논문 2017-12-42

플래시 디스크 기반 행렬전치 알고리즘 심층 분석 및 성능개선

(In-depth Analysis and Performance Improvement of a Flash Disk-based Matrix Transposition Algorithm)

이 형 봉*, 정 태 윤
(Hyung-Bong Lee, Tae-Yun Chung)

Abstract : The scope of the matrix application is so broad that it can not be limited. A typical matrix application area in computer science is image processing. Particularly, radar scanning equipment implemented on a small embedded system requires real-time matrix transposition for image processing, and since its memory size is small, a general matrix transposition algorithm can not be applied. In this case, matrix transposition must be done in disk space, such as flash disk, using a limited memory buffer. In this paper, we analyze and improve a recently published flash disk-based matrix transposition algorithm named as asymmetric sub-matrix transposition algorithm. The performance analysis shows that the asymmetric sub-matrix transposition algorithm has lower performance than the conventional sub-matrix transposition algorithm, but the improved asymmetric sub-matrix transposition algorithm is superior to the sub-matrix transposition algorithm in 13 of the 16 experimental data.

Keywords : Disk matrix transposition, Sub-matrix transposition, Virtual sub-matrix, Buffer cache

Ⅰ. 서 론

일반적인 행렬 연산은 프로그래밍 언어가 지원하는 2차원 배열 즉, 메모리 공간에서 이루어진다. 특히 행렬의 전치는 선형대수의 가장 기본적인 연산 중의 하나로서 응용분야가 넓고 그 기본 알고리즘은 이미 널리 알려져 있다. 최근에는 다중코어 프로세서나 분사처리를 활용하는 병렬처리 [1, 2]와 하드웨어 수준의 메모리 구조를 고려한 메모리 접근 전략 [3]에 의한 성능향상 방안 등이 연구되고 있다. 그러나 행렬의 규모에 비하여 메모리 크기가 매우 작을 경우에는 행렬 데이터의 배열 적재 자체가 불가능하다. 따라서 이런 경우에는 최소한의 버퍼 메모리를 이용하여 디스크 공간에 저장된 데이터에 직접 접근하여 연산을 시행해야 한다. 그런데 2 차원 메모리 배열에 적용하는 알고리즘을 디스크 배열에 그대로 적용하면 디스크 입·출력이 과도하게 발생하여 효과적이지 못하다.

행렬전치는 이미지 처리를 위한 2 차원 디지털 필터 (2D Digital Filter)나 2 차원 푸리에 변환 (2D-FFT: 2D Fast Fourier Transforms) 등에 필요한 연산이다. 특히 합성 개구 레이더 (SAR: Synthetic Aperture Radar) [4] 나 무인 항공기 (UAV: Unmanned Aerial Vehicle) [5], 그리고 기상 예보 영역 [6]에서는 촬영된 대규모 영상 데이터에 대한 실시간 처리가 필수적이어서 효과적인 디스크 전치 알고리즘이 요구된다 [7].

이 연구에서는 SSD, USB, HDD 등 블록 저장장치에 저장된 대규모 정방행렬을 전치하는 알고리즘 [7]을 리눅스 운영체제 환경에서 심층 분석하고 성능 개선을 모색한다. 이를 위하여 Ⅱ 장에서 운영체제의 블록장치 입·출력 관리가 어떻게 이루어지는지를 살펴보고, Ⅲ 장에서 일반적인 디스크 행렬전치 알고리즘들을 고찰한다. Ⅳ 장에서는 최근에 발표된 플래시 메모리 기반 비대칭 부분행렬전치 알

*Corresponding Author (tychung@gwnu.ac.kr)
Received: July 20 2017, Revised: Sep. 25 2017, Accepted: Oct. 12 2017.
H.B. Lee, T.Y. Chung: Gangneung-Wonju National University

ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2017.12.6.377

고리즘에 대한 분석 및 개선, 그리고 성능 평가를 실시하고 마지막 V장의 결론으로 맺는다.

