# 범용 병렬화일 시스템 상에서 MPI-IO 방안의 성능 평가 벤치마크

박 성 순†


## 요 약

IBM에서는 개발한 범용병렬 파일 시스템(General Parallel File System: 이하 GPFS와 혼용)상에서 작동하는 MPI-IO 방안을 개발하였다. 우리는 이 MPI-IO 방안의 입출력 성능을 평가하기 위해 다양한 종류의 행렬 곱(matrix multiplication) 벤치마크 방안을 설계, 구현하였다. 현재 GPFS 상에서 구축되어 있는 IBM MPI-IO 방안에서는 네 가지 데이터 접근 방식을 제공하고 있는데, 본 논문에서 기술하는 모든 벤치마크들은 이러한 데이터 접근 방식의 입출력 시간 및 계산 시간을 측정하기 위해 구현되었다. 본 논문에서는 그 구현방안을 기술하고, 성능평가 내용을 기술한다.


# Benchmarks for Performance Testing of MPI-IO on the General Parallel File System

Sung-Soon Park†


## ABSTRACT

IBM developed the MPI-IO, we call it MPI-2, on the General Parallel File System. We designed and implemented various Matrix Multiplication Benchmarks to evaluate its performances. The MPI-IO on the General Parallel File System shows four kinds of data access methods : the non-collective and blocking, the collective and blocking, the non-collective and non-blocking, and the split collective operation. In this paper, we propose benchmarks to measure the IO time and the computation time for the data access methods. We describe not only its implementation but also the performance evaluation results.


**키워드 : MPI-IO, parallel io, double buffering, matrix multiplication**

## 1. 서 론

병렬처리 분야에서 데이터 접근방식 및 시스템 환경을 고려하여 효과적인 성능을 얻기 위해 병렬 입출력(parallel IO) 방안에 대한 중요성이 증대되고 있다[7]. 이를 위해 다양한 입출력 유형을 제공하는 MPI-IO 방안이 제시되었다[5]. 그리고 이 방안은 병렬 컴퓨터뿐만 아니라, 클러스터링 시스템에 이르기까지 다양한 시스템 환경에 적용되어 사용되고 있다[1, 4]. 그러나 아직 각 시스템들에서 어느 입출력 방안이 나은 성능을 보이는지를 비롯한 다양한 성능평가 부분에 대한 연구는 아직 미흡한 상황이다[3, 8].

본 논문에서는 IBM에서 개발한 범용 병렬 파일 시스템(General Parallel File System)상에서 작동하는 MPI-IO 방안의 입출력 성능을 평가하기 위해 설계, 구현한 다양한 종류의 행렬 곱(matrix multiplication) 벤치마크들을 기술하고 그 성능평가 내용을 분석한다. 현재 범용 병렬 파일 시스템 상에서 제공하는 있는 MPI-IO 방안에서는 네 가지의 데이터 접근 방식을 제공하고 있는데, 이하에서 기술하는 모든 벤치마크들은 이러한 데이터 접근 방식의 입출력 시간 및 계산 시간을 측정하기 위해 구현되었다[4]. 이 네 가지 데이터 접근 루틴들은 다음과 같다 :

(a) MPI_FILE_READ_AT (MPI_FILE_WRITE_AT)
(b) MPI_FILE_READ_AT_ALL (MPI_FILE_WRITE_AT_ALL)
(c) MPI_FILE_IREAD_AT 및 MPI_WAIT
(MPI_FILE_IWRITE_AT 및 MPI_WAIT)
(d) MPI_FILE_READ_AT_ALL_BEGIN
MPI_FILE_READ_AT_ALL_END
(MPI_FILE_WRITE_AT_ALL_BEGIN
MPI_FILE_WRITE_AT_ALL_END)

† 정 회 원 : 안양대학교 컴퓨터학과 교수
논문접수 : 2000년 10월 20일, 심사완료 : 2001년 5월 25일

이들은 동기화(synchronization)와 일치(coordination)를 명시적인 옵셋으로 표현함으로서 위치 성질(positioning property)을 만족하는 데이터 접근 연산들이다. 첫 번째 루틴인 (a)은 동기화와 일치성질 측면에서, 비집적/블록킹(non-collective and blocking) 연산이다. 두 번째 루틴인 (b)는 집적/블록킹 연산이고, 세 번째 루틴인 (c)은 비집적/비블록킹 연산이고, 네 번째 루틴인 (d)는 분할집적(split collective)연산이다. 이 루틴들의 이름을 간단히 표현하기 위해 이하에서는 (a)는 비집적, (b)는 집적, (c)는 비집적/비블록킹, 그리고 (d)는 분할집적 방식이라 부른다.

## 2. 벤치마크 설계

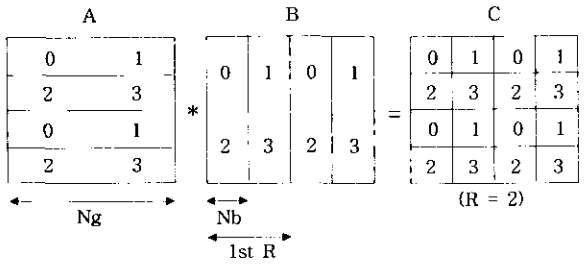
### 2.1 행렬 곱

본 연구에서는 행렬 곱 방안에 대해 총 11가지의 벤치마크 변형들을 구성하였다. 이들은 기존의 순수한 행렬 곱 벤치마크들을 기반으로 한다.

#### 2.1.1 TESTMAT1과 TESTMAT2

전통적인 행렬 곱 방안에 집적 루틴과 비집적 루틴 등의 두 데이터 접근 루틴들에 여러 MPI 프리미티브들을 추가하여 TESTMAT1을 구성하였다. 집적 루틴과 비집적 루틴들에 대한 여러 유형의 입출력 시간을 측정하는데 이 TESTMAT1을 사용한다. 이와 같은 방식으로 순수한 행렬 곱 방안에 비집적/비블록킹 루틴과 분할집적 방식으로 TESTMAT2를 구성하였다.

