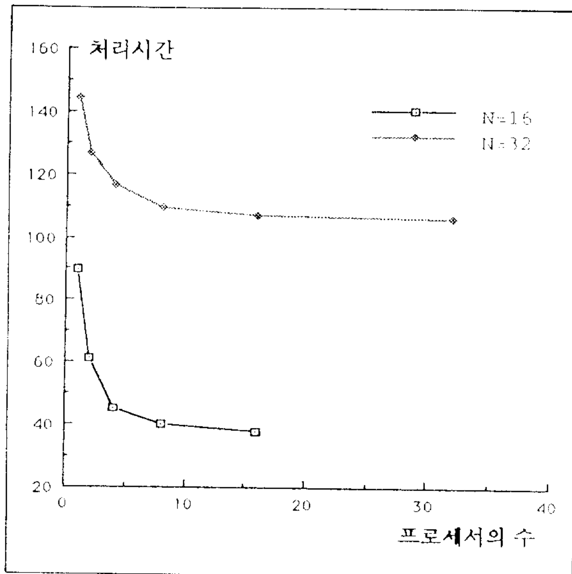
〈그림 12〉 여러 프로세서와 외부의 공통 메모리가 하나의 버스를 통하여 연결된 시스템



〈그림 13〉 〈그림 12〉의 추상화된 모델

분산 시뮬레이션을 함으로해서 얻어지는 성능 향상을 알아보기 위하여 〈그림 12〉과 같이 하나의 버스에 여러 프로세서들이 연결되어 있고 또 하나의 공통 메모리 영역이 있는 시스템에 대하여 시뮬레이션을 하였다.[12] 시뮬레이션은 SUN 4 Sparc 워크스테이션 상에서 수행되었다. 〈그림 13〉은 〈그림 12〉의 시스템에 대한 추상화된 모델로서 모두 6가지 종류의 atomic 모델로 구성되어 있다. 그림에서 Generator 모델은 Processor 모델들에게 보내어질 태스크 메시지를 만드는 역할을 한다. 하나의 태스크 메시지에는 전달되어야 할 Processor의 id와 로칼 메모리를 이용하는 시간, 공통 메모리를 이용하는 시간등의 정보가 포함된다. Processor 모델의 입력포트 'in'으로 하나의 태스크 메시지를 입력 받게되면 Processor 모델은 LOCAL-PROCESSING의 상태로 된다. 이 상태에서 태스크 메시지에 표시된 로칼 메모리 이용 시간 만큼 경과하게 되면 Processor 모델은 그 태스크 메시지를 출력 포트 'bus-connect'로 메시지를 받을때 까지 그 상태를 유지한다. 그래서 그 조건이 만족하게 되면 Processor 모델은 'COMMOM-PROCESSING'상태로 되고 태스크 메시지의 공통 메모리 이용 시간 동안 머무르게 된다. 시간이 다 경과하게 되면 processor 모델은 자신의 id를 출력포트 'prob-request'를 통하여 Buffer2 모델로 보낸다. 그 메시지는 결국 Generator 모델로 보내어지고 Generator 모델은 그 processor 모델을 위해 새로운 태스크 메시지를 발생하게 된다. Common-Memory 모델은 Global-Bus 모델로 부터 태스크 메시지를 받고 그 태스크의 공통 메모리 이용시간 동안 'CM-SERVICE'상태에 있다가 Buffer1 모델에게로 'CM-done'메시지를 받을때마다 자신의 큐를 점검하여 거기에 태스크들이 있을 경우 제일 첫번째의 태스크를 큐에서 빼내어 Global-Bus 모델을 통하여 Common-Memory 모델에게로 전송한다.

위의 모델에 대하여 N이 16과 32의 경우에 P-



〈그림 14〉 〈그림 13〉의 시뮬레이션 결과

DEVSIM++의 성능을 측정하였다. N개의 Processor 모델들을 제외한 모든 모델들을 하나의 시뮬레이션 환경의 프로세서에 mapping하고 N개의 Processor들을 M개의 시뮬레이션 프로세서에 mapping하였다. 이때 M은 1, 2, 4, 8, 16, 32 (N이 32일 경우에만)로 변화시켜 가면서 실험하였다. 두 시뮬레이션 프로세서간의 통신거리는 항상 1로 하였다. 프로세서들간의 통신에 소요되는 시간은 실제 시뮬레이션이 수행되는 환경에 의해 좌우되는 것이므로 정하기가 쉽지 않다. 그래서 우리는 실험환경에서 측정한 통신 소요시간에 어떤 상수 f를 곱한 값으로 정하였다.

〈그림 14〉은 통신 지연 요소 f를 1로 할때의 시뮬레이션의 결과다. N이 16일 경우에는 P-DEVSIM++이 분산의 환경에서 좋은 speed-up을 낸다는 것을 알 수 있다. 그러나 N이 32인 경우에는 별로 좋지 못한데 이것은 대부분의 태스크에 메시지들이 연결망(Bufferl, Global-Bus, Common-Memory 모델)에 몰려서 시뮬레이션의 로드가 제대로 분산되지 않은 것에 기인한다.

6. 결 론

본 논문에서는 DEVS 형식론의 abstract simulator를 병렬화시키는 것에 대하여 기술하였다. DEVS 모델의 계층적인 구조와 입출력 포트간의 연결 구조를 분산환경의 프로세서들에게 전달하기 위하여 configuration 파일이 이용되었다. 분산 시뮬레이션을 위하여 각 프로세서에는 모니터 프로그램을 두었고 시뮬레이션에 관련된 데이타들의 효율적인 관리를 위하여 ModelID 클래스가 도입되었다. PAR-INT 알고리즘이 abstract simulator의 내부 사건들을 병렬로 처리하기 위하여 도입되었다. 이 알고리즘에서는 Time Warp 알고리즘의 기술들을 이용하지만 rollback이 일어날때 다른 모델등에게는 영향을 주지 않는다는 장점이 있다. 알고리즘의 성능을 평가하기 위하여 여러 프로세서와 하나의 공통 메모리 모듈이 하나의 버스에 연결된 시스템에 대하여 시뮬레이션 하였다.

