최소 자료 이동을 위한 최적 병렬 정렬 알고리즘

홍 성 수† 심 재 홍††

요 약

본 논문은 p($p=n^{1-x}$, $0<x<1$)개 프로세서가 존재하는 EREW-PRAM 모델 병렬 컴퓨터에서 시간 복잡도가 $O(n^x \log n)$이며, 비용(최악의 실행시간 * 프로세서 수)은 $O(n\log n)$이고, 자료 이동도가 $O(n^{1-x}+n^x)$인 병렬 정렬 알고리즘을 제안한다. 병렬 정렬 알고리즘은 리스트를 p개 특정키를 중심으로 분리한 다음 블록의 크기를 거의 일정하게 할 수 있는 엔코딩 기법을 사용했다.

An Optimal Parallel Sort Algorithm for Minimum Data Movement

Sung Soo Hong† and Jae Hong Sim††

ABSTRACT

In this paper we propose parallel sorting algorithm, taking $O(n^x \log n)$time complexity, $O(n^x \log n)$ cost (parallel running time * number of processors) and $O(n^{1-x}+n^x)$data movement complexity under the ERWW-PRAM model. The methods for solving these problems similar. Parallel algorithm finds pivot for partitioning the data into ordered subsets of approximately equal size by using encording pointers.

1. 서 론

정렬은 이론적으로 큰 기본적 문제이고, 계산에 있어서나 자료를 처리하는데 실질적으로 중요한 알고리즘 중에 하나다[5]. 일반적으로 병렬 정렬 문제는 크게 2개의 범주로 나누고 있으며 그 하나는 병합을 기초로 하는 정렬 방식이고 또 하나는 분할 방식에 의한 정렬이 있다. 병합 방식에 의한 정렬은 리스트를 여러개 서브 리스트로 나누고 각 프로세서들을 서브 리스트에 할당해 서브 리스트들을 정렬한 다음 한개의 리스트가 될 때까지 병합시키는 방법이다. 이 방법은 스케줄 상이나 동기화 상에 많은 문제점이 있다[8]. 분할 방식에 의한 방식에 의한 방법은 특정키를 먼저 선택한 다음 리스트를 그 특정키를 중심으로 분리하며 이 방식을 리스트의 모든 원소들이 정렬될 때까지 반복해 수행한다.

분할 방식을 사용한 알고리즘들의 특징은 i번째 서브리스트의 모든 원소들은 (i+1)번째 서브리스트의 원소보다 작은 특성을 갖는다. 그러나 분할 방식 알고리즘의 가장 커다란 문제점은 분할된 서브 리스트의 자료 크기를 거의 같게 만드는데 그 초점을 맞추고 있다. 분할 방식의 특성을 만족하면서 가능한이면 자료의 이동을 적게하는 효율적인 병렬 알고리즘 개발이 시급하다. 왜냐하면 자료의 이동이 많으면 많을 수록 충돌 횟수와 여러가지 버스 연결등이 복잡해지기 때문이다. 이러한 점을 극복하기 위하여 Colemen 등은 입력 자료에서 임의적으로 통계적 표본을 얻어 그것으로 다중키를 사용하여 정렬하는 방식도 있다[11].

병렬처리를 위한 컴퓨터 시스템은 일반적으로 SISD, SIMD, MISD, MIMD 4가지로 분류하고 있다[10]. SIMD 병렬 시스템은 그 구조 형태에 따라 다시 SMM(Shared Memory Model), MCC

† 종신회원 : 호서대학교 컴퓨터공학과 부교수
†† 정 회 원 : 광운대학교 전자계산학과 교수
논문접수 : 1994년 4월 18일, 심사완료 : 1994년 8월 3일

(Mesh Connected Computer), CCC(Cube Connected Computer), PSC(Perfect Shuffle Computer)로 분류되고 있으며, SMM은 다시 공통 기억 장소를 프로세서가 동시에 접근 할 수 있느냐 없느냐에 따라 EREW-PRAM, CREW-PRAM, CRCW-PRAM로 나누어진다[6]. 충돌을 허용하는 CRCW-PRAM 모델 컴퓨터에서 개발된 병렬 알고리즘을 충돌을 허용하지 않는 EREW-PRAM 모델에 적용 시키는데는 n개 프로세서를 사용하여 O(T(n).logn)시간이 소요된다[6]. (단 T(n): CRCW-PRAM 모델에서의 시간복잡도)병렬 알고리즘에서 최적이라는 것은 병렬알고리즘의 비용(병렬 알고리즘의 최악의 실행시간 *프로세서 수)이 그 문제를 해결하기 위한 선형 알고리즘의 상수배로 나타날 수 있을 때 병렬 알고리즘을 최적이라고 한다[12, 13, 14].