(그림 1)은 프로세서의 수가 네 개일 경우에 TESTMAT1 알고리즘의 계산유형을 보여주고 있다. 여기서 각 숫자는 프로세서의 인덱스를 나타낸다. 그리고 $N_b$는 각 계산노드 상에서의 지역 블록(local block)의 크기를 나타내고, R은 중복 요소(replication factor)로서 out-of core 행렬들의 전역 크기(global size)를 나타낸다. $N_g$는 $N_b$*R*sqrtP로 계산된다. (이후에서 프로세서의 수를 P로 나타내는데, 이때, P를 sqrtP*sqrtP로 표현하기도 한다. 이 sqrtP와 관련된 상세 내용은 '3.2 매개변수'에서 다룬다.)




(그림 1) TESTMAT1의 비교방안

프로그램 TESTMAT1과 TESTMAT2 알고리즘은 동일한데, 다음과 같다.

```
for R
    read A (data amount is Nb * Ng)
    for R
        read B (data amount is Ng * Nb)
        compute C = A * B
        write C (data amount is Nb * Nb)
    endfor
endfor
```

그래서 (그림 1)의 예에서 행렬 A가 4개의 수평블록으로, 행렬 B는 4개의 수직블록으로 구분되어있다고 할 때, 알고리즘에 따르면 네 개의 프로세서들은 디스크 영역의 행렬 A에서는 $N_b$*$N_g$만큼의 데이터를 각각 동시에 읽은 후, B로부터는 $N_g$*$N_b$ 만큼의 데이터를 또한 동시에 읽는다. 그리고 A*B의 연산을 각각 동시에 수행한 후, 디스크 영역의 행렬 C로 $N_b$*$N_b$ 만큼의 데이터를 쓴다.

즉, 프로세서 0은 알고리즘 상의 첫 번째 루프구조와 두 번째 루프구조에서 R 값이 0일 때, 행렬 A로부터는 크기가 $N_b$*$N_g$인 첫 번째 수평블록(horizontal block)을 읽고, B로부터는 $N_g$*$N_b$인 첫 수직블록(vertical block)을 읽는다. 이와 같은 방식으로 프로세서 1은 행렬 A의 첫 번째 수평블록과 행렬 B의 두 번째 수직블록을 읽고, 프로세서 2는 행렬 A의 두 번째 수평블록과 행렬 B의 첫 번째 수직블록을 읽는다. 그리고, 프로세서 3은 행렬 A의 두 번째 수평블록과 행렬 B의 두 번째 수직블록을 읽는다. 물론 네 개 프로세서들이 행렬 A로부터 데이터를 읽는 작업은 동시에 진행된다. 그리고, 행렬 B로부터 데이터를 읽는 작업을 4개의 프로세서가 동시에 진행하고, C를 계산하는 곱 연산을 수행한다. 그 후, 프로세서 0은 그 계산결과를 행렬 C의 첫 번째 줄에서의 첫 번째 칸 디스크 영역에 쓰고, 프로세서 1은 첫 번째 줄의 두 번째 칸에 쓴다. 다른 프로세서들도 같은 방식으로 저장한다.

#### 2.1.2 SREVERSEMAT

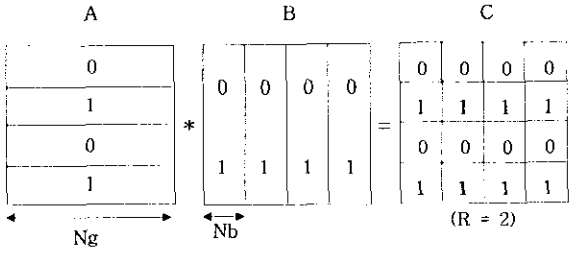
다양한 측정결과를 얻기 위해, 프로세서들과 행렬 상에서의 데이터 접근영역을 고려하면서 데이터 접근유형을 변경시키는 방식으로 TESTMAT1을 수정한 여러 변형들을 구현하였다. 그 첫 번째 방안으로는 행렬 A와 B 사이의 데이터 읽기 방식을 간단하게 바꾼 SREVERSEMAT이 있다. TESTMAT1와 같은 일반적인 행렬 곱에서는 행렬 A와 B를 읽고 A*B에 의해 C를 계산한 후 C 결과를 쓰지만, SREVERSEMAT에서는 A를 읽고 B를 읽는 순서를 B를 먼저 읽고 A를 읽는 순서로 바꾼다.

데이터 접근방식이 행 우선(column major)인 행렬 B의 데이터 접근 횟수를 줄일 수 있고, 일반적으로 열 우선(row major) 데이터 접근이 행 우선 경우보다 더 낫기 때문에, 접근유형이 열 우선인 경우에는 SREVERSEMAT이 TESTMAT1보다 더 나은 입출력 시간을 보여줄 수 있다. 그 알고리즘은 다음과 같다.

```
For R
    read B
    for R
        read A
        compute C = A * B
        write C
    endfor
endfor
```

### 2.1.3 MREVERSEMAT

SREVERSEMAT에서 프로세서 접근유형을 변경하여 MREVERSEMAT과 CREVERSEMAT를 구성하였다. (그림 2)는 MREVERSEMAT를 위한 접근방식을 보여준다.



