병렬성을 고려한 DEVS 모델의 파티션 알고리즘: 모델의 구조 정보를 이용*

A Concurrency Preserving Partitioning Algorithm of DEVS Models: Using Structural Information of Models

김기형** · 김탁곤*** · 박규호***
Ki Hyung Kim · Tag Gon Kim · Kyu Ho Park

Abstract

In this paper, we present a partitioning algorithm for distributed simulation of DEVS(Discrete Event System Specification) models. To preserve concurrency inherent in models, the proposed algorithm utilizes the structural information of models. Through benchmark simulation experiments, we show that the proposed algorithm can generate good partitions.

Key Words : Partitioning, Parallel/distributed discrete event simulation, Time Warp, DEVS formalism, Hierarchical simulation

1. Introduction

분산 (병렬) 시뮬레이션은 시뮬레이션 프로그램을 병렬 컴퓨터에서 실행시키는 것을 의미한다. 최근에 와서 분산 (병렬) 이산 사건 시뮬레이션은 상당한 관심을 끌고 있는데, 공학, 전산, 경제, 군사 등 여러 응용분야에서 시스템의 복잡도가 커짐에 따라 순차컴퓨터에서 수행하기에는 너무 많은 시간이 걸리게 되고 분산 (병렬) 환경에서의 시뮬레이션이 그 한가지 해결 방법이 되고 있기 때문이다.

분산 시뮬레이션은 크고 복잡한 시스템을 대상으로 하는 경우가 대부분이므로 모델의 검증(verification)과 타당성(validation)은 중요한 문제가 된다. 이를 위해 DEVS(Discrete Event Systems Specification) 형식론에 기반한 분산 시뮬레이션 기법이 제안되었다. DEVS 형식론은 다양한 시뮬레이션 세계관(world view)과 모델링 방법들을 통합하고 수학적으로 잘 정의된 모델 구성을 위한 시스템 이론적 방법론이다[15, 16]. DEVS 형식론은 모델의 구조(structure)와 행동(behavior)을 시뮬레이션 수행으로부터 추상화 시키기 위해 모델을 집합 이

* 이 논문은 1997학년도 영남대학교 학술연구 조성비 지원에 의한 것임.
** 영남대학교 컴퓨터공학과
*** 한국과학기술원 전기 및 전자공학과

론적 방법을 이용하여 기술하며, 이산 사건 시스템을 계층적(hierarchical)이고 모듈화(modular)된 형식으로 기술할 수 있다. 이 계층화와 모듈화 특성은 위에서 언급한 모델의 검증과 타당성 등에 많은 도움을 줄 수 있다. DEVS 모델의 분산 시뮬레이션에 대한 연구는 지금까지 많이 발표되었지만, 기존의 전통적인 분산 시뮬레이션과는 다른 방법들을 사용해 왔다. 그 이유는 DEVS 형식론이 모델의 외부 사건과 내부 사건을 분리하여 기술하고, 시뮬레이션 방법으로서 계층적 시뮬레이션이라는 방법을 사용해 왔기 때문이다[10]. 따라서 대부분의 병렬 DEVS 시뮬레이션 방법들은 DEVS 모델의 특정한 병렬성만을 이용해 왔는데, 최근에 DEVS 모델을 위한 일반적인(비동기적인) 분산 시뮬레이션 환경인 D-DEVSim++이 제안되었다[10, 11, 17]. D-DEVSim++은 낙관적 시뮬레이션 방법과 계층적 시뮬레이션 방법을 통합한 알고리즘을 사용하였다.

본 논문의 목적은 D-DEVSim++환경에서의 모델 파티션 알고리즘을 제시하는 것이다. 모델 파티션 문제는 분산 시뮬레이션에서 성능에 가장 큰 영향을 미치는 요소 중의 하나이다. 따라서 가장 빠른 시뮬레이션 시간을 얻기 위한 모델 파티션 문제는 분산 시뮬레이션의 연구에서 중요한 목표가 되어 왔다. 만일 모델 파티션시, 시뮬레이션에 대한 완전한 지식(각 시뮬레이터에서의 사건 처리 순서, 각 사건의 처리 시간, 사건들간의 선행 관계(precedence), 그리고 최적의 사건 스케줄 등)을 이용하려 한다면 이 문제는 기본적으로 NP-complete 문제이다[3]. 문제를 좀더 실제적으로 본다면, 이러한 시뮬레이션의 실행에 관계된 지식을 예측하기란 불가능하다. 더욱이 이러한 시뮬레이션 지식을 안다고 가정하고 이 지식을 이용하여 모델을 파티션하였다고 하더라도 이 파티션의 좋고 나쁨을 판단할 수 있는 알려진 알고리즘이 없다. 그러므로 모델 컴파일 시간의 불완전한 지식에 기반한 준 최적 모델 파티션을 하는 다항식 알고리즘이 합당하다. 그러한 알고리즘들은 다음과 같은 단순화된 목표에 기반을 두고 있다: (1) 프로세서간 통신의 최소화, (2) 파티션의 계산 로드간의 균형, (3) 독립적인 시뮬레이션 사이클간의 병렬 실행을 최대화. 그러나 아직 모델의 병렬성을 어떻게 측정하고 어떻게 컴파일 시간에 이 정보를 알아내는가 하는 문제는 아직 일반적인 해답이 없고 다음과 같이 많은 연구가 진행되어 왔다. 랜덤 파티션 방법은 가장 쉽게 사용될 수 있는 방법이다[14]. Smith et al.[14]은 동적인 행동을 가지는 복잡한 그래프에 대해 랜덤 파티션이 다른 방법들과 비교해서 훨씬 적은 계산양으로 좋은 성능을 낼 수도 있음을 주장했다. 분산 로직 시뮬레이션에서는 대부분의 파티션 방법들이 두 가지 최적화 목표((1) 로드의 균형과 (2) 파티션간 통신의 최소화)에 기본적인 바탕을 두고 있는 코스트 함수를 채용해 왔다[4, 1, 9, 2, 12]. Kapp et al.[9]은 위의 코스트 함수에 보수적 방법의 동기화 정도를 새로운 인자로서 포함시켰다. Cong et al.[4]과 Bagrodia et al.[1]는 combinational Boolean 회로에 대한 분산 시뮬레이션에서 *K-AFM*이라고 부르는 acyclic multi-way 파티션 알고리즘을 제안하였다. 이 알고리즘은 유명한 *FM* 알고리즘[5]에 기반을 두고 있고 다음과 같은 두 가지 특징이 있다. 첫째, 그들은 태스크 그래프에 방향성을 둠으로써 회로의 구조를 파악하는데 도움이 됨을 보였다. 둘째, 서로 파티션들 사이에 cyclic dependency가 있는 경우 롤백이 많이 일어난다는 정보를 이용하여 이러한 dependency가 될 수 있으면 없도록 하였다. Smith et al.[14]는 게이트 레벨 시뮬레이션에서 파티션간의 병렬성을 최대로 하기 위해 *cone*이라는 개념을 도입하였다. Cone은 한 게이트의 출력단에 의해 영향을 받는 게이트들의 집합으로 정의된다. 이러한 cone의 개념은 acyclic multi-way 파티션 알고리즘에서 사이클을 찾아내는데도 사용되었다[4, 1]. Chawla[3]는 모델의 병렬성에 대한 척도로서 *linearity fraction*라는 것을 도입하였다. 그는 시뮬레이션 태스크들을 *directed acyclic graph (DAG)*로 나타내고 그들 간의 병렬성을 linearity fraction으로 예측하고 파티션하였다. 지금까지는 컴파일 시간에 모델의 병렬성을 예측하고 파티션하는 정적인 알고리즘들을 살펴보았는데, 컴파일 시간에는 간단히 초기적 파티션을 하고, 실제 시뮬레이션을 진행하면서 모델들의 행동을 살펴보고 파티션을 변경하는 동적인 파티션 방법도 제시되었다[7]. 그러나 그러한 알고리즘들은 별로 성공을 거두지 못하였는데 가장 큰 이유로 모델들을 다른 노드 컴퓨터로 이동시키는 비용이 너무 크기 때문이었다. 지금까지 설명한 알고리즘들은 DEVS 모델의 분산 시뮬레이션 환경인 D-DEVSim++에서 사용될 수 없는데, 이는 모델이 계층적으로 구성

