시뮬레이션 도구 SMPLE의 개발 및 멀티프로세서 시스템 성능 분석에의 활용

Development of Simulation Tool SMPLE and Its Application to Performance Analysis of Multiprocessor Systems

조 성 만*, 조 경 산*
Sung-Man Jo*, Kyung-San Cho*

Abstract

This paper presents the development of event-driven system level simulation tool SMPLE(smpl Extended, an extention of smpl) and its application to the performance analysis of multiprocessor computer systems. Because of its data structure, it is very difficult to change, expand or add new functions to simulation language smpl implemented by MacDougall. In SMPLE, we change data structure with structure and pointer, add new functions, and enable dynamic memory management.

Using new data structure, facilities, and functions added in SMPLE, we simulate job processing of a shared bus multiprocessor system with autonomous hierarchical I/O subsystem. We set system performance criterion and analyze the performance contribution of subsystems and units. The impact of disk I/O on system performance is evaluated under various conditions of number of processors, processing power, memory access time and disk seek time.

Ⅰ. 서론

기존 컴퓨터의 확장 및 변경이나 새로운 컴퓨터의 설계에 영향을 주는 절대적 역활자인 성능의 평가는 대부분 컴퓨터 시스템 개발의 마지막 단계에나 고려되었으며, 특히 중형컴퓨터의 설계 및 구현에 있어서는 대형컴퓨터의 경우와 비교하여 성능분석은 일반적으로 무시되어 왔다[8]. 이러한 컴퓨터 시스템의 성능 평가 및 분석은 단순한

* 단국대학교 전산통계학과

시스템 운영의 모니터링만으로는 부족하고, 시스템내의 모든 연계된 분야의 연구가 병행되어야 한다. 이러한 목적을 위해 모델링 및 시뮬레이션을 통한 시스템 평가와 분석은 가장 적절하고 필요한 방법이다.

〈그림 1-1〉은 컴퓨터 시스템 성능평가 모델링을 위하여 사용되어 온 기술들을 보여주고 있다[6]. 분석적 큐잉모델(analytic queueing model)들은 컴퓨터 시스템 성능 모델링을 위한 비용 대비 가장 효과적인 기술을 제공한다. 그러나 대부분의 분석적 큐잉모델들은 모델링에 사용되는 가정들이 실제 시스템과는 상충되는 면이 많아 곧바로 새로운 컴퓨터 시스템의 설계에 적용될 수 없다. 벤치마크(benchmark)는 실제 시스템의 성능측정에 있어서 다른 방법들에 비하여 가장 정확한 결과를 보여준다. 그러나 벤치마크에 의한 프로세서의 성능지표인 MIPS의 수치나 특정한 벤치마크 프로그램을 수행한 결과수치는 시스템 성능을 제시하기에는 객관성이 부족하거나, 혹은 특정 컴퓨터의 구조 및 운영체제에 편파적일 수 있다. 또한 벤치마크에 의한 성능측정방식은 이의 수행을 위한 실제의 시스템이 존재 하여야만 한다는 단점이 있다. 반면에 시뮬레이션(simulation)은 대상시스템을 소프트웨어를 사용하여 가상적으로 흉내내는 것으로, 큐잉모델을 수행하는 것보다 훨씬 상세하게 실제 시스템을 모델링 할 수 있다.

컴퓨터 시스템의 개발 및 분석에 많이 활용되는 시뮬레이션 모델을 구축하는데는 SIMSCRIT, GPSS, SIMAN, SLAM Ⅱ 등과 같은 시뮬레이션 전용 언어를 사용하는 방법과 FORTRAN, PASCAL, C 등의 일반 범용언어를 사용하는 방법이 있을 수 있다. 시뮬레이션 전용언어는 시뮬레이션 모델링을 위한 자연스러운 구조와 기능을 제공하므로 프로그래밍하는 시간을 짧게 단축할 수 있고, 수정하기 쉬우며, 오류를 범할 수 있는 가능성이 적다는 장점이 있다. 그러나 시뮬레이션 전용언어는 일반 범용언어에 비하여 널리 보급되어 있지 않기 때문에 대부분의 사람들이 새로운 언어를 습득하여야 한다는 부담과 몇몇 시뮬레이션 언어는 수행을 위하여 특별한 컴퓨터를 요구하기도 한다. 또한 시뮬레이션 전용언어는 다양한 시스템을 모델화 하기위해 설계된 반면 범용언어는 특정 응용에 맞추어 설계될 수 있기 때문에 보다많은 융통성과 수행 속도면에서의 이점이 있을 수 있다[7].

현재 컴퓨터 설계 및 활용에 가장 널리 쓰이는 C 언어를 시뮬레이션 툴 구현에 활용함으로써, C언어 사용자의 편의를 도모할 수 있다. 일반 범용언어를 사용하여 컴퓨터시스템의 시뮬레이션 모델을 구현하기 위해 시뮬레이션에 필요한 기본구조를 제공하는 smpl이 제안되고 C언어를 이용하여 구현되었다[9].

하지만, smpl에서 사용되는 자료구조는 변경이나 확장 또는 필요한 기능의 추가에 많은 어려움이 따른다. 본 논문에서는 smpl의 분석 및 기능의 추가를 용이하게 하기 위하여 이를 변경, 확장한 SMPLE(smpl extended)의 개발 내용 및 컴퓨터 시스템 성능분석에의 활용예를 소개한다.

본 논문의 구조는 2장에서는 시뮬레이션 프로그램의 기본구성을 설명하고, 3장에서는 smpl의 취약점 및 본 연구에서 제시하는 SMPLE의 구현을 제시한다. 4장에서는 SMPLE에서 새로이 추가된 자료구조와 함수 및 설비 제어방식등을 활용하여 공유메모리 멀티프로세서 시스템의 작업수행 과정을 모델하고, 구성성분들의 성능기여도와 입출력 프로세서, 입출력 버스, 제어기, 디스크 등으로 구성된 입출력 부시스템의 성능 영향을 시뮬레이션에 의해 분석하고 5장에서 결론을 맺는다.

낮음 — 개발의 난이도 및 비용 — 높음

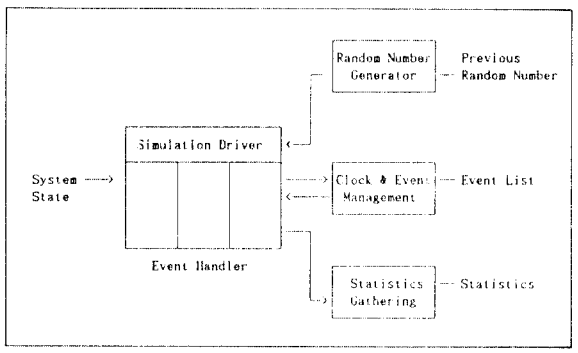
RULES OF THUMB	LINEAR PROJECTION (UTILIZATION and LOAD)	ANALYTICAL QUEUEING THEORY MODELS	SIMULATION	BENCHMARKING

〈그림 1-1〉 컴퓨터 시스템 모델링 방법

Ⅱ. 시뮬레이션 프로그램

2.1 이산사건(event driven) 시뮬레이션 프로그램 구성요소

컴퓨터 시스템의 동작 특성상 시스템에 대한 모델구축에는 이산사건 시뮬레이션 모델이 많이 사용된다. 〈그림 2-1〉은 이산사건 시뮬레이션 프로그램의 기본적인 구성요소들을 보여주고 있다. 시뮬레이션 프로그램은 〈그림 2-1〉과 같이 기본적으로 시뮬레이션 드라이버가 관찰하고자 하는 시스템의 상태를 입력으로 받고, 시뮬레이션 시계 및 사건(event)을 관리하는 부분, 난수 및 확률변수를 발생시키는 부분, 측정되는 데이타를 통계처리하는 부분 등으로 이루어진다.

