벡타 연산을 효율적으로 수행하기 위한 다중 스레드 구조

윤 성 대† 정 기 동††

요 약

본 논문에서는 벡타연산을 효율적으로 수행하고,대단위 병렬시스템을 지원하는 다중 스레드구조, MULVEC (MULtithreaded architecture for the VEctor Computations)을 제시한다. MULVEC은 데이타플로우 모델에 수퍼 스칼라 RISC 마이크로 프로세서를 갖는 기존의 폰 노이만 모델을 도입하였다. 그리고 동일한 스레드 세그멘트내에 벡타연산이 반복되는 경우에 상태필드를 이용하여 동기화의 수를 감축시켰으며, 이에 의해 문맥전환 횟수,통신량 등을 감소시켰다. 그리고 노드 수의 변화에 대한 MULVEC의 성능평가(프로그램들의 수행시간, 프로세서들의 이용율)와 *T의 성능평가(프로그램의 수행시간)를 SPARC station 20 (super scalar RISC microprocessor)에서 시뮬레이션을 하였으며, 노드의 수,루프의 반복횟수 등에 따라 프로그램의 수행시간이 MULVEC이 *T보다 약 1-2배 정도 빠르다는 것을 알 수 있었다.

A Multithreaded Architecture for the Efficient Execution of Vector Computations

Sung Dae Youn† Ki Dong Chung††

ABSTRACT

This paper presents a design of a high performance MULVEC (MULtithreaded architecture for the VEctor Computations), as a building block of massively parallel processing systems. The MULVEC comes from the synthesis of the dataflow model and the extant super sclar RISC microprocessor.The MULVEC reduces, using status fields, the number of synchronizations in the case of repeated vector computations within the same thread segment, and also reduces the amount of the context switching, network traffic, etc. After benchmark programs are simulated on the SPARC station 20(super scalar RISC microprocessor),the performance(execution time of programs and the utilization of processors) of MULVEC and the performance(execution time of a program) of *T according the different numbers of node are analyzed. We observed that the execution time of the program in MULVEC is faster than that in *T about 1-2 times according the number of nodes and the number of the repetitions of the loop.

1. 서 론

Signal processing,Image processing,Scientific computation등에서 연산속도가 TIPS(Trillions of Instructions Per Second)의 수준을 실현하기 위한 컴퓨터시스템은 단일 수퍼컴퓨터 시스템보다는 처리기들을 상호 연결한 다중 프로세서의 병렬처리 시스템이 예상된다. 최근의 폰 노이만 구조는 파이프라인에의해 명령어들을 중첩하여 수행시키므로 단일 스레드 처리시에는 매우 효율적이지만 단일 명령어 흐름으로 부터 병렬성을 이용하고자 하기 때문에 병렬 처리시에는 성능 향상의 한계가 있다. VLSI 의 기술발전에의해 RISC프로세서의 한계를 극복하고 한 사이클에 두 개이상의 명령어를 수행하도록 한 고성능 마

† 정 회 원 : 부산공업대학교 전자계산학과 교수
†† 준 회 원 : 부산대학교 전자계산학과 교수
논문접수 : 1995년 7월 19일, 심사완료 : 1995년 11월 20일

이크로프로세서(super scalar pipelined RISC architecture)는 전통적인 제어흐름에 의한 계산 모델의 컴퓨터 구조이기 때문에 병렬처리 시스템의 기본노드로 활용하기에는 많은 문제점이 있다. 이에 비하여 데이타흐름 구조는 비동기적 자료 병렬성과 자료 가용성에 따라 수행하므로 명령어 수준의 병렬성을 자연스럽게 이용할 수 있으나 계산의 지역성을 활용하지 못하고, 명령어 단위로 병렬 수행하기 때문에 스레드의 크기가 매우 작아서 스레드간의 문맥전환이 많이 필요한 단점이 있다[1].

최근에는 이들 단점들을 극복하기 위해 데이타 흐름과 제어흐름의 혼합형 시스템의 프로세서에 스레드단위로 스케쥴링하는 대규모 병렬성의 다중스레드 구조에 대한 연구가 활발히 진행되고 있다[2, 3, 4].

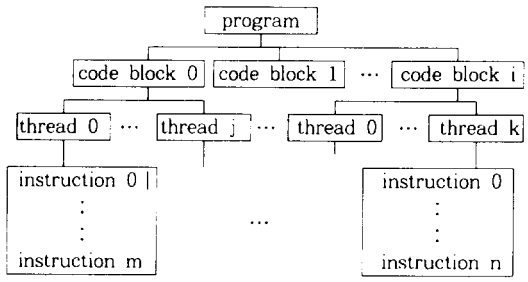
P-RISC구조는 RISC명령어 집합에 fork,join 등의 병렬처리 명령어들을 추가하여 RISC 마이크로 프로세서에서 다중 스레드의 처리가 가능하도록 명령어 수준에서 확장하였으나 동기화 처리, 스레드의 수행등을 한 프로세서가 전담하도록 하였기때문에 병렬처리의 효과가 감소되며, 여러 스레드가 파이프라인을 인터리빙(interleaving)하기 때문에 단일 스레드 처리시에 다른 스레드간의 문맥전환에 의해 성능이 감소된다[2].

*T구조는 프로그램의 병렬수행을 위한 동기화 처리와 프로그램의 계산부분에 해당하는 데이타 처리를 각각 다른 프로세서에서 독립적으로 수행하도록하여 P-RISC구조의 단점을 극복하였으나, 동일 프로세서내의 스레드는 큐에서의 대기시간 때문에 순차 처리되어 병렬성을 충분히 활용하지 못하는 단점이 있다[3]. 그리고 비교적 작은 크기를 갖는 다수의 스레드를 처리하는 경우에는 동기화의 부담이 시스템 성능을 저하시킬 수 있다.