되어 있고 또한 태스크 그래프 역시 트리의 구조이기 때문이다.

본 논문에서는 DEVS 모델의 분산 시뮬레이션을 위한 새로운 파티션 알고리즘을 제안한다. 제안한 알고리즘은 위의 세가지 단순화된 목표에 기반을 두고 있으며, 모델의 병렬성을 예측하기 위해 DEVS 모델 개발 방법론에서 자연스럽게 얻을 수 있는 모델의 계층적 구조 정보를 이용한다.

본 논문은 다음과 같이 구성된다. 2장에서는 본 연구의 기반이 되는 DEVS 형식론과 시뮬레이션 방법을 설명한다. 3장에서는 분산 시뮬레이션의 파티션 문제를 정의하고 분석한다. 4장에서는 제안된 계층적 파티션 알고리즘을 설명한다. 5장에서는 제안된 알고리즘의 성능을 벤치 마크 시뮬레이션을 통해 알아본다. 마지막으로 6장에서는 본 논문의 결론을 맺는다.

2. DEVS 형식론과 Time Warp

DEVS 형식론은 이산 사건 모델을 계층적이고 모듈화 된 형태로 표현할 수 있다[16]. 계층적인 모델 구성을 위해서, DEVS 형식론은 원소(atomic) 모델과 연결(coupled) 모델을 사용한다. 원소 모델은 시스템의 행동(behavior)을 나타내며, 연결 모델은 자신의 서브 모델들이 어떻게 연결되는가를 나타낸다. 연결 모델은 더 큰 연결 모델의 서브 모델이 될 수 있는데 이 특성을 이용하여 복잡한 모델이 계층적으로 구성될 수 있는 것이다. 원소 모델 AM은 다음과 같은 구조로 정의된다.

$AM = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$
where,
X : 외부 입력 사건 집합.
S : 순차 상태 집합.
Y : 외부 출력 사건 집합.
δ_{int} : $S \rightarrow S$: 내부 상태 전이 함수.
δ_{ext} : $Q \times X \rightarrow S$: 외부 상태 전이 함수.
λ : $S \rightarrow Y$: 출력 함수.
ta : $S \rightarrow R^{+}_{0,\infty}$: 시간 진행 함수.

여기서 Q는 M의 전체 상태들의 집합으로서 다음과 같이 나타낼 수 있다.

$Q = \{(s, e) \mid s \in S \text{ and } 0 \le e \le ta(s)\}$.

일반적인 모듈화 된 명세(specification)와 마찬가지로 원소 DEVS 모델은 외부 세계와 입력 및 출력 포트를 통하여 통신한다. 좀더 상세히 말하면, 다른 모델로 부터의 외부 입력 사건이 입력 포트로 들어오면, 모델은 외부 상태 전이 함수를 사용하여 어떻게 반응할 것인가를 결정한다. 만일 외부 사건이 다음 스케줄 시간(이 시간은 시간 진행 함수에 의해 결정된다)까지 들어오지 않으면 모델은 내부 사건 전이 함수를 사용하여 상태를 전이한다. 이때 외부 출력 메시지도 출력 함수를 사용하여 동시에 출력한다. 이 출력 메시지는 다시 다른 모델에 외부 입력 사건으로서 역할을 하게 되고 이러한 전체 과정이 되풀이 되는 것이다.

여러 원소 모델들은 모여서 하나의 연결 모델이 될 수 있다. 이러한 연결 모델은 DEVS 형식론의 모듈화 특성에 따라 동등한 원소 모델로 표현될 수 있다. 그러므로 연결 모델은 그 자체가 여러 개 모여서 또 다른 연결 모델을 구성할 수 있는 것이다. 연결 모델 DN은 다음과 같이 정의된다.

$CM = \langle D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, SELECT \rangle$
where,
D : 구성 요소(component) 이름 집합.
For each i in D
M_i : 구성요소 i에 대한 DEVS.
I_i : i의 influencee 모델들의 집합.
For each j in I_i
$Z_{i,j}$: $Y_i \rightarrow X_j$: i 모델의 출력을 j 모델의 입력으로 연결하는 함수.
$SELECT$: D의 부분 집합 $\rightarrow D$: 여러 구성요소 모델들이 같은 시간에 스케줄을 원할 때 그것들을 순서화 시키는 함수.

2.1 계층적 시뮬레이션 기법

DEVS 모델의 시뮬레이션은 추상 시뮬레이터(abstract simulator)라고 부르기도 하는 계층적 시뮬레이션 기법

에 기반을 두고 있다[16]. 추상 시뮬레이터는 DEVS 모델을 실행시켜주는 일종의 가상 프로세서(또는 알고리즘)로서, 각 DEVS 모델마다 하나씩 할당되고 또한 연결 모델의 연결 정보에 따라 서로 연결된다. 두 가지 종류의 추상 시뮬레이터가 있는데, 원소 모델을 위해서 SIMULATOR, 연결 모델을 위해서 COORDINATOR라고 부른다.

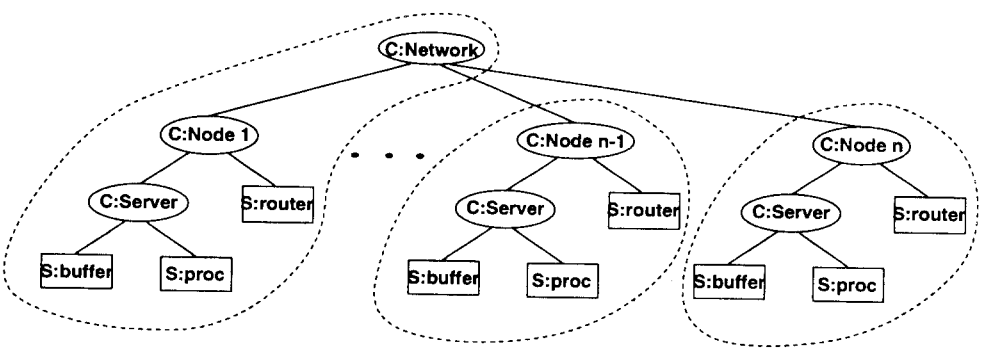
SIMULATOR와 COORDINATOR는 (*, t), (x, t), (y, t), (*done*, t_N)의 네가지 메시지 타입을 사용하여 보다 상위의 COORDINTOR와 시뮬레이션에 필요한 정보를 주고 받는다. 위에서 t는 시뮬레이션 시간이고 t_N은 다음 스케줄 시간이다. (x, t)와 (y, t) 메시지는 각각 외부 입력과 출력 사건 정보를 가지고 있다. (*, t)와 (*done*, t_N) 메시지는 스케줄링을 위해서 사용한다. 즉, (*, t) 메시지는 SIMULATOR에게 내부 사건(*)이 도착함을 나타낸다. 또한, (*done*, t_N) 메시지는 새로운 스케줄을 만들기 위해 사용된다. 이 메시지들을 사용하여 추상 시뮬레이터는 계층적으로 스케줄링 및 시뮬레이션을 진행한다. 추상 시뮬레이터의 연산에 대한 자세한 설명은 [16]을 참조하라.