II. 운영체제 블록 저장장치 입·출력 관리

1. 메모리와 블록 저장장치

메모리 (주기억장치)와 블록 저장장치 (디스크)의 근본적인 차이점은 저장단위가 각각 바이트와 블록이라는 데 있다. 8비트로 구성된 바이트 단위로 저장된 메모리에서 4 바이트로 구성된 정수 데이터를 읽거나 쓰는 일은 다른 바이트와 관계없이 독립적으로 수행될 수 있다. 그러나 특정 바이트의 일부 비트만을 접근하기 위해서는 해당 바이트 전체를 레지스터에 읽은 후 비트 연산을 통해 원하는 비트에 접근해야 한다. 블록 저장장치는 저장단위가 적게는 수 백 바이트에서 많게는 수 십 킬로바이트에 이른다. 특정 블록 내에 저장된 4 바이트 정수에 접근하기 위해서는 블록 전체를 메모리에 읽은 후 내부 거리 (offset)을 계산하여 접근해야 한다.

2. 디스크 버퍼 캐시

버퍼 (buffer)의 일반적인 개념은 입·출력 단위나 속도의 차이를 극복하기 위해 데이터를 일시적으로 보관하는 장소이고, 캐시 (cache)는 속도가 느린 저장장치에 대한 빠른 접근을 지원하기 위해 그 복사본을 저장해 두는 고속의 장소이다 [8]. 리눅스 등 거의 모든 상용 운영체제는 디스크 입·출력 성능 향상을 위해 디스크 블록에 대하여 버퍼와 캐시 개념을 동시에 적용한 디스크 버퍼 캐시를 운영한다 [9]. 디스크 버퍼 캐시는 그림 1과 같이 한 번 접근된 디스크 블록을 메모리에 캐시시킴으로써 이후의 해당 디스크 블록에 대한 읽기와 쓰기가 메모리에서 이루어지도록 한다. 대부분의 운영체제는 메모리 용량 및 용도 등 시스템 환경에 따라 다르나 보통 50% 이상을 버퍼 캐시에 할당하여 시스템 전체 성능 향상을 우선적으로 추구한다 [10]. 버퍼 캐시의 가장 큰 문제는 디스크에 업데이트되지 않은 불결 (dirty block) 블록이 존재하는 상태에서 단전 등 시스템 고장이 발생하면 그 때까지의 수정 내용이 사라진다는 점이다. 이를 보완하기 위해 운영체제는 기회가 있을 때마다 디스크에 출력하는 지연 쓰기 (delayed write) 전략을 도입한다. 따라서 디스크 입·출력 어플리케이션은 지연 쓰기가 이루어지기 전에 해당 블록에 대한 쓰기를 마치는 것이 성능 관점에서 유리하다. 그러나 어플리케이션이 접근 데이터의 위치와 블록 번호와의 대응관계까지 고려하는 일은 무리이므로 이를 위해서는 가급적 가까운 위치의 데이터를 차례로 접근하는 것이 최선이다. 디스크 버퍼 캐시의 또 다른 문제는 그 용량이 제한되어 있으므로 디스크의 광범위한 범위가 참조될 경우 기존의 캐시를 디스크에 업데이트한 후 다른 블록용으로 재사용해야 한다는 점이다. 이 때 어떤 블록을 선택하여 교체할 것인가의 방법은 버퍼 캐시의 성능을 좌우하는 중요한 문제이다. 특히 낸드 플래시 (NAND Flash) 디스크의 경우 특정 블록을 쓰기 위해서는 지우기 작업이 필요하므로 버퍼 캐시 운영전략의 중요성은 더욱 커진다 [11-13].