이들 혼합형 시스템은 Signal processing, Image processing, Scientific computation 등에 많이 이용되는 벡타연산인 경우에는 수많은 동기화를 처리해야 하므로 효율성이 매우 낮다[2, 3].

본 논문에서는 기존의 혼합형 컴퓨터의 구조를 개선하고 동기화 횟수, 문맥전환 횟수, 통신량등을 감소시켜 벡타연산을 효율적으로 수행하기 위한 대단위 병렬처리 시스템의 구조에 관하여 연구하였다. 본 논문에서 제안한 MULVEC (MULtithreaded architecture for the VEctor Computations)의 각 프로세서는 수퍼스칼라 RISC 마이크로프로세서를 활용하고 Remote memory and Vector memory control Processor(RVP)가 상태필드를 이용하여 원격데이타와 벡타데이타를 효율적으로 관리하도록 하여 동기화를 최적화함으로써 데이타 프로세서의 활용도를 높이도록 한다. 이와같은 MULVEC의 특징을 요약하면 다음과 같다.

(1) 프로그램은 (그림 1)과 같이 코드블럭,스레드, 단위명령어의 순서로 계층적으로 구성되며 기억장소는 코드블럭단위로 할당된다.



(그림 1) 프로그램 구조
(Fig. 1) Program structure

(2) 데이타 흐름에의한 점화규칙은 스레드단위이며, 스레드내의 명령어의 실행은 제어흐름에 의한다.

(3) 명령어간의 동기화는 카운타레지스타들을 이용하고, 스레드간의 동기화는 프레임 기억장소를 이용한다.

(4) RVP 에서는 원격 스칼라데이타,원격 벡타데이타,해당 노드 자체의 벡타데이타와 각 항목의 상태필드를 관리하도록 한다.

(5) 동기화 프로세서의 입력 우선순위는 message queue와 vector wait queue 순이며 최초로 동기화를 원하는 데이타는 먼저 동기화를 하도록 한다.

(6) 벡타연산인 경우에 데이타 프로세서에서는 레지스타를 이용하여, RVP에서 형성된 상

태 필드에 의해 다음 데이타의 유무를 알 수 있으므로, 다음 데이타가 존재하는 경우에는 동기화의 과정이 없이 연산을 하도록 하고, 다음 데이타가 존재하지 않는 경우에는 vector wait queue를 이용하여 다시 동기화를 하도록 함으로써 동기화 횟수, 문맥전환 횟수, 네트웨크 통신량등을 감소시킨다.

(7) 프레임 기억장소는 스레드 세그먼트 단위로 heap 구조를 형성하며 명령어, 스칼라데이타, 벡타데이타 부분으로 나눈다.

2. MULVEC의 구조

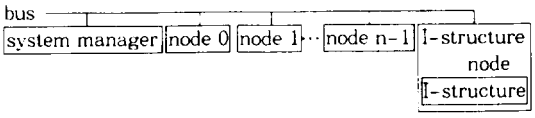
2.1 MULVEC에서 연산의 정의

일항연산에서는 토큰 값이 스칼라인 경우에는 스칼라연산, 벡타인 경우에는 벡타연산이라 정의하고, 이항연산에서는 연산 명령어의 입력인 두 개의 토큰 값이 모두 스칼라인 경우에는 스칼라 연산이라 정의하고,스칼라와 벡타인 경우에는 스칼라를 원소가 하나인 벡타화로하여 벡타연산이라 정의하고, 모두 벡타인 경우에는 벡타연산이라 정의한다.

2.2 MULVEC 시스템의 구조

시스템은 (그림 2)와 같은 방식으로 구성되며, 이 시스템 전체를 관리하는 system manager 노드와 n개의 처리기 노드와 non-strict한 자료구조와 동기화를 효율적으로 처리할 수 있는 I-structure 노드로 구성된다[6].

n개의 처리기 노드 구조는 각각 (그림 3)과 같이 3개의 장치로 구성되며,



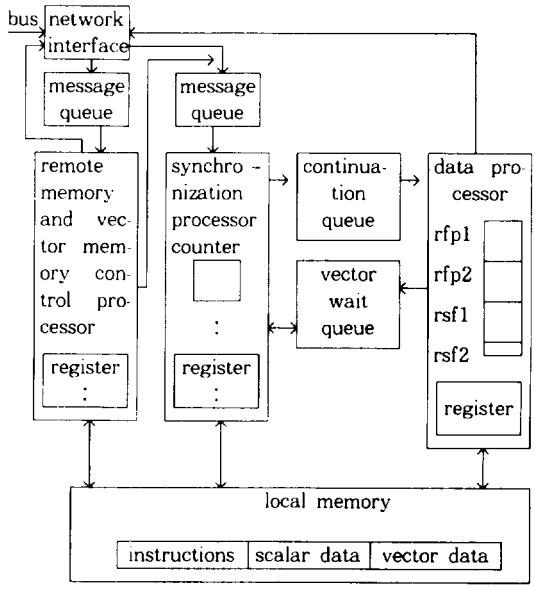
(그림 2) MULVEC system의 구조
(Fig. 2) Architecture of MULVEC system

각 장치간의 정보교환은 4개의 큐와 1개의 국부 기억장소로 이루어 지고,병렬 처리를 위해 각 프로세서들은 비동기적으로 처리되며,각 프로세서는 계층적 파이프라인 구조를 갖는다.그리고 각 프로세서들의 특징은 다음과 같다.