2.2 D-DEVSim++ 환경

D-DEVSim++은 DEVS 모델을 위한 분산 시뮬레이션 환경이다[10, 17]. D-DEVSim++은 시뮬레이션 알고리즘으로서 DOHS(Distributed Optimistic Hierarchical Simulation) 기법을 채용하였는데, 이 기법은 계층적 시뮬레이션 기법과 Time Warp 기법의 혼합 알고리즘이다.[18] Time Warp 기법은 전통적 분산 시뮬레이션에서 가장 많이 사용되고 있는 동기화 기법이다[8, 6]. Time Warp에서는 각 프로세스가 메시지가 도착하는 즉시 처리하고 나중에 인과관계 조건에 위배되는 것이 발견되면 그것을 수정(롤백이라고 부른다)하게 함으로써 시뮬레이션을 진행한다.

DOHS 기법은 기본적으로는 계층적 시뮬레이션에 기반을 두고 있으므로 시뮬레이션의 태스크 그래프로서 계층적으로 계층적으로 형성된 추상 시뮬레이터들을 가정한다. 시뮬레이션의 첫번째 단계는 이러한 계층적 추상 시뮬레이터를 파티션하고, 각 노드 컴퓨터에 이들 파티션을 맵핑시킨다. 한 예로서, 〈그림 1〉은 큐잉 네트워크 모델의 추상 시뮬레이터를 분산 시뮬레이션하기 위해 파티션한 것을 보여준다.

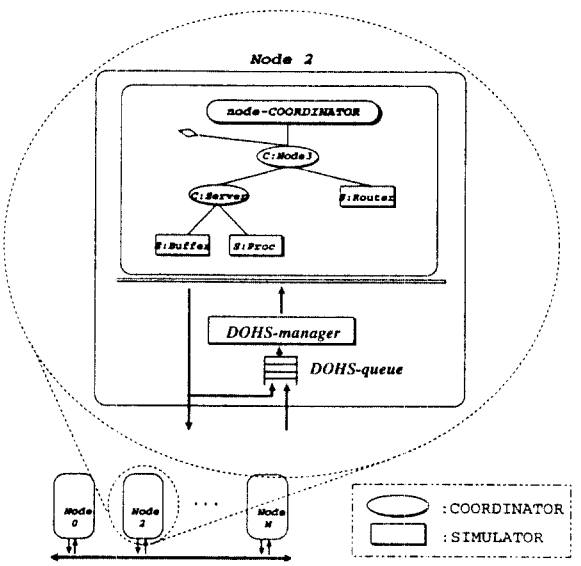
또한 각 노드 컴퓨터는 자신만의 스케줄러가 따로 있어서 독자적으로 시뮬레이션을 진행시킨다. 그리고, 각 노드 컴퓨터에서 진행되고 있는 시뮬레이션은 Time Warp 기법[8, 6]을 사용해서 서로 동기 시킨다. Time Warp 기법은 보수적 방법[13]과 함께 분산 시뮬레이션에서 가장 많이 쓰이는 동기화 알고리즘중의 하나이다. Time Warp 기법은 시뮬레이션상에서 미래의 사건이 과거의 사건의 실행에 영향을 미쳐서는 않된다는 인과관계 조건(causality constraint)을 만족시키기 위하여, 롤백이라는 실행 수정 방법을 사용한다. 즉 각 노드 컴퓨터는 인과관계를 고려하지 않고 시뮬레이션을 진행하다가, 인과관계 오류가 발견될 때마다 이를 롤



〈그림 1〉 CQN(Closed Queuing Network) 모델의 추상 시뮬레이터를 분산 시뮬레이션하기 위한 파티션

백을 통하여 수정하는 것이다.

DOHS 기법은 계층적 시뮬레이션 기법과 Time Warp을 서로 결합시키기 위하여 〈그림 2〉와 같은 시뮬레이션 구조를 가진다. 즉, 각 노드 컴퓨터는 지역 스케줄러 (node-Coordinator), DOHS-manager, DOHS-queue, 그리고 추상 시뮬레이터를 가진다. DOHS-queue와 DOHS-manager는 메시지들의 처리를 제어하는 큐와 제어기이다. 각 파트의 연산에 대한 자세한 설명은 [10]를 참조하라. 여기서는 다만 시뮬레이션이 계층적으로 이루어지고, 각 파티션간에는 Time Warp에 의해 동기화 된다는 것을 주의하자.



〈그림 2〉 DOHS 기법의 전체 구조

3. 분산 시뮬레이션에서의 파티션 문제

이 장에서는 분산 시뮬레이션에서의 파티션 문제의 특성을 분석해본다. 또한 이 분석에 근거하여 일반적인 해결 방법에 대해 알아본다. 본 논문에서 고려하는 분산 시뮬레이션은 일반성을 잃지 않으면서 시간 윈도우를 가지는 낙관적 분산 시뮬레이션으로 가정한다[6]. 여기서 시간 윈도우는 병렬로 실행 가능한 시뮬레이션 시간 폭을 의미하며 시뮬레이션의 병렬성을 제어하기 위해 흔히 사용된다. 이러한 시간 윈도우를 갖는 낙관적 분산 시뮬레이션은 가장 많이 사용되고 있는 기법이다. 시뮬레이션 태스크 i는 시뮬레이션 수행 중 다음과 같은 활동을 하게 된다.

1. 참 계산, T_i : 롤백되지 않는 시뮬레이션 사이클로서 다음과 같이 두 파트로 구성된다: (1) 모델 실행 시간(M_i)과 (2)상태 저장 시간(S_i).
2. 지역 동기(local synchronization), L_i : 지역 동기와 관련된 연산으로 다음과 같이 세 파트로 구성된다: (1) 거짓 계산(결국에는 롤백되는 시뮬레이션 사이클, F_i), (2) 과거 메시지에 의한 롤백 (R_i), (3) 아무 일도 하지 않는 휴지(idle) 사이클(I_i).
3. 전역 동기, G_i : 전체 시뮬레이션 진행을 관리하기 위해 사용하는 연산 (예: 전체 시뮬레이션 시간(GVT) 계산, 흐름 제어, 메모리 관리).