정렬 문제를 해결하기 위해서 병합 방식을 사용한 문헌으로는 [7, 13]등이 있으며 분할 방식에 의한 문헌은 [1, 12, 16]등이 있다. 이중 EREW-PRAM 모델에서 최적인 알고리즘은 [12, 13]이 있고, CRCW, CREW 모델에서의 최적알고리즘은 [4, 15]가 있다. 최소 최소 자료 이동을 위한 알고리즘은 permutation sort[7], address table sort[12]등이 있다.permutation 정렬방식은 최소 자료 이동을 갖는 알고리즘이기는 하나 프로세서수가 $O(n^2)$ 알고리즘으로 알고리즘이 최적이 아니다. Address table 정렬 방식은 알고리즘이 최악일 경우 알고리즘이 최적이 아닐 뿐만 아니라 여분의 기억장소와 포인터가 각 원소수 만큼 필요한 알고리즘이다.

본 논문은 EREW-PRAM모델 병렬컴퓨터에서 p개 프로세서들이 p개 피보트 원소들을 임의로 선정하고 p개 피보트 키들을 정렬한 다음 이진 검색 방법을 사용해 나머지 모든 원소들이 어느 블록에 속해 있나를 조사한 다음 모든 원소들을 해당 블록에 할당한다. 이때 각 블록의 크기가 일정하지 않으므로 블록의 크기를 거의 일정하게 할 수 있는 알고리즘을 제안한다.

2. Parallel Prefix와 병렬 선택 알고리즘

2장에서는 본 논문에서 사용되는 알고리즘중 KRUSKAL이 제안한 parallel prefix[3]와 Akl이 제안한 병렬 선택 알고리즘[14]을 살펴본다.

2.1 Parallel Prefix 알고리즘

Prefix 계산이란 식$(a1^{\circ} a2^{\circ} \ldots\ldots a_k, k=1\ldots,n)$의 결과를 구하는 것으로 최근 병렬처리 알고리즘에서 많이 사용하고 있다.(단 $^{\circ}$:associative 명령) KRUSKAL등은 EREW모델 병렬 컴퓨터에서 p $(p=n^{1-x})$개 프로세서를 사용하여 시간 복잡도가 $O(n^x)$인 알고리듬을 제안했으며, 개략적인 알고리즘은 다음과 같다.

```
procedure PREFIX(n, A, p)
  for j=0 to [logn] − 1 do
    p개 프로세서들은 동시에 다음식을 계산
    한다.(단,1≤i≤n)
    만약 (i MOD 2^j)=0이면
      NEWA[i]=A[i−2^j] ° A[i]을 계산한다.
    p개 프로세서들은 동시에 NEWA[i]를 A
    [i]로 옮긴다.
  endfor
END PREFIX
```

prefix 알고리즘에서 컴퓨터 모델은 EREW모델이고, 이 모델은 p개$(p=n^{1-x}, 0<x<1)$ 프로세서가 존재하며, 각 프로세서들은 지역 기억장소인 NEWA를 갖고 있다. 만약 $^{\circ}$ 가 합을 의미하는 연산 명령이라고 하면 결국 prefix알고리즘은 A[1]+A[2] ...A[n]을 계산하는 문제가 될 것이다.

2.2 병렬 선택 알고리즘

단일 프로세서 상에서 k번째 작은 원소를 구하는 알고리즘은 이미 잘 알려져 있다. 여기에서는 Akl이 제안한 병렬 선택 알고리즘을 살펴보자. Akl[3]이 제안한 컴퓨터 모델은 EREW모델이며, 프로세서수는 $n^{1-x}(0<x<1)$ 즉 P_1, P_2....P^{1-x}

일때 배열 A에서 k-번째 작은 원소를 구하는 문제다. Akl은 p개 프로세서를 사용하여 $O(n^x)$시간만에 k-번째 원소를 구하는 알고리즘을 제안했으며, 개략적 알고리즘은 다음과 같다. 이때 $|Si|$는 서브리스트 Si의 원소 갯수다.

procedure PSELECT(A,k)

1) 리스트 A를 S_i^{1-x} 개의 서브리스트로 분할하며, 각 서브리스트의 원소 갯수는 S_i^x개이다.
2) 각 프로세서들은 중위수를 구한다음 그것을 공동 기억장소 M에 기억 시킨다음 중위수 가운데 중위수를 구한다.
3) 배열 A의 모든 원소들을 중위수 가운데 중위수를 중심으로 크거나 (S_2) 같거나(S_2), 적은원소(S_3)들로 분리한다.
4) CASE
   : $|S_1| \geq k$:CALL PSELECT(S_1, k)
   : $|S_1| + |S_2| \geq |k$:return(중위수 가운데 중위수)
   : else :CALL PSELECT(S_3, k- $|S_1| - S_2|$)
   ENDCASE

