# GPSS 프로그램의 병렬화에 관한 연구

## A Study on the Implementation of GPSS Program on a Parallel Computer

윤정미*
Jungmi Yoon

**Abstract**

With the rapidly increasing complexity of decision-making or system development in the fields of industry, management, etc., modelling techniques using simulation has become more highlighted. Particularly, the advent of parallel computer systems not only has opened a new horizon of parallel simulation, but also has greatly contributed to the speed-up of the execution of simulation. The implementation of parallel simulation, however, is not a easy job for those who accustomed to the existing computer systems. And it is also necessarily confronted with the problem of synchronization conflicts in the process.

Thus, how to allow a wider community of users to gain access to parallel simulation while solving synchronization conflicts has become an important issue in simulation study. As a method to solve these problems, this paper is primarily concerned with the implementation of GPSS which is a generally used simulation language for discrete event simulation, onto a parallel computer using C-LINDA. For that, this paper is to suggest a model and algorithm and to experiment it using a case.

* 유한대학 전산과

## 1. 서론

오늘날 시뮬레이션은 시스템 개발이나 의사 결정 문제에 있어서 가장 중요한 모델링 기법의 하나로 각광받고 있다. 특히 최근에 와서 공학, 전산, 경제, 군사 등 여러 응용 분야에서 시스템의 복잡도가 커지고 순차 컴퓨터에서 수행하기에는 너무 많은 시간이 걸리게 됨에 따라 병렬 컴퓨터의 출현과 함께 프로그램의 실행 속도를 단축할 수 있는 분산 시뮬레이션에 대한 관심이 한층 고조되고 있다. 이산 사건 시뮬레이션(discrete event simulation)이 연속적인 시간상에서 이산적으로 발생하는 사건에 따라 실 시스템의 상태를 분석하는 과학적인 문제 해결 방법[3, 6, 7, 9, 11, 18, 20]인 반면, 분산 시뮬레이션은 시뮬레이션 프로그램을 일반 컴퓨터가 아닌 병렬 컴퓨터에서 실행시키는 것을 의미한다[10, 12, 19].

분산 시뮬레이션은 통신을 통하여 서로 정보를 주고받는 여러 프로세서(processor)들의 집합으로, 시스템간에 주고받는 메시지에는 메시지가 처리되어야 하는 시뮬레이션 시간을 표시하는 시간 기록표(time-stamp)가 붙게 된다. 따라서 시간 t에 메시지 m을 프로세스 p에서 처리하는 경우, 메시지 m의 시간 기록표에 t를 표시하여 그것을 프로세스 p에 전달하게 된다. 분산 시뮬레이션이 일반적인 분산처리 프로그램과 다른 점은 각 프로세스에 도착하는 메시지들을 시간의 순서대로 배열하여 인과관계(causality)의 순으로 시뮬레이션 한다는 점이다. 그러나 이러한 과정에서는 여러 프로세서에서 각각 실행되는 메시지들이 어떻게 인과관계 조건을 만족시켜 시간 기록표의 순으로 처리할 수 있느냐의 문제가 필연적 이슈로 등장하게 된다[1, 2, 10]. 즉, 동기화 충돌(synchronization conflicts)의 해결에 그 초점이 모아지고 있다. 이러한 동기화 충돌문제를 해결하기 위한 방법은 크게 보수적인 방법(conservative methods)과 낙관적인 방법(optimistic methods)으로 나누어진다. 보수적인 방법은 인과관계 조건이 만족될 때까지 기다렸다가 시간기록표 순으로 시뮬레이션을 진행하는 방법이고, 낙관적인 방법은 메시지가 도착하는 즉시 처리하고 나중에 인과관계 조건에 위배되는 것이 있으면 롤백(rollback)시켜 인과관계 조건을 만족시키도록 시뮬레이션 하는 방법이다[8, 13].

분산 시뮬레이션은 병렬 컴퓨터의 출현과 더불어 획기적인 발전기반을 마련하였다고 할 수 있다. 그러나 병렬 컴퓨터를 이용한 분산시뮬레이션은 고도의 숙련이 요구되고 프로그램 작성의 난해성과 복잡성 등으로 인하여, 아직까지 순차 컴퓨터 프로그램에 익숙한 일반 프로그래머조차도 접근이 용이하지 않은 것이 현실이다. 따라서 일반 프로그래머들이 쉽게 접근할 수 있는 분산 시뮬레이션의 기법을 개발하는 문제는 시급한 과제가 아닐 수 없다.

이에 본 연구는 일반 컴퓨터에서 널리 사용되고 있는 시뮬레이션 언어인 GPSS(General Purpose Simulation System)를 병렬 컴퓨터로 구현하기 위한 모델과 알고리즘을 제시하고, 병렬 프로그램 언어인 C-LINDA를 사용하여 그 타당성을 검증하는데 목적을 두었다. 이를 위하여 GPSS 프로그램의 실례를 사용하였고, 동기화 충돌의 문제를 해결하기 위한 방안으로는 낙관적인 기법을 채택하였다.

## 2. GPSS의 병렬화 모델과 알고리즘

### 2.1. 모델링

GPSS 프로그램을 분산 시뮬레이션 환경에서 구현하기 위해, 본 연구에서는 GPSS 프로그램을 몇 개의 프로세스로 재구성하였다. 이를 위하여 GPSS 프로그램을 GENERATE 문장 수에 따라 다른 프로세스로 구분하고(이때 GENERATE로 시작하는 논리적 블록을 LP1, LP2, ... 라고 명명하고, 각각의 프로세스 LP1, LP2, ... 는 제각기 다른 프로세서로 실행되도록 하였다), 여기에 더하여 제어 프로세스(control process)의 역할을 수행할 수 있는 또 하나의 주 프로세스(master process)를 설정하였다. 이때 주 프로세스는 프로그램 시작시에 각각의 LP들을 동시에 기동시켜 줌은 물론, 전역변수(global variable)를 포함하는 문장이 실행될 때마다 시스템의 상태를 시간에 의해 정렬된 하나의 선형 리스트(linked list)를 유지함으로써 실행 도중에 생기는 동기화 문제의 발생 여부를 확인시켜주고, 각각의 LP

들이 실행을 완료했을 때 자료들을 모아 결과로 산출해 주는 기능을 수행하도록 하였다.

