울산대학교 연구논문집 제21권(자연과학·공학편) 제1호 pp.65~71. 1990.
UOU Report Vol.21, (Natural Science and Engineeing) No.1, pp.65~71, 1990.
University of Ulsan

SPICE2G6의 벡터화 및 Multitasking화

구 자 록
전 자 계 산 학 과

〈요 약〉

본 논문은 회로 시뮬레이터인 SPICE2G6를 CRAY-2S 시스템에 효율적으로 수행할 수 있는 기법에 관하여 기술하고 있다. 프로그램 코드의 최적화를 위한 알고리즘에서의 병렬 처리를 전개함으로써 주어진 소프트웨어의 코드 변화을 최소화하면서 수행시간을 단축시킬 수 있는 벡터화와 Multitasking화를 위한 체계를 제안하고 있다.

Vectorization and Multitasking of SPICE2G6

Koo, Ja-Rok
Dept. of Computer Science

〈**Abstract**〉

This paper describes the efficient implementation technique of SPICE2G6 on the CRAY-2S super-computer system. A framework for the vectorization and multitasking of the algorithm is also proposed and illustrated with program examples, so that the execution time can be reduced with minimal changes to the software.

Ⅰ. 개 요

대량의 계산을 요하는 소프트웨어의 개발은 컴퓨터 구조의 성능에 크게 좌우된다. 지난 10여년간 단일 프로세서 스칼라 컴퓨터에서 다중 프로세서 벡터 컴퓨터에 이르기까지 매우 빠른 속도로 발전되어 왔다. 이러한 이동은 더 큰, 더 빠른 컴퓨터를 요하는 과학 계산분야의 수요에 부응하는 것이기도 하다. 국내에서도 이러한 흐름에 따라, 병렬처리를 효율적으로 수행하는 고속의 슈퍼컴퓨터인 CRAY-2S를 도입함으로써 여러 응용분야에 적용시킬 수 있는 환경을 구축하게 되었다. 기존의 시스템에서 구현된 회로 시뮬레이터인 SPICE2G6는 입력의 크기가 방대해지면 컴퓨터의 처리속도가 엄청나게 느려지는 한계점이 발생한다. 이를 극복하기 위해 여러 Special purpose machine이 제안되었지만, 실제로 그러한 기계의 구현은 막대한 비용과 기간을 요구하고 비용면에서의 효율성도 아직 명확히 밝혀지지 않은 실정이다.

이에 본 논문에서는, SPICE2G6를 CRAY-2S 슈퍼컴퓨터에 효율적으로 수행할 수 있는 구현 방법을 제시하고자 한다. 먼저 SPICE2G6 프로그램 코드와 CRAY-2S가 제공하는 소프트웨어에 대한 분석을 행하고, SPICE2G6를 이 슈퍼컴퓨터에 단순 이식을 시킨뒤, SPICE2G6의 코드중 실행시간을 크게 차지하는 모듈을 Multitasking을 위한 병렬처리 가능한 구조로 재구성함으로써 수행속도를 효율적으로 향상시키고자 한다.

II. CRAY-2S 시스템과 SPICE 프로그램

1. CRAY-2S의 구조

CRAY-2S Cray Research Inc.에서 개발한 컴퓨터로서 백터연산과 Multitasking을 지원하는 슈퍼컴퓨터이다. 이 CRAY-2S의 본체는 다음과 같은 구성요소로 이루어지있다. 크게 시스템의 동작을 제어하고 감시하는 Foreground 시스템과 2억 5천 6백만개의 글자를 저장할 수 있는 공유 기억장치, 그리고 백터 및 스칼라 연산을 할 수 있는 4개의 Background 프로세서로 이루어져 있다.

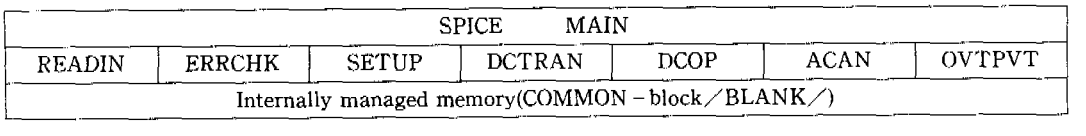
Foreground 프로세서는 Background 프로세서의 요구에 대한 응답을 하고, 채널통신 신호들의 순서를 정함으로써 시스템의 동작을 관할한다. Foreground 프로세서의 주요 기능은 Background 시스템의 여러 채널 노드로부터의 요구들에 대한 실시간 응답을 하는 것이다.