END PSELECT

PSELECT 알고리즘에서 배열 A는 특정 피보트 키를 중심으로 S_1, S_2, S_3(피보트 키보다 적은 원소의 블록, 같은 원소 블록, 큰 원소 블록)으로 나누워 지며, 각 블록은 순서가 있으므로 $S_{1max} < S_{2min}$, $S_{2max} < S_{3min}$의 결과를 얻을 수 있다. PSELECT알고리즘을 사용하여 리스트 A의 $\lfloor A/2 \rfloor$번째 원소를 구했을때 리스트 A는 $\lfloor A/2 \rfloor$번째 원소를 중심으로 분할(SPLIT) 된다. 분할된 2개의 서브리스트를 사용하여 $\lfloor A.\frac{1}{4} \rfloor$ 번째 원소, $\lfloor A.\frac{3}{4} \rfloor$) 번째 원소를 중심으로 분할 하는 것도 쉽게 구할 수 있다. 이러한 방법으로 분할된 서브리스트의 갯수가 $|c.n^x|$ 될때까지 수행하는 알고리즘을 이진 병렬 선택 알고리즘 이라고 하자.

[정리1] 리스트 A를 이진 병렬 선택 알고리즘을 사용하여 첫번째 ($\lfloor A.\frac{1}{2} \rfloor$), 두번째 ($\lfloor A.\frac{1}{4} \rfloor$, $\lfloor A.\frac{3}{4} \rfloor$), 세번째($\lfloor A.\frac{1}{8} \rfloor$ $\lfloor A.\frac{3}{8} \rfloor$ $\lfloor A.\frac{5}{8} \rfloor$ $\lfloor A.\frac{7}{8} \rfloor$) 등으로 서브리스트의 크기가 $c|n^x|$ 될때까지 분할하는 시간 복잡도는 는 $O(n^x)$이다.

〈증명〉 $p(p=n^{1-x})$개 프로세서를 사용하여, 리스트 A를$\lfloor A.\frac{1}{2} \rfloor$번째 원소를 중심으로 분할하는 시간은 cn^x이다.(k=1) 이때 리스트 A는 (S_1, S_2)로 분리되며, $|S_1| = |S_2| = |A/2|$ 이다. 프로세서수를 $|p/2|$ 개씩 S_1과 S_2에 할당해 독립적으로 $|S_1(1/2)|$번째 원소와 $|S_2(1/2)|$번째 원소를 구하는 시간 복잡도는 $\frac{1}{2}cn^x$이다.

.
.
.

k번째일 경우의 시간 복잡도는 kcn^x가 된다. 따라서 서브리스트의 크기가 $c|n^x|$ 될 때까지 리스트 A를 분할하는 시간은 다음과 같다.

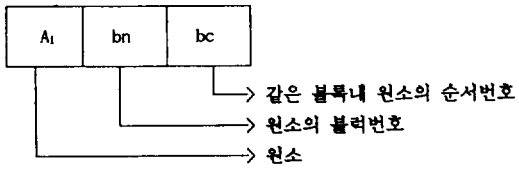
$$cn^x + \frac{1}{2}cn^x + \ldots\ldots \frac{1}{2^k}cn^x$$
$$= cn^x\left(1 + \frac{1}{2} + \frac{1}{2^2} + \ldots\ldots \frac{1}{2^k}\right)$$
$$= 2cn^x$$

그러므로 이진 병렬 선택알고리즘의 시간 복잡도는 $O(n^x)$이다.

3. 균형정렬 알고리즘

QUICK정렬은 리스트를 특정 키를 중심으로 리스트를 분리하여 특정키의 왼쪽은 특정키보다 적게 오른쪽은 특정키보다 큰 원소들로 분류한 다음 위와 같은 방법을 모든 원소들이 recursive하게 정렬될 때까지 수행한다. Akl[12] 알고리즘, parallel quicksort[2]알고리즘, hyper quicksort[1]알고리즘, balanced-binsort[15]알고리즘등이 이러한 부류에 속한다. 본 논문에서 제시하려는 알고리즘과 QUICK정렬 부류 알고리즘과 틀린점은 p개($p=n^{1-x}$)의 피보트키를 중심으로 리스트를 분류한다.(이때 나머지 모든 원소들은(p+1)개의 블록중 한개 위치에 선정하게 된다.) 분류된 리스트의 크기가 일정하지 않으므로 거의 일

정한 크기 $c|n^x|$ 개로 엔코딩한 다음 p개 프로세서로 독립적으로 분류된 리스트를 힙정렬 함으로써 리스트 전체를 정렬 할 수 있다. 이때 서브리스트의 각 원소의 자료구조는 (그림 1)과 같다. 분류방식은 먼저 p개 프로세서가 리스트를 분할하여 서브리스트를 만들고 각 서브리스트에서 임의의 원소를 공동기억 장소인 M에 보관한다(그림 2). (단 자료수 25, p=4) p개의 프로세서를 사용하여 M의 원소들을 정렬한 다음 M을 각 프로세서의 지역 변수 M으로 복제한다. 각 프로세서들은 독립적으로 각 서브리스트의 원소들이 어느 블록에 속하는지 이진 검색에 의하여 결정한다.