위에서 제시된 모델을 좀 더 구체적으로 설명하면 다음과 같다. 주 프로세스가 각각의 LP 들을 동시에 기동시킴으로써 프로그램이 시작되고, 각각의 LP는 전역변수가 사용되어진 GPSS 블록을 실행할 때마다 동기화 충돌 문제가 발생하는지를 확인하기 위해 주 프로세스 안에 있는 선형리스트에 하나의 레코드를 추가하게 된다. 이 때 롤백을 가능하게 하기 위해 각각의 LP에는 향후 발생하게 될 사건 리스트(future event list)와 트랜젝션(transaction)의 모든 것이 포함된 선형 리스트가 시뮬레이션 시간에 의해 정렬되어 유지되며, 주 프로세스와 각각의 LP 사이의 교류는 튜플 공간 안에 있는 튜플에 의해 실행된다. 동기화 충돌 확인 과정에서, 만약 동기화 충돌이 발생되지 않으면 각각의 LP는 개입중단 없이 자신의 작업을 계속하고, 동기화 충돌이 발생되면 모든 실행되어진 사건들은 충돌이 일어나기 전의 시점까지 롤백되어야 한다. 또한, 충돌이 일어난 시점 이후의 사건들은 제거되어지거나 갱신되어진다. 그리고 나서, 로칼 시간이 재 설정되어 지고 각각의 LP는 다시 실행을 계속한다. 이 과정은 프로그램이 끝날 때까지 반복되어지며 각각의 LP가 작업을 끝내면 주 프로세스에게 값을 반환하고 주 프로세스는 반환된 값들을 모아 결과를 리포트 하게 된다.

## 2.2 GPSS의 병렬화 알고리즘

GPSS 프로그램의 병렬 프로그램으로의 변환을 가로막는 결정적인 요인은 동기화 충돌의 문제이다. 따라서 GPSS 프로그램을 병렬 프로그램으로 변환하는 과정은 결국 동기화 충돌의 문제를 해결하는 과정이라고 할 수 있다. 이를 위해서는 GPSS의 각 문장(statement)들을 중심으로 동기화 충돌의 여부를 검토해보지 않으면 안된다.

그러나 본 연구에서는 GPSS의 전 문장들을 고려하기보다는 실험 사례로 사용할 GPSS Tugboat Program[16, 17]에 나오는 주요 문장인 GENERATE, ADVANCE, SEIZE/RELEASE, START, TERMINATE를 중심으로 동기화 충돌의 여부와 충돌의 발생시 이를 해결하기 위한 알고리즘을 고려해 보기로 한다. 제시된 모든 알고리즘은 두 개의 프로세스 LP1과 LP2로 제한하였고, LP1이 LP2에 선행하는 것으로 가정하였다. 이는 세 개 이상의 프로세스에 대해서도 적용되어질 수 있다.

### 2.2.1. 동기화 충돌이 발생하지 않는 문장

GENERATE 문장은 새로운 프로세스를 생성하는 단계이고, ADVANCE 문장은 시간의 경과만을 제공해주는 기능을 수행한다. 따라서 이러한 문장들에 대해서는 동기화 충돌이 전혀 발생하지 않으므로 동기화 충돌 문제를 고려하지 않고 순차적(sequential) 시뮬레이션에서와 같이 실행하여도 무방하다. 한편 START 문장은 GPSS 프로그램의 START A 에서 A가 TERMINATE A 문장이 있을 때 상수처럼 사용되기 위해 튜플 공간 안에 저장하는 기능만을 수행하므로 또한 동기화 충돌의 문제는 발생하지 않는다.

### 2.2.2 동기화 충돌이 발생하는 문장

#### 2.2.2.1 SEIZE/RELEASE 문장

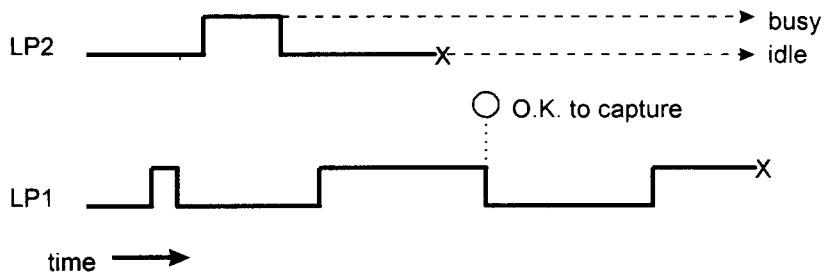
SEIZE/RELEASE 문장은 공유자원(common resource)을 핸들링하는 기능을 가지고 있다. 따라서 GPSS 프로그램에서 SEIZE/RELEASE 문장이 사용되는 LP들 사이에 공유자원이 존재하면 반드시 동기화 충돌 문제가 고려되어져야만 한다. 만약 동기화 충돌이 발생되면 동기화 충돌이 일어난 시점까지 실행된 모든 트랜젝션을 롤백시켜야 한다. 그러나 만약, SEISE/RELEASE 문장을 갖는 LP들 사이에 공유자원이 존재하지 않는다면 서로 영향을 미치지 않으므로 동기화 충돌을 고려하지 않아도 된다.

LP들 사이에 공유자원이 존재하는 경우, 동기화 충돌을 해결하기 위한 알고리즘은 아래와 같다.

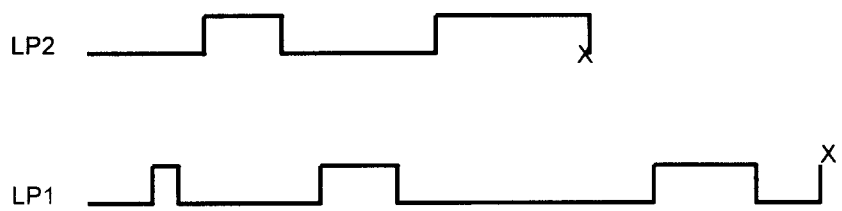
**IF** (LP2 is the executing process)
**THEN** {
**IF** (LP1 has already captured the common server when LP2 requests it)
case 1. **THEN** LP2 must wait until LP1 releases the server
**ELSE** {
LP2 captures the common server
**IF** (the time that LP2 releases the server < the time that LP1 captures the next server)
case 2. **THEN** LP1 and LP2 continue to run : no problem here
case 3. **ELSE** LP1 must be backed up until the time that LP2 releases the server