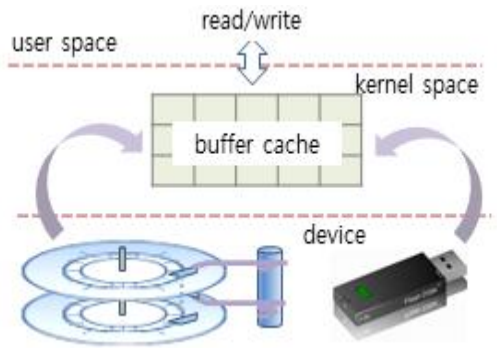

그림 1. 운영체제의 디스크 버퍼 캐시
Fig. 1 Disk buffer cache of operating systems

3. 리눅스 입·출력 시스템 콜

유닉스와 리눅스 등 POSIX (Portable Operating System Interface) [14] 규격을 준수하는 운영체제는 이 논문 내용과 관련하여 표 1의 입·출력 시스템 콜들을 표준으로 지원하다. 그림 2에는 표 1의 시스템 콜을 이용하여 행우선 (row major) 형태로 저장된 디스크 행렬의 i 행 j 열 원소 DM (i, j)에 대한 읽기 및 쓰기 프리미티브 예를 보였다.

표 1. POSIX 입·출력 시스템 콜
Table 1. POSIX I/O system calls

I/O system calls	Function
open(pathname, flag [,mode])	file open
lseek(fd, offset, reference)	access location
read(fd, location, size)	input
write(fd, location, size)	output
close(fd)	file close

```
DM_read(int i, int j, DATA *vp, int n)
{
   lseek(Fd, (i*COL+j)*sizeof(*vp), SEEK_SET);
   read(Fd, vp, sizeof(*vp)*n);
}
DM_write(int i, int j, DATA *vp, int n)
{
   lseek(Fd, (i*COL+j)*sizeof(*vp), SEEK_SET);
   write(Fd, vp, sizeof(*vp)*n);
}
```

그림 2. 디스크 행렬 원소 읽기/쓰기 프리미티브
Fig. 2 Read/Write primitives for disk matrix

```
:
  Fd = open("DM_DATA", O_RDWR);
  for (i = 0; i < ROW; i++) {
     for (j = i+1; j < COL; j++) {
       DM_read(i, j, &vr, 1);
       DM_read(j, i, &vc, 1);
       DM_write(i, j, &vc, 1);
       DM_write(j, i, &vr, 1);
     }
  }
  close(Fd);
:
```

그림 3. 원소 기반 디스크 행렬전치 알고리즘
Fig. 3 Element-based disk matrix transposition algorithm

Ⅲ. 디스크 행렬전치 알고리즘

1. 원소 기반 접근 방식

원소 기반 접근법은 그림 3과 같이 파일개방 절차를 제외하고는 메모리 행렬전치 알고리즘과 동일하다. 즉, 간단하고 별도의 버퍼를 요구하지는 않지만 운영체제의 버퍼 캐시에 대한 부담이 크다. 그림 4에서 행 (i)은 블록 내 가까운 부분을 참조하여 캐시 적중률이 높지만, 열 (j)은 계속 새로운 블록을 요구하기 때문에 캐시 적중률이 낮고 캐시 용량 한계에 따른 블록 교체 빈도가 높아진다. 뿐만 아니라 매 원소마다 입·출력이 요구되어 운영체제 실행부담 또한 증가한다.

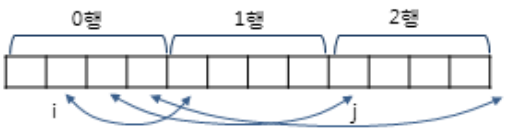

그림 4. 원소 기반 디스크 행렬전치의 블록 참조 패턴
Fig. 4 Block access pattern of element-based disk matrix transposition

```
:
  DATA  Buf[NB];
  for (i = 0; i < ROW; i++) {
     for (j = i + 1; j < COL; ) {
        bn = MIN(NB, COL - j);
        DM_read(i, j, Buf, bn);
        for (bi = 0; bi < bn; bi++, j++) {
           vr = Buf[bi];
           DM_read(j, i, &vc, 1);
           Buf[bi] = vc;
           DM_write(j, i, &vr, 1);
        }
        DM_write(i, j, Buf, bn);
     }
:  }
```