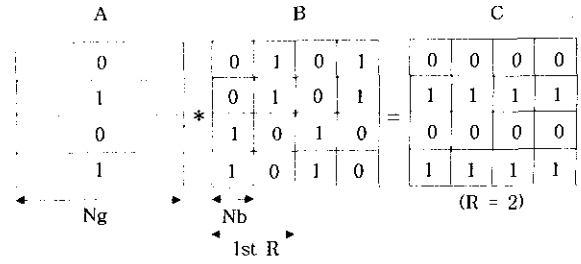

(그림 2) MREVERSEMAT의 접근방식

두 개의 프로세서를 사용한다고 가정할 때, 프로세서 0과 1은 동시에 행렬 B의 수직블록을 읽는다. 그리고 프로세서 0는 행렬 A의 첫 수평블록을, 프로세서 1은 두 번째 수평블록을 읽는다. 물론 이들의 데이터 크기는 이전 프로그램과 같은 $N_g$*$N_b$이다. 다음 단계로 각 프로세서는 C를 계산하고, 행렬 C에서 첫 번째 행의 첫 번째 결과와 두 번째 열에 대한 디스크 영역에 그 결과를 저장한다. 이 알고리즘은 다음과 같다.

```
For (iv = 0 ; iv < (R*P) ; iv++)
    read B
    For (ih = 0 ; ih < R ; ih++)
        read A
        compute C = A * B
        write C
    endfor
endfor
```

### 2.1.4 CREVERSEMAT

행렬 B의 접근 유형을 약간 수정하여, MREVERSEMAT로부터 CREVERSEMAT를 생성하였다. 그러나 CREVERSEMAT에서는, 프로세서 0이 행렬 B의 첫 번째 수직블록을 읽을 때, 프로세서 1은 두 번째 수직블록을 읽는다. 그래서 다음에 프로세서 0이 행렬 B의 두 번째 수직블록을 읽을 때 프로세서 1은 반대로 첫 번째 수직블록을 읽는다. (그림 3)은 CREVERSEMAT의 접근 방식을 보여준다. 그래서 MREVERSEMAT의 경우에 프로세서들이 행렬 B의 동일 수직블록을 읽는 반면, CREVERSEMAT는 디스크 영역의 같은 위치를 읽는 것을 피할 수 있다.




(그림 3) CREVERSEMAT의 접근방식

비록 MREVERSEMAT와 CREVERSEMAT간에 새로운 위치를 계산하는 옵셋이 다르긴 하지만 (실제적으로 CREVERSEMAT의 옵셋 계산은 좀 더 복잡하다), CREVERSEMAT의 알고리즘은 MREVERSEMAT와 유사하다.

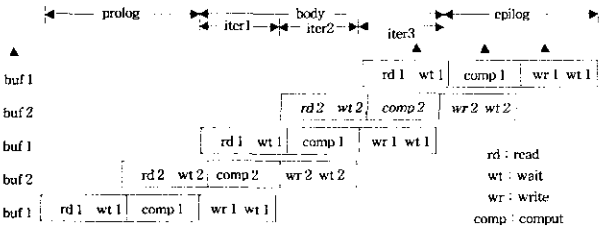
## 2.2 행렬 곱을 위한 이중 버퍼링

동기화 성질과 관련하여, MPI-IO는 비블록킹과 분할집적 방식 등의 두 가지 입출력 방안을 지원한다. 일반적으로 비블록킹/비집적 방식이나 분할집적 방식들을 비블록킹 입출력 호출이라 한다. 앞에서 이미 TESTMAT2에서 구현된 비블록킹 입출력 호출에 대해 언급하였다. 이러한 호출 방식에서는 입출력 연산을 시작시키긴 하지만, 비록 그 블록킹 입출력 호출이 입출력 요구가 끝났음을 알리지 않는다고 해도 그 연산이 끝나기를 기다리지 않는다. 그래서 입출력 연산의 종결을 기다리지 않고 다른 연산들의 수행을 시작할 수 있다. 이 성질을 사용하여 비블록킹 입출력 호출방식의 성능을 개선하기 위해, 호출방식들을 위한 이중 버퍼링(double buffering) 방안을 설계하고, 이상에서 언급하였듯이 행렬 곱 프로그램의 여러 유형을 구현하였다. 비블록킹/비집적 방식 및 분할 집적 방식 등을 위해 구분되는 두 가지 이중 버퍼링 방안을 설계하였다.

### 2.2.1 비블록킹/비집적 방식을 위한 이중 버퍼링

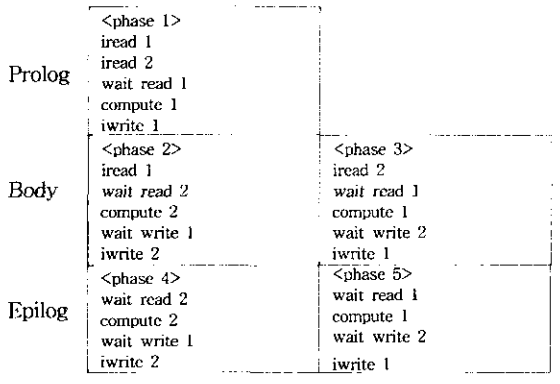
비블록킹/비집적 방식을 위한 기본적인 논리는 다음과 같다. 어떤 데이터를 디스크 영역에서 버퍼영역으로 읽어올 때, 이 읽기가 끝나기 전에 다른 데이터를 디스크 영역에서 다른 버퍼 영역으로 읽어 오기 시작할 수 있다는 것이다. 이는 이 방식이 디스크 영역으로부터 데이터를 읽어오는 작업과 디스크 영역으로 데이터를 쓰는 작업을 계산 작업과 동시에 수행할 수 있게 하기 때문이다. 그래서 계산 시간에 데이터를 읽거나 쓸 수 있다. 그러나 다른 통신을 위한 계산을 위해 버퍼 영역을 사용하면 오류가 발생한다.

(그림 4)는 비블록킹/비집적 방식을 위한 이중 버퍼링 방안을 보여준다. 읽기-계산-쓰기 과정이 5회 반복된다고 가정하면, 첫 번째 접근은 읽기 데이터를 버퍼 1(buf1)로 전송한다. 이때, 첫 번째 버퍼에서 첫 번째 연산(comp1)이 시작되기 이전에 두 번째 버퍼(buf2)로의 읽기(rd2)가 시작될 수 있다. 이 작업은 파이프라인 방식으로 진행된다.