그러므로, 파티션 P하에서 프로세서 i에 대한 전체 시뮬레이션 실행 시간 ET_{P_i}는 다음과 같이 나타낼 수 있다.

$$\forall i,\ ET_{P_i} = T_i + L_i + G_i \tag{1}$$

$$= M_i + S_i + F_i + R_i + I_i + G_i. \tag{2}$$

이때, 다음의 등식은 항상 성립한다.

$$T_i + L_i + G_i = T_j + L_j + G_j, \qquad \forall i,j. \tag{3}$$

스피드업은 순차 시뮬레이션 시간과 분산 시뮬레이션 시간의 비율로 나타내어진다. 순차 시뮬레이션에서는 동기 문제에 대한 연산이 필요 없기 때문에 모델 실행 시간(M)만이 포함된다. 이 모델 실행 시간은 시뮬레이션 방법이나 파티션 알고리즘에 상관없이 항상 일정하기 때문에 순차 시뮬레이션의 총 실행 시간(ET_S)은 다음과 같이 나타낼 수 있다.

$$ET_S = \left(\sum_{i=0}^{m-1} M_i\right) \tag{4}$$

위의 수식에서 m은 파티션의 수이다. 수식 (2), (3), and (4)를 이용하면 스피드업은 다음과 같다.

$$(\sum_{i=0}^{m-1} M_i) / (M_j + S_j + F_j + R_j + I_j + G_j), \quad \text{for any } j. \quad (5)$$

최대의 스피드업을 얻으려면, 수식 (2)의 각 항목을 최소화 하면 될 것이다. 수식 (2)에서 M_j만이 시뮬레이션을 위한 유용한 연산이다. 그러므로 평균 모델 실행 시간 M_{avg}를 다음과 같이 정의한다.

$$M_{avg} = (\sum_{i=1}^{m-1} M_i) / m \quad (6)$$

위 식에서 m은 프로세서의 숫자이다. 상태 저장 연산(S_j)과 전역 동기 연산(G_j)은 낙관적 분산 시뮬레이션을 위해 꼭 있어야 하는 항목들이다. 그러므로 스피드업을 최대로하기 위해서는 파티션 알고리즘은 일반적으로 다음과 같은 항목들을 고려해야 한다.

1. 각 프로세서의 계산 부하 (또는 모델 실행 시간 M_j)를 균형있게 나누어야 한다.

$$\max_{j=0:m-1} M_j \rightarrow M_{avg} \quad (7)$$

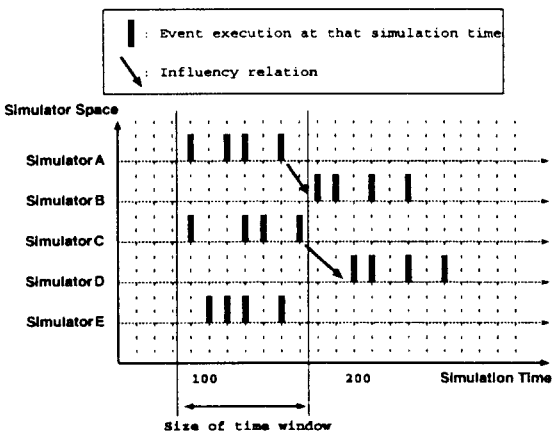
2. 지역 동기 연산 (L_i)는 최소화되어야 한다.

$$\forall i, \quad L_i = F_i + R_i + I_i \rightarrow 0. \quad (8)$$

그러므로 가능한 최대의 스피드업은 다음과 같이 나타내어질 수 있다.

$$(\sum_{i=0}^{m-1} M_i) / (M_{avg} + S_j + G_j), \quad \text{for any } j. \quad (9)$$

이러한 최대 스피드업을 얻을 수 있는 파티션을 최적의 파티션이라 부른다. 이제는 어떻게 L_i를 최소화할 수 있는지 생각해 보자. 롤백은 궁극적으로 각 프로세서에서 메시지들이 시뮬레이션 시간 순서대로 도착하지 않기 때문에 일어난다. 그러므로 각 프로세서의 지역 시뮬레이션 시간들의 차이를 최소화 한다면 이와 같은 순서 없는 메시지들의 숫자는 줄어들 것이다. 이를 위해 파티션 알고리즘은 다음과 같은 요소들을 고려해야 한다: (a) 프로세서간 통신을 최소화한다, (b) 독립적인 시뮬레이션 태스크의 숫자를 최대로 한다. 즉, 만일 시뮬레이션 태스크들이 서로 독립적이라면 그들 사이의 통신은 일어나지 않을 것이고 롤백 또한 일어나지 않을 것이다. 더욱이 메시지 통신 지연은 추가적인 롤백을 일으킬 것이다. 독립적인 시뮬레이션 태스크의 숫자를 최대화 하려면 시뮬레이션 사이클들 간의 영향 관계와 같은 실행 시간 행동을 알아야 한다. 〈그림 3〉은 시뮬레이션 태스크의 병렬성을 보여준다. 비록 모든 시뮬레이터가 같은 계산 로드를 가지더라도 임의의 시뮬레이션 시간에서 서로 영향 관계가 없는 태스크들만이 병렬로 실행될 수 있다. 예를 들어, 시뮬레이터 A와 B는 시간 100 부근에서 영향 관계를 가지므로 그 시간에 서로 병렬로 실행될 수 없다. 그러므로 만일 이들 태스크들을 (A,C,E)와 (B,D)의 두 그룹으로 나눈다면 우리는 병렬성을 전혀 얻을 수 없을 것이다. 그러나 컴파일 시간에 이러한 실행 시간 정보를 정확히 알아내기란 매우 어렵다. 따라서 제시된 방법은 계층적 DEVS 모델 디자인 방법론으로부터 고 수준 모델 구조 정보를 이용한다.



〈그림 3〉 시뮬레이션 태스크들의 병렬성

4. 모델의 고수준 구조 정보를 이용한 계층적 파티션 알고리즘

파티션을 위해 우선 시뮬레이션 태스크를 다음과 같이 정의된 가중치를 갖는 태스크 트리로 변환한다.

Definition 1 태스크 트리는 $G = (V, E, C, T)$로 정의된다. 이때 $V = \{n_j, j = 1 : v\}$, $v = |V|$는 노드의 집합,

$E = \{e_{i,j} = \langle n_i, n_j \rangle\}$는 통신 엣지의 집합, C는 엣지의 통신 코스트의 집합, 그리고 T는 노드의 계산 코스트의 집합이다.

태스크 트리의 설명을 위해 다음과 같은 명명법을 사용한다. 만일 $e_{i,j}$가 E에 있으면 n_i는 n_j의 부모이고 n_j는 n_i의 자식이다. 만일 n_i로부터 n_j로 경로가 있고 $n_i \neq n_j$이면 n_i는 n_j의 조상이고 n_j는 n_i의 후손이다. 후손이 없는 노드는 잎사귀 노드라고 부른다. 노드 n_i의 서브트리는 n_i 자신을 포함해서 n_i의 후손을 의미한다. 이때, n_i는 그 서브트리의 루트라고 부른다. 태스크 트리에서 노드 n_j의 깊이는 그 태스크 트리의 루트로부터 n_j까지의 경로의 길이이다.