(그림 1) 원소의 자료구조
(Fig. 1) data structure of element

그 다음에 parallel prefix방식을 사용하여 각 블록 원소들의 원소수B(i)(i=1...p)를 누적시킨다. 모든 원소들이 해당 블록의 위치가 결정 되었다면 $S_{1max}<S_{2min}$, $S_{2max}<S_{3min}$과 같이 블록간 순서가 존재하게 된다.(S_{imax}, S_{imin}:블록 S_i의 최대, 최소값) 각 블록내 원소의 갯수 $|S_i|$, i=1...p는 일정하지 않으므로 $|S_i|<c|n^x|$ 이면 $c_1|n^x| \le |S_i| \le c_2|n^x|$ 가 될 때까지 인접한 다른 S_j, j≠i로 포인터를 옮긴다. 만약 $|S_i| \ge c|n^x|$ 를 만족하면 병렬선택 알고리즘을 사용하여 Si를 블럭의 크기가 $c|n^x|$ 인 서브리스트로 분리하는 포인타 엔코딩 기법을 사용한다. 포인타 엔코딩 후 각 서브리스트의 크기는 (그림 5) 거의 $c|n^x|$ 개 이므로 p개 프로세서가 각 블록에 할당되어 독립적으로 각 서브리스트를 정렬한다. 이때의 알고리즘을 단계적으로 설명하면 알고리즘 3-1과 같다.

PROCEDURE PASORT(A, n, p)

단계 1> p개 프로세서들이 ($p=n^{1-x}$) 배열A를 분할하고 분할된 서브리스트에서 임의의 원소를 선택한 다음 그 원소를 공통기억장소 M에 보관한다.

단계 2> p개의 프로세서를 사용하여 M를 정렬시킨다음 M을 각 프로세서의 기억. 장소로 옮긴다.

단계 3> p개 프로세서들은 독립적으로 분할된 서브리스트의 원소들이 어느 블록에 속해 있나를 이진검색에 의해서 결정한다. 또한 원소들이 해당 블록내 이동될 위치도 카운터에 의해서 결정한다. 단계 3>가 끝나면 각 블럭내 원소의 최대 카운터 B(i)(i=1...p)가 결정된다.

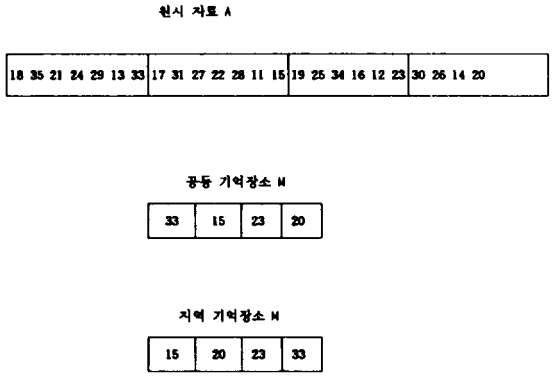
단계 4> B(i)(i=1...p)를 parallel prefix 알고리즘으로 계산한다.

단계 5>
CASE
: $|S_i| < c|n^x|$: $c_1|n^x| < |S_i| < c_2|n^x|$ 가 될때까지 인접한 다른 블록으로 포인터를 옮긴다. //c, c_1, c_2 : 상수 //
: $|S_i| \ge c|n^x|$: 이진 병렬 선택 알고리즘을 사용하여 S_i를 크기가 $c|n^x|$ 가 될때까지 분할한다.
END CASE
// $|S_i|$ i=1....p를 적당한 크기(거의 $c|n^x|$)로 될 수 있게 포인터를 옮기거나, S_i를 적당한 크기로 분할한다.//

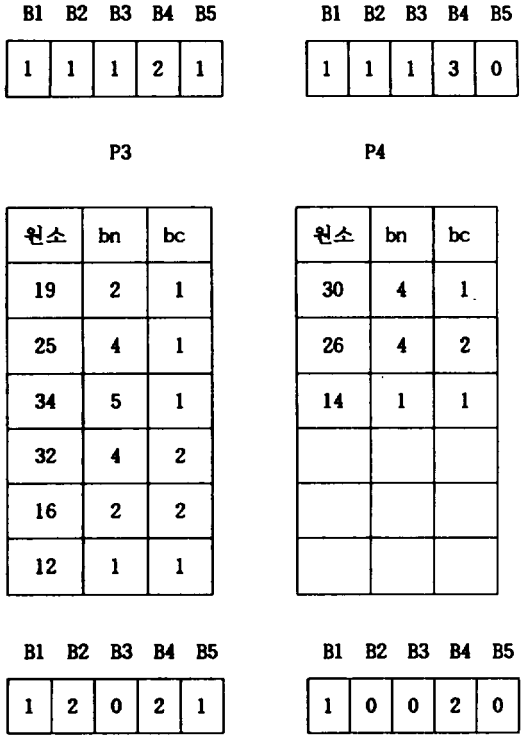
단계 6> 각 블록에 1개의 프로세서를 할당하여 힙정렬을 한다.