(그림 4) 비블록킹/비집적 방식을 위한 이중 버퍼링 방안

이 방식을 사용하여, 비블록킹/비집적 입출력 방식을 위한 이중 버퍼링 방안의 변형형태인 TESTDBMAT1과 DB1SREVERSEMAT 및 DB1MREVERSEMAT 등을 구현하였다. TESTDBMAT1는 비블록킹/비집적 입출력을 수행하는데 사용하는 행렬 곱 방식인 TESTMAT2에 이중 버퍼링을 적용한 방식이다. DB1SREVERSEMAT는 TESTMAT1에 대한 SREVERSEMAT 경우에서처럼 TESTMAT2의 역순 데이터 접근방식에 이중 버퍼링을 적용한 방식이다.



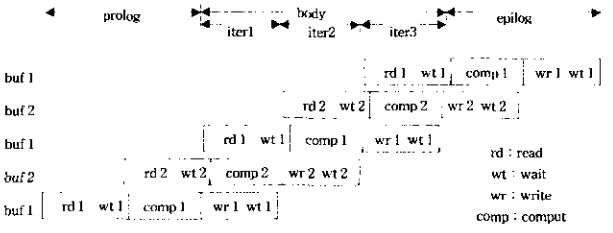

(그림 5) 비블록킹/비집적방식의 이중 버퍼링 수행

DB1MREVERSEMAT는 TESTMAT1에 대한 MREVERSEMAT 경우에서처럼 TESTMAT2의 또 다른 역순 방식에 이중 버퍼링을 적용한 프로그램이다.

우리는 (그림 4) 방식에 기반한 세 가지 벤치마크를 구현하였다. (그림 5)는 비블록킹/비집적 입출력 방식의 이중 버퍼링 수행 알고리즘을 보여주고 있다. 이 수행은 프롤로그(prolog)에서 시작하여, 몸체부분이 반복되는 형식으로 진행된다. 즉, 수행은 단계 1에서 시작하고, 몸체부분은 단계 2, 3, 2, 3, … 순으로 반복된다. 그래서 몸체부분의 마지막 수행이 단계 2에 이르렀을 때, 단계 5가 에필로그(epilogue)로 수행된다. 몸체부분의 마지막 수행이 단계 3인 경우에는 단계 4가 에필로그 단계로 수행된다. (그림 5)에서 'iread'와 'iwrite'는 MPI_FILE_IREAD_AT와 MPI_FILE_IWRITE_AT를 의미하고 'wait'는 MPI_WAIT를 의미한다.

2.2.2 분할집적방식을 위한 이중 버퍼링

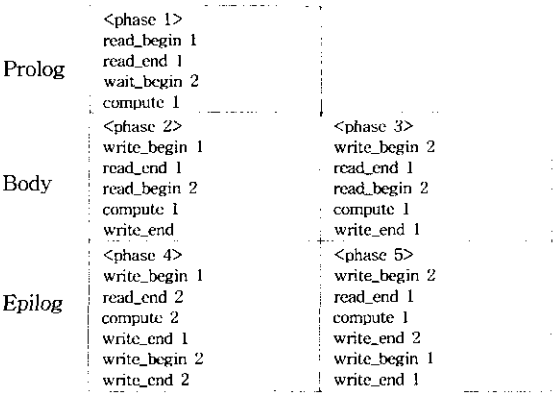
분할집적 연산은 비블록킹 방식보다 더 많은 제약된 규칙을 갖는다. 특히 분할집적 입출력 연산은 같은 파일을 동시에 접근하는 것을 허용하지 않는다. 그래서 새로운 read_begin과 read_end 쌍을 위치시킬 때, 이전에 나왔던 쌍의 read_end 다음에 위치시켜야 한다. (그림 6)은 분할집적을 위한 이중 버퍼링 방식을 보여준다. 읽기-계산-쓰기 과정이 5회 반복된다고 가정할 때, 버퍼로 데이터를 읽는 후속 read_begin은 이전 읽기 연산의 read_end 다음에 위치시킬 수 있다. 쓰기 또한 읽기와 동일한 연산 순서를 갖는다.




(그림 6) 분할집적을 위한 이중 버퍼링

이 또한 파이프라인 방식으로 진행된다.

이 방식을 사용하여 우리는 분할집적 입출력의 여러 이중 버퍼링 변형인 TESTDBMAT2, DB2SREVERSEMAT 및 DB2MREVERSEMAT 등을 구현하였다. TESTDBMAT2는 분할집적 입출력을 사용한 행렬 곱 방식인 TESTMAT2에 이중 버퍼링을 적용한 방식이다. DB2SREVERSEMAT는 TESTMAT1를 위한 SREVERSEMAT1 경우처럼, TESTMAT2의 역순 데이터 접근방식에 분할집적 연산을 위한 이중 버퍼링을 적용한 방식이다. DB2MREVERSEMAT은 TESTMAT1을 위한 MREVERSEMAT의 경우처럼, TESTMAT2의 또 다른 역순 방식에 분할집적 연산을 위한 이중 버퍼링을 적용한 방식이다.




(그림 7) 분할집적 연산을 위한 이중 버퍼링 수행

우리는 또한 (그림 6)의 방식에 기반한 세 가지 벤치마크를 구현하였다. (그림 7)은 분할집적 입출력의 이중 버퍼링 수행 알고리즘을 보여준다. 그 수행은 (그림 5) 알고리즘과 동일한 순서로 진행된다. (그림 7)에서, read_begin은 MPI_FILE_READ_AT_ALL_BEGIN을, read_end는 MPI_FILE_READ_AT_ALL_END을, write_begin은 MPI_FILE_WRITE_

AT_ALL_BEGIN을, 그리고 write_end는 MPI_FILE_WRITE_AT_ALL_END 등을 의미한다.

## 3. 구 현

### 3.1 시스템 환경

모든 성능측정은 다음 사양을 갖는 아이비엠 RS/6000 SP2 시스템 상에서 진행되었다.