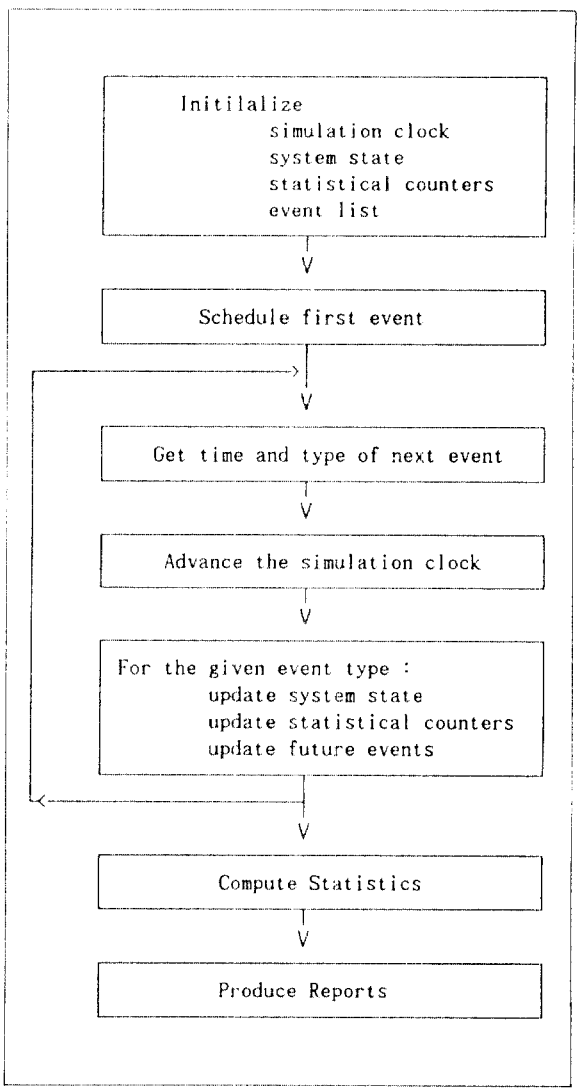


〈그림 2-1〉 시뮬레이션 프로그램 구성요소

2.2 제어의 흐름

일반적인 이산사건 시뮬레이션 프로그램은 〈그림 2-2〉와 같은 순서로 동작되게 된다. 시뮬레이션이 진행되는 과정을 간략히 살펴보면, 먼저 시뮬레이션 프로그램에 사용될 각종 변수들을 초기화 한다. 그 다음 최초의 사건을 사건 리스트(event list)에 예약한다. 다음의 시뮬레이션 진행 과정은 이 최초의 사건에 의해 연속적으로 발생되게 된다. 그 다음 동작은 사건 리스트에 예약되어 있는 최초의 사건을 꺼내어 이 사건의 종류에 의해 해당 사건을 발생시키게 된다. 시뮬레이션 시간은 이때 증가되며, 해당 사건이 발생됨에 의해 시뮬레이션하는 대상 시스템의 상태가 변화하게 되고 각종 측정치들의 값이 갱신되게 된다. 하나의 사건이 발생되고 그 사건에 대한 수행이 완료되었으면 시뮬레이션을 계속 진행시키기 위해 다음에 발생될 사건을 미리 사건 리스트에 예약하여 주어야 한다.

이와같은 동작이 계속 반복되어 어떤 종료조건을 만족하게 되면 최종적으로 이제까지 측정한 값들을 종합적으로 계산하여 최종 보고서로 작성하여 출력하고 시뮬레이션 프로그램을 종료하게 된다.



〈그림 2-2〉 시뮬레이션 제어의 흐름

```
#define     nl      256         /* element pool length   */

static int
        I1[nl],
        I2[nl],                 /*  facility descriptor.  */
        I3[nl];                 /*      queue. &          */
static double                   /*      event list        */
        I4[nl],                 /*      element pool      */
        I5[nl];
```

〈리스트 3-1〉 smpl에서 자료구조의 선언

```
#define     maxfct    100     /*  사용할 수 있는 최대 facility의 개수          */

struct tkn {
        long tk_id;           /*  Token 식별번호. 현재는 사용되지 않는다.      */
        int  tk_no;           /*  Token 번호                                   */
        int  ev_no;           /*  Event 번호                                   */
        int  pri;             /*  Token의 우선순위                             */
        int  ev_pri;
        real arrive;          /*  token 이 facility에 도착한 시간              */
        real start;           /*  이 token이 queue나 server에 들어간 시간      */
        real tm;              /*  Queue에서는 잔여 service시간,
                                  event list에서는 CLOCK시간                   */
        struct tkn *link;
};

struct fct {
        int  ns;              /*  Facility가 갖고 있는 server의 개수           */
        int nbs;              /*  현재 busy인 server의 개수                    */
        int inq;              /*  queue에 들어가 있는 token의 개수             */
        long Qexit_count;     /*  queue에서 빠져나간 token의 총개수            */
        long release_count;   /*  service 받고 빠져나간 token의 총개수         */
        long preempt_count;   /*  service 받다가 preempt된 token의 총개수      */
        real queuetime;       /*  queue에서 대기중이었던 시간의 총 합          */
        real busytime;        /*  busy이었던 시간의 총 합                      */
        real intime;          /*  token이 도착했다가 나갈때까지의 총 합        */
        char name[50];        /*  Facility의 이름                              */
        struct tkn *front;
        struct tkn *rear;
        struct tkn *Qfront;
        struct tkn *Qrear;
};

struct tkn *ev_list;          /*  event list head          */
struct tkn *tk_pool;          /*  token pool head          */
struct fct fhd[maxfct];       /*  facility 배열            */
```

〈리스트 3-2〉 변경된 SMPLE 데이타구조

Ⅲ. smpl의 분석 및 SMPLE의 개발

smpl은 C언어의 기능적인 확장으로서 라이브러리 함수들의 집합 형태를 갖는 시뮬레이션 부시스템으로, C언어 자체와 더불어 이산사건중심 시뮬레이션 언어를 구성하며, smpl 시뮬레이터는 C언어 프로그램으로 구현된다.

이번장에서는 smpl의 특성을 분석하고, SMPLE에서의 개선안을 설명한다.

3.1 자료구조의 변경

기존의 smpl에서 사용되는 프로그램의 자료구조는 〈리스트 3-1〉과 같이 배열과 인덱스 구조를 사용하도록 선언되어 있다. 이러한 구조는 완성된 프로그램내에서는 효율성 면에서 좋으나, 변경이나 확장하기 위해 이 프로그램을 분석한다거나 필요한 기능을 추가하기 위해서는 매우 많은 어려움이 따르게 된다. 따라서 본 논문에서는 smpl의 자료구조를 〈리스트 3-2〉에서 보인 C언어의 structure와 pointer를 기반으로 하게 변경시키고 이에 따르는 제반 변경 사항을 수정 보완하여, 프로그램의 분석을 용이하게 하며 기능의 변경 및 추가를 수월하게 하였다.