그림 5. 행 기반 디스크 행렬전치 알고리즘
Fig. 5 Row-based Disk Matrix Transposition Algorithm

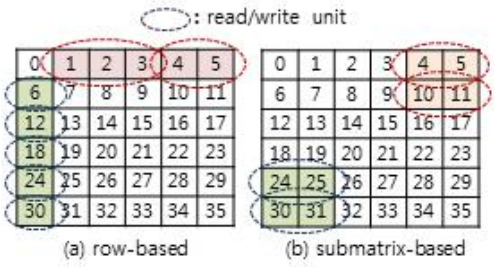

그림 6. 디스크 행렬 입·출력 단위 비교
Fig. 6 Comparison of disk matrix I/O unit

2. 행 기반 접근 방식

그림 4의 원소 기반 블록 참조 패턴에서, 연속되는 행 원소들을 그림 5, 그림 6 (a)와 같이 버퍼 단위로 묶음 처리하면 행에 대한 입·출력 횟수를 감소시킬 수 있다. 이를 열 위주로 변환해도 결과는 마찬가지다.

3. 부분행렬 기반 접근 방식

그림 5, 그림 6 (a)의 행 기반 전치에서의 단점은 열에 대해서는 여전히 원소 기반의 입·출력이 불가피하다는 점이다. 이를 보완하여 열 부분에 대해서도 묶음 처리가 가능하도록 하는 방법이 그림 6의 (b)에 보인 부분행렬기반 전치이다. 즉, 대칭 관계에 있는 두 부분행렬을 버퍼에 묶음 처리로 읽은 후 메모리 내에서 전치하고 그 결과를 다시 묶음 처리로 저장한다. 이 때 두 부분행렬의 크기가 동일하므로 각각에 전체 버퍼의 반을 할당해야 한다. 부분행렬 SM(si, sj)에 대한 읽기와 쓰기 프리미티브

```
SM_read(int si, int sj, DATA *vp, int sm)
{
   for (i = 0; i < sm; sm++, vp += sm)
      DM_read(si*sm, sj*sm, vp, sm);
}
SM_write(int si, int sj, DATA *vp, int sm)
{
   for (i = 0; i < sm; sm++, vp += sm)
      DM_write(si*sm, sj*sm, vp, sm);
}
```

그림 7. 디스크 부분행렬 읽기/쓰기 프리미티브
Fig. 7 Read/Write primitives for disk sub-matrix

```
:
  DATA  Buf1[NB/2], Buf2[NB/2];
  Sm = sqrt(NB/2);
  for (si = 0; si < Sm; si++) {
     for (sj = si + 1; sj < Sm; sj++) {
        SM_read(si, sj, Buf1, Sm);
        if (si == sj)
          SM_trans_diag(Buf1, Sm);
        else {
          SM_read(sj, si, Buf2, Sm);
          SM_trans(Buf1, Buf2, Sm);
          SM_write(sj, si, Buf2, Sm);
        }
        SM_write(si, sj, Buf1, sm);
     }
  }
:
```

그림 8. 부분행렬 기반 디스크 행렬전치 알고리즘
Fig. 8 Sub-matrix-based disk matrix transposition algorithm

예를 그림 7에 보였고, 그림 8은 이를 이용한 부분 행렬 기반 전치 알고리즘이다.

버퍼 단위 묶음 입·출력의 효과를 확인하기 위해 위 세 가지 유형의 알고리즘에 대한 성능을 비교한다. 이 논문의 실험 환경은 표 2와 같고, 디스크로는 USB 메모리 형태의 8GB 이동식 플래시 디스크를 사용하면서 임베디드 리눅스와 유사하게 최소한의 버퍼 캐시를 사용하도록 쓰기 캐싱을 오프 (off)

표 2. 실험 환경
Table 2. Experimental Environment

Items	Specifications(model)
CPU	• Intel E7200, Core 2 • 2.53GHz
Memory	• 4.0GB
OS	• Window 7 • Cygwin(Linux environment in Window)

표 3. 디스크 전치 알고리즘 성능 비교 (단위: 초)
Table 3. Performance comparison of disk matrix transposition algorithm (unit: sec)

Data set	Element-base	Row-base	Submatrix-base
1^2K^2	43.24	22.10	4.11
2^2K^2	171.86	153.05	14.58
3^2K^2	383.99	203.58	32.87
4^2K^2	688.29	357.25	43.80

로 설정하였다 [15]. 테스트 데이터로는 1K×1K ~8K×8K의 정수 정방 행렬을 n2K2 형태의 표기로 사용하고, 버퍼는 4K ~ 32K의 크기를 사용한다.

