공학 연구논문집 제22권 제2호 pp. 75~85, 1991. 울산대학교
Journal of Engineering Research Vol. 22, No. 2, pp. 75~85, 1991.
University of Ulsan.

Alliant FX/2812 시스템의 병렬처리 구현에 관한 연구

구 자 록
전자계산학과

〈요 약〉

본 논문은 Alliant FX/2812 시스템에서 주어지는 프로그램의 동시성과 벡터화 구현을 효율적으로 처리하기 위한 기법을 고찰하고, 간단한 실험 결과를 통하여 대단위 프로그램의 병렬화 작업의 가능성과 기준을 제시하였다. 이 논문은 또한 Alliant FX/2812 시스템의 기본 프로세서인 인텔의 i860 프로세서의 특징인 수퍼 스칼라 기능을 살펴보았다.

A Study on the implementation of Concurrency and Vectorization on Alliant FX/2812 system

Koo, Ja-Rok
Dept. of Computer Science

〈Abstract〉

Objectives of this paper are to study techniques and methods for effective implementation of concurrency and vectorization for large codes; and to determine the feasibility of parallelizing an existing large scientific code; and to estimate potential speedups attainable.

This paper presents the experimental results in order to study the above objectives on Alliant FX/2812 system. This system is based on the Intel's i860 processors. Also, this paper shows the feasibility of super-scalar processors for the benchmark applications.

I. 개 요

병렬 처리 프로세서가 대단위의 과학계산 분야에서 널리 사용되어 오고 있다. 이러한 추세는 앞으로도 계속 이어질 전망이다. 국내에서도 과학원의 시스템 공학센터에 이어 기업체 및 학교 연구기관에서 수퍼 컴퓨터 내지는 미니 수퍼컴퓨터를 도입하여 여러 응용 분야에서의 수요에 부응하기 위한 환경을 구축하기 시작하였다.

이에 본 논문은 서울 대학교 전자계산소에 설치된 미니 수퍼컴퓨터인 Alliant FX/2812에서 간단한 실험을 통하여 효율적인 동시성 및 벡터화 구현을 위한 기법을 살펴보았다.

Alliant FX/2812 다중 프로세서 미니-수퍼 컴퓨터 시스템은 과학 계산용 프로그램에서 발견되는 병렬성을 자동으로 찾아내도록 설계되었다. 물론 의도는 기존에 존재하는 Fortran 프로그램을 되도록이면 그대로 이용하여 병렬처리를 하는 것이다[13]. 이러한 계산 능력의 이점을 구현하기 위해, 인텔의 i860 64 비트 마이크로 프로세서를 이용하여 스칼라, 부동 소숫점 연산, 벡터 및 동시성 명령어를 지원하고 있다. 하드웨어 구조와 운영 체제는 고성능과 계산 능력을 구현하기 위하여 최적화 설계로 이뤄져 있다[9].

II. Alliant FX/2812의 구조

1) i860 CPU 구성

Alliant FX/2812 시스템의 기본 프로세서인 i860 CPU는 64-비트 RISC(Reduced Instruction Set Computer)형으로서, 대형 시스템 및 수퍼 컴퓨터 구조의 개념을 이용하여 병렬 연산을 수행할 수 있다. 성능 향상을 고려하여 칩을 크게, 정수형 및 부동 소숫점 연산 장치, 명령어 및 데이타 캐쉬 메모리, RISC 코아, 그래픽스 장치, 그리고 버스 제어 장치로 이뤄져 있다. 이 64-비트의 구조는 각 클럭 사이클마다 다중 연산을 구현하기 위해 데이타 및 명령어 밴드위스(Bandwidth)를 지원하고 있다. 다중 기능장치로부터 성능향상을 위하여 i860 CPU는 매 클럭사이클마다 세 개까지의 연산을 발생시킨다. 단일 명령어 형태에서, 프로세서는 매 사이클마다 RISC 코아 명령어나 부동 소숫점 명령어를 발생시켜 스칼라 연산 구현에 이용하며, 두 개의 연산 형태에서는 RISC 코아가 각 사이클마다 64-비트 너비의 명령어 캐쉬를 이용하여 두 개의 32-비트 명령어를 가져 온다. 한 개의 32-비트 명령어는 RISC 코아로, 다른 하나는 부동 소숫점 연산 장치로 가서 병렬 수행을 행한다(Dual Instruction Mode). 부동 소숫점 연산은 덧셈과 곱셈 연산 및 정수 연산을 포함하여 각 사이클마다 세 개의 연산을 구현하는데, 이는 세 단계로 파이프라인화된 덧셈과 곱셈 장치에 의한 것이다 (Dual Operation Mode). 하드웨어 벡터 명령어의 복잡한 제어 로직이 없이도, 이러한 미세한 병렬성을 갖는 하드웨어 벡터연산을 이용하여, 고전적인 벡터 연산을 지원한다[1, 2, 3, 4, 5, 6, 10]. i860 CPU의 하드웨어 구조는 그림 1.에 있다.