3.2 동시적인 사건 발생 순서의 조절

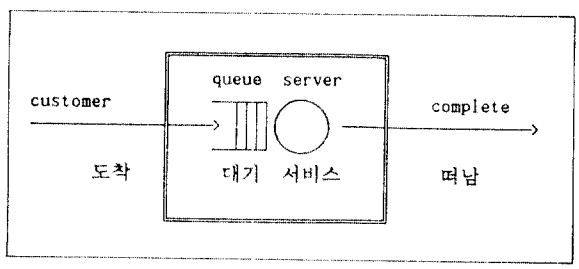
컴퓨터 시스템에 대한 시뮬레이션 수행시에 동일한 시뮬레이션 시각에 발생하는 여러 사건들의 순서를 조절하는 것은 매우 중요한데, 그 순서가 잘못 뒤바뀌게 되면 시뮬레이션하는 대상 시스템이 잘못 평가될 수도 있기 때문이다.

smpl에서는 새로운 시뮬레이션 사건을 사건 리스트에 예약하기 위한 함수로서 schedule()을 사용한다. 이때 함수 schedule()에서 넘어온 시간에 대한 인수를 이용하여 사건들이 정렬되어 있다가 cause()함수에 의하여 꺼내어져 시뮬레이션 사건이 발생되게 된다. smpl에서 schedule()함수를 사용하여 어떠한 사건을 예약할 경우에는 schedule(ev, tm, tkn);과 같이 인수로서 ev, tm, tkn만을 전달하고 사건 리스트에서는 인수 tm과 시뮬레이션 시계 CLOCK을 참조하여 CLOCK+tm의 크기에 의하여 다음에 발생될 사건들의 순서를 유지하게 된다. 이때 사건 리스트에서는 사건이 발생될 시각순으로 정렬하게 되고, 사건이 발생할 시각이 작은것이 사건 리스트의 앞쪽으로 오게 된다. 만일 2개 이상의 사건이 발생할 시각이 똑같다면 새로 들어오는 사건이 기존의 사건 뒤에 추가되어 나중에 발생되게 된다.

따라서 기존의 방법에서는 동시적인 사건의 발생은 사건의 예약 순서에 영향을 받게 되어 사건 발생순서의 조절에 매우 많은 어려움이 따르게 된다. 이러한 문제점을 해결하기 위해서 사건을 예약하기 위한 함수 schedule()에서 넘겨주는 인수에 그 사건의 우선순위를 나타내는 pri를 추가하여 함수 schedule()을 확장 하도록 한다. 이렇게 인수 pri를 추가함으로 해서 사건 리스트에서는 사건이 일어날 시간순으로 먼저 정렬이 되고, 만일 동일한 시각이라면 우선순위가 높은 것을 앞으로 놓아, 사건발생 시각이 빠르고 우선순위가 높은 사건이 먼저 발생되게 하여 사건 발생 순서를 좀더 용이하게 조절할 수 있게 된다. 이때 만일 사건 발생 시각이 똑같고, 우선순위도 동일 하다면 먼저 사건 리스트에 들어온 사건이 먼저 발생되게 된다.

3.3 설비의 이용

〈그림 3-1〉은 "단일서버 큐잉 시스템"을 나타내고 있다.



〈그림 3-1〉 단일서버 큐잉 시스템

smpl에서 설비의 선언은 함수 facility()를 호출함으로써 수행되어 진다.

smpl을 이용하여 어떠한 설비를 선언한 후, 하나의 토큰이 설비에 도착하였다가 떠나는 과정을 살펴보면 대략적으로 "도착", "대기", "서비스", "떠남"의 4가지 동작으

로 구분되어 진다. 토큰이 서버에 도착하는 동작은 request ( ), 토큰이 서비스를 다 받고 설비를 떠나는 동작은 release ( )의 2개 함수로 표현된다.

smpl에서 큐에서 대기하는 토큰은 우선순위에 의하여 정렬되게 된다. 그러나 경우에 따라서는 FIFO나 LIFO 등의 방법으로 동작시켜야 하는 필요도 있을 것이므로 SMPLE에서는 함수 set_queue( )를 추가하여 임의의 방법으로 큐를 운영할 수 있도록 확장하였다. 아래에 그 사용예를 보인다.

set_queue(f, FIFO);

위의 예는 디스크립터 f로 나타내어지는 설비의 큐를 FIFO방식으로 운영하겠다는 의미이다. 이때 넘겨주는 인수로는 PRI, FIFO, LIFO 등의 3가지가 가능하다.

3.4 데이타 측정

SMPLE는 기본적으로 〈그림 3-1〉과 같은 "큐잉 시스템"을 시뮬레이션 하기위한 함수들을 제공한다. 시뮬레이션이 진행되는 과정은 어떠한 설비를 선언하고 토큰으로 대표되는 작업이 설비를 통과해 나가면서 걸리는 시간을 측정하게 된다.

하나의 설비에 대하여 대기시간의 총합을 측정하기 위한 2가지 방법이 가능하다.

첫번째 방법은 토큰이 각 설비에 도착할 때마다 현재까지의 측정치를 계산하는 방법이다. 이 방법은 각 설비의 큐에 총 대기시간을 갖는 sum_queue_time, 큐에서 현재 대기중인 토큰의 개수 n, 마지막으로 큐에 토큰이 도착하였던 시각 last_change_time 등과 같은 변수를 두고, 새로운 토큰이 큐에 도착할 때마다 sum_queue_time+=n*(CLOCK - last_change_time);와 같이 총 대기 시간을 계산하고 last_change_time=CLOCK;와 같이하는 방법이다.

두번째로는 앞에서와는 달리 토큰이 각 설비를 떠날 때 측정치를 계산하는 방법이다. 이방법은 각 토큰이 설비에 도착할 때마다 각 토큰에 도착시각을 기억시켜 놓았다가 설비를 떠날 때 sum_queue_time+=(CLOCK - tk→intime);와같이 누적시간을 계산하는 것이다.

원래의 smpl에서는 첫번째의 방법을 사용하였고, 변경된 SMPLE에서는 두번째의 방법이 사용되었다. 원래의 smpl에서는 토큰 이동시 그 토큰의 번호만이 이동된다. 그러나 SMPLE에서는 그 토큰에 대한 실제 자료구조에 대한 포인터가 이동된다. 따라서 메모리가 추가적으로 요구되는 단점이 있으나 부가적인 정보표현에 있어서 좀더 자연스러운 구조를 제공하여 주고, 곱하기 연산이 덧셈연산으로 대체되므로 연산 시간이 짧아지는 장점이 있다.

3.5 기타 변경된 내용

기타 현재까지 추가된 내용은 아래와 같다.

① int who_server(int f) : 현재 설비 f에서 서비스를 받고 있는 토큰번호를 넘겨준다.

② int who_queue(int f) : 현재 설비 f의 대기 큐의 제일 앞에서 대기하고 있는 토큰번호를 넘겨준다.

③ int pop_Q(int f) : 현재 설비 f의 대기 큐의 제일 앞에서 대기하고 있는 토큰을 꺼내어 그 토큰 번호를 넘겨준다.

④ push_Q(int f, int tkn) : 설비 f의 대기 큐의 제일 뒤에 토큰번호 tkn을 삽입한다.

⑤ int pop_server(int f) : 현재 설비 f의 서버 제일 앞에서 서비스 받고 있는 토큰을 꺼내어 그 토큰 번호를 넘겨준다.

⑥ push_server(int f, int tkn) : 설비 f의 서버에 토큰번호 tkn을 삽입하여 서비스를 받게한다.

⑦ wc( ) : 현재 사건 리스트에 기록되어 있는 사건의 목록을 사건번호, 토큰번호의 순으로 보여준다. 필요시 약간의 내용을 추가하여 발생시간 및 토큰의 우선순위를 보여줄 수 있다.

⑧ wf(int f) : 설비 f의 현재 서비스 받고 있는 토큰과 대기하고 있는 토큰을 보여준다.

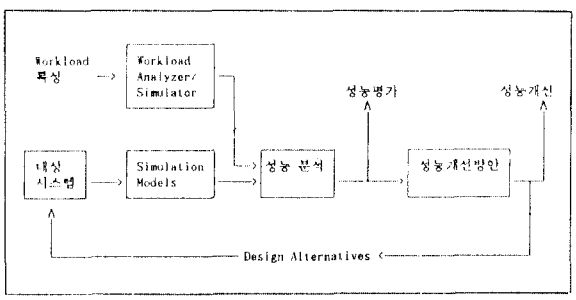
⑨ printfct(int f) : 설비 f의 현재 상태정보를 보여준다.