(1) RVP(Remote memory and Vector memory control Processor)는 기억장소를 할당하고 원격 스칼라데이타, 원격 벡타데이타, 노드 자신의 벡타데이타들을 전송 혹은 기억시키고,

(2) SP(Synchronization Processor)는 message queue 혹은 vector wait queue에서 메세지의 입력을 받아 동기화를 체크하여 동기화가 성공하면 continuation queue로 메세지를 전송하고, 동기화가 실패하면 vector wait queue로 메세지를 전송하며,

(3) DP(Data Processor)는 continuation queue의 메세지를 입력받아 연산을 시작하고, 벡타연산인 경우에는 상태필드를 체크하여 다음 데이타의 동기화가 필요없으면 벡타연산을 계속 수행하고, 동기화가 필요하면 동기화 메세지를 형성하여 vector wait queue로 메세지를 전송한다.



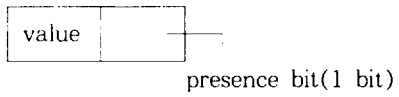
register : general register
rfp1,rfp2 : frame pointer register
rsf1,rsf2 : status field register

(그림 3) MULVEC의 노드구조
(Fig. 3) Node architecture of MULVEC system

2.3 프레임 기억장소의 구조

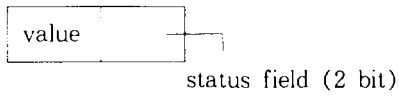
순차 프로그램에서 프레임은 스택방식으로 할당되는 반면에 병렬 프로그램에서는 여러 프레임이 동시에 할당될수 있으므로 프레임들은 트리구조를 형성한다[6].

MULVEC에서는, 프레임은 스레드 세그먼트에 대응하여 할당되며, 각 프레임은 명령어, 스칼라 데이타, 벡타데이타로 구분하여 국부기억장소에 저장되고 각 프레임의 크기는 컴파일시에 할당되어 heap구조를 형성한다. 스칼라데이타의 형태는 (그림 4)와 같으며, 존재 비트는 데이타 유무에 따라 0이나 1이 된다.



(그림 4) 스칼라데이타 형태
(Fig. 4) Format of scalar data

벡타데이타의 형태는 (그림 5)와 같으며, 상태필드는 〈표 1〉과 같이 현재 데이타와 다음 데이타의 관계를 나타낸다.



〈그림 5〉 벡타데이타 형태
(Fig. 5) Format of vector data

〈표 1〉 상태 필드
〈Table 1〉 Status field

null	initialize
0 0	present data is absent, and next data is absent
0 1	present data is existent, but next data is absent
1 0	present data is existent, and next data is existent
1 1	present data is existent (last data)

2.4 RVP (Remote memory and Vector memory control Processor)

컴파일시에 스레드 세그먼트의 프레임 기억장소(값과 존재비트나 상태필드)의 크기를 알 수 있으므로 스레드의 할당시에 기억장소를 미리 확보하고, 초기화를 한다. 원격 벡타데이타의 load, store는 split phase를 이용하므로 프로세서가 원격 데이타의 요청후에 데이타의 도착을 기다리지 않고 다른 일을 처리한다[7]. Revload와 revstore 메세지의 정의와 그 메세지를 전송받은 노드의 수행알고리즘은 다음과 같다.

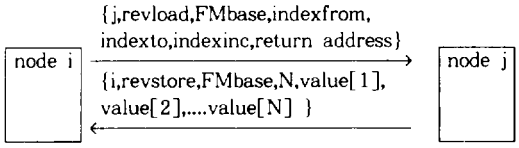
```
revload  =  {node number,revload,FPbase,inde-xfrom,
            indexto,indexinc,return address}
  loop
     /*** start a block ***/
     i=1  /*** first record of a block ***/
  if indexfrom >indexto then exit/*** end of all
                                   blocks ***/
    loop /*** concatnate N records ***/
  if i>N or indexfrom>indexto then exit /*** N is
                blocking factor, end of a block ***/
  FM[i].value=value[indexfrom]
  indexfrom=indexfrom+indexinc
  i=i+1
  endloop /*** end a concatnation of records ***/
 format revstore message
 save revstore message into the message queue
 /*** end a block ***/
 endloop

revstore  =  {node number,revstore,FPbase,N,value[1],
             value[2],....,value[N] }
   i=0
   loop
   i=i+1
   save the value[i] to the FM.value
   if FM[1].statusfield=null
     then do /*** processing first entry of table ***/
       make and save FM.statusfield
       format sync message
       save sync message into the message queue
       exit
       enddo
     else do /*** processing all entry except first of
                                          table ***/
       make and save to the FM.status field
       if i=N then exit
       enddo
 endloop
```

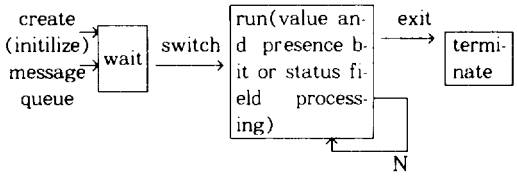
그리고 노드 자신의 데이타를 load, store하는 경우에는 노드번호를 자신의 번호로 부여하고, revload인 경우에는 요청되는 노드의 프레임 시작주소(FPbase), 인덱스 초기치, 증가분, 최종치, 반환되어 저장될 주소를 전송하고, revstore인 경우에는 전송되어 저장될 주소 (FPbase)와 값들이 전송되어 벡타데이타가 첫번째 인덱스이

면 값과 존재비트나 상태필드를 저장한후에 동기화 프로세서로 {sync,Ip,Fp,value}메세지를 전송하고,그외의 경우에는 값과 존재비트나 상태필드를 N(전송되는 값의 수)번 저장한다.

노드 i와 노드 j의 원격 데이타의 요청과 응답은 (그림 6)과 같이 수행되며, (그림 7)은 RVP의 상태전이도를 나타낸다.