표 3은 버퍼가 8K일 때, 12K2 ~ 42K2 행렬에 대한 원소 단위, 행 기반 버퍼 단위, 부분행렬 기반 버퍼 단위 입·출력을 적용한 각각의 전치 시간 측정 결과이다. 이 표로부터 버퍼 단위 입·출력의 효과가 크다는 사실을 알 수 있다.

Ⅳ. 부분행렬 기반 디스크 행렬전치 알고리즘 분석 및 개선

1. 비대칭 부분행렬 기반 알고리즘 분석

그림 7, 8의 부분행렬 기반 디스크 행렬전치는 대각선상에 위치한 부분행렬 전치 시 한 쪽 버퍼를 활용하지 못하는 단점이 있다. 이를 보완하기 위해 [7]은 버퍼를 반으로 나누어 대칭 관계에 있는 두 부분 행렬에 분배하는 대신, 대각선상의 부분행렬에는 전체 버퍼를 할당하는 방안을 제시하였다. 이렇게 하면 대각선상의 부분 행렬들은 버퍼 전체를 모두 활용한다. 그러나 대칭 관계에 있는 두 부분행렬들은 한꺼번에 버퍼에 읽을 수 없기 때문에 그림 9

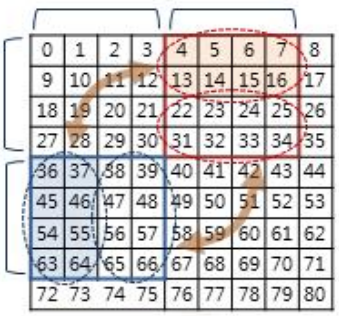

그림 9. 비대칭 디스크 행렬전치 개념
Fig. 9 Concept of asymmetric sub-matrix transposition

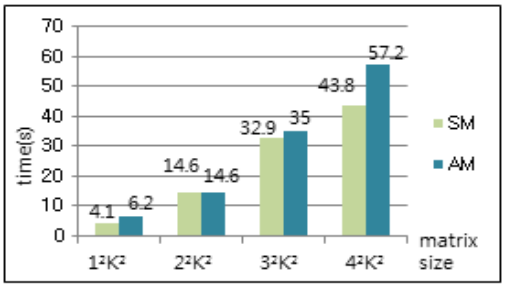

그림 10. 부분행렬 기반 (SM)과 비대칭행렬 기반 (AM) 알고리즘의 전치 성능 비교
Fig. 10 Performance comparison between sub-matrix-based (SM) and asymmetric sub-matrix-based (AM) algorithm

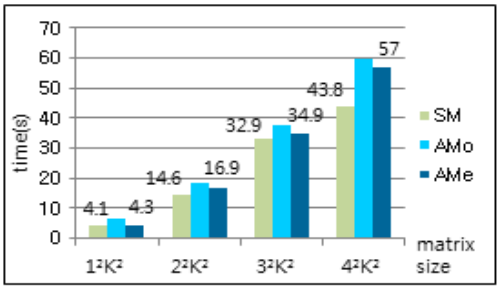

그림 11. 부분행렬 크기 짝수화에 의한 AM 알고리즘 개선 결과 (AMe)
Fig. 11 Improvement result of AM algorithm using even-sized sub-matrix (AMe)

와 같이 부분행렬 각각을 반으로 분리하여 두 번에 걸쳐 처리하고, 이 방안을 비대칭 부분행렬전치라 명명하였다. 만약 부분 행렬의 크기가 홀수라면 두 번 처리 후 나머지 부분을 행 기반 접근 방식으로 전치한다. 부분 행렬 단위에서 남는 부분 역시 행 기반 접근 방식으로 전치하도록 제시하였다.