CRAY-2S 본체는 4개의 동일한 Background 프로세서를 가지고 있다. 각각의 Background 프로세서는 산술 및 논리연산을 수행하기 위한 레지스터와 기능장치를 가지고 있다. 이러한 연산들과 Background 프로세서의 다른 기능들은 제어장치의 역할로 통합될 수 있다. 각각의 Background 프로세서는 동일의 독립된 제어장치를 포함한다. 이들 제어장치는 32-bit의 프로그램 주소 레지스터와 공유 기억장치의 사용 없이도 프로그램 루프를 실행할 수 있는 8개의 독립된 16글자 단위 명령어버퍼, 그리고 여러 Background 프로세서가 하나의 일을 수행하고자 할 때 공유 기억장치를 보호하기 위한 8개의 semaphores를 포함하고 있다. Background 프로세서들은 각각 독립된 9개의 기능 장치를 가지고 있다. 이들 기능 장치는 레지스터로부터 데이타를 받아 연산을 수행하고 그 결과를 다시 레지스터로 보낸다. 기능장치들은 동시에 수행을 하게 된다. 각각의 기능장치들은 하나의 클럭주기마다 하나의 결과를 전달한다. 각각의 Background 프로세서는 주소 덧셈 및 곱셈 연산장치와 스칼라 정수, 이동, 논리 기능장치, 백터 정수 및 논리 연산 기능 장치, 그리고 실수 덧셈 및 곱셈 연산 기능장치들로 이뤄져 있다.

CRAY-2S에서는 슈퍼컴퓨터의 수행속도와 처리능력을 활용하기 위해 표준 UNIX에 I/O, File 시스템, Multitasking 등의 개념을 첨가하여 이뤄진 UNICOS를 운영체계로 사용하고 있다. 또한 CRAY-2S에서는 이전의 모든 슈퍼컴퓨터와 마찬가지로 FORTRAN프로그램을 주로 사용한다. UNICOS 아래에서 Pascal로 쓰여진 CFT77 FORTRAN컴파일러가 사용되고 있다. CFT77은 CRAY 컴퓨터 시스템 사용자들의 고도의 계산요구에 부합하도록 특별히 고안되어진 것이다. 이 CFT77 컴파일러의 특징을 살펴보면, 전적으로 주기억 장치에서 작동하며, 큰 프로그램의 경우에 대비해 기억장소의 동적인 확장이 가능하며, 프로그램의 원시코드를 모듈단위로 재구성하며, 컴파일 순서가 원시 프로그램 처리, 최적화, 백터화, 그리고 코드생성으로 이뤄져 있다.

2. SPICE 프로그램 분석

SPICE2G6는 약 15000줄의 FORTRAN 언어로 작성된 프로그램으로서 주프로그램과 7개의 주요 모듈, READIN, ERRCHK, SETUP, DCTRAN, DCOP, ACAN, OVTPVT로 이루어진다. SPICE2G6의 전체 블럭 도표는 그림 2.1과 같다.



(그림 2.1) SPICE2G6의 전체 블럭 도표

SPICE2G6의 주 프로그램은 전체 순환루프에 대한 제어를 담당한다. 프로그램이 시작되면 우선 SPICE2G6에서 사용되는 상수가 초기화되고 READIN모듈이 호출되어 입력화일을 읽어 들인다. READIN 모듈이 입력화일을 읽으면 회로에 대한 자료구조가 형성되는데, 이 자료구조는 각 회로소자, 소자모델 및 출력변수를 정의하는 Linked List의 집합으로 이루어 진다. 또한 수행되어야 할 분석의 유형 및 제어변수의 지정을 위해 COMMON 블럭 변수가 준비된다. READIN 모듈이 수행된 후 ERRCHK 모듈이 호출되어 입력된 회로의 기술에 대한 오류검사를 하며, 아울러 회로와 소자 모델의 매개변수 목록 및 노드의 테이블을 출력해 낸다. READIN과 ERRCHK 모듈의 수행이 끝나면 회로