⑩ Time( ), Random( ) : 기타 변경 내용으로 TURBO-C 내장함수와 이름이 동일하여 충돌을 일으키는 함수 time( )와 random( )의 첫자를 대문자로 바꾸어 충돌을 피했다.

Ⅳ. 멀티프로세서 컴퓨터 시스템에 대한 시뮬레이션 구현과 시스템 분석 예

시스템의 성능 분석은 시스템으로서의 총체적 성능과 시스템을 구성하는 각 부시스템 또는 모듈의 개별적 성능의 이원적 분석이 필요하다. 총체적 성능과 각 개별적 성

능과의 관계를 분석하여, 문제점을 분석하고, 이에대한 개선책을 각 부시스템 또는 그들의 연결을 통해 제시하여야 한다. 이러한 시스템 성능분석은 〈그림 4-1〉과 같이 수행될 수 있다[1]. 선정된 workload에 대한 대상 시스템에서의 성능적 자료가 시뮬레이션 모델을 통한 예측을 통해 얻어지면, 이 자료를 이용하여 대상 시스템의 구조 및 운영 관리 체제의 연계를 통한 성능적 문제점을 분석한 후, 시뮬레이션 모델 및 시스템 분석의 일반적 결과를 이용하여, 자원의 부족 및 높은 경쟁율에 의한 시스템 병목현상의 점진적 제거를 위해 구조의 수정을 통한 성능 개선 방법을 추구한다. 본 장에서는 이러한 측면을 고려하여, 앞



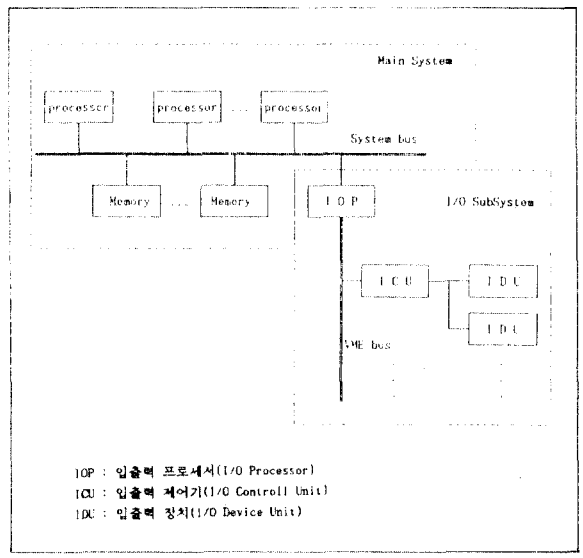
〈그림 4-1〉 시스템 성능 평가 연구의 구성도

장에서 제시한 SMPLE을 활용한 중형컴퓨터의 성능 분석 및 개발을 위한 시뮬레이션을 소개한다.

4.1 시뮬레이션 대상 시스템의 설정

본 논문에서는 버스구조의 멀티프로세서 시스템의 성능 모델을 제시한다.

대상 시스템은 단일 버스 공유메모리 방식의 멀티프로세서 시스템으로 하나의 버스를 여러개의 프로세서 모듈과 메모리 모듈이 공유하고 있으며 입출력을 전담하는 입출력 프로세서가 역시 이를 공유하고 있다. 시스템은 크게 주 시스템과 입출력 프로세서 부시스템으로 나뉜다. 주 시스템에는 프로세서와 그 주변회로로 구성된 프로세서 모듈과 공유메모리 모듈을 포함하며 입출력 부시스템은 입출력 처리기(IOP)와 입출력 장치(IDU) 그리고 이들을 구동하는 입출력 제어기(ICU)를 포함한다. 그 전체적인 구성은 〈그림 4-2〉와 같다[2].



〈그림 4-2〉 멀티 프로세서 시스템의 구조

본 연구에서는 주 시스템 및 입출력 부시스템이 전체 시스템에 미치는 영향을 분석하기 위해 공유메모리 멀티프로세서 시스템의 작업수행 과정을 모델하고, 구성성분들의 성능기여도와 입출력 프로세서, 입출력 버스, 제어기, 디스크 등으로 구성된 입출력 부시스템의 성능 영향을 시뮬레이션에 의해 분석한다.

시뮬레이션 수행결과 각 장치들의 평균 활용도, 평균 데이타 처리 주기, 평균 대기시간등에 대한 자료와 하나의 데이타가 처리되는데 걸리는 각 시간성분들을 측정할 수 있다.

4.2 시스템의 동작과정

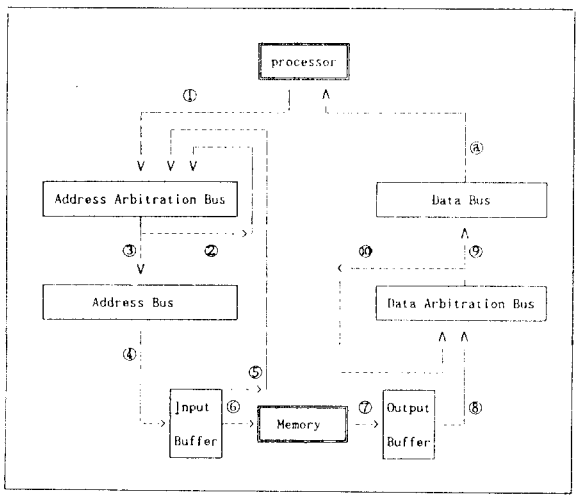
프로세서와 IOP 및 메모리는 하나의 시스템 버스로 연결되어 있고, 이를 통하여 서로간의 데이타를 전송한다. 프로세서는 메인 메모리로부터 데이타를 읽어오라는 메모리 요구신호와 IOP에게 디스크에서 데이타를 읽으라는 명령을 발생시킨다.

이러한 작업수행과정을 시스템 버스를 통한 메모리접근과 입출력 처리의 2부분으로 나누어 설명한다.

메모리 접근에 대한 시뮬레이션은 프로세서(CPU), 시스템버스, 메모리, 입출력 처리기(IOP) 사이 에서의 데이

타 흐름을 분석한다. 프로세서 및 입출력 처리기가 메모리 읽기 요구를 발생시키면, 이 신호는 시스템버스를 통하여 메모리에까지 도달하여 메모리에서 데이타를 읽은 다음에, 이 데이타를 다시 시스템버스를 통하여 이를 요구한 프로세서에게 전달하는 과정을 시뮬레이트하게 된다. 이때 버스는 packet switching 및 독립적 부버스 구조의 특성을 가지며, 메모리 모듈에는 입력 및 출력의 대기장소(Input/Output Buffer)를 가정한다. CPU로부터 입출력 명령이 발생되어 시스템버스를 통과하여 데이타가 전달되는 과정을 〈그림 4-3〉에 보인다.

입출력 부시스템에서는 IOP, ICU, IDU 사이에서의 데이타 흐름에 관하여 분석한다. CPU에서 발생한 입출력 명령이 IOP에게 전달되면 IOP는 자신의 DBR(Data Buffer RAM; 입출력 프로세서의 디스크 버퍼)에서 먼저 데이타를 찾고, 없으면 이 신호를 VME 버스를 통하여 ICU에게로 전달하고, 이 명령을 받은 ICU는 자신의 디스크 캐쉬에서 데이타를 찾는다. 이때에도 요구한 데이타가 없으면 디스크에서 데이타를 읽어 자신의 디스크 캐쉬에 먼저 가져다가 놓고 VME 버스를 통하여 IOP의 DBR에 갖다가 놓는다. 이렇게 하여 IOP의 DBR에 데이타가 놓이게 되면 이 데이타를 메모리에 갖다놓은 다음 하나의 입출력 명령이 끝난다. CPU로부터 입출력 명령이 발생되어 IOP가 이를 처리하여 다시 CPU에게로 전송할 준비가 되기까지의 과정을 〈그림 4-4〉에 도시하였다.