- 16개의 200MHz thin 노드 : 두 개의 PowerPC 630 프로세서 ; 하나의 GB 실(real) 메모리
- 전체 시스템을 위한 하나의 SP 스위치 ; 노드 당 하나의 SPSMX 스위치 어답터
- AIX 4.3.2 + PSSP 3.1 기반 소프트웨어 (이것은SPMD 작업을 관리하는 Parallel Operating Environment를 가짐)
- GPFS Release 1.2 ; RVSD(Recoverable Virtual Shared Disk) Release 3.1.1

노드들의 8개는 범용 병렬 화일 시스템을 위한 VSD 서버들로 구성되었다. 각 서버노드는 "JBOD" 모드 (no RAID or mirroring)라 불리는 8개의 4.5GB 5,400RPM SSA 디스크 드라이브의 두 loop에 붙은 하나의 Feature 2125 Enhanced SSA 어답터를 장착하고 있다. 다른 8개의 노드들은 수행 측정을 위한 클라이언트로 제공된다. 각 클라이언트 노드들은 VSD 소프트웨어뿐만 아니라 범용 병렬 화일 시스템 클라이언트 코드들도 수행하였다. 전체 범용 병렬 화일 시스템의 크기는 약 560GB 정도이다. 8개의 클라이언트 노드들 상에서의 측정은 1, 2, 4, 8, 또는 16병렬 태스크들을 수행하도록 조정되어 있다. 노드들은 8개 또는 그 이하의 프로세스들을 위해 태스크들에 할당되어 있다. 각 노드는 하나의 태스크를 수행했다. 16 태스크 수행을 위해 각 클라이언트 노드는 두 개의 독립적인 태스크들을 수행했다.

### 3.2 매개변수

이 절에서는 범용 병렬 화일 시스템 상에서 MPI-IO를 위한 여러 유형의 성능 측정을 위해 매개변수들을 정의한다. 이 매개변수들은 그 벤치마크에 따라 분류된다.

행렬 곱 프로그램을 위한 성능측정을 위해 이하의 매개변수들을 사용한다.

- 프로세서 수, P : 2, 4, 8, 16.
- 행렬 크기 : 8M.
- 프로그램
- 중복 요소, R : 1, 2, 4, 8.
- 각 계산 노드 당 지역블록 크기, $N_b$ : 32, 64, 128, 256, 512.
- out-of-core 행렬의 광역크기, $N_g$

프로세서 수와 관련하여, 이하의 벤치마크들은 매개변수 sqrtP를 프로세서 수 P를 제공하는 입력 매개변수로 사용되는데, 그 이유는 이하의 프로그램들이 sqrtP에 기반하여 코딩되었기 때문이다. 이들 프로그램을 위한 P 값은 sqrtP*sqrtP로 계산된다. 그리고 이들 프로그램을 위한 매개변수 $N_g$는 sqrtP*$N_b$*R로 계산된다.

- TESTMAT1, TESTMAT2, SREVERSEMAT, TESTDBMAT1, TESTDBMAT2, DB1SREVERSEMAT, DB2SREVERSEMAT

그러나 이하의 벤치마크들은 프로세서 수을 위한 입력 매개변수로 P를 사용한다. 그것들의 Ng는 P*Nb*R로 계산된다.

- MREVERSEMAT, CREVERSEMAT, DB1MREVERSEMAT, DB2MREVERSEMAT

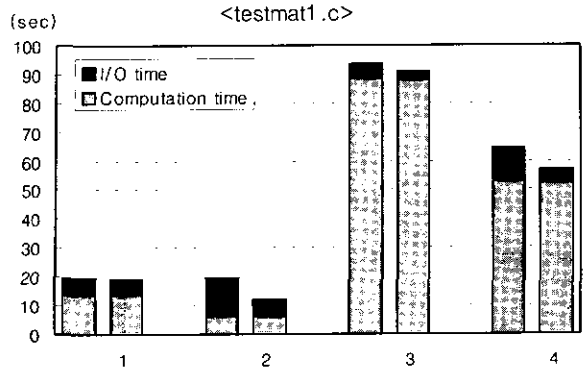
데이터 쉬핑(data shipping)을 위해서는 하나의 매개변수를 사용한다. 그래서 우리는 그 모드를 데이터 쉬핑 경우와 비데이터 쉬핑(non-data shipping) 경우로 나누었다. 비데이터 쉬핑 경우는 물론 데이터 쉬핑이 적용되지 않은 경우이다. 입출력 작업에 데이터 쉬핑을 적용하였을 때, 행렬 곱 벤치마크를 위한 버퍼크기를 1, 2, 4, 8, 그리고 비데이터 쉬핑 등의 경우로 분류할 수 있다. 데이터 쉬핑 값이 1이면, 그 버퍼크기는 1*256KB이다.

## 4. 측정결과 및 분석

수행시간과 성능을 얻고, 분석하기 위해 우리는 벤치마크들을 실행하여 여러 측정 모형을 구성하였다. 그 수행시간은 최소 5번의 반복 수행을 통해 평균값으로 얻었고, 또한 그 평균시간은 사용하는 모든 프로세서에서의 평균시간이기도 하다. 먼저, 우리는 행렬 크기와 프로세서 수, 그리고 각 방식들에 대해, 집적 입출력 연산과 비집적 입출력 연산간의 수행시간을 비교하였다. 둘째, 여러 방식들에 대해 프로세서 수를 바꿀 때의 수행시간을 측정하였다. 세째, 여러 방안에 대하여 데이터 쉬핑 매개변수를 변경할 때의 데이터 쉬핑 방안의 효과를 측정하였다. 우리는 비블록킹/비집적 방안과 분할집적 방안들을 위한 여러 이중 버퍼링 방안들을 구현하여, 이들에 대한 이중 버퍼링 방안의 효과를 측정하였다.