$c_{i,j} \in C$는 엣지 $e_{i,j} \in E$에서의 통신 코스트이다. 노드 n_i의 계산 코스트, $\tau_i \in T$는 두개의 튜플 $\langle p_i, s_i \rangle$로 구성된다. 이때 p_i는 n_i의 계산 코스트이고 s_i는 n_i의 서브트리의 코스트를 합산한 합산 코스트이다. 노드 n_i의 계산 코스트 p_i는 순차 시뮬레이션 수행중 모든 입력 메시지를 처리하는데 걸린 모델 실행 시간이다. 즉 이 코스트는 분산 시뮬레이션의 오버헤드 $(L + G)$는 포함하지 않는다.

$$p_j = M_j \tag{10}$$

노드의 계산 코스트는 사실 모델의 크기에 비례하지는 않는다. 즉 사이즈는 작더라도 더욱 자주 실행되는 노드가 더 코스트가 클 수 있는 것이다.

서브 트리 G_i의 합산 코스트 s_i는 다음과 같이 G_i의 계산 및 통신 코스트의 합이다.

$$s_i = \sum_{n_k \in G_i} p_k + \sum_{n_k \in G_i} (c_{k,v} + c_{w,k}). \tag{11}$$

이제 파티션 문제를 지금까지 설명한 명명법으로 나타낼 수 있다. 즉 파티션 문제는 태스크 트리 G의 노드들을 최대 m개의 파티션 $(K_0, K_1, ..., K_{m-1})$으로 맵핑하는 문제이다. 즉 파티션 문제는

$$min \max_{i=0:m-1} \left(\sum_{n_j \in K_i} (p_j + (c_{j,v} + c_{w,j})) \right). \tag{12}$$

의 목적 함수를 만족시키도록 태스크 트리 G의 노드들을 다음과 같이 맵핑시키는 것이다.

$$map(n_j) = i, \qquad j = 1 : v \text{ and } i = 0:(m-1) \tag{13}$$

Algorithm 1 Hipart(n_v)

▷ *Initially,* $L_{avg} = s_v / m + (\sum_i \sum_j c_{ij}) / |E|$, *where* $|E|$ *is the number of edges in* E.

▷ *Check if there are imminent children* n_w*s whose loads are greater than* L_{avg}.

1. **while** (child n_w and $s_w > L_{avg}(1 + \alpha)$) **do**

   ▷ n_v *is too big to be inserted into one partition and should be partitioned more.*

2. Hipart(n_v);
3. **end while**

   ▷ *Now,* n_v *has only those children whose loads are smaller than* L_{avg}

4. **while** ($s_v > L_{avg}(1 + \alpha)$ and number of partitions already made $< m$) **do**
5. Make a partition P for satisfying the objective function $H = min|(2 \cdot c_{v,i} + \sum_{n_i \in P, n_i \in K} s_i - L_{avg}|$, where K is the set of children of n_v;

   ▷ *Update* L_{avg}.

6. $L_{avg} = \sum_{n_i \in P, n_i \in P} (c_{i,k} + p_j)$;
7. After partitoning, update the computation and communication costs of parents of n_v, including n_v itself;
8. **end while**

〈그림 4〉 계층적 파티션 알고리즘

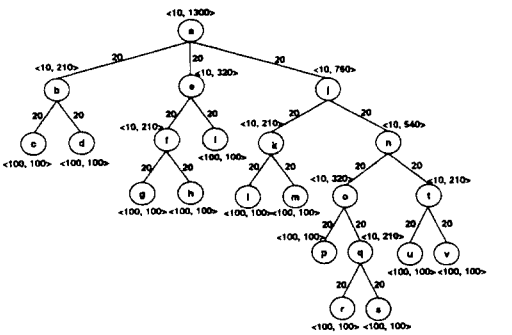
이때 파티션의 갯수가 최대 m임을 주의하자. 파티션 문제의 최종 목적은 전체 시뮬레이션 시간을 최소로 하는 것이지 주어진 프로세서들을 최대한 이용하자는 것이 아니기 때문이다. 시뮬레이션 모델의 특성에 따라서 이 두가지 목표는 서로 상반될 수 있다. 즉 더 많은 프로세서를 사용한다고 하더라도 시뮬레이션은 적은 프로세서를 사용했을 때보다 더 길게 걸릴 수 있는 것이다.

〈그림 4〉는 제시된 알고리즘을 보여준다. 제시된 알고리즘은 파티션 수행중 모델의 구조를 유지하려고 노력한다. 태스크 그래프가 트리이므로 재귀형태의 알고리즘이 가능하다. 알고리즘은 태스크 트리의 루트 노드로부터 시작한다. 알고리즘이 노드에 들어가면, 우선 목적 파티션의 평균 로드 (L_{avg})보다 큰 코스트를 가지는 자식이 있는가를 체크한다. 만일 그러한 자식이 있다면, 그 자식은 한 파티션에 들어가기에는 너무 크므로 더욱 잘게쪼개어 져야 한다. 그러므로 알고리즘은 그 자식 노드로 들어간다. 이러한 재귀적 프로세스를 통해 알고리즘은 L_{avg}보다 작은 코스트를 가지는 자식들만을 가지는 노드를 발견하게 된다. 이때, 우리는 알고리즘이 그 노드에 방문한다 라고 한다. 이렇게 알고리즘이 어느 노드에 방문하게 되면 그 노드의 자식들을 최소의 코스트를 가지는 파티션으로 나눈다.

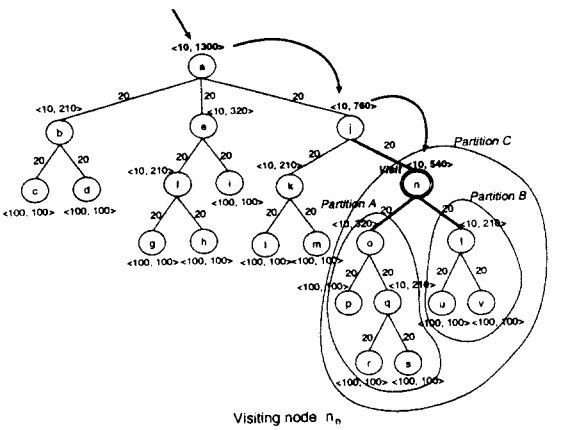
예를 들어 〈그림 5〉의 태스크 트리를 생각해 보자. 태스크 트리의 정의에 따라 각각의 노드 n_j와 에지 $e_{i,j}$는 $\langle p_j, s_j \rangle$와 $c_{i,j}$를 코스트로 가진다. 파티션전의 각 노드의 합산 코스트는 엣지의 통신 코스트를 포함하지 않음을 주의하자. 정의로 부터 같은 파티션안에 있는 엣지의 통신 코스트는 0이고, 서로 다른 파티션을 연결하는 엣지만이 0이 아닌 통신 코스트를 가지기 때문이다.

이 예제에서는, 태스크 트리를 세개로 나누려고 한다. 초기단계에서, 알고리즘은 각 파티션의 평균 로드 L_{avg}를 다음과 같이 예측한다.