}
}
**ELSE** LP1 and LP2 continue to run

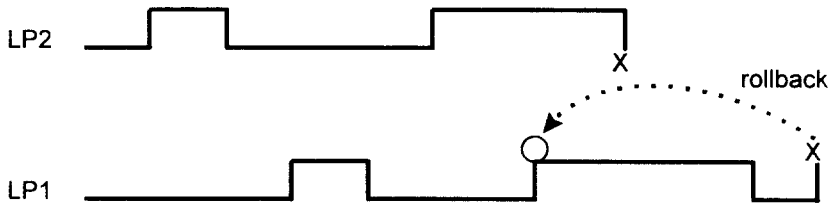
**( CASE 1 )**




**( CASE 2 )**



**( CASE 3 )**



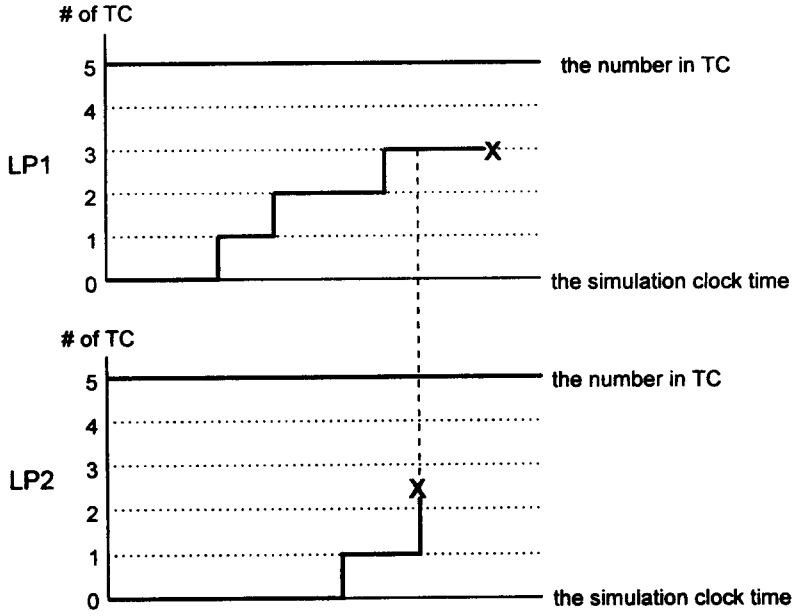

2.2.2.2 TERMINATE 문장

TERMINATE 문장은 이벤트 종료의 기능을 하고, 앞에서 언급한 START A에서의 A와 함께 사용된다. 만약 GPSS 프로그램 안에 하나의 TERMINATE 문장이 존재한다면 동기화 충돌 문제는 고려하지 않아도 되지만, 두 개 이상의 TERMINATE 문장이 존재한다면 반드시 고려되어져야 한다. 이를 위해 각각의 LP는 TERMINATE 문장을 실행할 때마다 시뮬레이션 시간과 종료 계수기(termination counter : 알고리즘에서는 TC로 표기함) 값을 포함한 정보를 튜플 공간에 저장해서 동기화 충돌의 발생 여부를 확인한다.

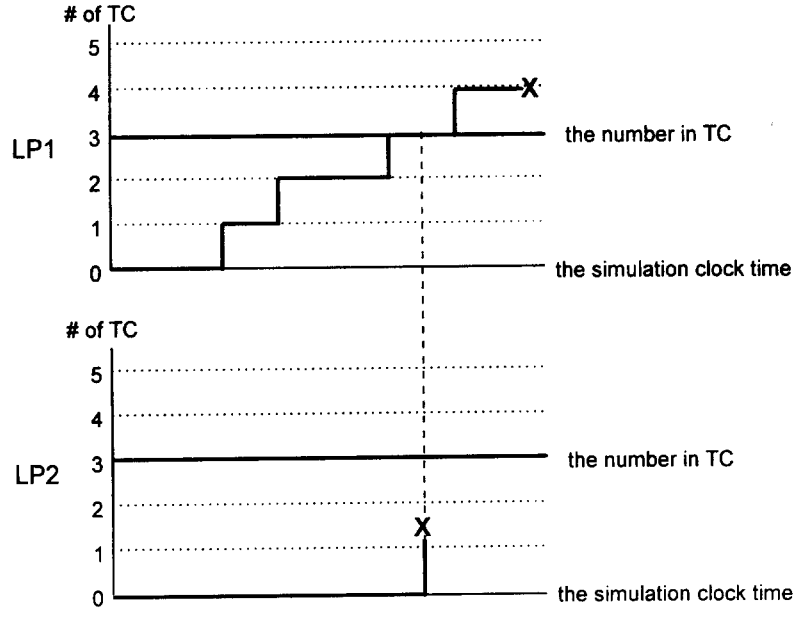
동기화 충돌을 해결하기 위한 알고리즘은 다음과 같다.

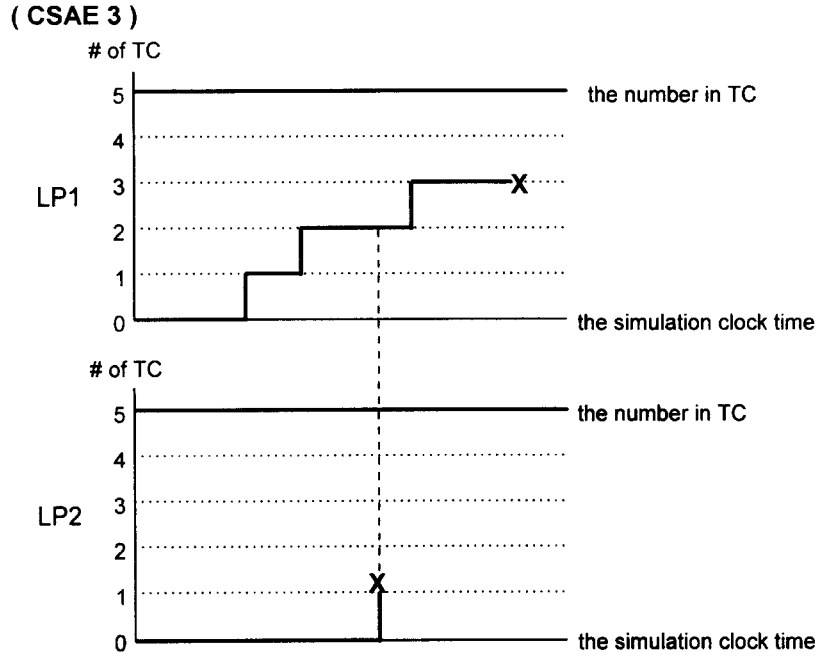
**IF** (LP1 is the executing process)
**THEN** the number in TC of LP1 is saved and LPs continue to run
**ELSE** {
. find the right place for the number in TC of LP2 by its time
**IF** (the number in TC of LP1 + the number in TC of LP2 >= the number in TC)
**THEN** {
**IF** (the number in TC of LP1 + the number in TC of LP2 == the number in TC)
Case 1. **THEN** all LPs stop to run with the simulation clock time of LP2
Case 2. **ELSE** all LPs stop to run with the simulation clock time of LP1
}
Case 3. **ELSE** the number in TC of LP2 is saved and LPs continue to run
}