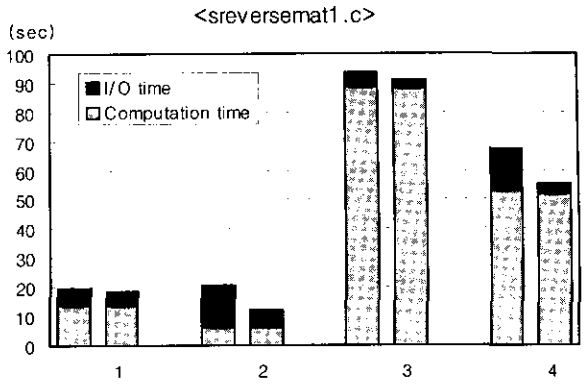
### 4.1 집적 대 비집적

네 가지 행렬 곱 벤치마크들에 대해 매개변수 값을 변경하면서 네 가지 경우들로 분류하여 집적 연산과 비집적 연산 사이의 전체 수행시간 (입출력 시간 + 계산시간)을 측정하였다. 각 행렬 크기는 1024x1024이다. 우리는 그 버퍼크기가 2*256KB인 데이터 쉬핑을 사용하였다.



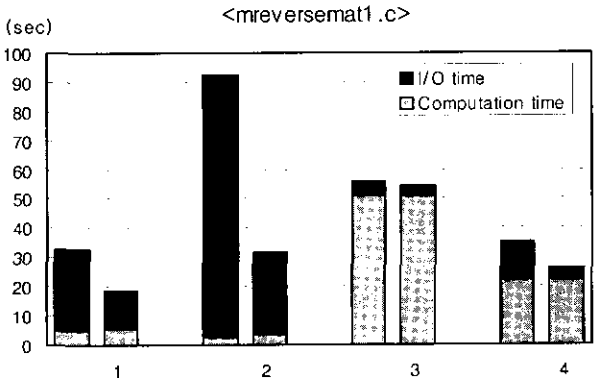

• 1 : P가 16, R이 1, $N_b$가 256인 경우
• 2 : P가 16, R이 2, $N_b$가 128인 경우
• 3 : P가 4, R이 1, $N_b$가 512인 경우
• 4 : P가 4, R이 2, $N_b$가 256인 경우

(a) TESTMAT1



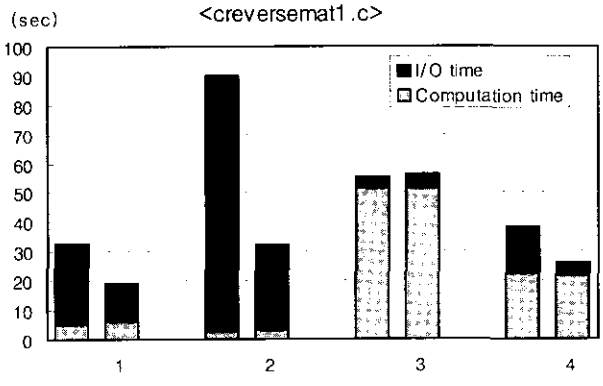

• 1 : P가 16, R이 1, $N_b$가 256인 경우
• 2 : P가 16, R이 2, $N_b$가 128인 경우
• 3 : P가 4, R이 1, $N_b$가 512인 경우
• 4 : P가 4, R이 2, $N_b$가 256인 경우

(b) SREVERSEMAT




• 1 : P가 16, R이 1, $N_b$가 64인 경우
• 2 : P가 16, R이 2, $N_b$가 32인 경우
• 3 : P가 4, R이 1, $N_b$가 256인 경우
• 4 : P가 4, R이 2, $N_b$가 128인 경우

(c) MREVERSEMAT




• 1 : P가 16, R이 1, $N_b$가 64인 경우
• 2 : P가 16, R이 2, $N_b$가 32인 경우
• 3 : P가 4, R이 1, $N_b$가 256인 경우
• 4 : P가 4, R이 2, $N_b$가 128인 경우

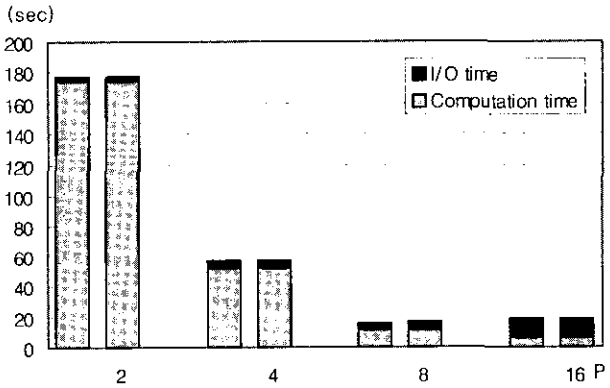
(d) CREVERSEMAT

(그림 8) 집적 대 비집적 연산 경우

대부분의 실험으로부터 비집적 연산에서의 전체 수행시간은 집적연산보다 좋음을 볼 수 있다. 지역블록 크기를 감소시키면서 중복 요소를 증가시킬 때, 전체 수행시간에서의 입출력 시간은 증가한다. 이는 디스크영역에 대한 접근빈도 수가 증가하고 그 입출력 시간이 이 빈도에 종속되기 때문이다. 그 대표적인 현상을 MREVERSEMAT와 CREVERSEMAT에서 볼 수 있다. 지역블록 크기를 감소시키면서 프로세서 수와 중복 요소를 증가시킬 때, 그 입출력 시간은 극도로 증가한다. 이들 알고리즘에서 MREVERSEMAT와 CREVERSEMAT 경우를 TESTMAT1과 SREVERSEMAT 경우들과 비교하면 그 수행 복잡도는 (R*P)*R이다.