(그림 6) 벡타데이타의 revload,revstore 수행
(Fig. 6) Revload,revstore execution of vector data



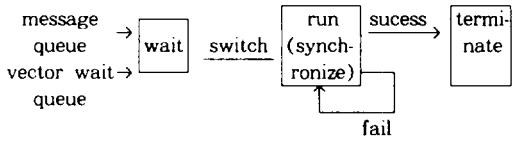
(그림 7) RVP의 상태전이도
(Fig. 7) State transition diagram of RVP

2.5 SP (Synchronization Processor)

동일한 스레드 세그먼트내의 동기화는 카운타를 이용하고, 다른 스레드세그먼트내의 동기화는 프레임기억장소를 이용하고, 입력 우선순위는 message queue와 vector wait queue순이며 입력되는 메세지의 정의와 수행 알고리즘은 다음과 같다.

```
sync = { sync,Ip,Fp,FP or value }
  save contents of sync message to the
  registers in the synchronization processor.
  synchronize using the counter register or
  frame memory.
  if sucess then make the message and
    save into the continuation queue
    else make the message and save into
    the vector wait queue
exit
```

동기화가 성공하는 경우에는 continuation queue로 {Ip, Fp, FP or value} 메세지를 전송하고, 실패하는 경우에는 vector wait queue로 {sync,IP,FP,FP or value} 메세지를 전송하여 message queue가 비게되면 vector wait queue의 메세지를 다시 동기화 한다. (그림 8)은 SP의 상태전이도를 나타낸다.



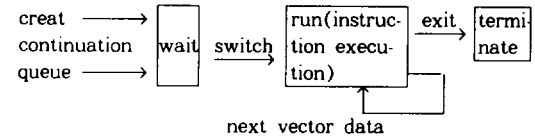
(그림 8) SP의 상태전이도
(Fig. 8) State transition diagram of SP

2.6 DP (Data Processor)

continuation queue에서 {IP, FP, FP or value} 메세지를 입력받아서 연산을 한후에 레지스타를 이용하여 FP1.status, FP2.status가 둘중에 하나라도 00 이나 01 이면 동기화를 위해 sync메세지를 형성후 vector wait queue로 저장하고 이 스레드는 소멸되며, 10,10 이나 11,10이면 동기화를 하지않고 다음 데이타들을 fetch하여 연산하며,11,11이면 모든 연산이 종료된다. 이와같은 과정을 비중단 조건으로 계속 수행하는데,이에대한 명령어의 정의와 수행알고리즘은 다음과 같다.

```
chk={chk,rsf1,rsf2,label}
  if(rsf1=10 and rsf2=10) or (rsf1=11 and rsf2
  =10) or (rsf1=10 and rsf2=11) then exit.
  if rsf1=11 and rsf2=11 then go to label
  make the message { sync,IP,FP,FP or value }
  save this message into the vector wait queue.
  terminate this thread
```

그리고 벡타데이타가 지역성이 있는 경우에는 동기화를 하지않으므로 동기화 횟수, 문맥전환 횟수등을 감소시켜 데이타의 공간적 병렬성을 활용한다.(그림 9)는 DP의 상태전이도를 나타낸다.



(그림 9) 데이타 프로세서의 상태전이도
(Fig. 9) State transition diagram of DP

3. 코드블럭의 예

코드블럭이 (그림 10)과 같이 정의되어 있을 때,

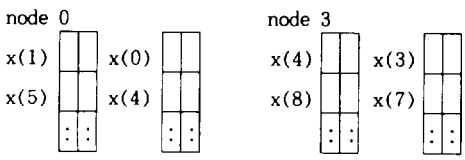
```
do i=1,m
  y(i)=x(i)+x(i-1)
enddo
```

(그림 10) 코드블럭의 예제
(Fig. 10) Example of code block

P-RISC구조나 *T구조는 x의 모든 데이타가 원격지에 존재하더라도, 연산을 위해 x(i)와 x(i-1)의 원격데이타를 한 원소씩 요청하고, 문맥전환을 하여 다른 일을 처리하다가 요청한 원격데이타가 도착하면 큐에 저장한후에 다시 문맥전환을 하여, 동기화를 한후에 연산을 한다. 이 과정을 m번 수행한다. 그러나 MULVEC구조에서는 x의 모든 데이타가 원격지에 존재하면 x(i)와 x(i-1)의 원격데이타들을 전부 요청하고, 문맥전환을 하여 다른 일을 처리하다가 요청한 원격데이타들이 도착하면 첫번째 원소들만 메세지 큐에 저장한후에 다시 문맥전환을 하여,동기화를 한후에 연산을 한다. 그리고 두번째 원소부터는 상태필드와 chk명령어등을 이용하여 동기화와 문맥전환이 없이 계속 연산만을 한후에 이 코드블럭의 수행을 마친다. 그러므로 동기화 횟수의 감소, 문맥 전환의 횟수 감소,통신량의 감소등이 이루어진다.

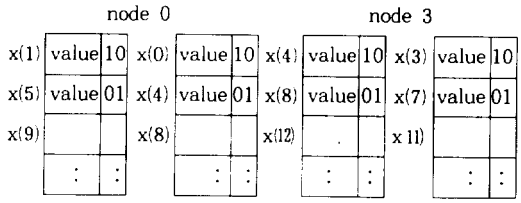
3.1 프레임 기억장소

(그림 10)의 코드블럭을 예로하여, 4-bounded loop경우에대한 컴파일시의 초기 기억장소의 형태는 (그림 11)이 된다.



(그림 11) 요구전의 vector memory format
(Fig. 11) Vector memory format before request

RVP에서 N은 2(전송되는 값의 수)이고 각 노드가 revstore를 한번 수행했을때 벡타데이타가 저장된후의 기억장소 형태는 (그림 12)가 된다.