의 분석에 대한 준비가 끝난 상태이며, SETUP 모듈이 호출되어 DCTRAN과 ACAN 모듈에 의해 사용되는 정수형 포인터 구조를 만든다. SETUP 모듈의 실행이 끝나면 회로에 대한 분석이 진행되며, 사용자가 만든다. SETUP 모듈의 실행이 끝나면 회로에 대한 분석이 진행되며, 사용자가 정의한 온도에 따라 주프로그램의 순환루프가 반복된다. 각 온도에 대한 주프로그램의 분석 루프가 수행된 후, 프로그램은 해당 사항에 대한 통계 사항을 인쇄하고, 다음 입력을 계속 읽어 들인다. DCTRAN 모듈은 SPICE2G6에서 가장 크고 복잡한 부분으로, DC 동작점 분석, 과도 초기조건 분석, DC전달곡선 분석 및 과도 분석을 행한다. 이 모듈의 주요 구성 부프로그램으로는 DCTRAN, TRUNC, TERR, SORUPD, ITER8, DCDCMP, DCSOL, LOAD, INTGR8, DEODE, BJT, JUNCT, FETLIM, JFET, MOSFET 등이 있다. DCTRAN 프로그램은 전체 모듈을 관장하는 부분으로 4가지의 서로 다른 유형의 분석을 수행하게 된다.

III. Multitasking과 Vectorization

응용 프로그램머들이 컴퓨터 하드웨어를 자세히 알아야 할 필요는 없다. 사용자들이 Multiprocessing과 Vector 연산을 수행하기 위해 그 기능을 갖는 내장함수를 불러 처리할 수 있는 소프트웨어의 보조가 제공되어진다. 이 부분에서는 최적화 소프트웨어를 위한 Multitasking과 Vectorization에 관련된 몇가지 문제점들을 언급하고자 한다. FORTRAN 최적화 코드를 Vector Multiprocessor 환경에 적용하기 위한 일반적인 체계와 효율성을 향상시킬 수 있는 몇 가지 척도를 요약한다.

1. Multitasking

Multitasking이란 프로그램을 동시에 수행할 수 있는 두 개 이상의 부분으로 재구조화하는 것을 의미한다. 수행을 위해 시간상의 순서를 정할 수 있는 계산단위를 Task라 칭한다. 일반적으로 한 개 이상의 프로세서가 주어진 job의 여러 Tasks에 전적으로 할당되어진다거나, 그 Tasks들이 특정한 순서로 수행되어 진다거나, 특정 Task가 먼저 끝난다는 보장은 없다. 이점에서 Multitasking화 된 프로그램은 시간에 대해서 비결정적이다. 그러나, 적당한 동기화 기법에 의해 프로그램을 결과에 대해 결정적이게 할 수 있다. 여러 Tasks들에 의해 사용되어질 어떤 모듈에 대해 복제를 허용하는 프로그램의 성질을 Reentrancy라 부른다. 그 모듈은 하나의 부프그램이나 프로그램 문장들의 집합이 될 수 있으며, 그 모듈의 환경은 매번 task로서 불려질 때 마다 다시 만들어진다. Reentrancy는 multitasking화된 프로그램을 정확히 수행함에 있어서 충분한 조건은 아니나 필요한 성질이다. 병렬처리 가능 tasks들을 서로 독립성을 유지할 때만이 정확하게 수행이 가능해 진다.

2. Vectorization

Vectorization이란 가장 안쪽 루프에서 병렬성을 전개하기위해 프로그램을 재구성하는 것을 의미한다. 이것은 스칼라연산에 필요한 시간 내에서 긴 벡터의 처리를 가능하게 하는 벡터 하드웨어의 성질에 의해서 가능해진다. 벡터화의 주요 대상은 반복 수행문(안쪽 DO—루프)이다. 루프가 N번 수행되어진다면, 벡터컴퓨터는 이것을 K개의 세그먼트로 나눠 N/K 크기의 벡터를 처리하게된다. (K는 컴퓨터의 하드웨어 구성에 의해 결정됨.) 나눠진 벡터는 주어진 기능 장치에 의해 연산이 이뤄진다. 예를 들면 한 장치가 벡터 덧셈연산을 하고 있는 동안 다른 장치는 벡터 곱셈연산을 수행하게 된다. 이런 식으로 여러 장치들이 서로 겹쳐지거나(Overlap) 순차적으로 연결되어진다(Chained). 전자의 경우에는 한 개 이상의 장치가 동시 수행되며, 후자의 경우에는 한 장치에서 계산된 결과가 다른 장치로 곧장 옮겨질 수 있어서 기억장소의 참조 필요성을 줄일 수 있다. 한 루프에 존재하는 연산들이 서로 독립적일 때 소프트웨어 시스템의 효율적인 벡터화가 이뤄질 수 있다.