2) Alliant FX/2812의 구조

이 시스템의 구조는 2개의 수퍼 계산용 프로세서(SCE)와 2개의 수퍼 대화형 프로세서(SIP)로 구성된 프로세서 모듈로 이뤄져 있다. 이들 2개의 수퍼 계산용 프로세서와 2개의 수퍼 대화형 프로세서는 기능적으로 완전히 독립된 프로세서들로서 위에서 설명한 인텔의 RISC i860 표준 프로세서를 이용하고 있다. CE는 벡터와 부동 소숫점연산을 통한 고성능의 계산 능력(병렬처리 기능도 포함)을 지원하며, IP는 하드웨어 장치 및 비계산형 응용 지원(예, 그래픽

처리)을 한다. 각각의 CE는 동적으로 동시-제어버스를 이용할 수 있다. 이들은 또한 병렬 수행을 위하여 융통성있는 클라스터 형태를 구성하는데, 최대 14개의 CE에서 2개의 CE까지 가능하며, 서로 배타적인 CE로 이뤄진 6개까지의 분리된 병렬 클라스터도 동시 수행이 가능하다. 그림 2.는 FX/2800의 구조를 나타내고 있다.

케쉬는 케쉬 모듈(CM)당 512KB로서 최대 8개의 CM으로 4MB까지 가능하며, 주기억장치의 모듈(MEM)은 모듈당 64MB로서 최대 16개의 MEM으로 1GB의 실기억용량을 가지며, 4GB의 가상적인 기억용량을 갖는다. 이 기억 모듈은 16개로 분리된 형태를 취하며, 또한 두 갈래의 포트를 가진다. 입출력 모듈의 SCE는 프로세서 모듈의 SCE와 완전한 호환성을 유지하는데, 주로 입출력 인터럽트 처리용으로 이용된다[9].

Concentrix 운영 체제는 UNIX(버클리 버젼 4.3)를 기본으로하여 병렬 초고속 계산과 실시간 처리를 지원하기 위한 기능을 가지며, 6개 까지의 병렬 클라스터를 자동적으로 처리한다. 또한 오픈 시스템 네트워킹(예, NFS, NCS, NQS, X11)을 지원한다[13,14].

이 시스템이 지원하는 언어는 Fortran, C, Ada로서, 단일 i860 프로세서가 제공하는 파이프라이닝(이중 연산 모드: Dual Operation Mode)과 명령어 수준의 병렬성(이중 명령어 모드: Dual Instruction Mode)을 근간으로 하여, 다중 프로세서를 통한 루프와 테스크 수준의 병렬성(Loop and Task Level Parallelism)을 구현한다[9].

Alliant FX/2812 Fortran 컴파일러는 코드에 존재하는 루프 수준의 병렬성(일명, Alliant 병렬성이라 함.)을 찾아내는 최적화 Fortran 컴파일러이다. 이 기법을 이용하여 시스템은 다시 프로그램을 작성함이 없이 적용되는 CE의 수에 따라 응용 코드의 실행 속도를 증가시킨다. 이 자동 병렬화 기법은, 쉽게 찾을 수 없는 동시 호출 부프로그램이나 루프의 변형등을 처리하기 위하여 사용자 명령어를 첨가함으로써 확장할 수 있다.[7,8,11,12]