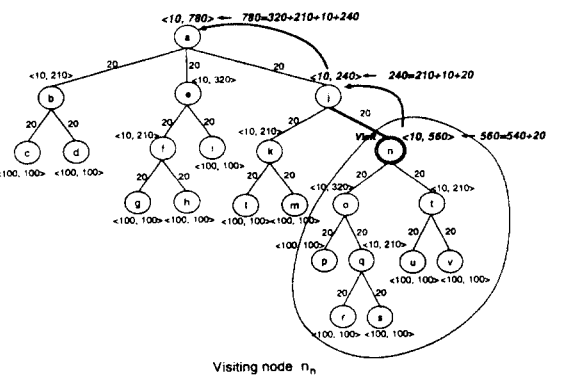
$$L_{avg} = s_a / m = 1300 / 3 = 433.$$



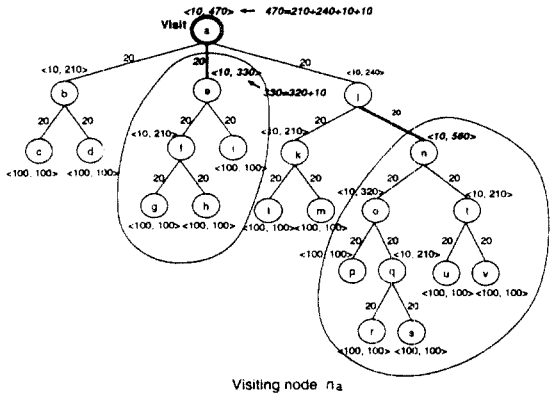
〈그림 5〉 예제 태스크 트리 T



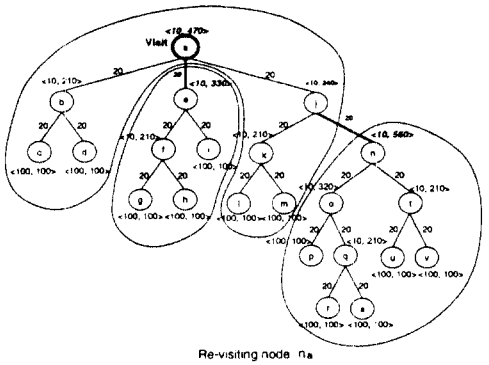
〈그림 6〉 예제 태스크 트리 T에 대한 파티션 과정 (a)



〈그림 7〉 예제 태스크 트리 T에 대한 파티션 과정 (b)



〈그림 8〉 예제 태스크 트리 T에 대한 파티션 과정 (c)



〈그림 9〉 예제 태스크 트리 T에 대한 파티션 과정 (d)

파티션 알고리즘은 파티션간의 병렬성을 최대로 하면서 각 파티션을 구해진 평균 로드 L_{avg}에 최대로 가깝게 나누려고 하는 것이다. 알고리즘 HIPART가 허용하는 각 파티션의 로드의 한계는 $L_{avg}(1 \pm \alpha)$이다. 여기서 $L_{avg} \cdot \alpha$는 각 파티션간의 평균 로드의 최대 허용 차이(mismatch)이다. α가 클수록 파티션간의 로드의 차이가 클 수 있는 것이다. 이 값은 파티션과정이 진행함에 따라 최신 값으로 변경된다. 알고리즘 HIPART는 루트 노드 n_a에서 시작해서 다음과 같은 과정을 통해 태스크 트리를 파티션한다.

1. 알고리즘 HIPART가 노드 n_a에 들어가면 세 자식 노드 n_b, n_e, 그리고 n_j를 발견하게 된다. n_j의 합산 코스트 s_j는 760(〉/433(1 + α)/)이고, 이 값은 한 파티션안에 들어가기에는 너무 큰 값이다. 그러므로, 알고리즘은 n_j를 더 잘게 자르기 위해 n_j에 들어가게 된다.
2. n_j에서 알고리즘은 n_n의 합산 코스트 s_n이 540(〉|433(1 + α)|)이고 이 값 역시 한 파티션안에 들어가기에는 너무 크다는 것을 발견하게 된다. 따라서 알고리즘은 다시 n_n에 들어간다.
3. n_n에 들어 가서야 알고리즘은 모든 자식 노드들이 L_{avg}보다 작은 합산 코스트를 가짐을 알게 된다. 이제 알고리즘은 n_n을 방문하게 되고 n_n의 자식노드의 일부로 구성된 파티션을 만들려고 한다〈그림 6〉. 이 경우 세가지 파티션이 가능하다. 첫째는 n_n앞에서, 두번째는 n_t앞에서, 마지막으로 세번째는 n_n앞에서 파티션하는 것이다(〈그림 6〉에 이 세가지 파티션이 나타내어져 있다.). 각 파티션의 합산 코스트는 각각 320, 210, 그리고 540이다. 알고리즘은 다음과 같이 L_{avg}와 이들 합산 코스트 사이의 차이가 최소가 되는 파티션을 선택하게 된다.

$$H_p = min(\,/(320 + (2 \cdot 20)) - 453/,\; /(210 + (2 \cdot 20)) - 453/,\; /(540 - (2 \cdot 20)) - 453/\,)$$

알고리즘은 결국 540을 합산 코스트로 가지는 세번째 파티션을 선택하게 된다〈그림 7〉. 이때 주의할 점은 540은 이 파티션을 다른 파티션과 연결할 엣지인 e_{nt}의 통신 코스트 c_{nt}를 포함하지 않았다는 것이다. 따라서 540에 c_{nt}를 더하면 이 파티션의 최종 합산 코스트는 560(540+20) 된다. 이제 알고리즘은 L_{avg}와 방금 전에 만든 파티션의 합산 코스트와의 차이값을 반영하여 새로운 α의 값을 얻어 낸다. 또한, 노드 n_n에 대한 방문의 마지막 절차로서 n_n부터 태스크 트리 T의 루트 노드까지의 계산 코스트를 재계산한다(이 재계산된 결과는 〈그림 7〉에 나타내어져 있다).

4. 알고리즘이 노드 n_j에 다시 들어가게 되면 이제는 n_j의 합산 코스트가 L_{avg}보다 작음을 알게 되고, 다시 n_a에 들어가게 된다.
5. 알고리즘이 노드 n_j에 다시 들어가면, 모든 자식 노드들의 합산 코스트가 L_{avg}보다 작음을 알게 된

다. 그러므로, 알고리즘은 L_{avg}에 가장 가까운 파티션을 찾으려고 노력하게 되고, 결국은 〈그림 8〉에 보인바와 같이 n_r앞에서 파티션을 만들게 된다.

6. 이제는 〈그림 9〉와 같이 모든 남은 노드들이 마지막 파티션에 들어가게 된다.