〈그림 4-3〉 시스템버스에서 하나의 메모리 요구신호 처리과정

4.3 시뮬레이션 프로그램

〈그림 4-3〉과 〈그림 4-4〉와 같이 동작되는 대상시스템에 대한 시뮬레이션 모델을 SMPLE를 사용하여 구축할 때 사용된 함수들간의 연결도를 〈그림 4-5〉에 보인다. 그림에서 네모상자안의 명칭들은 각각의 사건들을 구분하는 사건 명칭들이고 이를 그대로 함수명칭으로 하여 프로그램하였다. 입출력 처리기에 대한 입출력 요구는 인터럽트를 통해 이루어지며 인터럽트 버스 및 입출력 명령 메세지를 처리하기 위한 버스참조는 그림에서 표시되어 있지 않다.

〈리스트 4-1〉은 실제로 작성된 프로그램의 일부분이다. 〈리스트 4-1〉에서 밑줄쳐진 함수들이 SMPLE에서 제공되는 함수이다. 시뮬레이션 프로그램을 수행시킬 때에는

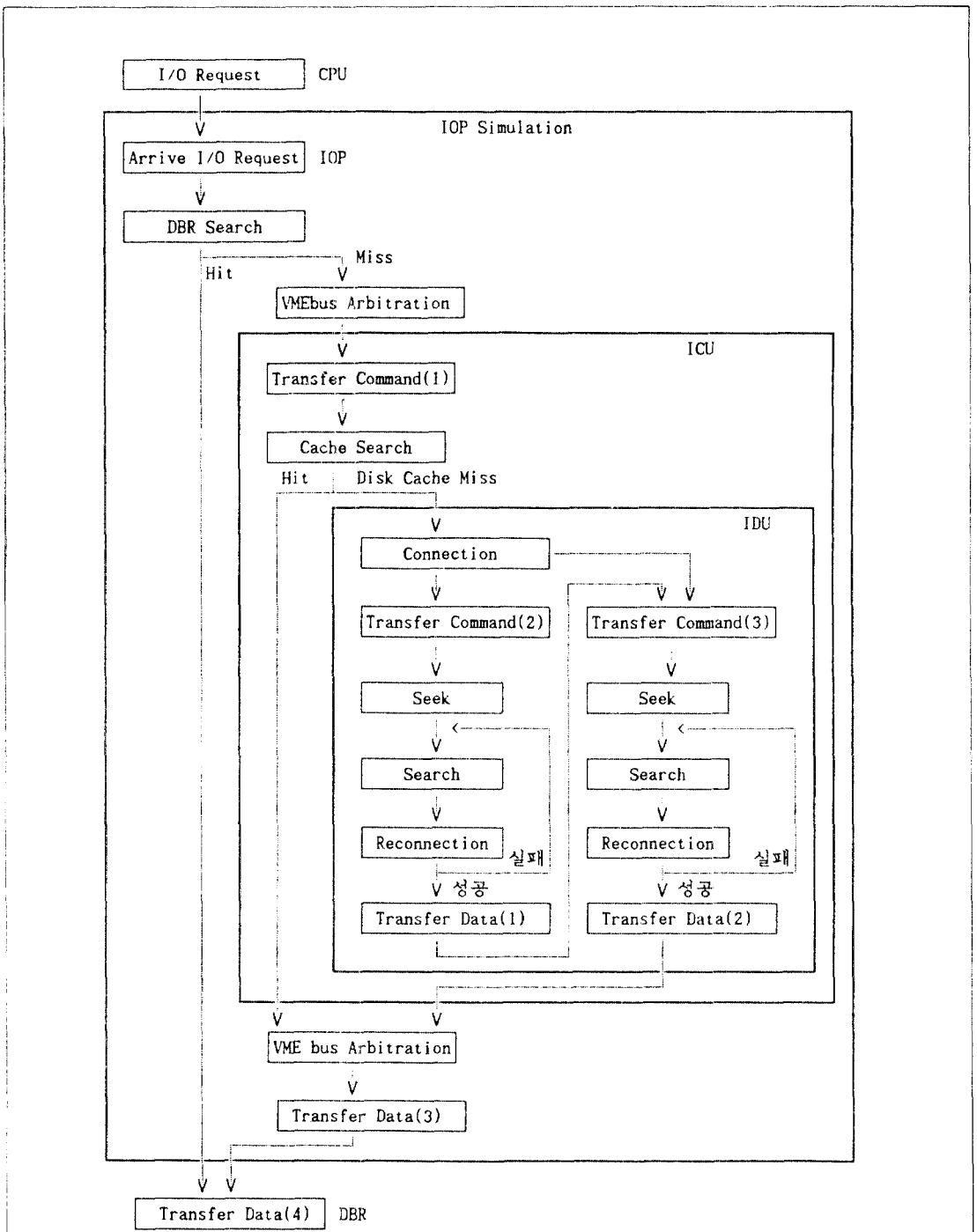
Usage : main count P1 P2 dbr_hit icu_hit

와 같은 형식으로 5개의 파라메터를 입력으로 받는다.

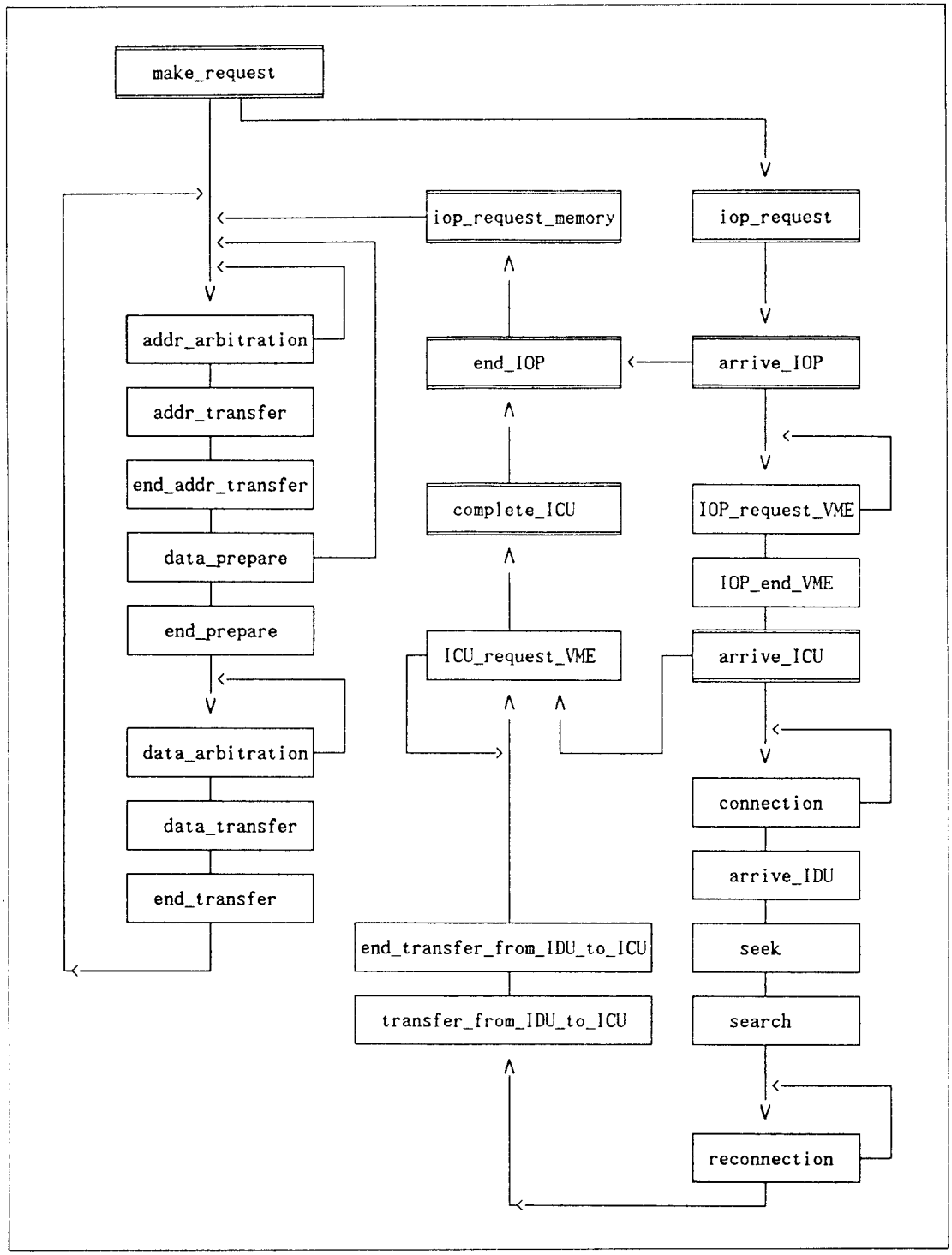
〈리스트 4-1〉에서 함수 smpl()은 시뮬레이션 프로그램을 초기화 시키는 루틴이고, define_facility()에 의해 기본적인 설비들이 정의되며, schedule()에 의해 초기 사건들을 사건 리스트에 예약하여 주며, cause()에 의해 사건 리스트에 예약되어 있는 사건들을 하나씩 꺼내어 그에 해당되는 사건들을 수행시킨다. 아래부분에 report()와 report2()의 2개 결과 보고서 출력함수가 있는데 report()는 SMPLE에서 기본적으로 제공되는 시뮬레이션 수행결과 출력함수이며, report2()는 SMPLE에서 측정되지 않는 값들을 추가로 살펴보기 위하여 덧붙인 수행결과 출력함수이다.