III. 벡터화 및 동시성 (Vectorization and Concurrency)

FX/Fortran-2800은 벡터화 및 동시성을 위해 배열 연산, 반복적인 do 루프 및 If 루프 연산을 최적화한다. 수행 형태를 분류해 보면, 스칼라(Scalar), 동시성(Concurrent), 벡터(Vector), 벡터-동시성(Vector-Concurrent), 그리고 외부-동시성 내부-벡터(Concurrent-outer vector-inner, COVI)로 나눌 수 있다.[7]

1) 스칼라(Scalar)

모든 연산은 순차적으로 이뤄진다. 한 개의 연산을 수행하는 데 걸리는 시간은 부분적인 연산시간과 오버헤드를 합한 양이다. 예를 들어, A(I) = A(I) + S 연산은 스칼라 모드로 다음과 같이 이뤄진다.

```
A(1) = A(1) + S
A(2) = A(2) + S
.....
A(N) = A(N) + S
```

2) 동시성(Concurrent)

모든 연산은 여러 개의 프로세서에서 동시성을 갖고 처리된다. 사용되는 프로세서의 수는 수행시간에 결정되는 데, 한 개에서 프로그램의 수행에 따라 시스템이 제공하는 최대 갯수까지 다양하다. 한 개의 연산을 수행하는 데 걸리는 시간은 부분적인 연산의 총 수를 프로세서의 갯수로 나눈

뒤, 한 개의 부분적인 연산에 필요한 시간을 곱하고 오버헤드를 합한 양이다. 예를 들어, A(I) = A(I) + S 연산은 동시성 모드로 다음과 같이 이뤄진다(4개의 프로세서인 경우).

processor 0	*processor 1*	*processor 2*	*processor 3*
A(1) = A(1) + S	A(2) = A(2) + S	A(3) = A(3) + S	A(4) = A(4) + S
A(5) = A(5) + S	A(6) = A(6) + S	A(7) = A(7) + S	A(8) = A(8) + S
.....			
A(N-3) = A(N-3)+S	A(N-2) = A(N-2)+S	A(N-1) = A(N-1)+S	A(N) = A(N) + S

3) 벡터(Vector)

연산이 한 프로세서의 기계 명령어 수준으로 중첩되어 있다. 한 개의 벡터 연산에서 처리되는 부분적인 연산의 갯수는 프로세서 타입과 다른 요인으로 결정된다. FX/2800과 같은 명령어 수준의 파이프라이닝을 갖는 기계에서는 적정 연산의 수는 그 프로세서의 케쉬의 크기에 달려 있다. 스칼라 명령어에 대한 벡터 연산의 속도는 FX/2800의 경우 대략 1.5배에서 최고 3.0배까지 증가한다. 예를 들어, A(I) = A(I) + S 연산은 벡터 모드로 다음과 같이 이뤄진다.

Time A(0) = A(0) + S A(1) = A(1) + S A(2) = A(2) + S A(3) = A(3) + S…

Cycle 1	load A(0)				
Cycle 2	add A(0)	load A(1)			
Cycle 3	add A(0)	add A(1)	load A(2)		
Cycle 4	add A(0)	add A(1)	add A(2)	load A(3)	…
Cycle 5	store A(0)	add A(1)	add A(2)	add A(3)	…
Cycle 6		store A(1)	add A(2)	add A(3)	…
Cycle 7			store A(2)	add A(3)	…
Cycle 8				store A(3)	…
.....					

4) 벡터-동시성(Vector-concurrent)

연산이 여러 개의 프로세서에서 그룹을 이루어 동시성을 띠고 수행되고, 각각의 그룹은 벡터 연산으로 수행된다. 프로세서의 수로 나눠지는 부분적인 연산의 수가 벡터 연산의 적정 크기보다 같거나 작다면, 연산을 수행하는 데 필요한 총시간은 한 개의 벡터 연산 시간에 오버헤드를 더한 양이다. 연산이 하나의 벡터 연산으로 수행될 수 없다면, 전체 수행시간은 프로세서의 수로 나눠진 벡터 연산의 수에 하나의 벡터 연산을 수행하는 데 걸린 시간을 곱한 값에 오버헤드를 더한 것이다.

