공학 연구논문집 제27권 제2호 pp. 1~18, 울산대학교
Journal of Enginering Research Vol. 27, No. 2, 1~18, 1996
University of Ulsan.

하이퍼큐브 구조에서의 대용량 데이터 정렬을 위한 병렬 Bitonic 정렬 알고리즘 연구

양 명 국
전기공학과

<요 약>

데이터 정렬은 다양한 데이터 조작 응용 프로그램의 핵심 요소로 널리 활용되어, 효율 적인 정렬 알고리즘 개발을 위한 연구가 다각도에서 수행되고 있다. Bitonic 정렬 알고리즘은 단계별로 전체 데이터가 데이터 쌍을 형성하고 상호 비교-정렬을 반복하는 특성이 있어, 알고리즘의 병렬화가 용이하고, 대상 병렬 시스템의 상호 연결망 특성에 따라 효과적인 데이터 정렬을 실행할 수 있다. 본 연구에서는 하이퍼큐브 구조의 상호 연결 망이 Bitonic 정렬의 단계별 비교 패턴과 일치하는 점에 착안, 먼저 하이퍼큐브 시스템에 Bitonic 정렬 알고리즘을 적용하고, 알고리즘 수행에 소요되는 시간을 분석, 계산하였다. 나아가서 프로세서 개수가 한정된 시스템에서 다량의 데이터를 정렬할 경우 네트워크 내부의 데이터 충돌을 피하고 전체 프로세서 이용률을 최대화시키는 두 가지 병렬 Bitonic 정렬 알고리즘(High-Order Interleaving(HOI)정렬 기법과 Low-Order Interleaving(LOI)정렬 기법)을 제안하고, 이들 알고리즘 수행에 소요되는 시간을 구하였다. HOI 방식은 알고리즘이 비교적 단순할 뿐 아니라, 알고리즘 수행에 소요되는 시간도 경제적인 것으로 나타나 성능 면에서 우수한 방식으로 판명되었으나, 응용 분야에서 요구하는 정렬 후 데이터 배열 상태에 따라 정렬 기법이 선택되어야 한다.

Design and Evaluation of Parallel Bitonic Sorting Algorithm on Hypercube System For Large Data Sorting

Yang, Myung-Kook
Department of Electrical Engineering

<Abstract>

Data sorting is well known as an elementary process of the various data manipulation problems. Bitonic sorting algorithm is an efficient method to be parallelized due to its systematic repetition characteristic of the partial comparison and rearrangement. The comparison pattern of Bitonic sorting is completely match with the Hypercube interconnection. This provides the motivation of this research. Mapping technique of the Bitonic sorting algorithm on a Hypercube system and its time complexity are discussed and calculated. A comparative study is conducted to show the effectiveness of the Hypercube architecture to support the Bitonic sorting algorithm. Two parallel Bitonic algorithms are proposed and analyzed to sort a large data under the limited number of Hypercube connected processors. These are High-Order Interleaving(HOI) Bitonic sorting algorithm and Low-Order Interleaving(LOI) Bitonic sorting algorithm. Both algorithms are designed not only to avoid the data collision on the Hypercube network but also to maximize the processor utilization. It is shown that the HOI technique is simple to apply and takes less time to execute the sorting. The selection of algorithm, however, can be determined according to the requirement of the application problems.

1. 서 론

정렬 알고리즘은 다양한 응용 프로그램의 데이터 검색, 보완, 저장 등 많은 데이터 조작의 핵심 요소로써 널리 활용되며, 여러 각도에서 연구의 대상이 되어왔다. Bitonic 정렬 알고리즘은 부분 정렬과 병합(merge)을 일정한 패턴으로 반복 수행하여 전체 정렬을 완성하는 기법이다[1]. 이때 단계별 정렬 작업(operation)에서 모든 데이터가 비교-부분 정렬의 과정을 반복하게 되므로, 다중 프로세서 시스템을 활용할 경우 다수의 프로세서를 동시에 작동시켜, 각 프로세서 집합이 해당 부분 정렬을 수행하게 함으로써 빠른 시간에 대량의 데이터를 효율적으로 정렬할 수 있다.