IV. Multitasking

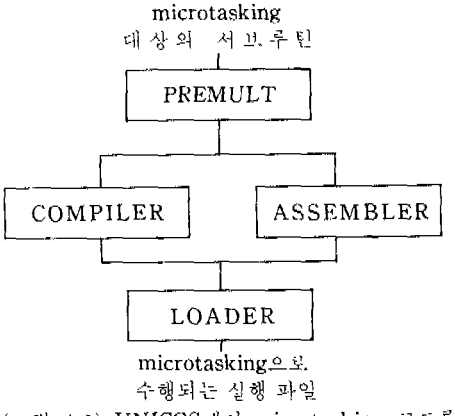
1. Multitasking 기법의 선택

본 논문에서는 CRAY-2S에서 제공하는 Multitasking 기법인 Macrotasking과 Microtasking의 기법중 Microtasking 기법을 이용하여 SPICE2G6를 병렬처리 하였다. 하나의 프로그램을 Macrotasking을 위한 코드변환이 Microtasking에서의 것보다 더 많은 부분의 변환과 변환의 어려움, 또한 Macrotasking Library 루틴에 따르는 CPU Time 오버헤드등의 관점에서 볼 때, 수치해석을 이용한 반복적인 방법에 의한 회로 방정식의 해를 구하며, 빈

번한 GO TO문의 사용으로 인한 비구조적인 프로그램으로 이뤄져 있으며 COMMON문에 의한 수많은 공유변수를 사용하는 SPICE2G6와 같은 software System은 Vectorization과 더불어 DO 루프의 병렬 처리가 가능하며, 낮은 Synchronization 오버헤드 및 프로세서 디렉티브의 효율적인 수행이 가능한 Microtasking이 더 유효 적절하기 때문이다.

2. Microtasking 기법에 의한 Multitasking

Macrotasking에서와는 달리, Microtasking에서는 Multitasking이 가능한 프로그램의 코드 부분을 Microtasking 디렉티브를 사용하여 표시함으로써 다수의 프로세서가 병렬적으로 수행을 가능하게 한다. Microtasking 기법을 사용하기 위해서는, 원시 프로그램의 병렬 처리 가능 부분에 첨가한 Microtasking 디렉티브를 이 원시 프로그램을 컴파일하기 전에 Preprocessor인 PREMULT를 수행시켜 Microtasking이 가능한 코드로 변경시켜야 한다. 이 PREMULT는 Microtasking화된 포트란 부프로그램에 대해 두개의 포트란 부프로그램과 CAL(Cray Assembly Language)로 쓰여진 하나의 파일을 생성하는데, 두개의 포트란 코드 중 하나는 Multitasked 버전이고, 다른 하나는 그렇지 않은 Single Procesor 버전이다. 그래서, 수행시에 Microtasking화된 부프로그램을 호출할 경우, 이 CAL 루틴이 먼저 수행되어져 이미 Microtasking이 가능한 상태이면 Single Processor 포트란 부프로그램으로 수행이 진행되며, 그렇지 않은 경우는 Multitasked 버전으로 수행이 진행된다. 다음 그림은 UNICOS 환경하에서 Microtasking의 과정을 보여주고 있다.



(그림 4.1) UNICOS에서 microtasking 코드를 얻기 위한 과정

다음은 Microtasking을 위한 프로그램의 분석을 어떻게 할 것인가에 대해서 알아본다. 일반적으로, Vectorizing이 Microtasking보다 성능 평가에 있어서 뛰어나다. 짧은 길이의 벡터 경우에는 스칼라 프로세싱이 벡터 프로세싱보다 뛰어나며, 작은 크기의 Task 경우에는 벡터 프로세싱이(또는 스칼라 프로세싱의 경우까지)Multitasking보다 뛰어나다. 다음의 간단한 프로그램 루프를 생각해 보자.

```
DO 10 I=1, N
   A(I)=B(I)+S*C(I)
10 CONTINUE
```

N의 값에 따라 스칼라, 벡터, 또는 Multitasking 프로세싱을 선택함에 있어서 처리속도는 달라질 것이다. N의 값이 매우 작으면 스칼라 프로세싱이, N의 값이 매우 크면 Multitasking이 적절할 것이다. 중첩된 루프의 경우에는, 가장 안쪽 루프는 벡터 처리를 하고, 다중 프로세서하에서 벡터화를 최대한 활용하기 위해 바깥 루프의 경우에는 Microtasking을 행한다.