END PASORT

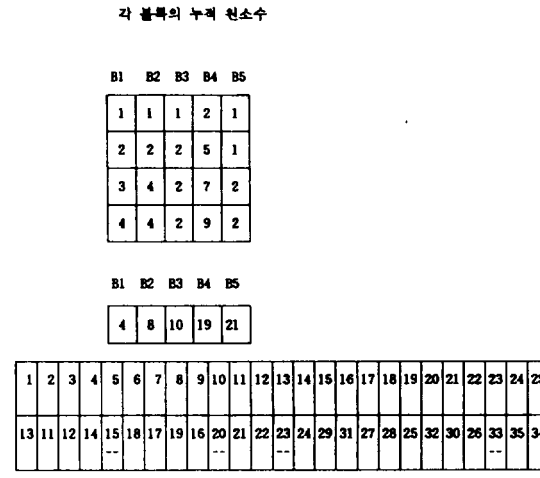
알고리즘 3-1 엔코딩 방식에 의한 균형 정렬 알고리즘



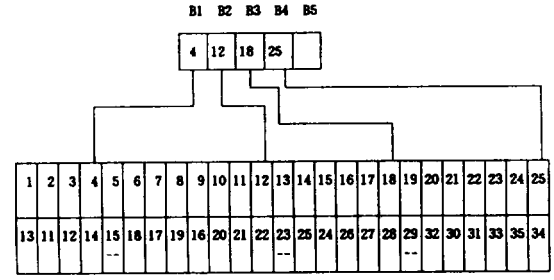
(그림 2) 공동기억장소와 지역기억장소 M
(Fig. 2) shared memory and local memory



(그림 3) 서브리스트의 자료들이 어느블록에 속하는가 (bn), 같은 블록내 자료카운터(bc)를 나타낸 결과.
(Fig. 3) to which blocks the data blongs(bn), result of data counter in same block(bc)



(그림 4) P개의 피보트 키를 중심으로 A를 분류시킨 결과
(Fig. 4) result of split A with number of p pivot key



(그림 5) 블록을 카운터 엔코딩한 결과
(Fig. 5) result of block using encoding pointers

4. 알고리즘 분석

알고리즘 3-1의 단계1)은 임의의 원소를 선택함으로 다음과 같이 O(1)에 수행할 수 있다. 단계2)는 p개 프로세서가 p개($|n^{1-x}|$)개 원소를 정렬함으로 $O(n^x \log n^x)$를 넘지 않는다. 단계3)는 p개 프로세서들이 동시에 수행하며, 이진검색을 수행하므로 $O(n^x \log n^x)$을 넘지 않는다. 단계4)는 $|n^x|$개의 원소를 parallel prefix 알고리즘으로 수행하므로 $O(n^{1-x})$를 넘지 않는다.

4.1 평균적 경우

정렬되지 않은 리스트의 크기가 $(-\infty, +\infty)$라

고 가정하고 p개의 프로세서들에 의해서 p개의 분리(split)키를 중심으로 리스트는 (p+1)개의 블록으로 구분 된다. 단계 1)에서 표본 원소들을 일정하게 선택하였다면 각 블록의 원소 갯수는 거의 일정 한 크기의 비표본 원소가 존재한다. 우리가 평균의 경우를 구하기 위하여 가장 관심이 있는 부분은 블록의 크기이다. 왜냐하면 단계 5>에서 블록이 결정되면 단계 6>에서 각 블록에 1개의 프로세서가 할당되어 각 블록을 정렬하기 때문이다.

(보조정리 1)

$1 \le i \le (p+1)$이고 $1 \le j \le (n-p)$일때, i번째 블록에 원소의 갯수가 j개일 확률은

$$P_i(j) = \binom{n-j-1}{p-1} \cdot \binom{n}{p}$$이다.

(보조정리 2)

$$\sum_{j=1}^{n-p} \binom{n-j-1}{p-1} j^2 = \binom{n}{p} \frac{(n-p)(2n-p)}{(p+1)(p+2)}$$

증명)

$u_j = \binom{n-j-2}{p-1} \cdot (j+1)^2$, $u_0 \neq 0$라고하면

$$\frac{u_{j-1}}{u_j} = \frac{(j+(2))}{1} \cdot \frac{(j+(2))}{(j+(1))} \cdot \frac{(j+(-n+p+1))}{(j+(-n+2))} \cdot \frac{1}{(j+1)}$$

위 식을 이용하여 점화식으로 풀면,

$$\sum_{j=1}^{n-p} \binom{n-j-1}{p-1} j^2 = u_0 {}_3F_2 \left(\begin{matrix} 2 & 2 & (-n+p+1) \\ - & 1 & (-n+2) \end{matrix} \middle| 1 \right)$$