**( CSAE 1 )**


# of TC
5
4
3
2
1
0
LP1
X
the number in TC
the simulation clock time
# of TC
5
4
3
2
1
0
LP2
X
the number in TC
the simulation clock time

**( CSAE 2 )**


# of TC
5
4
3
2
1
0
LP1
X
the number in TC
the simulation clock time
# of TC
5
4
3
2
1
0
LP2
X
the number in TC
the simulation clock time




## 3. GPSS 프로그램의 병렬화 구현 사례

### 3.1. 사례 : GPSS Tugboaut 프로그램

분산시뮬레이션 환경에서 구현시켜 볼 GPSS 프로그램은 Schriber[17]의 "An Introduction to Simulation using GPSS/H"에서 인용하여 사용하였다. 그는 이 프로그램의 이름을 Tugboat Program이라고 붙였다(이하에서는 GPSS Tugboat Program으로 칭함: <그림 1> 참조).

### 3.2. C-LINDA를 통한 GPSS의 구현

3.2.1 C-LINDA[4, 5, 14, 15]

C-LINDA는 C 언어와 LINDA를 결합한 병렬 프로그램 언어로서, 공용 기억장치 컴퓨터(shared-memory computer)와 분산 기억장치 컴퓨터(distributed memory computer), 네트워크를 포함한 여러 개의 병렬 컴퓨터 시스템에서 많이 이용되고 있다. 이는 모든 프로세스에 의해 접근(access)될 수 있는 튜플 공간(tuple space)과 그 위에 작용하는 몇 개의 함수를 사용하여 병렬성을 구현한다.

C-LINDA에서 주로 사용되는 6개 함수와 그 기능은 다음과 같다.

■ EVAL 함수 : 각각의 새로운 프로세스를 시작하기 위한 기능을 가지며, 보통 데이터 튜플(data tuple)처럼 튜플 공간 안에 존재한다. EVAL 함수는 최고 16개의 파라미터까지 사용할 수 있고, 그들은 int, long, short, char, float, double의 값들을 갖는다. 여기에 사용되는 데이터는 항상 튜플 공간을 통해 또 다른 하나의 프로세스로 전달되며, 함수의 인수로 structure, pointer, array, union은 사용할 수 없다.

■ IN 함수 : 인수에 제시된 템플릿(template)과 일치하는 데이터 튜플을 튜플 공간에서 찾아 그 튜플을 튜플 공간에서 제거하고 읽어오는 기능을 수행한다.

■ INP 함수 : 일치하는 데이터 튜플이 존재하는지를 테스트해서 일치하는 튜플이 검색되면 1을, 그렇지 않으면 0을 반환하는 기능을 수행한다.

■ RD 함수 : IN 함수와 비슷하나 튜플 공간으로부터 일치되는 튜플을 찾아, 그 튜플을 제거하지 않고 읽어오는 기능을 수행한다.

■ RDP 함수 : INP 함수와 같은 기능을 가지나 일치되는 튜플을 튜플공간에서 제거하지 않고 수행한다.

■ OUT 함수 : 새로운 튜플을 생성해서 튜플 공간에 추가하는 기능을 수행한다.

```
        SIMULATE                    base time unit: 1hour
* LP1
GENERATE    15, 5           Type A ships arrive, one by one
SEIZE       BERTHArequest / capture the A berth
SEIZE       TUGBOAT     request/capture the tugboat
ADVANCE     2               berthing time
RELEASE     TUGBOAT         let the tugboat go
ADVANCE     8, 2            unloading time
SEIZE       TUGBOAT         request/capture the tugboat again
ADVANCE     1               deberthing time
RELEASE     TUGBOAT         let the tugboat go
RELEASE     BERTHAno longer occupying the A berth
TERMINATE   1               Type A ships leave, one by one
* LP2
GENERATE    20, 5           Type B ships arrive, one by one
SEIZE       BERTHBrequest/capture the B berth
SEIZE       TUGBOAT         request/capture the tugboat
ADVANCE     2               berthing time
RELEASE     TUGBOAT         let the tugboat go
ADVANCE     10, 4       unloading time
SEIZE       TUGBOAT         request/capture the tugboat again
ADVANCE     1               deberthing time
RELEASE     TUGBOAT         let the tugboat go
RELEASE     BERTHBno longer occupying the B berth
TERMINATE 1         Type B ships leave, one by one

START  500              start the Xact-Movement phase
END                         end of Model-File execution
```