smpl에서 수행될 사건을 미리 사건 리스트에 예약시키는 함수는 schedule()인데 이 함수는 schedule(ev, tm, tkn); 과 같이 인수로서 사건번호(ev), 시간(tm), 토큰번호(tkn)의 3개 만을 전달하도록 되어있으나, 이곳의 확장된 SMPLE에서는 schedule(make_request, 0.0, p, p); 와 같이 4개의 인수가 전달되고 있다. 여기에서 make_request는 새로운 메모리 요구신호를 발생시키기 위한 사건번호이고, 0.0은 현재 시각으로, 처음의 p는 메모리 요구신호를 발생시키는 프로세서번호, 뒤의 p는 이 프로세서의 우선순위로서 현재의 예에서는 프로세서번호에 따라 우선순위가 주어지도록 되어 있다.

구현된 시스템 버스는 주소 중재버스, 주소버스, 데이타 중재버스, 데이타버스, 상태버스 등으로 이루어지며,



〈그림 4-4〉 IOP의 동작 과정



〈그림 4-5〉 주요 함수간의 연결

```
/*****************************************************************************************
Input Parameter :
argv[1] : Count        CPU가 발생시키는 메모리 request의 개수
argv[2]  P1            CPU가 발생시키는 메모리 request가 메모리 cache에서 hit 될 확률
argv[3]  P2            CPU가 발생시키는 메모리 request가 메인메모리에서 hit 될 확률
argv[4]  dbr_hit       I/O request 신호가 IOP의 DBR에서 hit 될 확률
argv[5]  icu_hit       I/O request 신호가 ICU의 DISK CACHE에서 hit 될 확률
*
int main(int argc, char *argv[])
{
      int i, p, ev;

      if (argc < 6)
      {
            printf("Usage : main count P1 P2 dbr_hit icu_hit\n")
            exit(1);
      } else {
            init(argv); /* 초기치 설정 */
      }

      smpl(0, "Memory System Simulation");

      define_ facility();
      for (p=no_of_IOP, p<no_of_RQ; P++) /* 작업을 시작하기 전에 초기 event를 만들어 준
      다. */
      {
            schedule(make_request, 0.0, p, p);
      }              /* p번째 CPU가 memory request를 하기 위한 준비 event */
      incount = 0L;
      while (exit_flag' = TRUE)
      {
            cause(&ev  &p); /* p는 request를 발생시키는 CPU(or IOP)번호 */
            switch(ev)
            {
            case  make_request       make_request_event(p);              break;
            case  addr_arbitration   addr_arbitration_event(p);          break
            case  addr_transfer      addr_transfer_event(p);             break

                                                .
                                                .
                                                .

            case reconnection        reconnection_event(p);              break;
            case end_IOP             end_IOP_event(p);                   break;
            default : printf("ERROR CASE EVENT NUMBER %d \n", ev); exit(1);
            }  /* switch */
      }  /* while */
      report();
      report2();
      exit(0);
}  /* END MAIN */
```

〈리스트 4-1〉 작성된 시뮬레이션 프로그램 예

이중에서 주소 중재버스는 여러 프로세서 및 입출력 처리기가 동시에 시스템 버스를 요구할 경우의 충돌을 중재하기 위해 필요하다. 주소 중재버스의 중재를 거친 프로세서만이 주소버스를 통하여 메모리 모듈에 주소를 전달할 수 있으며, 메모리 모듈에서는 입력 대기장소를 가정하여 요구한 메모리 모듈이 busy일 경우 이 장소에서 대기한다. 이때 만일 대기장소에 여유 공간이 없을 경우에는 다시 중재버스를 거쳐 주소가 전달되도록 한다. 주소 중재버스에서는 동시에 요구된 메모리 요구 신호에 대해서는 프로세서 번호순에 의한 우선순위 방법을 적용하고, 공정한 중재를 위하여 앞에서와 같이 1번이상 주소버스를 사용하고 다시 되돌아온 프로세서에 대해서는 우선순위를 가장 낮게 하여 앞에서 요구되었던 신호들이 모두 처리된 후에 처리되도록 한다. 입출력 처리기는 블럭 전송기능을 갖는다. 입출력 처리기는 1번의 중재에 의해 1블럭 만큼의 전송이 수행되며 일단 1번의 중재를 거친 후에는 1클럭에 1번씩 파이프라인식으로 주소를 전달한다.

4.4 시뮬레이션 수행출력

본 논문에서는 1명령어 수행당 1비트의 입출력을 나타내는 Amdahl/Case의 법칙에 일치하며 관련 논문에서 제시한 입출력 작업 부하의 특성을 반영하기 위해 다음의 기본 환경을 채택하였다.

-2MIPS 프로세서
-1.2 메모리 참조 / 명령어
-1 공유 메모리 요청 / 10 메모리 참조
-1 디스크 I/O 요청 / 1000 공유 메모리 요청
-1 Kbyte의 페이지

공유 메모리 및 입출력 요구는 exponential arrival rate로 발생한다.

〈그림4-6〉과 〈그림 4-7〉은 시뮬레이션 수행결과 출력 예이다.