5. 실험 결과

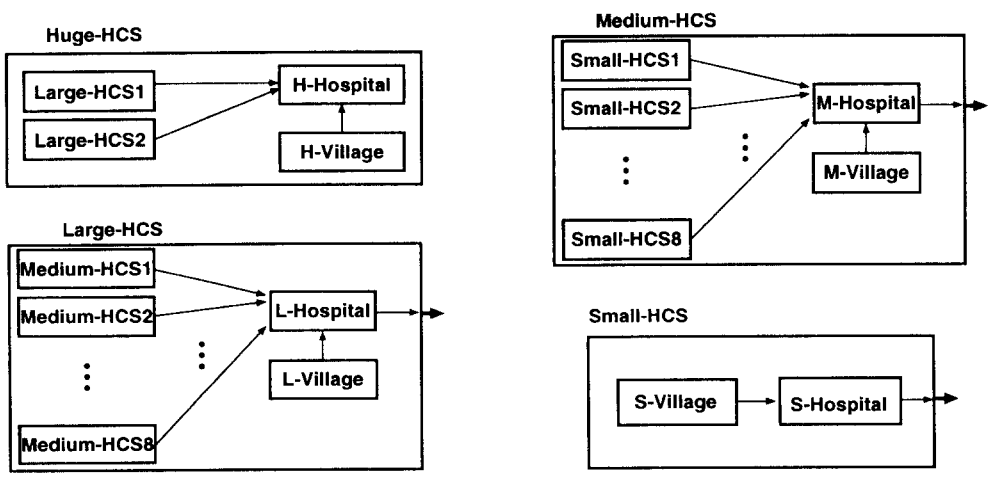
이 장에서는 실험을 통해 제안된 계층적 파티션 알고리즘 HIPART의 성능을 평가해 본다. 성능평가에 사용된 모델은 콜롬비아 보건 관리 시스템(CHCS)이라고 부르는 큐잉 모델이다. 이 모델은 여러 분산 시뮬레이션 연구들에서 사용되어 왔다(Baezner *et al.* 1988). CHCS 모델은 마을과 보건소를 기본단위로 하는 여러 계층의 보건 관리 시스템이라고 할 수 있다. 각 마을에는 한 개의 보건소가 있다. 마을 사람들이 병에 걸리면 그 지역의 보건소로 가서 진찰을 받고 가능하면 치료도 받게 된다. 만일 그 보건소에서 치료할 수 없는 병이라면 좀더 큰 병원으로 가게 되고 다시 진찰과 치료를 받게 된다. 병원에 환자가 도착하면, 환자는 큐에 넣어지고, 순서대로 진찰을 받는다.

마지막으로 가장 높은 레벨의 병원은 모든 병을 다 치료할 수 있다고 가정한다.

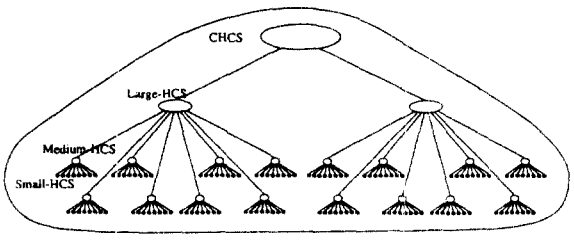
〈그림 10〉은 계층적으로 구성된 CHCS 모델을 보여준다. 가장 높은 레벨의 연결 모델은 SLARGE-HC이다. SLARGE-HC는 2개의 LARGE-HC로 구성되고 LARGE-HC는 MIDDLE-HC로, MIDDLE-HC는 8개의 SMALL-HC로 구성된다. 각각의 HC 모델은 HOSPITAL과 VILLAGE의 두 원소 모델을 가지고 있다. 실험틀(experimental frame)로서 각 VILLAGE는 환자들을 발생시키는 소스가 된다. 각 HOSPITAL은 큐와 정해진 수의 의사들을 갖는 서버가 된다. 각 1/2/3/4계층의 HOSPITAL은 16/4/4/1명의 의사들을 각각 가지고 있다고 가정한다.

〈그림 11〉은 파티션의 수가 각각 2,4,8,16, 그리고 32일 때, 제안된 알고리즘에 의해 얻어진 파티션 결과이다.

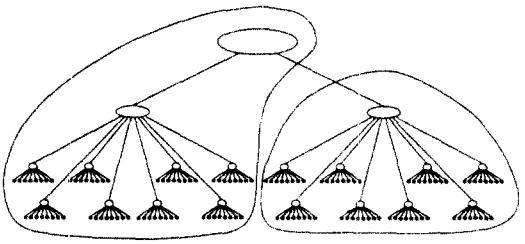
파티션 결과에 대해 평가를 하기 위해 D-DEVSim++ 상에서 시뮬레이션을 하였다. 시뮬레이션은 KAICUBE860 시스템에서 이루어 졌다. KAICUBE860은 KAIST에서 개발된 5 차원의 초격자형(hypercube) 병렬 컴퓨터이다. DOHS 방법의 성능 평가를 위해서, 노드 컴퓨터의 수를 1에서 32개 까지 변화시켜가면서 시뮬레이션 시간을 측정해 보았다. 전체 시뮬레이션 시간에 대한 speedup은 순차적인 구현에 의한 시뮬레이션 시간과 병렬적 구현에 의한 시뮬레이션 시간과의 비율이 된다. 이 때 주의해야 할 점은 순차적 구현에서는 동기화와 관련된 연산들 (전체 시뮬레이션 시간 계산, fossil collection, 상태 저장 등)은 포함되지 않아야 한다는 것이다.



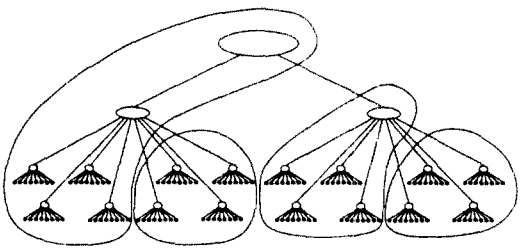
〈그림 10〉 콜롬비아 보건 관리 시스템의 계층적 모델 구성



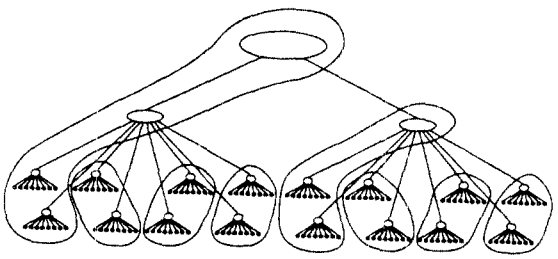
(a) Number of computer nodes = 1



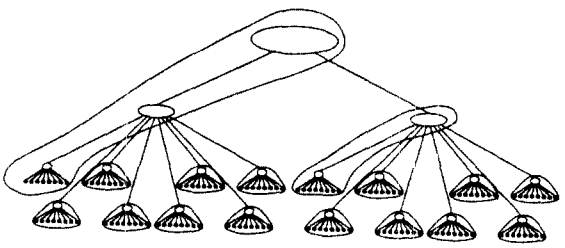
(b) Number of computer nodes = 2



(c) Number of computer nodes = 4

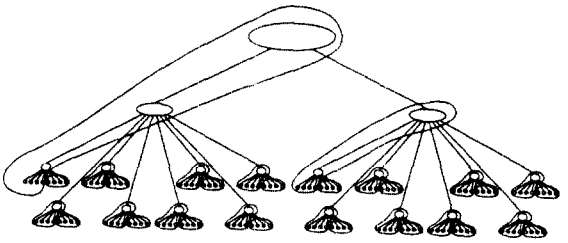


(d) Number of computer nodes = 8



(e) Number of computer nodes = 16

〈그림 11〉 CHCS 모델에 대한 파티션 결과

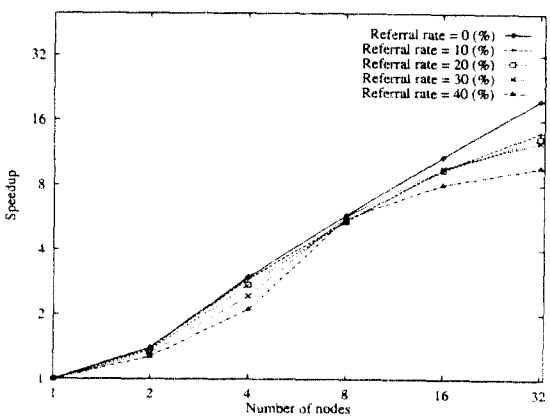


(f) Number of computer nodes = 32

〈그림 11〉 CHCS 모델에 대한 파티션 결과(계속)