### 4.2 프로세서 수

프로세서 수를 변경하면서, 비집적 연산을 위한 전체 수행시간 (입출력시간 + 계산시간)을 측정하였다. 여기서 각 행렬의




• 2, 4, 8, 16인 경우
• 2, 4, 8, 16인 경우

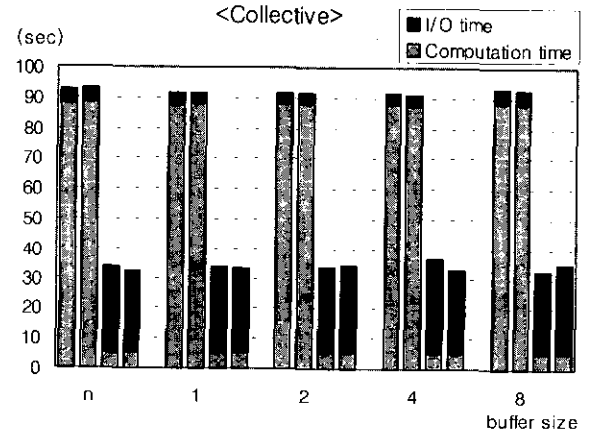
(그림 9) 프로세서 수에 따른 경우

크기는 1024 x 1024이다. 버퍼크기는 2 * 256KB인 데이터 쉬

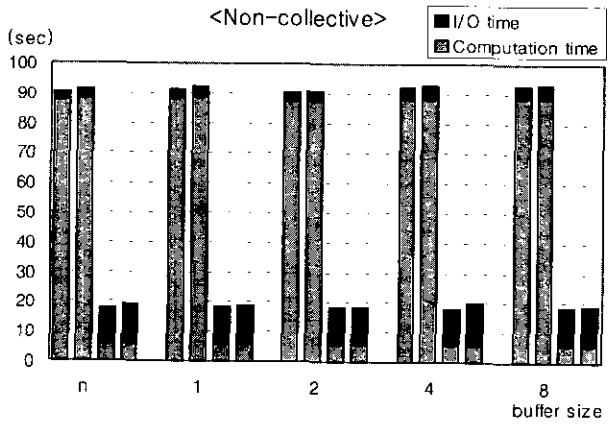
핑을 사용한다. 각 프로세서를 위한 행렬 크기는 8MB이고, 중복 요소는 1이다. 프로세서 수를 증가시켰을 때, 비록 전체 수행시간은 감소하지만 입출력 시간은 증가한다. 이는 전달되는 데이터 크기가 작아져서 그 전달이 더 빈번해지기 때문이다.

### 4.3 Data shipping

데이터 쉬핑 효과를 측정하기 위해 TESTMAT1, SREVERSEMAT, MREVERSEMAT 및 CREVERSEMAT 등을 수행하였다. 이 실험을 위한 행렬크기는 1024x1024이고, 중복 요소는 1이다. TESTMAT1과 SREVERSEMAT에서 집적연산과 비집적연산을 위해 데이터 쉬핑에서의 버퍼크기를 1, 2, 4, 8로 변경시키면서 측정결과를 얻었고, 비데이터 쉬핑 경우에 대해서도 측정했다. 이 실험을 위해 4개의 프로세서를 사용하였다. 행렬크기는 8MB이고 지역블록 크기는 512다. MREVERSEMAT와 CREVERSEMAT 등의 경우에 대해서도 16개의 프로세서를 사용한 것과 지역블록 크기가 64인 것을 제외하고는 같은 환경 하에서 진행하였다.




(a) 집적 : TESTMAT1, SREVERSEMAT, MREVERSEMAT 및 CREVERSEMAT



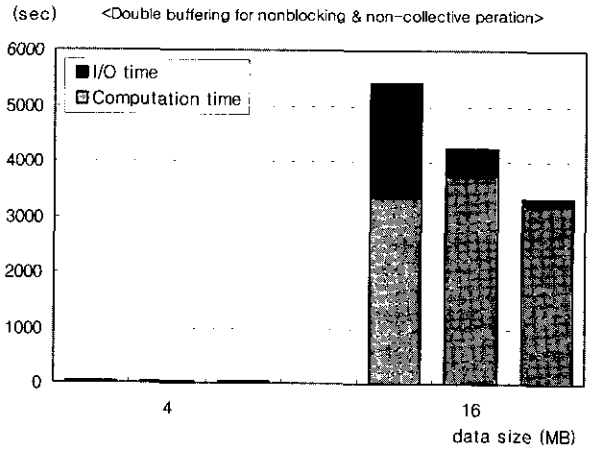

(b) 비집적 : TESTMAT1, SREVERSEMAT, MREVERSEMAT 및 CREVERSEMAT

(그림 10) 비데이터 쉬핑 경우와 4가지 데이터 쉬핑 경우들

여기서 비록 데이터 쉬핑을 적용하여도 전체 수행시간이 그리 많이 감소되지 않음을 발견하였다. 그래서 계산 시간과 형태의 관계, 지역블록 크기, 지역 데이터의 크기, 그리고 전송 블록크기 등을 고려하면서 데이터 쉬핑 효과를 측정하고 분석하는 것이 필요하다.

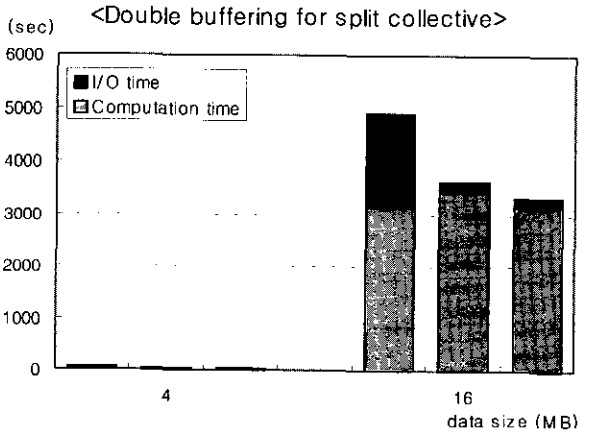
### 4.4 이중 버퍼링

우리는 비집적/비블록킹 연산과 분할집적 연산을 위한 이중 버퍼링 방안을 설계하고 구현하였다. 그 효과를 보이기 위해, 이중 버퍼링 방안을 TESTMAT2와 SREVERSEMAT 에




- TESTMAT2에서 비집적/비블록킹 연산 경우
- TESTDBMAT1 : TESTMAT2의 비집적/비블록킹 연산을 위한 이중 버퍼링 경우
- DB1SREVERSEMAT : SREVERSEMAT의 비집적/비블록킹 연산을 위한 이중버퍼링 경우

(a) 비블록킹/비집적 연산을 위한 이중 버퍼링 (버퍼 크기가4MB와 16MB)




- TESTMAT2에서 분할집적 연산 경우
- TESTDBMAT2 : TESTMAT2의 분할집적을 한 이중 버퍼링
- DB2SREVERSEMAT : SREVERSEMAT의 분할 집적을 위한 이중 버퍼링

(b) 분할집적을 위한 이중 버퍼링 (데이타크기가 4MB와 16MB)

(그림 11) 이중 버퍼링 경우

적용하였다. 4개의 프로세서(sqrtP 값을 2로하는)들을 사용하고, 그 지역블록 크기는 128로 한다. 중복 요소를 4에서 16으로 증가시키면서, 데이터 크기가 4MB인 경우와 16MB인 경우의 입출력 시간을 측정하였다.
여기서 이중 버퍼링을 입출력 연산에 적용하였을 때, 대부분의 전체 실행시간이 감소하였음을 알 수 있다. 특히 입출력 시간은 급격히 감소하였다.