〈그림 4-6〉은 SMPLE에서 시뮬레이션을 수행하는 과정에서 자동적으로 측정되는 값들을 report( ) 함수를 이용하여 작성된 것이다. 〈그림 4-6〉에서 FACILITY는 선언된 설비들의 명칭이고, UTIL.는 각 설비들의 활용도이고, PERIOD는 각 설비에 하나의 토큰이 서비스 받는 평균 시간이고, LENGTH는 각 설비의 대기큐에서 대기하는 토큰들의 평균 대기시간, RELEASE는 각 설비가 처리한 토큰의 개수, QUEUE는 대기큐를 거쳐간 토큰의 개수이다. 예를 들어 설비 MEMORY[0]는 총 13862개의 메모리 요구신호를 처리하였으며 그중 1904개가 곧바로 처리되지 못하고 큐에서 대기를 하였다가 처리되었다. 이때 큐에서 대기하는 입출력 명령의 평균 대기시간은 0.000036msec(=36nsec)이며 1개의 메모리 요구신호가 처리되는 시간은 0.000320msec(=320nsec)이다. 그리고 이때 이 메모리 모듈의 활용도는 0.009011이다.

smple SIMULATION REPORT

MODEL: Memory System Simulation

TIME: 492.256
INTERVAL: 492.256

		MEAN BUSY	MEAN QUEUE	OPERATION COUNTS		
FACILITY	UTIL.	PERIOD	LENGTH	RELEASE	PREEMPT	QUEUE
IOP[0]	0.055962	0.027275	5.524054	1010	0	966
CPU[0]	0.004581	0.000902	0.000000	2501	0	0
CPU[1]	0.004582	0.000904	0.000000	2494	0	0
CPU[2]	0.004582	0.000903	0.000000	2497	0	0
CPU[3]	0.004581	0.000899	0.000000	2508	0	0
MEMORY[0]	0.009011	0.000320	0.000036	13862	0	1904
MEMORY[1]	0.009152	0.000320	0.000029	14078	0	1638
			⋮			
VME BUS[0]	0.019509	0.007410	0.005499	1296	0	817

〈그림 4-6〉 시뮬레이션 수행결과 출력 예 1

IOP[0].incount = 505
IOP[0].outcount = 505
IOP[0].totaltime = 12224.616682
IOP[0].avg_time = 24.207162
IOP[0].Level[0] = 1312.5844 Count[0] = 361 Period[0] = 3.6360
IOP[0].Level[1] = 383.3403 Count[1] = 92 Period[1] = 4.1667
IOP[0].Level[2] = 10528.6920 Count[2] = 52 Period[2] = 202.4748

	COUNT	HIT	MISS	UPDATE	TIME
ICU[0] :	72	43	29	0	6329.0648
ICU[1] :	72	49	23	0	4025.3771

	count	wait	seek	search	reconnection	transfer	total
IDU[0] :	19	235.4414	16.0000	8.3333	0.8772	0.6510	261.3029
IDU[1] :	10	109.7327	16.0000	8.3333	0.0000	0.6510	134.7171
IDU[2] :	7	70.6827	16.0000	8.3333	0.0000	0.6510	95.6670
IDU[3] :	16	183.5420	16.0000	8.3333	0.0000	0.6510	208.5263

〈그림 4-7〉 시뮬레이션 수행결과 출력 예 2

〈그림 4-7〉은 SMPLE에서 자동적으로 측정되지 않는 값들을 측정하기 위하여 따로 프로그램내에서 작성한 루틴에 의하여 측정된 값들을 report2() 함수에 의하여 출력된 결과이다. IOP[0].incount는 설비 IOP[0]에 요구된 입출력 요구의 개수이다. 이 예에서는 총 10000개의 데이타 중에서 505개의 입출력 명령이 IOP[0]에 요구되었다. 참고로 〈그림 4-6〉에는 IOP[0]가 처리한 토큰의 개수가 2배로 나타나는데, 이것은 프로그램 작성과정의 편리를 위하여 토큰이 IOP에 도착할 때 한번 IOP를 요구하고 해제하였다가, 처리가 끝날 때 다시한번 요구하고 해제하기 때문이다. IOP[0].totaltime은 입출력 요구 명령이 IOP[0]에 도착하여 처리되기 까지의 입출력 동작에 소요된 총 누적 시간이며, IOP[0].avg_time은 1개의 입출력 요구 명령이 처리되는 평균 시간으로, 이 예에서는 개당 24.207162msec의 시간이 소요되었다. 그 아래의 IOP[0].Level[0]는 IOP에 요구된 입출력 명령이 IOP의 DBR에서 히트하는 경우, IOP[0].Level[1]는 ICU의 캐쉬에서 히트하는 경우, IOP[0].Level[2]는 IDU에서 데이타가 처리되는 경우의 측정치들을 나타내며, 각 레벨마다 앞에서부터 각 설비가 동작한 총 시간, 처리된 개수, 개당 평균 주기이다. 그 밑의 측정치들은 각 시간 성분별로 측정한 데이타 값들이다.

4.5 시뮬레이션 결과 분석

본 논문에서는 실 작업부하가 아닌 기존 연구의 작업부하 특성을 반영하여 확률적으로 제조된 작업부하를 이용함으로 짧은 시간에 광범위한 설계 공간의 성능을 예측할 수 있도록 하였다.

위의 기본 모델에서 프로세서의 처리능력, 프로세서의 수, 메모리 접근시간, 페이지 사이즈, 디스크 seek time 등을 가변화 한 다양한 환경하에서 시뮬레이션을 수행하였으며, 그 결과의 일부를 〈표 4-1〉에 보인다.

〈그림 4-8〉과 〈그림 4-9〉에 〈표 4-1〉의 1-4행에 있는 프로세서 수에 따른 작업수행주기의 변화 및 활용도의 변화를 그래프를 이용하여 나타내었다.

시뮬레이션이 제공하는 작업 수행 시간 및 이의 구성도는 시스템을 구성하는 각 부시스템 및 장치의 성능 기여도와 병목 구역에 관한 정보를 제공한다. 각 부시스템의 성능 기여도는 Amdahl's Speedup Law과 비교가 가능하다. 예를 들어 프로세서의 성능향상에 따른 시스템 성능의 향상을 비교해 보자(〈표 4-1〉의 1-4행). 프로세서가 2배 혹은 4배 향상되면 Amdahl's Speedup Law와 본 연구 결과가 근사하지만, 8배 향상의 경우에는 입출력 등 공유 자원에

〈표 4-1〉 시뮬레이션 결과

NO.	CASE	Time			Utilization				Tio/Tj
		Tc	Tio	Tj	Um	Uiop	Uioc	Uiod	
1	MIPS = 1.0	7849	561	8410	8.9	8.2	1.2	15.9	0.067
2	2.0	4467	623	5090	16.5	15.5	2.3	29.6	0.122
3	4.0	2524	791	3315	26.9	28.1	4.1	52.2	0.239
4	8.0	1865	1882	3748	31.7	46.1	6.4	84.3	0.502
5	p's = 1	3905	534	4439	4.2	3.9	0.6	7.4	0.120
6	2	3940	561	4501	8.5	7.8	1.2	14.9	0.125
7	4	4467	623	5090	16.5	15.5	2.3	29.6	0.122
8	8	3822	873	4695	33.5	31.1	4.6	59.7	0.186
9	Tmac = 1.0	3483	597	4080	4.4	13.2	2.5	31.6	0.146
10	2.0	3771	601	4372	8.4	14.0	2.4	30.8	0.137
11	4.0	4467	623	5090	16.5	15.5	2.3	29.6	0.122
12	8.0	4609	674	5283	30.7	18.7	2.1	27.0	0.128
13	Tsk = 8.0	4467	465	4932	16.5	15.5	2.3	20.4	0.094
14	16.0	4467	623	5090	16.5	15.5	2.3	29.6	0.122
15	24.0	4467	823	5290	16.5	15.5	2.3	38.5	0.156

Tc : 입출력 요구 발생 사이에서 프로세서와 메모리의 작업시간
Tio : 입출력 요구의 발생에서 처리 완료까지의 작업시간
Tj : 시스템의 작업시간 즉, Tj = Tc + Tio

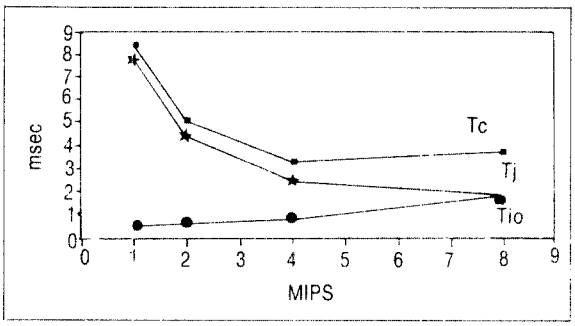
Um : 메모리 활용도
Uiop : 입출력 프로세서 활용도
Uioc : 입출력 제어기 활용도
Uiod : 디스크 활용도
P's : 프로세서 개수

Tsk : 디스크 seek 시간
Tmac : 메모리 접근시간

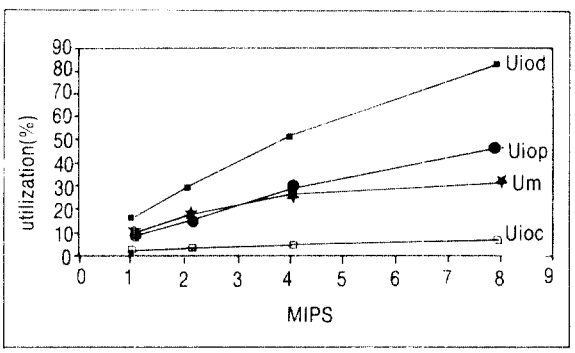
의 과부하로 인해 시스템 성능은 오히려 저해됨을 시뮬레이션에서는 예측하고 있다.