위 식을 Gauss함수 공식을 이용하면 APPENDIX III[9]

$${}_3F_2 \left(\begin{matrix} a & 1+(1/2)a-N \\ (1/2)a & b \end{matrix} \middle| 1 \right) = (b-a-1-N) \frac{(b-a)N-1}{(b)N}, u_0 = \binom{n-2}{p-1}$$

이므로

$$\sum_{j=1}^{n-p} \binom{n-j-1}{p-1} j^2 = \binom{n-2}{p-1} \cdot (p-2n) \cdot \frac{(-n)_{n-2-p}}{(-n+2)_{n-1-p}}$$

$$\sim \binom{n}{p} \cdot \frac{(n-p)(2n-p)}{(p+1)(p+2)}$$

〈정리 2〉 리스트 A를 $p(p=n^{1-x})$개의 피보트 키를 중심으로 (p+1)개의 블록으로 리스 트를 분리했을때 블록의 평균 원소의 갯수는 $O(n^x)$이다.

(증명) 임의의 블록의 원소 갯수는 많아야 (n-p)이고, 블록의 수는 많아야 (p+1)개이므로 블록의 평균 원소 갯수는 다음과 같다.

$$\sum_{i=1}^{p+1} \left[\frac{1}{n} \sum_{j=1}^{n-p} (p_i(j)) j^2 \right]$$

보조 정리 1)에 의하여

$$= \frac{1}{n} \frac{(p+1)}{\binom{n}{p}} \sum_{j=1}^{n-p} \binom{n-j-1}{p-1} \cdot j^2$$

보조 정리 2)에 의하여

$$= \frac{1}{n} \frac{(n-p)(2n-p)}{(p+2)} < \frac{2n}{p}$$

$p = n^{1-x}$이므로 $= \frac{2n}{n^{1-x}} = 2 * n^x$

따라서 블록의 평균 원소 갯수는 $O(n^x)$이다.

4.2 최악의 경우

알고리즘 3-1의 최악의 경우는 S_i블록에 $|n-p|$개의 원소가 몰려 있을 경우이다. 따라서 단계5)에서 S_i 를 $c|n^x|$크기 될 때까지 이분 하므로 그때의 시간복잡도W(n)은 정리 1에 의하여 다음과 같다.

$$W(n) = c(n^x + \frac{1}{2} n^x + \cdots + \frac{1}{2^k} n^x)$$

$$= 2cn^x$$

$$= O(n^x)$$이다.(단 $\frac{1}{2^k} n^x$는 서브리스트의 크기가 $c|n^x|$가 될때까지 시간)

단계 6)에서는 p개 프로세들이 독립적으로 힙정렬 하므로 시간복잡도는 $O(n^x \log n^x)$ 이다. 따라서 알고리즘 3-1의 시간 복잡도는 평균의 경우 $O(n^x \log^n)$ 이고, 최악의 경우도 $O(n^x \log^n)$이다. 비용은 (프로세서수 * 최악의 경우 시간복잡도)=$(O(n^x \log n^x * n^{1-x})) = O(n \log^n)$이다.

4.3 자료 이동도

알고리즘 3-1의 단계 1에서는 p개 프로세서들이 자료를 임의로 선택하여 공통 기억장소에 M에 보관하므로 원시자료의 이동은 없다. 단계 2에서도 공통 기억장소의 자료를 사용하므로 원시자료의 이동은 없다.

단계 3과 단계 4에서는 각 원시자료의 위치를 계산만 하므로 자료의 이동은 없다. 단계 5에서 최악의 경우 이진 병렬 선택 알고리즘을 사용하여 자료의 크기를($c \mid n^x \mid$)가 될 때까지 분할하므로 자료이동도는 $O(n^{1-x})$를 넘지 않는다. 단계 6에서는 각 블록의 크기가 거의 $c \mid n^x \mid$ 각 블록을 p개 프로세서들이 독립적으로 처리하므로 $O(n^x)$를 넘지 않는다. 따라서 알고리즘 3-1의 자료 이동도는 $O(n^{1-x}+n^x)$이다.

4.4 다른 알고리즘들과 비교

일반적으로 병렬정렬 문제는 크게 2개의 범주로 나뉘어지고 있다. 그중 하나는 병합 방식에 의한 정렬이고 또 하나는 분할 방식에 의한 정렬이다. 그러나 전자의 문제점은 많은 자료의 이동에 따른 프로세서들의 스케쥴이나 동기화 상에 문제가 있으며, [8] 후자의 문제점은 자료를 분할한 후 분할된 자료의 크기가 일정하지 않다는 점이다. 이러한 점을 극복하기 위해서 Colemen 등은 [11] 통계적 자료 표본을 사용했으나 유감스럽게도 자료의 특성을 미리 안다는 것은 거의 불가능한 일이다. 또한 [11] 방법을 개선해서 분할된 자료의 크기가 거의 일정하게 하는 방식이 Hanmao와 Jonathan[8]에 의해서 제안됐다.