<그림 1> GPSS Tugboat 프로그램

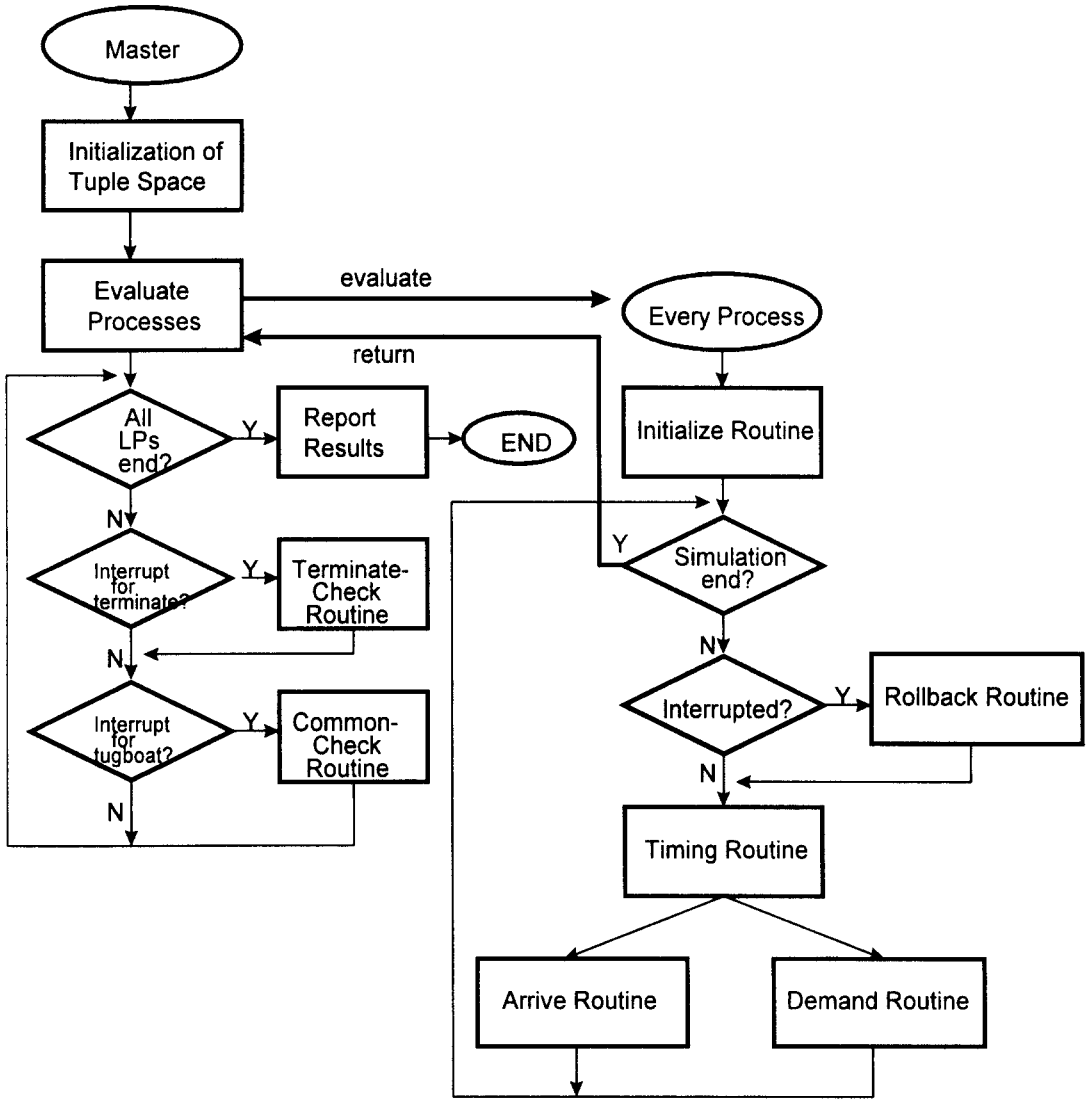
3.2.2 GPSS의 구현방법

<그림 2>는 2.1에서 제시한 모델을 C-LINDA로 구현하기 위한 순서도이다.

이를 GPSS 프로그램의 실례로 주어진 Tugboat program에 적용하여 설명하면 다음과 같다. 여기서는 복잡성을 피하기 위해 LP1의 실행을 중심으로 설명하였다.

■ 전체를 조절하기 위한 하나의 주 프로세스와 사용된 실례의 GENERATE 문장 두 개에 대해 두 개의 프로세스 LP1, LP2가 필요하다. 주 프로세스가 C-LINDA의 EVAL 함수를 통해 각각의 LP 들을 동시에 기동시킴으로써 프로그램이 시작된다.

■ LP1과 LP2는 제각기 「Initialize Routine」을 실행한다. 「Initialize Routine」에서 시뮬레이션 로




< 그림 2 > 순서도

칼 시간, 상태변수, 그리고 통계학적인 자료들이 초기화되고, 「Timing Routine」에서 발생될 다음 이벤트의 형태가 결정된다. 각각의 LP는 GENERATE 문장을 위한 도착 이벤트(arrive event)와 ADVANCE 문장을 위한 요구 이벤트(demand event)를 갖는다. 이때 이벤트의 성격에 따라 「Arrive Routine」과 「Demand Routine」이 실행된다.

■ 주 프로세스와 각각의 LP 사이의 교류는 튜플 공간 안에 있는 튜플에 의해 실행된다. 이때 각각의 LP는 C-LINDA의 IN이나 RD 함수를 사용해서 튜플 공간으로부터 필요한 데이터를 얻고, OUT 함수를 사용해서 튜플 공간의 정보를 갱신한다.

■ LP1이 공유자원인 터그보트(tugboat)를 사용하려고 할 때마다 「Common-Check Routine」이 실행된다. LP1은 동기화 충돌의 발생여부를 확인하기 위해 튜플 공간을 이용해 로칼 시간과 터그보트의 지속 시간이 표시된 메시지를 주 프로세스에게 전달함으로써 주 프로세스를 중단시킨다. 이때 2.2.2.1에서 제시된 SEIZE/RELEASE 문장의 동기화 충돌문제를 해결하기 위한 알고리즘이 사용된다. 동기화 충돌 문제의 발생여부를 확인하기 위해 주 프로세스는 개입중단시킨 LP1의 로칼 시간으로 주 프로세스 안에 있는 선형리스트에서 트랜젝션의 제 위치를 찾아 인접한 리스트의 터그보트 지속 시간과 LP1의 터그보트 지속시간을 비교한다. 만약 동기화 충돌이 발생되지 않는다면 주 프로세스는 튜플 공간을 통해 LP1에게 메시지를 전달하고 메시지를 전달받은 LP1은 자신의 작업을 계속한다. 만약 동기화 충돌 문제가 발생되면 LP1은 또 다른 하나의 프로세스인 LP2를 개입중단시킨다. LP2는 자신의 선형리스트에서 개입중단된 시간으로 제 위치를 확인하고 「Rollback Routine」을 실행한다.

■ 「Rollback Routine」에서 개입중단된 시간 이후의 트랜젝션들이 LP2의 선형리스트에서 제거되고, 이때 제거된 트랜젝션의 수만큼 롤백된 트랜젝션의 수와 롤백의 횟수도 증가하게 된다.

■ 하나의 트랜젝션이 종료될 때마다 「Terminate-Check Routine」이 실행된다. 이 루틴에서 LP는 종료 계수의 값을 확인하기 위해 튜플 공간을 통해 종료 계수의 값과 해당 시간을 포함한 메시지를 주 프로세스에 전달함으로써 주 프로세스를 중단시킨다. 이때 2.2.2.2에서 제시된 TERMINATE 문장의 동기화 충돌 해결을 위한 알고리즘이 사용된다. 동기화 충돌여부를 확인하기 위해 주 프로세스는 개입중단시킨 LP1의 로칼 시간으로 주 프로세스 안에 있는 선형리스트에서 트랜젝션의 제 위치를 찾아서 선형리스트 안에 있는 LP2 의 종료 계수기 값과 개입중단시킨 LP1의 종료 계수기 값의 합이 주어진 종료 계수기 값인 START A에서의 A값과 비교한다. 만약 합이 A보다 적으면 주 프로세스는 튜플 공간을 통해 LP1에게 동기화 충돌 문제가 발생하지 않았다는 메시지를 전달하고, 메시지를 전달받은 LP1은 작업을 계속한다. 그러나 합이 A와 같거나 많으면 모든 LP는 시뮬레이션을 끝내야 하는 정확한 시간과 함께 시뮬레이션을 중단한다.