CPU speedup	System speedup Amdahl's Law	System speedup Our Model
1	1	1
2	1.78	1.65
4	2.77	2.54
8	4.35	2.24

Akella[10]의 논문에 의하면, 블럭 입출력 요구사이의 수행 시간은 242.45μsec이고, Reddy[11]의 연구 결과는 0.99 - 15.95sec이다. 즉, 성능 결과 수치는 대상시스템의 구조 및 작업 부하를 고려해야만 의의가 있다. 본 연구에서는 입출력 요구사이의 평균 수행 명령어 수는 약 8000이며, 수행 시간은 1.8 - 7.8msec의 다양한 환경에 대한 성능을 보인다. 이 경우에, 입출력의 성능 기여도는 7 - 50%로 입출력 영향의 다양성을 보인다. 프로세서 수의 증가에 따른 시스템 성능의 향상은 〈표 4-1〉의 5 - 8행에 의



〈그림 4-8〉 프로세서 수에 따른 작업수행 주기의 변화



〈그림 4-9〉 프로세서 수에 따른 활용도의 변화

해 분석 가능하다.

13 - 15행의 자료에 의하면, 입출력 시간은 디스크의 성능보다는 디스크 버퍼의 히트율 및 관리 방식에 의해 가장 큰 영향을 받는다. 실제 디스크의 seek 시간을 3배 증가하여도 입출력 시간은 1.8배, 작업 수행시간은 6%만이 증가한다. 즉, 디스크 버퍼로 인해 실제 디스크의 접근율은 입출력 요구율에 비해 매우 떨어진다.

입출력에 의한 버스 및 메모리에의 영향은 대기시간과 요청에 대한 비율 및 활용도에의 비율로 분석이 가능하다. 메모리 요청에 대한 입출력 비율은 11% 증가시키며, 이로 인해 메모리 활용율은 0.5 - 4%의 영향을 준다. 실제 동일한 입출력 요구율에 대해, 페이지 크기를 증가하거나, 메모리의 활용도가 높은 경우에는 이 영향은 매우 커지게 된다. 입출력 처리시간은 입출력 부시스템의 자료 접근 시간 및 공유메모리의 접근 시간에 의해 영향을 받는데, 입출력에 의해 메모리가 과부하되어져 메모리 대기시간이 길어지고 이로인해 다시 입출력 처리시간이 길어지게 된다.

V. 결론

본 연구에서는 기존의 C언어 사용자들이 손쉽게 시뮬레이션 모델을 구축할 수 있도록 기존의 smpl을 확장하여 SMPLE(smpl extended)를 개발하였다. SMPLE는 smpl의 자료구조를 C언어의 structure와 pointer를 기반으로 하게끔 변경시키고 이에 따르는 제반 변경 사항을 수정 보완하여 프로그램의 분석을 용이하게 하며 기능의 변경 및 추가를 수월하게 하였고, 메모리를 동적으로 관리할 수 있게 하였다. SMPLE는 시스템 레벨의 시뮬레이션 모델을 손쉽게 구축할 수 있고, 그 수행 결과도 자동으로 작성하여 주므로 매우 편리하게 사용할 수 있다. 또한 자료구조 및 프로그램이 공개되어 있으므로 프로그래머가 원하는 기능을 쉽게 추가할 수 있는 장점도 있다.

SMPLE에서 새로이 추가된 자료구조와 함수 및 설비 제어방식 등을 활용하여 실제 중형급 컴퓨터 시스템에 대한 시뮬레이션 구현과 시스템 분석의 예를 보였다. 대상 시스템은 단일 버스 공유메모리 방식의 멀티프로세서 시스템으로, 주 시스템 및 입출력 부시스템이 전체 시스템에 미치는 영향을 분석하기 위해 공유메모리 멀티프로세서 시스템의 작업수행 과정을 모델하고, 구성성분들의 성능 기여도와 입출력 프로세서, 입출력 버스, 제어기, 디스크 등으로 구성된 입출력 부시스템의 성능 영향을 시뮬레이션에 의해 분석하였다. 또한 구성 부시스템(또는 장치)의 성능향상 및 프로세서 수에 따른 시스템 성능 개선 정도를 제시하였다.

SMPLE는 사용자 인터페이스 및 보고서 형태를 다양하게 하고 기능을 계속적으로 보강하여 컴퓨터 시스템의 시뮬레이션 모델 구축을 위해 발전될 것이다.

참고문헌

[1] 서울 대학교 컴퓨터 신기술 공동 연구소, 『고속 입출력 시스템 최종 보고서』, 1992.

[2] 대한 전자공학회 전자계산 연구회, 『컴퓨터 기술-국가기간전산망 특집』, 제7권 제1호, Dec. 1990.

[3] 정보과학회지, 『특집 시뮬레이션』, 90.2.

[4] 김홍준, 조경산, "디스크 입출력 및 디스크 버퍼의 성

능분석", 『정보과학회 학술 발표 논문집』, 제19권 제1호, pp. 195~198. 1992.

[5] 조성만, "시뮬레이션을 이용한 컴퓨터 시스템 성능평가에 관한 연구", 석사 논문, 단국대학교, 1992.

[6] Arnold O.Allen IBM System Science Institute "Queueing Models of Computer Systems", *COMPUTER*, April 1980, pp. 13~24.

[7] Averill M.Law and W.David Kelton, *SIMULATION MODELING AND ANALYSIS*, McGraw-Hill Book Company, 1991.

[8] Ivan Loffler, "Computer Performance Measurement and Capacity Planning for DEC VAXes and other Mini-computers", *Proc. of CMG International Conference*, 1988, pp. 733~737.

[9] M.H.MacDougall, *Simulating Computer Systems - Techniques and Tools*, The MIT Press, 1987.

[10] Janaki Akella and Daniel P. Siewiorek, "Modeling and Measurement of the Impact of Input and Output on System Performance", *Proc. of Symposium on Computer Architecture*, 1991, pp. 390~399.

[11] A.L. Narasimha Reddy, "A study of I/O Behavior of Perfect Benchmarks on a Multiprocessor", *Proc. of 17th Int. Symp. on Computer Architecture*, 1990, pp. 312~321.

저자소개

조성만

1990년 단국대학교 전자공학과 졸업
1992년 단국대학교 전산통계학과 졸업, 이학석사
현재 단국대학교 전산통계학과 박사과정 재학중

조경산

1979년 서울대학교 전자공학과 졸업
1981년 한국과학원 산업전자공학과 졸업, 공학석사
1988년 미국 텍사스주립대(오스틴), 공학박사
1988~1990년 삼성전자 컴퓨터부문 책임연구원
1990~현재 단국대학교 전산통계학과 조교수
관심분야: 시스템 성능분석, 컴퓨터 구조, 시뮬레이션응용