(그림 12) 저장후의 vector memory format
(Fig. 12) Vector memory format after store

3.2 단일 프로세서

모든 배열 값이 동일 노드의 프레임에 있다고 가정하고 앞의 코드블럭을 예로하면 다음과 같은 단일 프로세서 코드로 수행시킬수 있다.(오프랜드의 r은 레지스타)

```
      load rxifp1,fm[xif];x(i)의 초기항 포인타
      load rxitp,fm[xit];x(i)의 최종항 포인타
      load ryi,fm[yif];y(i)의 초기항 포인타
      sub rxifp2,rxifp1,16;x(i-1)의 초기항 포인타
      cmp ra,rxifp1,rxitp
      jgt ra,out
loop: move rxi1,rxifp1
      move rxi2,rxifp2
      fadd rtmp,rxi1,rxi2;x(i)+x(i-1)
      store fm[yif],rtmp;y(i)의 저장
      add rxifp1,rxifp1,16;포인타 증가
      add rxifp2,rxifp2,16;포인터 증가
      add ryi,ryi,16;포인타 증가
      cmp rb,rxif,rxit
      jle rb,loop
out:                  :
```

3.3 다중 프로세서

모든 배열값이 I-structure에 있다고 가정하고 앞의 코드 블럭을 예로하면 다음과 같은 다중 프로세서 코드로 수행시킬 수 있다.

```
/*  RVP    */
r1: store   fm[rxifp1],rv1     ;x(i)의 초기항 포인타
    sync    l1s
    next
r2: store   fm[xifp2],rv2      ;x(i-1)의 초기항 포인타
    sync l2s
    next
```

```
/*  SP    */
l1s : store fm[xi],rv1
      join c1,2,loop
l2s : store fm[xi1],rv2
      join c1,2,loop
l3s : load rv3,fm[yi]
      join c2,rn,out

/*  DP    */
      load rxif,fm[xif] ;x(i)의 초기항 포인타
      load rxit,fm[xit] ;x(i)의 최종항 포인타
      load ryif,fm[yif] ;y(i)의 초기항 포인타
      cmp ra,rxif,rxit
      jgt ra,out
      revload rxifp1,rxit1,rxic1,r1 ;x(i)의 remote value
      request(index from,to,inc)
      revload  rxifp2,rxit2,rxic2,r2 ;x(i-1)의  remote
      value request(index from,to,inc)
loop : move rxifp1,rxif ;x(i)의 초기항 포인타
      load rxifp2,fm[xif1] ;x(i-1)의 초기항 포인타
      load rxisf1,fm[xifsf] ;x(i)의 상태필드 포인타
      load rxisf2,fm[xifsf1] ;x(i-1)의 상태필드 포인타
ad : fadd rtmp,rxifp1,rxifp2 ;x(i)+x(i-1)
      revstore fm[yi],rtmp,l3s ;y(i)의 저장
      chk rxisf1,rxisf2,out ;상태필드 체크
      add rxifp1,rxifp1,16 ;포인타 증가
      add rxifp2,rxifp2,16 ;포인타 증가
      add rxisf1,rxisf1,16 ;포인타 증가
      add rxisf2,rxisf2,16 ;포인타 증가
      add ryi,ryi,16 ;포인타 증가
      jmp ad
out :          :
```

4. 시뮬레이션및 성능 평가

버스 전송시간은 8 nano second로 가정하고 [9], SPARC station 20 (super scalar RISC microprocessor)에서 어셈블리어로 에뮬레이터 하여 각 명령어와 메세지의 수행시간을 〈표 2〉와 같이 구한 후에 N(전송되는 값의 수)을 변화시켜 〈표 3〉을 구하였으며, 이들에 의해 4.1의 단일 프로세서 코드, 4.2의 다중 프로세서 코드를 SPARC station 20에서 다음과 같은 가정하에 MULVEC을 시뮬레이션 하였는데 이것은 이상적인 경우이다.

〈표 2〉명령어/메세지의 수행시간
〈Table 2〉 Execution time of instructions/messages

nano second

ALU	10	join	36
load(register)	21	chk	126
load(memory)	170	sync	52.5
store(register)	4.5	branch	2
store(memory)	138		

〈표 3〉 revload,revstore의 수행시간
〈Table 3〉 Execution time of revload,revstore

nano second

N	1	2	4	8
revload	1023	2203.5	4131.5	7810.5
revstore	759	1910.5	3781.5	7343.5

1. 버스에서는 다른 노드와의 충돌이없이 데이타를 즉시 전송하고,
2. 지능적인 번역에의해 모든 벡타데이타가 연산전에 I-structure에 저장되어 있어 요청이오면 시간지연이 없이 즉시처리 되고,
3. 데이타가 연속으로 도착하므로 동기화는 한번만 이루어지고,
4. 데이타 프로세서에서는 동기화후 즉시 연산이되고, 다음 데이타가 미리 도착되어 있으므로 비중단 조건으로 계속 연산이 된다.

루프 반복횟수를 128, 16,384로 하였을때, MULVEC에서 N과 노드 수의 증가에 따른 수행시간 (RVP, SP, DP는 비동기적으로 처리)은 〈표 4〉이며, *T에서 노드 수의 증가에 따른 수행시간(동기화처리와 스레드처리를 비동기적으로 처리)은 〈표 4〉이며 (*T에서는 N은 항상 1), 루프 반복횟수가 128일때 *T와 MULVEC의 수행시간에대한 그래프는 (그림 13)이 되며, m은 코드블럭의 루프 반복횟수, n은 노드의 수, N은 revload,revstore에서 전송되는 값의 수를