■ 각각의 LP가 작업을 끝내면 주 프로세스에게 값을 반환하고 주 프로세스는 반환되어진 값들을 모아 결과를 리포트 한다.

## 4. 실험 및 결과분석

### 4. 1 실험

4. 1. 1 GENERATE 문장이 두 개인 경우

GPSS Tugboat 프로그램에 대해 하나의 프로세서를 사용한 경우와 세 개의 프로세서(각각의 LP를 위해 두 개, 주 프로세스 한 개)를 사용한 경우의 실행시간을 50, 100, 500의 종료 계수기 값과 함께 변화시켜 가면서 실행하였다. 10번을 반복 실행하고 각각의 결과는 ( ) 안에 표시하였다.

<표 1>은 실행의 결과를 요약한 비교표이다. 「CDS」는 주 프로세스와 모든 LP들이 하나의 프로세서로 실행되고, 「NETWORK」은 주 프로세스와 각각의 LP가 다른 프로세서로 실행됨을 의미한다. 「ORIGINAL」은 변화 없이 주어진 GPSS 프로그램을 실행하였고 「CHANGE1」과 「CHANGE2」에서는 LP2에 있는 선박의 입항에 소요한 시간(berthing time)과 출항에 소요한 시간(deberthing time)을 0.0으로 변화시켜 실행하였다.

<표 1> LP = 2인 경우의 실행결과

| | | TC=50 | TC=100 | TC=500 |
|---|---|---|---|---|
| CDS | ORIGINAL | 3 (2,3,3,3,3,3,3,2,4,2) | 5 (6,4,6,5,5,5,4,6,4,5) | 23 (23,23,24,21,24,22, 22,25,23,22) |
| | # of rollback | 13 (7,17,12,17,14,14,15, 12,12,13) | 28 (23,18,39,31,24,22, 25,42,28,26) | 126 (136,122,120,113, 124,128,155,110,135, 120) |
| | # of rollback transaction | 135 (110,117,72,140,92, 87,146,99,237,252) | 235 (236,141,320,308, 186,164,174,411,236, 174) | 1351 (1066,1681,897,948, 1147,1192,2092, 1166,1444,1873) |
| | CHANGE 1 | 2 (1,2,1,2,2,1,2,1,2,2) | 3 (2,3,3,3,4,4,3,3,2,3) | 12 (11,16,11,10,12,13, 14,11,11,11) |
| | CHANGE 2 | 3 (2,3,3,3,2,2,3,3,3,2) | 4 (4,5,4,4,5,4,4,4,4,4) | 17 (20,16,18,15,14,21, 18,17,15,17) |
| NETWORK | ORIGINAL | 2 (2,2,2,2,2,2,3,2,3,2) | 4 (5,4,4,3,6,4,3,4,4,4) | 18 (16,20,17,21,13,22, 15,17,16,19) |
| | # of rollback | 14 (10,12,16,18,13,18,25, 9,12,11) | 28 (24,26,35,38,17,25, 31,29,30,26) | 130 (119,113,119,158, 140,138,127,119,126, 140) |
| | # of rollback transaction | 127 (103,106,146,152,149, 128,99,120,150,118) | 291 (165,307,400,220, 179,205,367,345,422, 298) | 1319 (1359,1254,1438, 1253,1388,887,1361, 1434,1376,1440) |
| | CHANGE 1 | 1 (1,1,1,1,1,1,1,1,1,1) | 2 (2,2,2,2,2,2,1,2,2,2) | 7 (8,8,6,8,8,6,6,7,7,8) |
| | CHANGE 2 | 2 (2,1,1,1,2,2,2,2,1,1) | 3 (3,3,3,2,2,4,3,3,3,4) | 12 (11,9,14,17,10,12,14, 13,9,13) |

따라서, 「CHANGE1」과 「CHANGE2」 모두는 LP2가 공유자원인 터그보트를 사용하지 않기 때문에 동기화 충돌의 문제는 전혀 발생되지 않으며, 또한 롤백도 일어나지 않는다. 그러나 LP2가 SEIZE/RELEASE TUGBOAT 문장을 실행할 때 「CHANGE1」은 공유자원인 터그보트에 대해 동기화 충돌문제 자체를 전혀 고려하지 않는 반면, 「CHANGE2」는 동기화 충돌문제가 발생되지 않음에도 불구하고 이의 확인과정을 거친다는 것에 차이가 있다. 따라서 「CHANGE2」는 동기화 충돌문제를 확인하기 위한 탐색시간을 추정하는데 사용되는 방법이다.

4.1.2 GENERATE 문장을 추가한 경우

위에서 제시한 GPSS 프로그램의 LP를 3개 혹은 4개로 확장시킨 프로그램은 <그림3>과 같고, <표 2>는 이를 실행시킨 결과이다.

```
        * LP3                          * LP4
GENERATE    40              GENERATE    50, 5
SEIZE       BERTHC          SEIZE       BERTHD
SEIZE       TUGBOAT         SEIZE       TUGBOAT
ADVANCE     2               ADVANCE     2
RELEASE     TUGBOAT         RELEASE     TUGBOAT
ADVANCE     10,4            ADVANCE     20, 5
SEIZE       TUGBOAT         SEIZE       TUGBOAT
ADVANCE     1               ADVANCE     1
RELEASE     TUGBOAT         RELEASE     TUGBOAT
RELEASE     BERTHC          RELEASE     BERTHD
TERMINATE   1               TERMINATE   1
```