시뮬레이션의 결과에서 계층적인 이산 사건 시스템의 모델이 계층적 분산 optimistic 시뮬레이션 알고리즘으로 시뮬레이션될 때 좋은 속도의 향상을 얻을 수 있다는 결론을 얻었다. 우리는 현재 개발된 알고리즘을 KAIST에서 개발된 5차원의 hypercube 컴퓨터인 KAICUBE-Ⅱ 컴퓨터에서 실현하는 연구가 진행중이다.[15][16] PAR-INT를 위한 효과적인 mapping 알고리즘과 configuration 파일의 사용자 인터페이스 보다 편리하게 하기 위한 그래픽 환경의 구현등이 추후 과제로 남아있다.

참고문헌

[1] B.P.Zeigler, and Guoquing Zhang, "*Mapping Hierarchical Discrete Event Models to Multiprocessor System : Concepts, Algorithm, and Simulation*", Journal of Parallel and Distributed Computing 9, pp.271-281, 1990

[2] K.M.Chandy, and J.Misra, "*Asynchronous Distributed Simulation via a Sequence of Parallel Computations*", Comm. ACM, April, 1981

[3] David Jefferson, and Henry Sowizral, "*Fast Concurrent Simulations using the Time Warp Mechanism*", Distributed Simulation 1985, V15 N2 pp.63-69

[4] B.P.Zeigler, *Theory of Modeling and Simulation*, John Wiley, NY, 1976

[5] B.P.Zeigler, *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, Orlando, Florida, 1984

[6] Tag-Gon Kim, and B.P.Zeigler, "*The DEVS-Scheme Simulation and Modeling Environment*", Chapter 2 of

the Book Knowledge Based Simulation : Methodology and Application(eds : Paul A.Fishwick and Richard B. Modjeski) Springer Verlag, Inc., pp. 20-35, 1990

[7] Yung-Hsin Wang, "*The Implementation of the Hierarchical Abstract Simulator on the iPSC Computer*", Master's thesis, University of Arizona, 1987

[8] Arturo I. Concepcion, "*Distributed Simulation on Multiprocessor : Specification, Design and Architecture*", Ph.D. dissertation, Dep. Comput. Sci., Wayne State University, Detroit, MI, Jan 1985

[9] Doo-Kwon Baik, "*Performance Evaluation of Hierarchical Simulators : Distributed Model Transformations and Mappings*", Ph.D.Dissertation, Dept. Comput. Sci., Wayne State University, 1986

[10] Eric R. Christensen, "*Hierarchical Optimistic Distributed Simulation : Combining DEVS and Time Warp*", Ph. D. desertation, University of Arizona, 1990

[11] 김탁곤, 박성봉, "*The DEVS Formalism : Hierarchical Modular Systems Specification in C++*", European Simulation Multiconference, 1992

[12] Fannie Tallieu, and Frank Verbove, "*Using Time Warp for Computer Network Simulations on Transputers*", Proceedings of the 24th Annual Simulation Symposium, 1991

[13] Yi-Bing Lin, and Edward D. Lazowska, "*A Study of Time Warp Rollback Mechanisms*", ACM Transactions on Modeling and Computer Simulations, Vol.1, January, 1991

[14] Bonian Dai, "*A Hierarchical, Modular Methodology for Multiprocessor System Modeling and Simulation*", Master's thesis, University of Kansas, 1991

[15] 이승섭, "*Implementation of Kernel for Parallel Processing in Hypercube Computer Systems*", M. S. Thesis, E. E. Dept., KSIST, 1989

[16] 김종욱, "*Implementation of Communicaton Module for a Hypercube Multicomputer*", M. S. Thesis, E. E. Dept., KAIST, 1990

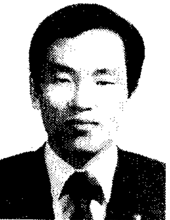
저자소개

성영락
1989년 2월 한양대학교 공과대학 전자공학과 졸업(공학사)
1991년 2월 한국과학기술원 전기 및 전자공학과 졸업(공학석사)
1991년 3월~현재 한국과학기술원 전기 및 전자공학과 박사과정
관심분야: 시뮬레이션 형식론, 병렬 시뮬레이션, 병렬처리

정성훈
1988년 2월 한양대학교 공과대학 전자공학과 졸업(공학사)
1991년 2월 한국과학기술원 전기 및 전자공학과 졸업(공학석사)
1991 3월~현재 한국과학기술원 전기 및 전자공학과 박사과정
관심분야: 지능제어시스템, 지식구축방법, 모델링 및 시뮬레이션

김탁곤
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 전산공학과 조교수
1991년 9월~현재 한국과학기술원 전기 및 전자공학과 조교수
관심분야: 모델링이론, 병렬/지능형 시뮬레이션 환경, 컴퓨터 시스템 등



박규호
1973년 서울대학교 전자공학과 졸업(공학사)
1975년 한국과학기술원 전기 및 전자공학과 졸업(공학석사)
1983년 프랑스 파리 11 대학 졸업(공학박사)

1975~1978년 동양정밀 개발과장
1983~1988년 한국과학기술원 전기 및 전자공학과 조교수
1988~1992년 한국과학기술원 전기 및 전자공학과 부교수
1992~현 재 한국과학기술원 전기 및 전자공학과 정교수
관심분야: 병렬처리, 컴퓨터 구조, 컴퓨터 비젼