Microtasking을 위한 여러 가지의 보조시스템이 있다. Flowtrace(flow, flodump, prof, and ftref)는 부프로그램을 호출한 횟수와 그들 부프로그램에서 소비한 시간을 그리고 프로그램의 Calling Tree를 제공한다.

3. Microtasking으로의 변환

다음은 Microtasking 디렉티브를 사용하여 SPICE2G6 프로그램을 Microtasking화 한 예를 설명하기로 한다. 먼저, Microtasking을 적용하기 위해서는 주 프로그램에 CMIC$ GETCPUS 디렉티브가 나타나야 하는데, 이는 Microtasking화된 프로그램에 사용가능한 최대의 프로세서의 수를 나타낸다.

```
      PROGRAM SPICE
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
      . . . . .
CMIC$ GETCPUS
      IPOSTP=0
```

그림 4.3.1 주 프로그램에서의 Microtasking의 선언

SPICE2G6의 부프로그램인 DCSOL은 회로 방정식의 행렬식을 LU-Decomposition한 결과를 Forward 및 Backward Substitution에 의해 해를 구하는 루틴으로서, DO루프로 구성되어 있어 CMIC$ DO GLOBAL FOR 디렉티브를 사용하여 Microtasking화 한다.

```
CMIC$ MICRO
      SUBROUTINE DCSOL
      IMPLICIT DOUBLE PRECISION (A-H,
      O-Z)
```

```
      . . . . .
C FORWARD SUBSTITUTION
CMIC$ DO GLOBAL FOR 20
      DO 20 I=2, NSTOP
      LOC=I
      IORD=NODPLC(IRSWPF+I)
   10 LOC=NODPIC(JCPT+LOC)
      IF (NODPLC(JCOLNO+LOC).GE.I) GO
TO 20
      J=NODPLC(JCOL+LOC)
      JORD=NODPLC(IRSWPF+J)
      VALUE(LVN+IORD)=VALUE(LVN+
IORD)-
     *  VALUE(LVN+LOC)*VALUE(LVN+
JORD)
      GO TO 10
   20 CONTINUE
```

그림 4.3.2 Single DO 루프의 Microtasking의 예

CMIC$ DO GLOBAL 디렉티브에는 네 가지의 유형이 있는데, CMIC$ DO GLOBAL, CMIC$ DO GLOBAL LONG VECTOR, CMIC$ DO GLOBAL BY expression, 그리고 CMIC$ DO GLOBAL FOR expression이 있다. CMIC$ DO GLOBAL은 2개 이상의 중첩된 루프에서 안쪽 루프는 벡터 처리로, 바깥쪽 루프는 Microtasking으로 수행한다. 이 디렉티브는 단순한 루프에서는 비효율적이며, 여기에는 나머지 3개의 디렉티브를 사용한다. CMIC$ DO GLOBAL BY expression과 CMIC$ DO GLOBAL FOR expression은 반복 횟수를 고정하지 않고 사용자가 임의로 정할 수 있는데, 전자는 루프를 expression 크기의 반복횟수를 갖는 프로세스들로 나누는 것을 말하며, 후자는 루프를 expression 만큼의 프로세스들로 나누는 것을 의미한다.