〈표 4〉 (그림 10)의 수행시간
〈Table 4〉 Execution time of (Fig.10)

m=128 nano second

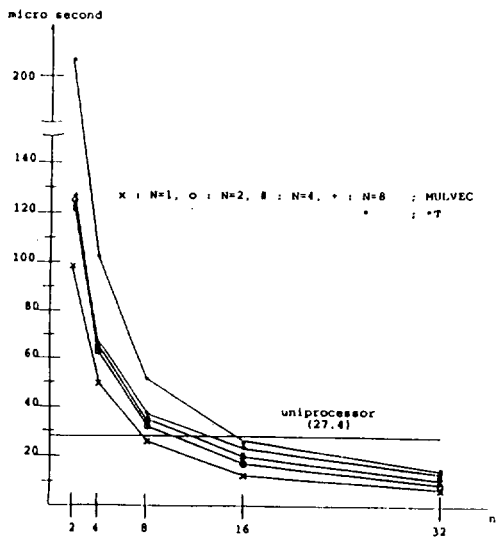
	n=2	n=4	n=8	n=16	n=32
N=1	99,097	50,521	26,233	14,089	8,017
N=2	125,222	64,086	33,518	18,234	10,592
N=4	125,966	65,462	35,210	20,084	12,521
N=8	126,476	67,728	38,354	23,667	12,521
*T	207,744	103,872	51,936	25,968	12,984

m=16,384 nano second

	n=2	n=4	n=8	n=16	n=32
N=1	12,437,401	6,219,673	3,110,809	1,556,377	779,161
N=2	15,653,766	7,828,358	3,915,654	1,959,302	981,126
N=4	15,493,982	7,749,470	3,877,214	1,941,086	973,022
N=8	15,048,468	7,528,724	3,768,852	1,888,916	948,948
*T	26,591,232	13,295,616	6,647,808	3,323,904	1,661,952

의미하고, 단일 프로세서의 수행시간은 m=128일때 27,412 nano second,m=16,384일때 3,801,620 nano second 이다.

〈표 4〉에 의하면 m=128이고 노드의 수가 많을 때는 연산이 각 노드로 분산 처리되어 *T와 MULVEC의 수행시간이 큰 차이가 없으나, m=16,384일때는 노드의 수가 많더라도 *T보다 MULVEC의 수행시간이 빠르다는 것을 알 수 있으며, N=1과 N=2의 수행시간의 차이는 N=1일때에 블럭을 형성하는 시간이 없기때문이며, 〈표 4〉와 (그림 13)에 의하면 m=128일 때 MULVEC의 N의 수가 증가하는데도 수행시간이 증가하는 것은 데이타와 상태필드를 구조화로 만드는데 시간이 필요하였으며, 이들의 증가된 시간은 반복되는 동기화 횟수에 의한 시간, 문맥전환 횟수에 의한 시간, 통신 시간 및 동기화 시간 등의 합보다는 적게 걸릴 것이며, N이 증가함에 따라 통신 교통량도 감소할 것이다. 그리고 m=16,384일 때 N의 수가 증가하면 수행시간은 감소하는데 이것은 데이타와 상태필드를 만드는데 소요되는 시간은 전체 수행시간에 거의 영향이 없다는 것을 알 수 있다. 그러므로 루프의 반복횟수가 많으면 많을수록 MULVEC은 효율적이라는 것을 알 수 있다. m=128,n=32일 때 N=4와 N=8의 수행시간이 같은 것은 연산의 횟수가 같기 때문이다. 〈표 4〉에 의하면 MULVEC에서는 m이 굉장히 클 때는 N의 변화가 전체 수행시간에 약간의 영향을 준다는 것을 알 수 있으며, 노드의 수가 약 12개 이상일 때 단일 프로세서보다는 수행속도가 빠르 다는 것을 알수있다. m=128, N=8일때, 데이타가 불연속적으로 도착하여 동기화가 여러번 발생하는 비이상적인 경우에 대한 (그림 10)의 수행시간은 〈표 4〉이며, s는 동기화 횟수를 의미한다.



(그림 13) m=128 일때 수행시간 그래프
(Fig. 13) Graph of execution time for m=128

〈표 5〉 동기화들을 포함한 (그림 10)의 수행시간
〈Table 5〉 Execution time of (Fig.10) contained synchronizations

m=128, N=8 nano second

	n=2	n=4	n=8	n=16	n=32
s=2	126,529	67,781	38,407	23,720	12,574
s=4	126,634	67,886	38,512	23,825	12,679
s=8	126,844	68,096	38,722	24,035	12,679
s=16	127,264	68,516	39,142	24,035	12,679

동기화가 16번 일어날때, 〈표 5〉의 수행시간이 〈표 4〉의 수행시간보다는 노드의 수에 따라 약 0.6 -2.1 % 가 증가한다는것을 알 수 있다. 그것은 〈표 2〉에서 sync메세지의 수행시간이 52.5 nano second이기 때문이다. 그리고 〈표 5〉에서 MULVEC의 수행시간이 〈표 4〉에서 *T의 수행시간보다 적으므로, 동기화가 여러번 발

〈표 6〉 프로세서 이용율
〈Table 6〉 utilization of processor

m=128, RVP %

	n=2	n=4	n=8	n=16	n=32
N=1	98.0	96.2	92.6	86.2	75.7
N=2	97.6	95.4	91.2	83.8	72.1
N=4	96.1	92.4	85.9	75.3	60.4
N=8	92.9	86.7	76.6	62.1	58.6

m=128, SP %

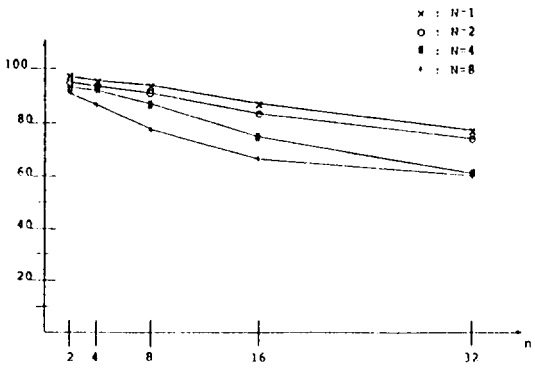
	n=2	n=4	n=8	n=16	n=32
N=1	5.9	6.0	6.1	6.3	6.6
N=2	4.7	4.7	4.8	4.9	5.0
N=4	4.6	4.6	4.5	4.4	4.2
n=8	4.6	4.4	4.2	3.7	4.2