그림 10은 표 3에서와 마찬가지로 버퍼 크기 12K일 때 네 가지 정방 디스크 행렬에 대한 부분행렬 기반 (SM: Sub-matrix)과 비대칭 부분행렬 기반 (AM: Asymmetric sub-matrix) 알고리즘의 전치 성능 비교인데, 발표된 AM 알고리즘의 성능이 기존의 SM 알고리즘보다 오히려 더 좋지 않음을 보이고 있다.

2. 비대칭 부분행렬 기반 (AM) 알고리즘의 개선

그림 10에서 AM 알고리즘 성능저하 원인의 하나로 부분행렬 크기가 홀수라는 점을 들 수 있다. 즉, 버퍼 12K는 정수 타입 3,072 (=12,288/4)개의 저장이 가능하고 이는 55×55 (=3,025)의 홀수 크기 부분 행렬을 감당할 수 있다. 이 경우 비대칭 부분행렬 전치가 세 번에 걸쳐 이루어져야 한다. 약간의 버퍼 낭비를 무릅쓰고 짝수 크기인 54×54 부분행렬을 사용하면 성능저하가 완화될 가능성이 있다. 이를 확인하기 위해 버퍼 12K를 55×55의 홀수 크기 부분행렬로 사용하는 경우 (AMo: AM odd)와 54×54의 짝수 크기 부분행렬로 사용하는 경우 (AMe: AM even)로 분류하여 그림 10과 동일한 데이터에 실험한 결과를 그림 11에 보였는데, 개선 효과는 있으나 성능이 여전히 SM 알고리즘보다 낮은 것으로 나타난다.

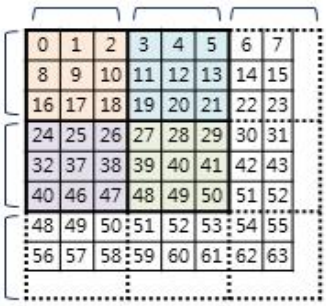

그림 12. AM 알고리즘 개선을 위한 가상 부분행렬 개념
Fig. 12 Concept of virtual sub-matrix for improvement of AM algorithm

AM 알고리즘의 성능 향상을 위한 또 다른 시도로서 부분행렬 단위를 제외한 나머지 부분을 행 기반으로 하나씩 처리하지 않고, 그림 12와 같이 가상 부분행렬로 확장하여 묶음 처리하는 방안을 제시한다. 이 방안을 따르면 연속된 디스크의 인접 데이터 접근 빈도를 높일 수 있다. 이와 같이 AM 알고리즘에 가상 부분행렬 개념을 적용한 알고리즘 AMv (AM with virtual sub-matrix)의 그림 10, 11의 동일 데이터에 대한 전치 성능을 그림 13에 비교해 보였다. 부분행렬 크기의 짝수화에 의한 개선보다 가상 부분행렬에 의한 개선 폭이 더 크고, 네 가지 중 세 가지 데이터에서 SM보다 성능이 더 나아진 현상을 볼 수 있다.

AMe와 AMv 알고리즘을 조합한 AMev의 성능을 분석하기 위하여 두 가지 관점에서의 실험을 실시하였다. 첫 번째는 주어진 42K2 디스크 행렬에 대하여 버퍼 크기가 4K에서 32K까지 4K 단위로

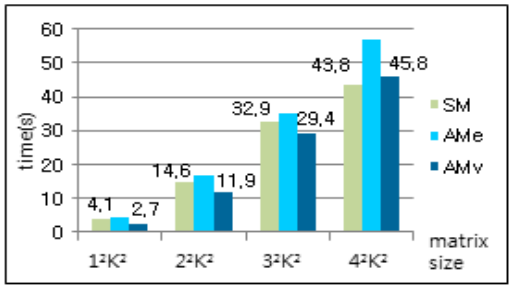

그림 13. 가상 부분행렬에 의한 AM 알고리즘 개선 결과 (AMv)

Fig. 13 Improvement result of AM algorithm using virtual sub-matrix (AMv)