위의 부프로그램 DCSOL의 예는 20개의 프로세서들로 이뤄진 Microtasking을 나타낸다. 수치적분에 사용되는 부프로그램인 COMCOF에는 CMIC$ DO GLOBAL디렉티브를 사용하여 Microtasking화 하였는데, 이는 이 부프로그램에 2중 및 3중 DO 루프가 존재하기 때문이다.

```
CMIC$ MICRO
      SUBROUTINE COMCOF
      ..........
C LU DECOMPOSITION
CMIC$ DO GLOBAL
      DO 70 I=2, ISTOP
      DO 70 I=2, ISTOP
      JSTART=I+1
      IF (JSTART. GT. ISTOP) GO TO 70
      DO 60 J=JSTART, ISTOP
      GMAT(J,I)=GMAT(J,I)/GMAT(I, I)
      DO 50 K=JSTART, ISTOP
      GAMT(J, K)=GMAT(J, I)/GMAT(I, I)
   50 CONTINUE
   60 CONTINUE
   70 CONTINUE
C FORWARD SUBSTITUTION
CMIC$ DO GLOBAL
      DO 90 I=2, ISTOP
      JSTART=I+1
      IF (JSTART. GT. ISTOP) GO TO 90
      DO 80 J=JSTART, ISTOP
      AG(J)=AG(J)—GMAT(J, I)*AG(I)
   80 CONTINUE
   90 CONTINUE
C BACKWARD SUBSTITUTION
CMIC$ PROCESS
      AG(ISTOP)=AG(ISTOP)/GMAT
(ISTOP, ISTOP)
CMIC$ END PROCESS
      IR=ISTOP
CMIC$ DO GLOBAL
      DO 110 I=2, ISTOP
      AG(IR)=AG(IR)—GMAT(IR, J)*AG(J)
  100 CONTINUE
      AG(IR)=AG(IR)/GMAT(IR, IR)
  110 CONTINUE
```

그림 4.3.3 중첩된 DO 루프의 Microtasking의 예

CMIC$ PROCESS는 Control Structure의 시작을 나타내며, 이 뒤를 따르는 코드는 Single Process임을 의미한다. CMIC$ ALSO PROCESS 디렉티브는 일련의 서로 독립적인 문장을 병렬 처리하기 위해서 사용하는데, MOS 트랜지스터의 계수행렬을 구성할 때 사용되는 부프로그램 MOSFET에서의 그 예를 보기로 한다.

```
CMIC$ MICRO
      SUBROUTINE MOSFET
      IMPLICIT DOUBLE PRECISION(A—H,
O—Z)
```

```
      .....
      LOC=LOCATE(14)
      CCMIC$ PROCESS
      LOCV=NODPLC(LOC+1)
      NODE1=NODPLC(LOC+2)
      NODE2=NODPLC(LOC+3)
      NODE3=NODPLC(LOC+4)
CMIC$ ALSO PROCESS
      NODE4=NODPLC(LOC+5)
      NODE5=NODPLC(LOC+6)
      NODE6=NODPLC(LOC+7)
CMIC$ ALSO PROCESS
      LOCM=NODPLC(LOC+8)
      IOFF=NODPLC(LOC+9)
      TYPE+NODPLC(LOCM+2)
      LOCM=NODPLC(LOCM+1)
      LOCT=NODPLC(LOC+26)
CMIC$ END PROCESS
```

그림 4.3.4 병렬처리 가능한 일련의 독립적인 문장의 예

이와 같이, 본 논문에서는 CMIC$ DO GLOBAL, CMIC$ DO GLOBAL LONG VECTOR, CMIC$ DO GLOBAL FOR/BY expression, CMIC$ PROCESS, CMIC$ END PROCESS, 그리고 CMIC$ ALSO PROCESS등을 사용한 Microtasking 기법으로 병렬처리를 행하였다.

V. 성능 평가

여기서는 다른 시스템 환경 아래에서 행한 SPICE2G6의 경우와 CRAY-2S에서 행한 경우를 비교해 본다. CRAY-2S에서의 경우에는 UNICOS에서 제공하는 time 기능을 사용하였고, 다른 시스템의 경우에는 SPICE2G6에서 제공하는 CPU-time을 사용하여 비교를 행하였다. Multitasking화된 SPICE2G6가 제공하는 시간은, 여러 프로세서에 의해 병렬적으로 처리되는 시간을 산술적으로 합하거나, Control Structure의 포함 유무로 부정확한 결과를 낳기 때문이다. Microtasking을 행한 SPICE2G6의 성능평가를 위해 UNICOS 환경하에서 timex 기능을 동일한 입력회로에 대하여 수행시켰지라도 항상 같은 결과를 나타내지는 않는데, 이는 UNICOS시스템의 multi-user 환경하에서의 프로세서 스케쥴링에 차이가 존재하기 때문이다. 따라서 본 논문에서는 동일한 벤치마크 프로그램에 대하여 10회 반복수행하여 얻은 결과를 평균한 값으로 성능평가를 행하였다.