<그림 3> Tugboat 프로그램의 확장

<표 2> LP의 수에 따른 실행결과 비교

| | # of GENERATE | 2 개(2 LPs) | 3 개(3 LPs) | 4 개(4 LPs) |
|---|---|---|---|---|
| CDS | Execution Time | 23 (23,23,24,21,24,22,22, 25,23,22) | 42 (42,39,42,42,36,46,46, 36,52,39) | 71 (83,70,58,58,58,91,70,7 4,87,62) |
| | # of Rollback | 126 (136,122,120,113,124, 128,155,110,135,120) | 273 (293,247,257,216,275, 324,238,297,365,216) | 412 (480,334,333,365,338,5 58,404,441,432,426) |
| | # of Rolledback Transaction | 1351 (1066,1681,897,948, 1147,1192,2092,1166, 1444,1873) | 2970 (3212,2488,3080,2756, 2684,3244,3076,2568, 4072,2516) | 4672 (5440,4548,39403828, 3704,6008,4776,4872,5 732,3876) |
| NETWORK | execution time | 18 (18,18,20,17,16,17,19, 19,19,19) | 30 (32,30,31,30,31,28,31, 29,29,30) | 45 (46,52,35,48,35,41,50,5 5,50,37) |
| | # of rollback | 130 (119,113,119,158,140, 138,127,119,126,140) | 235 (315,242,278,230,230, 117,247,221,247,221) | 505 (557,657,365,575,391,4 55,585,558,531,368) |
| | # of Rolledback Transaction | 1319 (1359,1254,1438,1253, 1388,887,1361,1434, 1376,1440) | 2459 (3172,2836,2940,2024, 2600,1384,3036,1956, 2176,2468) | 5627 (5532,6724,4352,6092, 4444,5208,6852,6792, 5968,4304) |

### 4.2 실험의 결과분석

프로세스의 수와 종료 계수기 값을 증가시켜가면서 실행 속도를 비교해 본 결과, 다음과 같은 사실을 확인할 수 있었다.

■ 한 개의 프로세서를 사용해서 TC=500으로 실행시킨 「CHANGE1」의 경우, 실행시간이 12초였다. 이는 동기화 충돌 문제나 롤백을 전혀 고려하지 않은 상태에서 12초의 실행시간이 걸렸음을 의미한다. 따라서 C-LINDA는 1개의 프로세서를 사용했을 때 많은 오버헤드가 발생됨을 알 수 있었다(<표 1> 참조).

■ 한 개의 프로세서를 사용해서 TC=500으로 실행시킨 「CHANGE2」의 경우, 실행시간이 「CHANGE1」의 12초에 비해 5초가 더 증가한 17초로 나타났다(<표 1> 참조). 이는 동기화 충돌의 문제가 발생되지 않음에도 불구하고 이의 확인과정을 반드시 거치는 「CHANGE2」의 기능에서 비롯된 것이다. 이때 SEIZE/RELEASE TUGBOAT 문장이 2번 사용되고 TC=500이므로 동기화 충돌 문제를 확인하기 위한 탐색은 1000번 일어난다. 따라서 1000번에 대한 탐색시간은 5초가 소요되었고, 이러한 결과는 TC=50인 경우와 TC=100인 경우에도 일률적으로 적용되었다.

■ 롤백이 일어나는 횟수가 많을수록 병렬 프로그램에서의 실행시간은 더욱 증가함을 확인할 수 있었다(<표 1> 참조).

■ TC=500으로 2개의 GENERATE 문장(두 개의 LP)을 한 개의 프로세서로 실행시킨 경우에는 실행시간이 23초였고, 세 개의 프로세서로 실행시킨 경우에는 18초로 나타났다. 즉, (23-18)*100/23 = 21.7% 감소하였다. 또한 3개의 LP를 실행시킨 경우 42초에서 30초로 28.6% 감소하였고, 네 개의 LP를 실행시킨 경우 71초에서 45초로 36.6% 감소하였다. 따라서 LP의 수를 고정해두고 프로세서의 수를 증가시킨 경우 실행시간이 현저히 감소됨은 물론, LP의 수를 증가시키면서 동시에 프로세서의 수를 증가시킬 경우 실행시간의 감소율이 점차 커짐으로써 더욱 효과적임을 알 수 있었다(<표 2> 참조).

■ 시뮬레이션의 과정에서 발생된 롤백의 수와 롤

<표 3> 프로세스별 시뮬레이션 복구시간

| | # of LPs | | LP1 | LP2 | LP3 | LP4 |
|---|---|---|---|---|---|---|
| CDS | 2 LPs | Rollbacks | 54 | 72 | | |
| | | Transactions | 601 | 750 | | |
| | | Simulation Backup Time | 1117.495 | 2268.687 | | |
| | 3 LPs | Rollbacks | 78 | 97 | 98 | |
| | | Transactions | 903 | 965 | 1102 | |
| | | Simulation Backup Time | 1126.198 | 2293.133 | 8022.754 | |
| | 4 LPs | Rollbacks | 104 | 113 | 97 | 98 |
| | | Transactions | 1026 | 1130 | 1280 | 1236 |
| | | Simulation Backup Time | 980.663 | 2450.206 | 8874.804 | 11097.224 |
| NETWORK | 2 LPs | Rollbacks | 63 | 67 | | |
| | | Transactions | 786 | 733 | | |
| | | Simulation Backup Time | 1746.955 | 1883.062 | | |
| | 3 LPs | Rollbacks | 74 | 92 | 69 | |
| | | Transactions | 774 | 836 | 849 | |
| | | Simulation Backup Time | 1003.048 | 2359.926 | 5920.075 | |
| | 4 LPs | Rollbacks | 111 | 144 | 120 | 130 |
| | | Transactions | 1341 | 1432 | 1448 | 1406 |
| | | Simulation Backup Time | 1595.080 | 3349.305 | 9681.135 | 13267.180 |

백된 트랜젝션의 수, 그리고 시뮬레이션 복구시간을 각 프로세스별로 나누어 정리해 보면 <표 3>과 같다.

두 개의 LP가 하나의 프로세서에서 실행될 때 전체 롤백의 횟수는 54+72=126이고 그 때의 시뮬레이션 복구 시간은 1117.495+2268.687=3386.182로, 하나의 롤백에 대하여 소요되는 평균 시뮬레이션 복구시간은 3386.182/126=26.874로 나타났다. 같은 방식으로 두 개의 LP를 세 개의 프로세서에서 실행했을 경우 평균 시뮬레이션 복구시간은 26.048이었고, 세 개의 LP를 실행했을 경우 하나의 프로세서를 사용하면 평균 시뮬레이션 복구시간은 41.912, 네 개의 프로세서를 사용하면 39.502로 나타났다. 또한 네 개의 LP를 실행했을 경우 하나의 프로세서를 사용하면 평균 시뮬레이션 복구시간은 56.803, 다섯 개의 프로세서를 사용하면 55.233으로 나타났다. 이로써 LP의 수가 증가할수록 롤백당 걸리는 평균 시뮬레이션 복구 시간이 증가함을 확인할 수 있었다.