5) 외부-동시성 내부-벡터(Concurrent-outer vector-inner; COVI)

중첩된 루프에서의 연산은 외부 루프는 동시성 형태로, 내부 루프는 벡터 형태로

수행이 이뤄진다. 루프의 반복 횟수가 알려지면 필요할 경우 루프를 재배치하여, 적은 반복 횟수의 루프는 내부 루프로, 많은 반복 횟수의 루프는 외부 루프로 처리한다. 각각의 내부 루프를 하나의 벡터 연산으로 처리할 수 있다면, 총 수행시간은 프로세서의 수로 나눠진 외부 루프의 수에 하나의 벡터 연산을 수행하는 데 걸린 시간을 곱한 값에 오버헤드를 더한 것이다.

IV. 실험 및 결과

실험에서 사용한 기본 프로그램은 Babb [12]이 저술한 책에서 인용하였는데, π 를 수치해석에 의한 구분 구적법으로 근사치 계산하는 프로그램이다. 그림 2.은 이 프로그램의 순차적인 버젼이며, 그림 3.은 병렬처리에 의한 버젼이다. π 를 사각형 법칙에 의한 수치적인 방법으로 계산하면 다음의 식으로 유도된다.

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx \sim \frac{1}{N}\sum_{j=1}^{N}\frac{4}{1+x_j^2}$$

여기서,

$$x_j = (j - 1/2)/N$$

는 j 구간의 중간점을 나타낸다. 그림 3.의 기본 프로그램은 사실상 모든 계산이 단일 루프로 처리된다.

FX/2812 시스템은, Fortran 라이브러리 루틴을 이용하여 각 프로그램마다 소요된 시간을 측정 목적으로 제공한다. 그 중의 etime Fortran 라이브러리 루틴은 프로세스 기준의 경과된 cpu 시간을 초 단위로 알려준다[7,14]. 그림 3.와 4.의 프로그램에서 etime Fortran 라이브러리 루틴을 이용하여 소요 시간을 계산하였다. 그림 4.의 프로그램은, 구분 구적법 계산을 프로세서 갯수의 단위로 나누어 처리하는 부프로그램을 갖고 있다. 동시성 및 벡터화에 최적화된 루프는 부프로그램 간의 데이타 종속 관계를 찾을 수 없으므로, 부프로그램을 부를 수 없다. 이 경우에는 명확하게 cvd$ cncall 명령어를 코드에 삽입하여 병렬 수행이 가능하게 한다.

그림 5.에서 보는 바와 같이, 네 가지 방법, 즉 스칼라, 벡터, 동시성, 그리고 벡터-동시성 모드로 최적화하여, 구분 구적법의 간격 수에 따른 경과 시간 및 오차를 나타 내었다. 그림 5.-(a)의 기본 프로그램 결과에서 살펴 보면, 구분 구적법의 간격 수가 100과 1000의 두 경우 모두 동시성 모드가 적합하며, 그림 5.-(b)의 변형된 프로그램 결과에서 살펴 보면, 구분 구적법의 간격 수가 100일 경우에는 벡터 모드가, 1000일 경우에는 동시성 모드가 적합한 것을 알 수 있다. 다만 오차는 구분 구적법의 간격 수가 100과 1000의 두 경우 각각에 대해서 일정하며, 1000의 경우가 더 정확함을 쉽게 찾을 수 있다. 단일 루프나 배열 연산의 경우, 다중 프로세서에서의 동시성 모드가 가장 뛰어남을 알 수 있다. 또한 부프로그램 간의 데이타 종속 관계로 부프로그램을 부를 수 없는 경우에는 cvd$ cncall 사용자 명령어를 코드에 삽입하여 동시성 모드로 병렬 수행하는 것이 바람직하다.

V. 결 론

i860 마이크로 프로세서로 이뤄진 미니 수퍼컴퓨터, Alliant FX/2812에서 프로그램 코드로 부터 최적화된 병렬성을 구현하는 방법을 살펴본 뒤, π 를 수치해석에 의한 구분 구적법으로 근사치 계산하는 프로그램을 실험한 결과로 부터 벡터 및 동시성 개념의 효율성을 인지하였다.