먼저 성능평가를 위해 사용한 7개의 벤치마크의 구성을 요약해 본다.

회 로	분석의 종류	능동소자의 갯수	회로의 노드 수
자동 증폭기	1, 2, 3, 4	4	14
터널 다이오드	2, 4	0	3
RTL 인버터	2, 4	2	11
슈미트 트리거	4	4	17
MOS MEMORT	4	12	10
MOS 증폭기 1	2, 3	27	21
MOS 증폭기 2	4	27	21

* 참조 : 1. 소신호 특성 분석 2. DC 전달 특성 분석
3. AC 주파수 분석 4. 과도 분석

(그림 5.1) 시뮬레이션의 벤치마크로 사용되는 회로들

다음에는 VAX-11/780, IBM/3090, CRAY-2S에서 각각 수행한 결과를 비교해 본다.

위의 결과에서 보는 바와 같이 CRAY-2S에서 수행시킨 속도가 가장 빠름을 알 수 있다. 미니 컴퓨터인 VAX-11/780은 IBM/3090 및 CRAY-2S에서의 것보다 최고 15배 처리 시간이 지연됨을 알 수 있다. 파이프라인 아키텍처를 사용한 IBM/3090에서는 능동소자의 갯수가 5개 이하인 경우에는 수행속도가 CRAY-2S에서의 경우보다 빠르나 능동소자의 갯수가 10개 이상이 될 경우에는 수행속도가 현저히 떨어짐을 볼 수 있다.

시뮬레이션 회로	VAX-11780	IBM 3090	CRAY-2S 단순이식	CRAY-2S 병렬처리
차동 증폭기	18.31	0.78	1.52	1.15
터널 다이오드 오실레이터	16.77	0.82	1.00	0.69
RTL 인버터	14.08	0.69	1.08	0.80
슈미트 트리거	13.64	0.68	1.15	0.83
MOS MEMORY	16.76	1.27	1.35	0.85
MOS 증폭기 1	40.04	2.42	2.63	1.73
MOS 증폭기 2	14.49	3.62	4.51	2.90
합 계	134.09	10.30	13.24	8.95

(그림 5.2) 수행시간을 비교한 결과의 도표

또한 Microtasking화한 경우와 단순 이식한 경우를 비교하여 보면, Microtasking화한 경우가 평균 1.5배의 빠른 처리속도를 나타냄을 알 수 있다. 능동소자의 갯수가 5개 이하의 경우에는 큰 차이가 없지만, 회로의 크기가 커져서 능동소자의 갯수가 증가할 경우에는 Microtasking화한 경우가 단순이식의 경우 보다 그 처리시간이 빨라짐을 알 수 있다.

참 고 문 헌

1) Cray Research, Inc.(1988). "CRAY-2S Multitasking Programmer's Manual."
2) Cray Research, Inc.(1988). "Technical Training And Developments."
3) Cray Research, Inc.(1986). "CRAY-2S Computer System Multitasking."
4) Alan H. Karp & Robert G. Babb II.(1988). "A Comparison of 12 Parallel Fortran Dialects." IEEE Software. Sept, pp.52~67.
5) Hwang, K., Su, S. P, and Ni, L. M.(1981). "Vector Computer Architecture And Processing Techniques." Advances in Computers. vol.20, pp 116~197.
6) Hwang, K. and Priggs, F. A.(1984). "Computer Architecture And Parallel Processing." Mcgraw-Hill.
7) Willi Schonauer.(1987). "Scientific Computing On Vector Computers." North-Holland.
8) August, M. C., et al.(1989). "Cray X-MP: The Birth of a Supercomputer." IEEE Computer. Jan, pp.45~52.
9) Zenios, S.A. and Mulvey, J. M.(1988). "Vectorization and Multitasking of Nonlinear Network Programming Algorithms." Mathematical Programming. vol.42, pp.449~470.
10) Allen, J.R. and Kenny, K.(1982). "PFC: A Program to Convert Fortran to Parallel Form." The Proceedings of the IBM Conference on Parallel Computers and Scientific Computations.