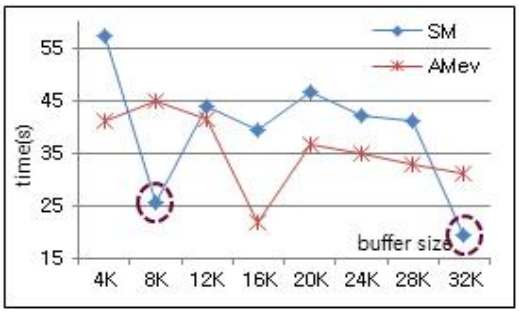

그림 14. 버퍼 크기 변화에 따른 AMev 알고리즘의 성능

Fig. 14 Performance of AMev Algorithm with Varying Buffer Size

표 4. 디스크 행렬 크기 변화에 따른 AMev 알고리즘의 성능 (단위: 초)

Table 4. Performance of AMev algorithm with varying disk matrix size (unit: sec)

Algo-rithm	1^2K^2	2^2K^2	3^2K^2	4^2K^2	5^2K^2	6^2K^2	7^2K^2	8^2K^2
SM	4	15	33	44	68	90	127	150
AMev	2	10	24	42	58	88	119	**158**

변화하는 관점이고, 두 번째는 주어진 버퍼 크기 12K에 대하여 디스크 행렬 크기가 12K2에서 82K2까지 1K 단위로 변화하는 관점이다. 두 관점의 측정 결과를 그림 14와 표 4에 보였다. 총 16가지 측정 결과 중, 그림 14의 버퍼 크기가 8K와 32K인 경우와, 표 4에서 행렬 크기가 82K2인 경우 등 총 3 가지 경우에만 AMev 알고리즘의 성능이 낮게 측정되었을 뿐, 나머지 13 가지에서는 우수한 것으로 나타났다. 이는 전체적으로 AMev의 성능이 우수하지만, 알고리즘 버퍼 크기와 부분행렬의 크기 및 운영체제 디스크 버퍼 캐시 크기의 관계, 그리고 버퍼 캐시의 현재 상태 등에 따라 극단적으로 어느 한 쪽에 유리한 경우가 존재할 가능성이 있는 것으로 분석된다.

V. 결 론

디스크 행렬 전치는 메모리가 한정된 소형 임베디드 시스템으로 구현되는 레이더 촬영 장비 등에서 필수적이다. 이 논문에서는 최근에 발표된 디스크 비대칭 부분행렬전치 알고리즘의 성능을 분석하고 개선하였다. 분석결과 비대칭 부분행렬전치 알고리즘은 기존의 부분행렬전치 알고리즘보다 성능이 높지 않다는 의외의 결과가 나타났다. 그 원인 분석을 토대로, 부분행렬 크기의 짝수화와 부분행렬 단위를 벗어난 나머지 부분에 대한 가상 부분행렬 처리 등 두 가지 개선점을 제시하였다. 그 개선 결과 모든 경우에서 개선 전 비대칭 부분행렬전치 알고리즘보다 우수하고, 16 가지 실험 데이터 중 13 가지에서 기존의 부분행렬전치 알고리즘보다 우수함을 확인하였다. 향 후, 플래시 디스크에 특화된 운영체제 디스크 버퍼 캐시 전략을 고려한 또 다른 개선점을 발견할 수 있을 것으로 기대된다.

References

[1] R.A. Na'mneh, W.D. Pan, S.M. Yoo, "Efficient Adaptive Algorithms for Transposing Small and Large Matrices on Symmetric Multiprocessors," Informatica, Vol. 17, No. 4, 535. pp. 535-550, 2006.

[2] J.C. Bowman, M. Roberts, "Adaptive Matrix Transpose Algorithms for Distributed Multicore Processors," Proceedings of Applied Mathematics, Modeling and Computational Science, Vol. 117, pp. 97-103, 2015.

[3] S. Huanghui, W. Zhensong1, Z. Weimin, "An Efficient Memory Access Strategy for Transposition and Block Operation in Image Processing," Journal of Computer Research and Development, Vol. 50, No. 1, pp. 188-196, 2013.