VI. 참고 문헌

1. Tekla S. Perry, "Intel's secret is out", IEEE SPECTRUM APRIL 1989 22-28.
2. Frank Hayes, "Intel's Cray-on-a-Chip", MAY 1989 BYTE 113-114.
3. Les Kohn, Neal Margulis, "Introducing the Intel i860 64-Bit Microprocessor", IEEE MICRO August 1989 15-30.
4. Leslie Kohn and Neal Margulis, "THE i860-BIT SUPERCOMPUTING MICROPROCESSOR", Proceedings SUPERCOMPUTING '8, November, 1989, 450-456.
5. Neal Margulis, "THE INTEL 80860 Superscalar architectures bring a new level of performance to microprocessors", DECEMBER 1989 BYTE 333-340.
6. Stephen S. Fried, "Personal Supercomputing with the Intel i860", JANUARY 1991 BYTE 347-358.
7. FX/FORTRAN-2800 Programmer's Handbook, ALLIANT COMPUTER SYSTEMS CORPORATION, JUNE, 1990.
8. FX/FORTRAN-2800 Language Manual, ALLIANT COMPUTER SYSTEMS CORPORATION, JUNE, 1990.
9. FX/2800 Technical Infromation, ALLIANT COMPUTER SYSTEMS CORPORATION, January 1990.
10. COMPUTER ARCHITECTURE A QUANTITATIVE APPROACH, JONNL HENNESSY & DAVID PATTERSON, MORGAN KAUFMANN PUBLISHERS INC. 1990.
11. Alan H. Karp, Robert G. Babb II, "A Comparison of 12 Parallel Fortran Dialects, IEEE Software, September 1988, pp. 52-67.
12. Programming Parallel Processors, R.G. Babb II, ed., Addison-Wesley, Reading, Mass., 1988.
13. Jack A. Test, Mat Myszewski, Richard C. Swift, "The Alliant FX/Series: A Language Driven Architecture for Parallel Processing of Dusty Deck Fortran", PARLE Parallel Architecture and Languages Europe, LNCS 258, Netherlands Vol. I : Parallel Architectures June 15-19, 1987.
14. Walid Abu-Sufah and Allen D. Malony, "VECTOR PROCESSING ON THE ALLIANT FX/8 MULTIPROCESSOR", 1986 International Conference on Parallel Processing, 1986, pp. 559-566.