m=128, DP %

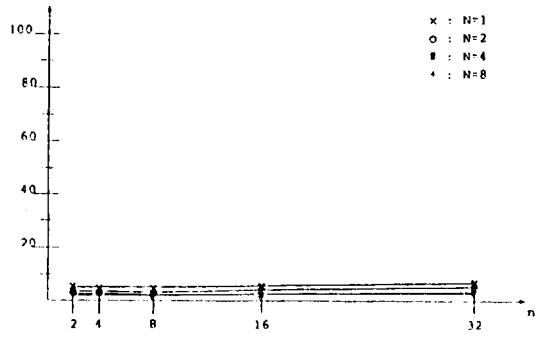
	n=2	n=4	n=8	n=16	n=32
N=1	62.2	61.9	61.5	60.8	59.6
N=2	49.2	48.8	48.2	47.0	45.1
N=4	48.9	47.8	45.2	42.7	38.2
N=8	48.7	46.2	42.1	36.2	38.2

생하는 비이상적인 경우라 하더라도 MULVEC이 효율적이다는 것을 알 수 있다.

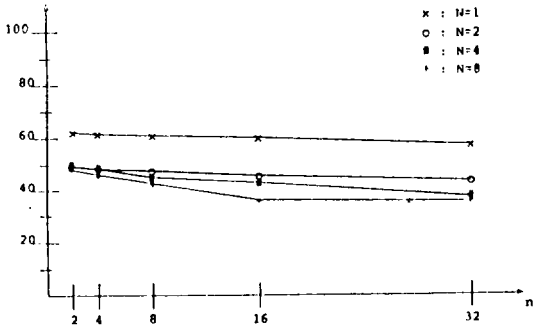
m=128일 때 RVP, SP, DP 이용율은 〈표 5〉이며 그 그래프는 (그림14, 15, 16)인데 RVP의 이용율이 SP, DP의 이용율보다 높은것은 모든 데이타가 원격지에 있으므로 값의 load, store와 상태필드의 형성및 저장에 많은 시간을 소모하기 때문이며, SP의 이용율이 낮은것은 한번의 동기화와 다른 스레드와 동기화를 위해서 y(i) 의 결과를 m번 저장 (revstore)하기 위한 오버헤드



(그림 14) RVP의 이용율 그래프
(Fig. 14) Utilization graph of RVP



(그림 15) SP의 이용율그래프
(Fig. 15) Utilization graph of SP



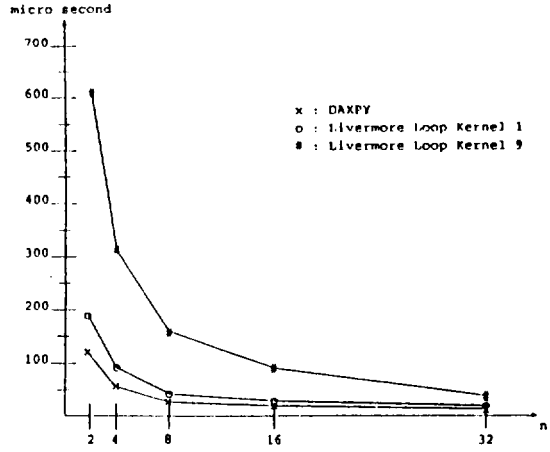
(그림 16) DP의 이용율
(Fig. 16) Utilization graph of DP

```
**********************
***        DAXPY       ***
for i = 1 to m do
        x[i] = a * x[i] + y[i]
**********************
***  Livermore Loop Kernel 1  ***
for i = 1 to m do
x[k]=q+y[k]*(r*z[k+10]+t*z[k+11])
**********************
***  Livermore Loop Kernel 9  ***
for i = 1 to m do
  px[1,i]=dm28*px[13,i]+dm27*px[12,i]+dm26*
          px[11,i]+dm25*px[10,i]+dm24*px[9,
          i]+dm23*px[8,i]+dm22*px[7,i]+co*
          (px[5,i]+px[6,i])+px[3,i]
```

(그림 17) 예제 프로그램
(Fig. 17) Example program

〈표 7〉 (그림 17)의 수행시간
(Table 7) Execution time of (Fig.17) nano second

	n=2	n=4	n=8	n=16	n=32
DAXPY	125,314	64,178	33,610	18,326	10,684
Livermore loop kernel 1	187,085	95,381	49,529	26,603	15,140
Livermore loop kernel 9	619,210	313,530	160,690	84,270	46,060



(그림 18) (그림17)의 수행시간 그래프
(fig. 18) Execution time graph of (Fig.17)

때문이며, DP는 벡타데이타를 비동기적으로 처리하므로 RVP에서의 데이타저장 및 상태필드의 처리에 따른 시간지연에 의한 오버헤드때문에 프로세서 이용율이 약 40-60%이다.

그리고 병렬 수행의 시물레이션에 많이 이용되는 DAXPY, Livermore Loop Kernel 1, Livermore Loop Kernel 9의 프로그램은 (그림 17)이고, (그림 17)의 프로그램에서 m=128, N=2에 대한 프로세서 수에 따른 MULVEC의 수행시간은 〈표 6〉이며 그래프는 (그림 18)이다.

(그림 18)에서 DAXPY는 벡타연산이 다른 두개의 프로그램보다는 작기 때문에 수행시간이 가장 짧고, Livermore Loop Kernel 9가 다른 두개의 프로그램보다 수행시간이 긴것은 연산자가 많을 뿐만 아니라, 그에 따른 자료구조의 접근이 많기 때문이다. 그리고 루프 반복 횟수가 128 보다 많으면 각 프로그램에 대한 수행시간의 차이가 매우 클것으로 예상된다.