## 5. 결론

병렬 컴퓨터의 등장은 컴퓨터 공학의 무한한 가능성과 새로운 차원을 개척했음에도 불구하고 이의 활용에는 고도의 숙련이 요구되고 프로그램 작성의 복잡성과 난해성으로 인하여 병렬 컴퓨터의 일반적인 활용에 걸림돌이 되고 있다.

이에 본 연구는 일반 컴퓨터에 널리 사용되고 있는 시뮬레이션 언어인 GPSS를 사용하여 병렬 컴퓨터 상에서 시뮬레이션을 구현함으로써 병렬 시뮬레이션에 대한 일반 프로그래머의 접근이 용이한 쉽고 편리한 방안을 제시하고자 하였다. 따라서 본 연구는 이를 위한 모델과 알고리즘을 제시하고 병렬 프로그램 언어인 C-LINDA를 사용하여 GPSS 프로그램의 실례인 Tugboat 프로그램에 적용, 구현하였다.

타당성의 검증에 있어서는 실험 사례로 사용한 Tugboat 프로그램을 하나의 프로세서로 실행시킨 경우와 여러 개의 프로세서로 실행시킨 경우로 나누어 프로세스(LP) 수와 종료 계수기 값의 변화에 중점을 두고 이들 값을 서로 비교·분석하였다. 그 결과 GPSS 프로그램 내에 LP의 수를 증가시킬수록 (GENERATE 문장이 많을수록), 순차적 시뮬레이션(하나의 프로세서에서 실행시킨 경우)에 비하여 분산시뮬레이션(여러 개의 프로세서에서 실행시킨 경우)에서의 실행시간이 현저하게 단축됨을 알 수 있었다. 따라서, GPSS 프로그램의 병렬 컴퓨터에서의 실행은 분산 시뮬레이션의 주목적인 실행속도의 감축에 크게 기여하고 있으며, GPSS에 대한 기본적 이해가 있는 일반 프로그래머라면 별반 어려움 없이 쉽게 접근할 수 있음을 알 수 있었다.

그러나 본 연구의 결과는 GPSS Tugboat Program에 한정된 것으로, 본 사례에서 나타난 GENERATE, ADVANCE, SEIZE/RELEASE, START, TERMINATE 문장들만을 고려하였다는 점에서 한계를 가지고 있다. 따라서 향후 본 연구를 더욱 진척시키기 위해서는 GPSS의 다른 문장, 즉 ENTER/LEAVE, PRIORITY, STORAGE, DEPART, TEST, INITIAL/SAVEVALUE, ASSIGN, SELECT, TABLE/TABULATE, LOGIC, GATE, TRANSFER, MARK 문장 등에 대해서도 심도있는 검토가 뒤따라야 할 것으로 본다. 또한 더 나아가 각 GPSS 문장을 병렬화 시킬 수 있는 알고리즘을 각기 독립된 함수로 만들어 쉽게 분산 시뮬레이션에 적용할 수 있는 방안에 대한 연구도 의미있을 것으로 본다.

## 참 고 문 헌


[1] 김기형, 김탁곤, 박규호, "DEVS에 기반한 분산 시뮬레이션 환경 D-DEVSim++의 설계 및 구현", 「한국 시뮬레이션 학회 논문지」, 제 5권, 제 2호(1996), pp. 41-58.

[2] 손진곤, "Petri Net과 DEVS 형식론을 이용한 컴퓨터 통신 프로토콜의 모델링," 고려대학교 박사 학위 논문(1991).

[3] 정영식, "SPN에 의한 이산사건 시뮬레이션 결과 예측에 관한 연구", 「한국 시뮬레이션학회 논문지」, 제 4권, 제 1호(1995), pp. 13-24.

[4] Ahuja, S., Carriero, N. J., Gelernter, D. H. and Krishnaswamy, V., "Matching Language and Hardware for Parallel Computation in the LINDA Machine", IEEE Transactions on Computers, Vol. 37, No. 8(1988), pp. 921-929.

[5] Carriero, N. and Gelernter, D., "LINDA in Context", Communications of the ACM, Vol. 32, No. 4(1989), pp. 444-458.

[6] Davies, R. and O'keefe, R., *Simulation Modelling with PASCAL*, Englewood Cliffs, NJ:Prentice Hall, Inc., 1989.

[7] Denning, P. J., Dennis, J. B., and Qualitz J. E., *Machine, Languages, and Computation, Englewood Cliffs*, NJ:Prentice Hall, Inc., 1978.

[8] Fujimoto, R. M., "Optimistic approaches to parallel discrete event simulation", Trans. of the Society for Computer Simulation, 7(2) (1990), pp. 153-191.

[9] Hoover, S. V. and Perry, R. F., *Simulation a Problem-Solving Approach*, Addison-Wesley Pub. Co. Inc., New York, 1989.

[10] Jefferson, D. and Sowizral, H., "Fast concurrent simulations using the Time Warp mechanism", Distributed Simulation, 15(2) (1985), pp. 63-69.

[11] Law, A. M. and Kelton, W. D., *Simulation Modelling and Analysis*, McGraw-Hill Book Co., 1982.

[12] Misra, Jayadev, "Distributed discrete-event simulation", ACM Computing Surveys, 18(1) (1986), pp. 39-65.

[13] Nicol, D. and Fujimoto, R., "Parallel Simulation Today", ORSA Journal on Computing(1994), pp.1-35.

[14] Original LINDA : C-Linda User's Guide & Reference Manual V2.5.2, Scientific Computing Associates Inc., 1993.

[15] Quinn, M, *Parallel Computing : Theory and Practice*, McGraw-Hill, Inc., 1994.

[16] Schriber, T. J., *Simulation Using GPSS*, John Wiley & Sons, 1974.

[17] Schriber, T. J., *An Introduction to Simulation using GPSS/H*, John Wiley & Sons, 1991.

[18] B.P. Zeigler, *Theory of Modelling and Simulation*, John Wiley, 1976.

[19] B.P. Zeigler, *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, 1984.

[20] B.P. Zeigler, *Theory of Modelling and Simulation*, A Wiley Interscience Pub., 1984.

## 저자소개

윤정미

1981년 성심여자대학 수학과(이학사)

1984년 이화여자대학교 수학과 전산전공(이학석사)

1995년 The City University of New York(전산학 박사)

1990~현재 유한대학 전산과 조교수

관심분야 : 모델링과 분산 시뮬레이션, 병렬 알고리즘