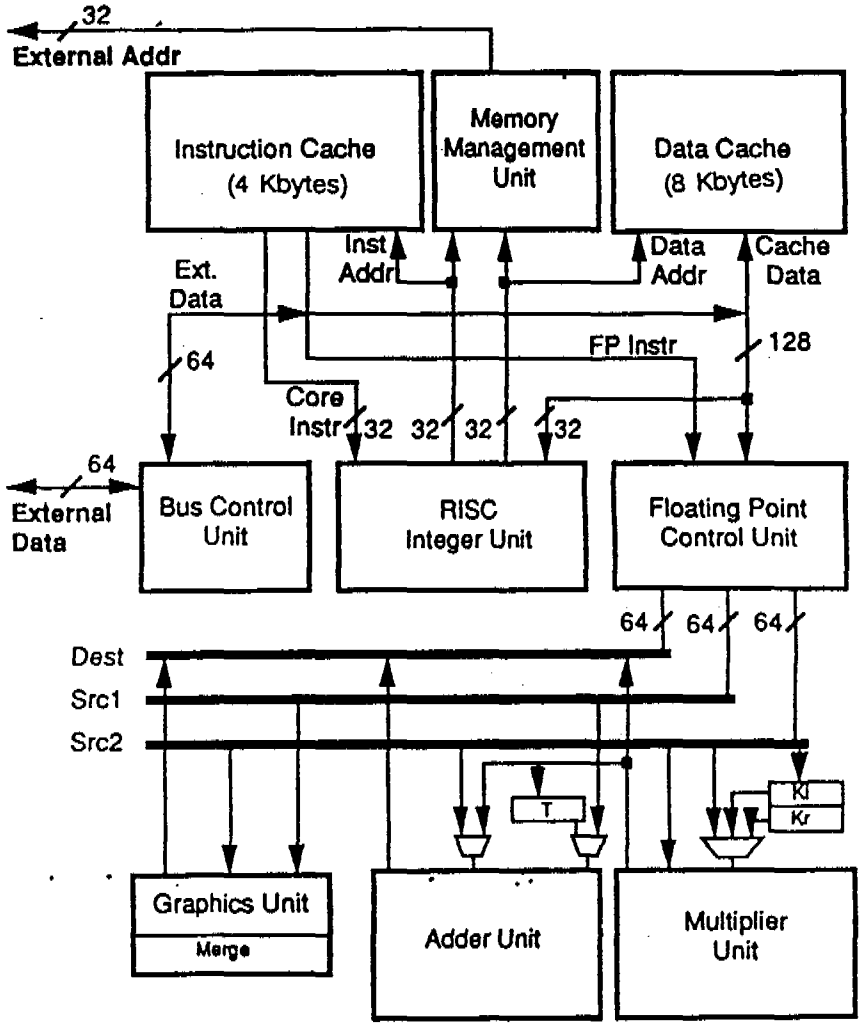

그림 1. i860 마이크로 프로세서의 기능 장치 및 데이타 패스

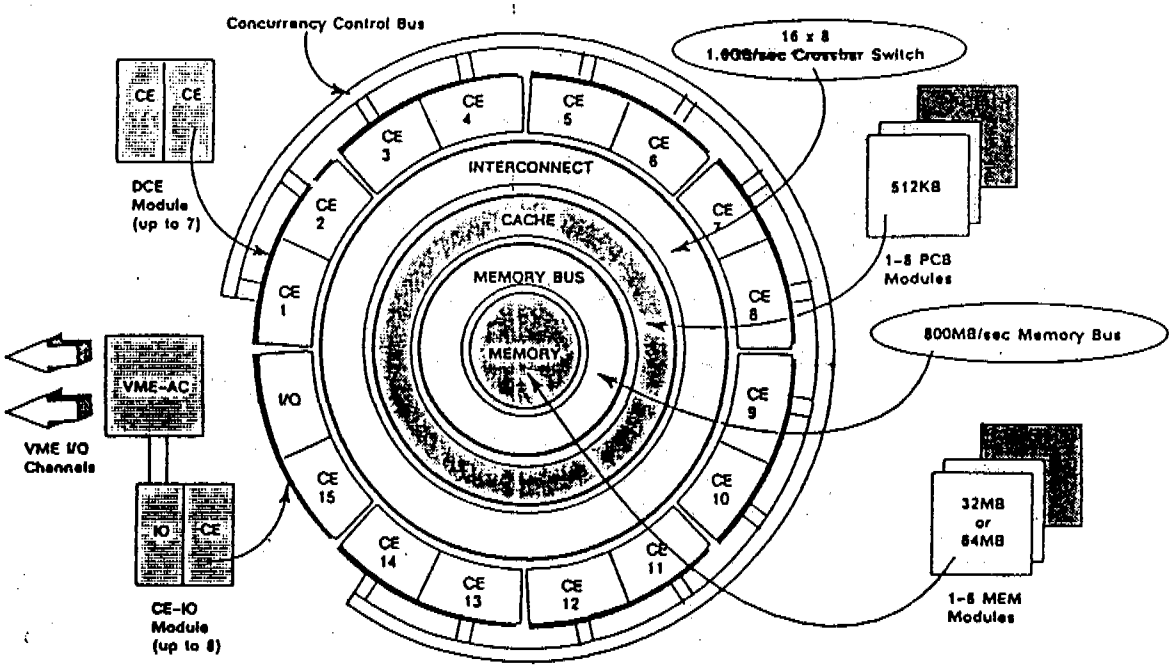

그림 2. FX/2800 구조

```
C - - Pi - Program loops over slices in interval, summing
C - -  area of each slice
C
      real tt1(2), tt2(2)
      integer*4 intrvls, cut
      double precision sumall, width, f, x
C
      f(x) = 4d0 / (1d0 + x * x)
C
      read(*,*) intrvls
      t2 = etime(tt1)
C
C - - Computer width of cuts
C
      width = 1.d0 / intrvls
      sumall = 0.0d0
C
C - - Loop over interval, summing areas.
```

```
C
      do 100 cut = 1, intrvls
        sumall = sumall + width * f((cut - .5D0) * width)
 100  continue
C
C - - Finish overall timing and write results
C
      t1 = etime(tt2)
      write(6, *) 'Time Interval =', 1.0 / float(intrvls),' seconds'
      write(6, *) 'Time in main =', t1 - t2,', sum =', sumall
      write(6, *) 'Error = ', sumall - 3.14159265358979323846d0
      stop
      end
```