[4] K.Y. Kwak, "Trend of SAR Technology," Journal of the Korean Institute of

Electromagnetic Engineering and Science, Vol. 22, No. 6, pp. 4-16, 2011 (in Korean).

[5] K.S. Chung, Y.S. Kim, S.Y. Oh, “Development of a Flood Damage Assessment Technology using UAV,” Journal of Korea Water Resources Association, Vol. 48, No. 1, pp. 51-59, 2015 (in Korean).

[6] J.G. Park, “Storage Format of Domestic Weather Radar Raw Data,” Proceedings of 2015 Korean Meteorological Society Conference, pp. 34-35, 2012 (in Korean).

[7] S.C. Kim, W.K. Park, B.W On, I. Lee, G.S. Choi, “An Asymmetry Matrix Transposition Scheme Based on NAND Flash Memory,” IEMEK J. Embed. Sys. Appl., Vol. 10, No. 21, pp.81-89, 2015 (in Korean).

[8] https://en.wikipedia.org/wiki/Data_buffer, https://en.wikipedia.org/wiki/Cache_(computing)

[9] A. Silberschatz, P. B. Galvin, G. Gagne, “Operating System Concepts,” 9th Ed., pp. 565-612, Wiley, 2013.

[10] https://unix.stackexchange.com/questions/25316/restrict-size-of-buffer-cache-in-linux

[11] S.W. Lee, D.J. Park, T.S. Chung, D.-H.Lee, S. Park, H.J. Song, “A log Buffer Based Flash Translation Layer Using Fully Associative Sector Translation,” ACM Transactions on Embedded Computng Systems, Vol. 6, No. 3, pp. 436-453, 2007.

[12] H.S. Lee, H.S. Yun, D.H Lee, “HFTL: Hybrid Flash Translation Layer Based on hot Data Identification for Flash Memory,” IEEE Transactions on Consumer Electronics, Vol. 55, No. 4, pp. 2005-2011, 2009.

[13] S. Kim, T. Kim, “A Write Buffer Management Scheme Considering the Command Queue in SSD,” Journal of KIISE Vol. 39, No. 1A, pp. 313-315, 2012 (in Korean).

[14] http://standards.ieee.org/develop/wg/POSIX.html

[15] http://www.thewindowsclub.com/enable-disable-disk-write-caching-windows-7-8

Hyung-Bong Lee (이 형 봉)

He received the B.S. and M.S. degrees in Computer Science from Seoul National University, Seoul, Korea, in 1984 and 1986 respectively. He received his Ph.D. degree in Computer Science from Kangwon National University, Chuncheon, Korea, in 2002. From 1986 to 1994, he was a senior engineer in Computer R&D Division of LG Electronics. From 1995 to 1998, he was with DEC (Digital Equipment Corporation) as a UNIX consultant. From 1999 to 2003, he was an Associate Professor at Honam University, Gwangju, Korea. Since 2004, he has been a Professor in the Department of Computer Science & Engineering at Gangneung-Wonju National University, Wonju, Korea. His current research interests include embedded systems, wireless sensor networks, and data mining algorithms.

Email: hblee@gwnu.ac.kr

Tae-Yun Chung (정 태 윤)

He received the B.S., M.S., and Ph.D. degrees in the School of Electrical & Computer Engineering at Yonsei University, Seoul, Korea in 1987, 1989, and 2000 respectively. From 1989 to 1996, he was a Research Engineer of Samsung Advanced Institute of Technology. From 1996 to 2001, he was with Samsung Electronics as a Senior Research Engineer. From 2000 to 2001, he was a vice chair of International DVD-Forum. Since 2001, he has been with the Department of Electronics Engineering at Gangneung -Wonju National University, Gangneung, Korea, and is currently a Professor. Since 2004, he has been with GEMS-CRC (Gangwon Embedded Software Cooperative Research Center, Gangneung, Korea) as the Chef. His major fields are image signal processing, digital video encoding, multimedia, copy protection, and his current research interests are embedded system, sensor network, video encoding.

Email: tychung@gwnu.ac.kr