5. 결 론

본 논문은 기존의 제어흐름과 데이타흐름의 개념을 혼합한 컴퓨터 구조에서 벡타연산을 수행할 때마다 발생하는 동기화에 의한 오버헤드를 효율적으로 개선한 대단위 병렬처리시스템의 구조에 관한 연구이다.

MULVEC은 RVP, SP, DP로 구성되며, 이들은 수퍼 스칼라 RISC 마이크로프로세서를 활용하도록 하므로 폰 노이만 구조를 사용함과 동시에,벡타데이타는 RVP에서,동기화는 SP에서, 스레드는 DP에서 처리되도록 일을 분담시켜 기존의 다중스레드에대한 벡타데이타 처리, 동기화처리를 데이타 처리와 분리함으로써 데이타 처리기의 과중한 부담을 줄였다. 아울러 벡타데이타는 RVP에서 상태필드의 조절에 의해 동기화의 횟수를 감축시켰고, 이에 의해 DP에서 모든 데이타를 비중단 조건으로 처리되도록 하였으므로 문맥전환 횟수, 통신 시간, 통신 량, 메세지 큐의 길이등을 감소시켰다.

그리고 본 논문에서는 MULVEC의 수행시간, 각 프로세서의 이용율에 대하여 SPARC station 20(super scalar RISC microprocessor)에서 루프의 반복횟수, 노드의 수, 전송되는 값의 수에 따른 변화를 시물레이션하였는데, 노드의 수가 약 12개 이상일때 MULVEC은 단일 프로세서보다 효율적이라는 것을 알 수 있었으며, RVP의 이용율이 SP, DP 보다는 높다는 것을 알 수 있었다. 그리고 코드블럭을 예로하여 *T와 MULVEC의 수행시간을 비교한 결과에 의하면, 루프의 반복횟수가 적고 노드의 수가 많을때는 큰 차이가 없으나, 벡타연산이 많고 루프의 반복횟수가 많으면 매우 효율적이 된다. 그리고 MULVEC은 벡타데이타의 지역성을 활용하였으므로 벡타연산이 많은 대단위 병렬처리에 적합한 구조이고 Signal processing, Image processing, Scientific computation 등에 매우 효율적이다.

앞으로의 연구과제로는 이 시스템에 맞는 시스템경영, 지능적인 번역기법, 효율적인 기억장소 관리법, RVP의 처리속도를 증가시키는 방법등이 남아있다.

참 고 문 헌

[1] G.Papadopoulous and O.E.Culler, "Monsoon : an Explicit Token Store Architecture",In proc. of Ann. Symp.on Computer Architecture,Seatle, WA,pp.82-91, 1990.

[2] R.S.Nikhil and Arvind, "Can Dataflow Subsume Von Neumann Computing?",In proc.of the 16th Ann.Symp.on Computer Architecture,Israel,pp.262-272,1989.

[3] R.S.Nikhil, G.M.Papadopoulous and Arvind, "*T : Multithreaded Massively Parallel Architecture",proc. of the 19th Ann.Symp.on Computer Architecture, Australia, pp.156-167, 1992.

[4] P.C.Treleaven, R.P.Hopkins and P.W. Rautenbach, "Combining Data Flow and Control Flow computing",J.of Computer, Vol.25,no.2,1982.

[5] Arvind, and R.S.Nikhil, "I-Structures : DataStructure for Parallel Computing", Trans. on ACM,Vol.11,no.4,Oct,pp.598-

632,1989.

[6] D.E.Culler,A.Sah, K.E.Schauser, T.V. Eicken, and J.Wawrzynek, "Fine-Grain Parallelism with Minimal Hardware Support : A Compiler-Controlled Thread Abstract Machine", In proc. 4th Intl. Conf.on ASPLOS,Santa Clara,CA,pp.164-175.Apr.,1991.

[7] J.G.Lbic, "Advanced Topics in Dataflow Computing", Prentice Hall, Englewood Cliffs, New Jersey,pp.11-21,1991

[8] R.L.Fink and F.E.Ross, "Following the Fiber Distributed Data Interface", Tran. on IEEE Network,March,pp.50-55,1992.

[9] E.H.Rho, S.H.Ha, S.Y.Han, H.H.kim, and D.J.Hwang, "Compilation of a Functional Languages for the Multithreaded Architecture : DAVRID", '94 Intl conf. on Parallel Processing, pp.2-239-2-242, 1994.

[10] P.R.Fenstermacher, J.E.Hicks, and R.P. Johnson, "ID World Reference Manual", Tech.Report, M.I.T.,Nov,1989

[11] H.Ahned, "An Enhanced Data-Driven Architecture for Efficient Scalar and Vector Processing", Ph.d. of the Royal Institute of Technology,1982

[12] "Sun-4 Assembly Language Reference Manual",Sun co.,May,1988

[14] "System Library Reference for the Sun2 and Sun3",Sun co.,may,1988.

윤 성 대

1980년 경북대학교 졸업(공학사)
1984년 영남대학교 대학원 졸업(공학석사)
1992년 부산대학교 대학원 전자계산학과 박사과정 수료
1986년~현재 부산공업대학교 전자계산학과 부교수
1992년~95년 부산공업대학교 전자계산소 소장
관심분야 : 병렬처리, 계산기 구조임

정 기 동

1973년 서울대학교 졸업(공학사)
1975년 서울대학교 대학원 졸업(공학석사)
1986년 서울대학교 대학원 계산통계학과 졸업(공학박사)
1990년~91년 M.I.T., South Carolina대학 교환교수
1978년~현재 부산대학교 전자계산학과 교수
1993년~현재 부산대학교 정보통신연구소 소장
관심분야 : 병렬처리, 데이타플로우, 멀티미디어임