## 5. 결론 및 향후 연구

우리는 행렬 곱 벤치마크의 여러 방안들을 설계, 구현하였고, 범용 병렬 화일 시스템 상에서의 MPI-IO 성능을 얻기 위해 읽기/쓰기만을 적용한 경우와 이중 버퍼링을 적용한 경우를 위한 실험 프로그램을 작성하였다. 또한 여러 측면을 고려하며 매개변수를 변경하면서 그 성능을 측정하였다. 그래서 다음과 같은 결과를 얻고, 향후 연구방향을 세웠다.

첫째, 집적연산과 비집적 연산에서의 수행을 비교해 볼 때, 비집적 경우가 집적 경우보다 일반적으로 나은 성능을 보였다. 그러나 부분적으로 그 역의 경우도 발견하였다. 그래서 캐싱이나 그 블록크기, 전송 데이터 크기 등을 고려하는 더 이상의 분석이 필요함을 알 수 있었다.

둘째, 프로세서 수를 고려할 때, 입출력 시간이 프로세서 수와 더불어 입출력 연산형태나 지역블록의 크기, 중복 요소 등과 같은 여러 다른 매개변수에 의해 좌우됨을 발견하였다. 그래서 우리는 매개변수들간의 관계를 고려한 더 이상의 측정이 필요하고, 성능의 최고점을 찾는 것과 그것을 특성화시키는 것이 필요하다. 그래서 우리는 범용 병렬 화일 시스템에서 MPI-IO를 위한 일반화된 성능모형을 구성할 수 있다.

셋째, 데이터 쉬핑 방식을 적용할 때, 비록 이미 testrdwr1.c 프로그램에서 데이터 쉬핑의 효과에 대한 결과를 보이기는 하였지만, 입출력 시간에 영향을 미치는 더 이상의 매개변수를 고려하는 것이 필요함을 발견하였다.

마지막으로, 우리는 이중 버퍼링을 비블록킹/비집적, 분할 집적, 그리고 그 외 다른 변형들에 적용할 때, 성능향상 결과를 얻었다. 이중 버퍼링과 더불어 우리는 더 이상의 성능향상을 위해 쓰레드(thread) 개념을 고려해야 한다.

이상에서 얻은 실험결과로부터 다음과 같은 개선점이 얻어질 수 있다.

① 행렬 곱에 대해서, 보다 의미있는 결과를 얻기 위해 보다 커다란 행렬을 사용한다. 그러나 커다란 행렬에 대해서는 계산시간이 급격히 커지므로 이를 줄이기 위한 방안이 필요하다.
② 데이터가 두 번째 또는 그 이후에 접근될 때 대부분의 데이터가 이미 범용 병렬 화일 시스템에 있기 때문에 읽기 시간을 대폭 줄이는 경향이 있는 범용 병렬 화일 시스템 캐쉬 효과를 얻을 수 있도록 testrdwr 프로그램에 대해서, 보다 커다란 파일을 사용한다.
③ 적어도 64개 이상의 노드를 갖는 보다 큰 시스템 상에서 벤치마크를 실행시켜 집적연산의 확장성(scalability)을 검사한다.
④ 결과의 가변성을 줄이기 위해 각 측정을 더 반복한다. (적어도 10회 이상 반복하고, 평균계산 시간의 최대시간과 최소시간은 하는 것이 바람직하다.)

## 참 고 문 헌


[1] "An Introduction to GPFS 1.2," IBM T. J. Watson Research Center, Dec, 1998. (Source : http : //www.rs6000.ibm.com/reso urce/technology/paper1.html).
[2] "General Parallel File System : High Performance Parallel File Systems," IBM T. J. Watson Research Center, Dec, 1998. (Source : http : //www.rs6000.ibm.com/sof tware/sp_products/gpfs.html).
[3] "GPFS Performance," IBM T. J. Watson Research Center, Feb, 1999. (Source : http : //www.rs6000.ibm.com/resource/technology/ gpfs_perf.html).
[4] IBM Internal Documents, "Supported Functions of MPI-I/O," IBM T. J. Watson Rescarch Center, 1998.
[5] Message Passing Interface Forum, "MPI-2 : Extensions to the Message Passing Interface, Univ. of Tennessee," Jul, 1997.
[6] P. S. Pachew, "Users Guide to MPI," Univ. of San Francisco, Mar, 1998.
[7] R. Bordawekar, J. M. del Rossario, and A. Choudhary, "Design and Evaluation of Primitives for Parallel I/O," Preceedings of Supercomputing 93, pp.452-461, 1993.
[8] R. Thakur, W. Gropp and E. Lusk, "Achieving High Performance with MPI-IO," ANL/MCS pp.742-0299, Argonne National Laboratory, Feb, 1999.



**박 성 순**

e-mail : sspark@aycc.anayng.ac.kr
1984년 홍익대학교 전자계산학과 졸업 (학사)
1987년 서울대학교 대학원 계산통계학과 졸업(석사)
1994년 고려대학교 대학원 전산과학과 졸업(박사)
1988년~1990년 공군사관학교 전임강사
1996년~1998년 Northwestern Univ. Postdoctoral fellow
1994년~현재 안양대학교 컴퓨터학과 부교수
관심분야 : 병렬처리, SAN