제시된 알고리즘과 Hanmao[8] 알고리즘이 틀린점은 Hanmao 알고리즘은 regular sample 방식

〈표 1〉 Ak1 알고리즘과 제시된 알고리즘의 비교
〈Table 1〉 compare algorithm 3-1 with AKL at el algorithm

	Akl 알고리즘	제시된 알고리즘
컴퓨터 종류	SIMD-PRAM	SIMD-PRAM
프로세서 수	$p(p=n^{1-x}, 0<x<1)$	$p(p=n^{1-x}, 0<x<1)$
응용여부	응용 가능하지 않음	응용 가능하지 않음
기본알고리즘	p개 프로세서를 사용해서 $\lfloor A/4 \rfloor$, $\lfloor A/2 \rfloor$, $\lfloor A(3/4) \rfloor$번째 원소를 구한다음 자료들을 네개의 서브리스트로 분할하고, 분리된 서브리스트들을 recursive하게 처리한다.	p개 프로세서를 이용하여 크기 $\lfloor n^x \rfloor$ 인 서브리스트를 만든다음 그 중 임의의 원 $\lfloor A/2 \rfloor$, $\lfloor A/(3/4) \rfloor$번째 원소들을 선택해서 그것을 정렬시키고 나머지 모든 원소들이 어느 블록에 위치하는지를 계산해서 (parallel prefix 사용) 모든 원소들을 (p-1)개의 블록으로 나눈다. 이때 각 블록의 크기가 일정하지 않음으로 엔코딩 방법으로 크기가 거의 비슷하게 할 수있게 포인터를 변형한 다음 p개 프로세서들이 각 블록에 할당되어 정렬 시킨다.
시간 복잡도	$O(n^x \log n)$	$O(n^x \log n)$
자료 이동도	$O(n^x \log n)$	$O(n^{1-x}+n^x)$
비 용	$O(n^{1-x}) * O(n^x \log n) = O(n \log n)$	$O(n^{1-x}) * O(n^x \log n) = O(n \log n)$

〈표 2〉 다른 정렬알고리즘과 제시된 알고리즘의 비교표
〈Table 2〉 compare algorithm 3-1 with other algorithm

이 름	컴 퓨 터	프로세서수	시간 복잡도
Kruskal	CRCW-PARM	n	O(nloglog n /logloglog n)
Hirschberg	CRCW-PARM	$n^{1-(1/x)}$	O(xlog n)
Preparata	EREW-PARM	n^2	O(log n)
Shiloach	EREW-PARM	n/nlog n	O(n(log n))
Akl	CRCW-PARM	p(1<p<n)	$O((\log^2 p + n/p) * \log n))$
Akl	EREW-PARM	n^{1-x}	$O(n^x \log n)$
Chleous	CRCW-PARM	n	O(log n)
제시된 알고리즘	EREW-PARM	n^{1-x}	$O(n^x \log n)$
Hanmao	MIMD	$p^2<=n$	O((n/p)·log n)

을 사용하여 블록의 크기를 거의 일정하게 한 반면 제시된 알고리즘은 각 블록의 자료 크기를 계산해 적당한 크기로 엔코딩함으로써 블록의 크기를 거의 일정하게 했다. 또한 Hanmao 알고리즘은 병합 방식을 사용한 반면 제시된 알고리즘은 복수 피보트 키에 의한 분할 방식을 사용했다. 〈표 1〉은 본 논문에서 제시한 알고리즘과 Akl등이 제시한 알고리즘을 컴퓨터종류, 프로세서 수, 기본 알고리즘, 시간복잡도,자료이동도, 비용등을 비교한 도표이다. 또한 〈표 2〉는 지금까지 알려진 알고리즘과 제시한 알고리즘의 컴퓨터, 시간복잡도, 비용등을 비교한 도표이다. 특히 제시한 알고리즘과 Akl 등이 제시한 알고리즘과의 차이점은 다음과 같다.

(1) Akl 알고리즘은 3개의 피보트 키를 사용하였으나 제시된 알고리즘은 $p(p=n^{1-x}, 0<x<1)$개의 피보트 키를 사용했다.

(2) Akl 알고리즘은 자료들을 피보트 키를 중

심으로 자료들을 이동함으로써 분류 했으나 제시된 알고리즘은 자료들을 피보트 키를 중심으로 자료들의 위치를 계산함으로써 자료의 이동 시간을 줄였다.

(3) Akl 알고리즘은 자료들을 정렬 시키기 위해서 서브리스트들을 병합시켰으나 제시된 알고리즘은 자료들의 위치를 계산하는 방법으로 자료들을 정렬 시켰다.