그림 3. 기본 프로그램

```
C
C - - Main - This program starts the workers and writes out the
C      final answer as well as the times for all workers
C
      double precision sumall, time(50)
      real tt1(2), tt2(2)
      integer*4 prcnum, nprocs, intrvls
      common /comm/ sumall, time
C
      read(*,*) nprocs, intrvls
      t2 = etime(tt1)
C
C - - Call subroutine concurrently to do work
C
CVD$L CNCALL
      do 100 prcnum = 1, nprocs
        call work(prcnum, intrvls, nprocs)
 100  continue
      t1 = etime(tt2)
      write(6, *) 'Time Interval =' , 1.00 / float(intrvls),'seconds'
      write(6, *) 'Time in main =', t1 - t2,', sum =', sumall
      write(6, *) 'Error = ', sumall - 3.14159265358979323846d0
      do 200 i = 1, nprocs
        write(6, *) 'Process ', i, ' Time = ', time(i)
```

```
 200  continue
      stop
      end
      subroutine work(idproc, intrvls, nprocs)
CVD$R NOCONCUR
c
c - Computes integral for every nprocs-th slice, using a
c - rectangular approximation.  Number of slices is passed
c - to routine as integer * 4 message.
c
      Integer * 4 cut, intrvls, idproc, nprocs
      Real dummy(2)
      Double precision sum, sumall, width, f, x, time(50)
      common /comm/sumall, time
      common /synch/ lock
      f(x) = 4d0 / (1d0 + x * x)
c
      t1 = etime(dummy)
c
c - - Get number of cuts and compute width of cuts
c
      width = 1.d0 / intrvls
c
c - - Calculate area in every "nprcss" cuts and sum
c - - (This is the WORK part)
c
      sum = 0.0d0
      do 100 cut = idproc, intrvls, nprocs
         sum = sum + width * f((cut - .5d0) * width)
 100  continue
      time(idproc) = etime(dummy) - t1
c
c - - - Return answer to the base node
c
cvd$1 sync
      sumall = sumall + sum
      return
      stop
      end
```

그림 4. 변형된 프로그램

구분 구적법 구간의 간격 수(사각형 갯수)				
방	100		1000	
법	경과 시간	오 차	경과 시간	오 차
1	1.3799965E-03	8.333333330057258E-006	1.0259997E-02	8.333333756382899E-008
2	5.6999922E-04	8.333333330057258E-006	2.1799989E-03	8.333333756382899E-008
3	5.6000054E-04	8.333333330057258E-006	2.1499991E-03	8.333333756382899E-008
4	5.6999922E-04	8.333333330057258E-006	2.1810010E-03	8.333333756382899E-008

(a) 그림 3.의 기본 프로그램의 결과

구분 구적법 구간의 간격 수(사각형 갯수)				
방	100		1000	
법	경과 시간	오 차	경과 시간	오 차
1	8.9589953E-03	8.333333331389525E-006	1.8309999E-02	8.333333356702610E-008
2	7.9400018E-03	8.333333331389525E-006	9.6599981E-03	8.333333356702610E-008
3	8.4600002E-03	8.333333331389525E-006	8.1899948E-03	8.333333356702610E-008
4	8.3099976E-03	8.333333331389525E-006	8.2000010E-03	8.333333356702610E-008

(b) 그림 4.의 변형된 프로그램의 결과

방법 1 : 스칼라(Scalar) 모드
방법 2 : 벡터(Vector) 모드
방법 3 : 동시성(Concurrent) 모드
방법 4 : 벡터-동시성(Vector-concurrent) 모드

그림 5. 결 과