〈그림 12〉는 실험 결과로서, referral rate가 변할 때의 speedup 변화를 측정한 것이다. Referral rate는 각 병원의 전체 환자 중에서 치료받지 못하고 상급 병원으로 후송되는 환자의 비율을 의미한다. Referral rate가 0(%)라면 모든 환자는 소속 마을의 지역 병원에서 전부 치료받음을 의미한다. 반대로 100(%) 라면 모든 환자가 지역 병원에서 상급 병원으로 후송됨을 의미한다. 따라서, referral rate가 높을 수록 모델의 병렬성이 떨어지게 된다. 이 실험에서는 0부터 40(%)까지 rate를 변화시켰다.

이 결과에서 보듯이 제안된 파티션 알고리즘이 만들어낸 파티션 결과가 실제 시뮬레이션에서 좋은 성능을 보여주었다. 즉 모델의 병렬성을 구조 정보를 이용하여 잘 예측할 수 있었다.



〈그림 12〉 Speedup v.s. referral rate

(# nodes = 32)

6. 결 론

본 논문에서는 DEVS 모델의 분산 시뮬레이션을 위한 새로운 파티션 알고리즘을 제안하였다. 제안된 알고리즘은 독립적인 시뮬레이션 태스크의 병렬 실행을 최대로 하기 위해서 계층적 모델 디자인 방법론으로부터 얻을 수 있는 모델 구조 정보를 이용하였다. 본 논문에서는 또한 벤치 마크 시뮬레이션 실험을 통해 제시된 알고리즘이 좋은 성능을 냄을 확인하였다.

참고문헌

[1] Rajive Bagrodia, Yu an Chen, Vikas Jha, and Nicki Sonpar. Parallel gate-level circuit simulation on shared memory architectures. In *Proceedings of the 1995 Workshop on Parallel and Distributed Simulation*, pages 170--174, Lake Placid, New York, 1995.

[2] Azzedine Boukerche and Carl Tropper. A static partitioning and mapping algorithm for conservative parallel simulations. In *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*, pages 164-172, Edinburgh, Scotland, U.K., 1994.

[3] Praveen Chawla. *Assignment Strategies for Parallel Discrete Event Simulation of Digital Systems*. PhD thesis, University of Cincinnati, 1994.

[4] Jason Cong, Zheng Li, and Rajive Bagrodia. Acyclic multi way partitioning of boolean networks. In *31th ACM/IEEE Design Automation Conference*, pages 670-675, 1994.

[5] C. M. Fiduccia and R.~M. Mattheyses. A linear-time heuristic for improving network partitioning. In *Proc. 19th Design Automation Conf.*, pages 175--182, 1982.

[6] Richard M. Fujimoto. Optimistic approaches to parallel discrete event simulation. *Trans. of The Society for Computer Simulation*, 7(2):153--191, October 1990.

[7] David W. Glazer and Carl Tropper. On process migration and load balancing in time warp. *IEEE Transactins on Parallel and Distributed Systems*, 4 (3):318--327, 1993.

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

[9] Kevin~L. Kapp, Thomas C. Hartrum, and Tom S. Wailes. An improved cost function for static partitioning of parallel circuit simulations using a conservative synchronization protocol. In *Proceedings of the 1995 Workshop on Parallel and Distributed Simulation*, pages 78--85, Lake Placid, New York, July 1995.

[10] Ki Hyung Kim. *Distributed Simulation Methodology Based on System Theoretic Formalism: An Asynchronous Approach*. PhD thesis, Korea Advanced Institute of Science and Technology, 1996.

[11] Ki Hyung Kim, Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park. Distributed optimistic simulation of hierarchical DEVS models. In *Proceedings of the 1995 Summer Simulation Conference*, pages 32--37, Ottawa, Canada, 1995.

[12] Pavlos Konas and Pen Chung Yew. Partitioning for synchronous parallel simulation. In *Proceedings of the 1995 Workshop on Parallel and Distributed Simulation*, pages 181--184, Lake Placid, New York, July 1995.

[13] Jayadev Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 18(1):39--65, March 1986.

[14] S. P. Smith, M. R. Mercer, and B. Underwood. An analysis of several approaches to circuit partitioning for parallel logic simulation. In *International Conference on Computer Design*, pages 664-667, 1987.

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

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

[17] 김기형, 김탁곤, 박규호, DEVS에 기반한 분산시뮬레이션 환경 D-DEVSim++의 설계 및 구현, 한국시뮬레이션학회 논문지 5권 2호: 41--58, 12월, 1996.

[18] Ki Hyung Kim, Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park. Kistributed Simulation of Hierachical DEVS Models: Hierarchical Scheduling locally and Time Warp Globally, Trans. of The Society for Computer Simultion 13(3), 1997.

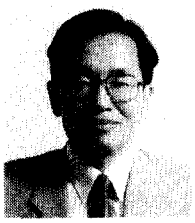
● 저자소개 ●

김기형

1990. 2. 한양대학교 공과대학 전자통신공학과 졸업(공학사)
1992. 2. 한국과학기술원 전기 및 전자공학과 졸업(공학석사)
1996. 8. 한국과학기술원 전기 및 전자공학과 졸업(공학박사)
1996. 9.～1997. 2. 한국과학기술원 위촉연구원
1997. 3.～현재 영남대학교 컴퓨터공학과 전임 강사
관심분야 분산시뮬레이션, 시스템 모델링 및 성능 평가, 멀티미디어 운영체제

김탁곤

1975. 2. 부산대학교 전자공학과 졸업(공학사)
1980. 2. 경북대학교 전자공학과 졸업(공학석사)
1988. 5. 아리조나대학교 전기 및 전산 공학과 졸업(공학박사)
1975. 2.～1977. 6. 육군 통신장교
1980. 9.～1983. 1. 부산수산대학교 전자통신공학과 전임강사
1987. 8.～1989. 7. 아리조나 환경연구소 연구 엔지니어
1989. 8.～1991. 8. 캔사스대학교 전기 및 전산공학과 조교수
1993. 9.～1993. 8. 한국과학기술원 전기 및 전자공학과 조교수
1993. 9.～현재 한국과학기술원 전기 및 전자공학과 부교수
국제학술지 편집 부위원장
Associate Editor : Simulation (SCS)
Associate Editor : Transactions of SCS (SCS)
Associate Editor : Simulation Digest (IEEE/ACM)
Associate Editor : Int J. in Intelligent Control and Systems
관심분야 모델링 이론, 병렬/지능형 시뮬레이션 환경, 컴퓨터/통신 시스템 해석



박규호

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
관심분야 병렬처리, 컴퓨터 구조, 컴퓨터 그래픽스