5. 결 론

SIMD EREW-PRAM 모델 병렬 컴퓨터에서 병렬 알고리즘의 효율성을 평가하기 위한 가장 중요한 요소는 비용(병렬알고리즘의 최악의 실행 시간 * 프로세서 수)과 자료 이동에 따른 프로세서들의 스케쥴 문제이다. 본 논문에서 제시하는 알고리즘은 이 두가지를 만족할 수 있는 알고리즘을 제시하는데 목적이 있다. 즉 제시된 병렬 알고리즘을 Parallel prefix 알고리즘을 사용하여 자료들을 다중 피보트키를 중심으로 각 원소의 해당 블록의 위치를 계산하여 원소를 옮김으로서 자료의 이동 시간을 줄일 수 있었다. 또한 p개의 프로세서들에 의해서 분할되는 각 블록의 크기를 적당한 크기로 엔코딩하는 기법을 사용해 각 블록의 크기가 거의 일정하게 했다. 본 논문에서 제안한 알고리즘을 분석해 본 결과 시간복잡도는 평균의 경우 $O(n^x \log^n)$이고, 최악의 경우도 $O(n^x \log^n)$이며 비용은 $O(n \log^n)$이고, 자료이동도는 $O(n^{1-x}+n^x)$이다.

따라서 제시된 균형 정렬 알고리즘은 최적이다.

참 고 문 헌

[1] B. Wagar, 'Hyper quicksort-a fast sorting algorithm for hypercubes. In Hypercubes Multiprocessors, SIAM, 1987.

[2] B. C. Chleous and I.Vato. 'Parallel Quicksort', journal of parallel and Distributed computing 11, pp. 332-337, 1991.

[3] C. P. Kruskal, L. Rudolph, and M. Snir, 'The power of parallel prefix', IEEE Trans, Computers, Vol. C-34, No. 10, pp. 965-968, 1985.

[4] C. P. Kruskal, "Searching, merging and sorting in parallel computation", IEEE Trans comput., Vol. C-32, pp. 942-946, 1983.

[5] D. E. Knuth, 'The Art of Computer programming Vol. 3: sorting and searching', Addison-Wesley, 1973.

[6] D. S. Hirschberg, 'Parallel graph algorithms without memory conflicts', Proc. 20th Allerton Conf. Commun, Comtrol. Comput, Monticello, pp. 257-263, 1982.

[7] Evans, D. J., and Yousif, N. Y. "The parallel neighbor sort and two-way merge algorithm. Parallel comput. 3. pp. 85-90, 1986.

[8] Hanmao Shi and Jonathan Schaeffei. "Parallel sorting by Regular Sampling." journal of parallel and Distributed computing. Vol. 14, No 4, pp. 361-372, 1992.

[9] I. J. Slater, 'Generalized Hyper geometric Functions', Cambridge University, press, 1966.

[10] M. J. Flynn, 'Very high speed computing systems', proc. IEEE 54 pp. 1901-1909, 1966.

[11] N. S. Colemen, D. J. Quammen and P. Y. Wang, 'A parallel Randomized Sorting Algorithm', International conferance on parallel processing, pp. Ⅲ-293-Ⅲ-296, 1992.

[12] S. G. Akl, The Design and Analysis of Parallel Algorithms, Prentice-Hall, 1989.

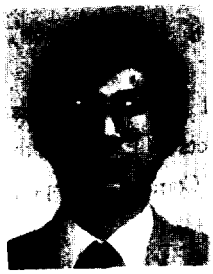
[13] S. G. Akl and N. Santoro, 'Optimal Parallel Merging and Sorting without Memory conflicts', IEEE Trans. Computers, Vol. C-36, No. 11, pp. 1367-1369, 1987.

[14] S. G. Akl, 'An Optimal Algorithm for Par-

allel Selection', Information Processing Letters 19, pp. 47-50, 1984.

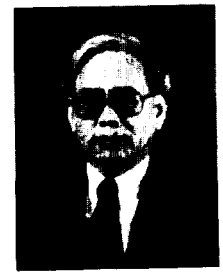
[15] Y. Shiloach and U. Vishkin, "Finding the maximum, merging and sorting in a parallel computation model." J. algorithm, Vol. 2, pp. 88-102, 1981.

[16] Y. Won and S. Sahni, 'A balanced binsort for hypercube multicomputers', Journal of supercomputing 2, pp. 435-448, 1988.



홍 성 수

광운대학교 전산학과 졸업
쌍용 컴퓨터실 근무
1983년 광운대학교 전산학과 석사학위 취득
1990년 광운대학교 전산학과 박사학위 취득
현재 호서대학교 컴퓨터공학과 부교수 재직
관심분야 : 알고리즘, 병렬처리, 자연어 처리



심 재 홍

1967년 서울대학교 수학과 졸업
1980년 고려대학교 대학원(이학석사)
1981~1988년 경희대학교 대학원(이학박사)
1984~1986년 정보과학회 부회장 역임
현재 광운대학교 교수로 재직중
관심분야 : 그래프알고리즘, 그래픽스, 수치해석