Batcher[1]와 Stone[2]은 Bitonic 정렬 알고리즘을 Multistage Interconnection Network(MIN)을 기반으로 하는 시스템에 적용하고, 이의 성능을 분석하였다. 이들은 각각 Omega 네트워크와 Perfect shuffle 네트워크를 대상 네트워크로 설정하고, 네트워크 내부의 스위칭 소자에 간단한 비교 기능을 추가하여 $N(=2^n)$개의 데이터를 정렬하였다. 정렬의 성능을 평가한 결과 N개의 데이터를 정렬하는데 소요되는 시간은 $O(\log^2 N)$으로 계산되었다[1,3]. 또한, Thompson과 Kung[4] 그리고 Nassimi와 Sahni[5] 등은 메쉬 구조의 다중 처리 시스템상에서 병렬 Bitonic 정렬 알고리즘을 구현하였다. 메쉬 구조는 프로세서간의 거리가 상호 위치에 따라 일정하지 않으며, 또한 시스템 크기가 커질 경우 프로세서간의 평균 거리가 증가하는 특성을 가지고 있다. Thompson과 Kung은 이같은 메쉬 구조의 특성을 고려하여 각 프로세싱 소자(PE)의 인식표(id, identifier) 부여에 변화를 준 Shuffled row-major indexing 기법을 제안하고, 이를 이용한 Bitonic 병합 알고리즘을 발표하였다. Shuffled row-major indexing 기법은 메쉬 구조상에서 Bitonic 정렬을 수행 함에 있어 PE간의 통신을 국지적 패턴으로 유도하여 데이터 전송 거리를 줄여 주는 효과를 제공한다.

본 연구에서는 먼저 하이퍼큐브 구조를 갖는 다중 처리 시스템을 대상 시스템으로 선정하고, 병렬 Bitonic 정렬 알고리즘을 적용하여 성능을 분석하였다. 또한, 분석 결과를 다른 시스템에서의 성능과 비교하여, 하이퍼큐브 시스템이 Bitonic 정렬 알고리즘을 구현하기에 적합한 구조를 가지고 있음을 입증하였다. 나아가서 병렬 알고리즘의 성능이 대상 시스템의 구조에 따라 변화함[6]을 재조명 하였다. 하이퍼큐브 시스템의 상호 연결 패턴(Interconnection pattern)은 Bitonic 정렬의 비교 패턴과 일치하기 때문에 알고리즘 수행 시 병렬 처리가 용이하며, 인접 프로세서 간의 통신에 소요되는 시간이 미세하다고 가정할 경우 프로세서의 이용도가 100%에 접근한다.

기존에 발표된 연구들은 PE의 개수$(P=2^p)$와 정렬하고자 하는 데이터의 개수

($N = 2^n$)가 동일한 경우를 가정하고 알고리즘을 구현하였다. 본 연구에서는 이같은 가정을 좀 더 일반화시켜, 데이터의 개수가 프로세서의 개수보다 많은 경우($n = p + i, I = 1, 2, 3, \ldots$), 병렬 처리 시 네트워크 충돌을 피하고 전체 프로세서의 이용률을 최대화할 수 있는 두 가지 병렬 Bitonic 정렬 알고리즘을 제안하고, 이를 하이퍼큐브 시스템에 적용하여 성능을 분석하였다.

2. 하이퍼큐브 시스템상에서의 Bitonic 정렬

그림 2.1 은 하이퍼큐브 시스템의 상호 연결 패턴과 Bitonic 정렬 과정의 상호 비교 패턴(Comparison patterns)을 보여주고 있다. 그림 2.1(a)의 4_큐브 시스템과 그림 2.1(b)의 16-데이터 Bitonic 정렬 과정을 면밀히 검토해 보면, 하이퍼큐브 시스템의 상호 연결 패턴과 정렬의 단계별 비교 패턴이 오차 없이 일치하는 것을 발견할 수 있다. 따라서 정렬하고자 하는 데이터가 하이퍼큐브 시스템의 프로세싱 소자(PE)에 하나씩 분산 배치되어 있을 경우 전체 데이터 정렬은 하이퍼큐브 시스템의 상호 연결 패턴을 따라 정렬 단계별로 1-Link(1-Hop) 데이터 이동만으로 가능하게 된다.

그림 2.1(b)에 보인 16 개 데이터의 Bitonic 정렬 알고리즘을 구체적으로 4_큐브 시스템에 적용하여 병렬 처리 과정을 단계별로 정리하면 다음과 같다.

1 단계: 1_큐브 정렬,
step 1: [*(0-1), (3-2), . . ., (15-14)*]

2 단계: 2_큐브 정렬,
step 1: [*(0-2), (1-3), . . ., (15-13)*]
step 2: [*(0-1), (2-3), . . ., (15-14)*]

3 단계: 3_큐브 정렬,
step 1: [*(0-4), (1-5), . . ., (15-11)*]
step 2: [*(0-2), (1-3), . . ., (15-13)*]
step 3: [*(0-1), (2-3), . . ., (15-14)*]

4 단계: 4_큐브 정렬,

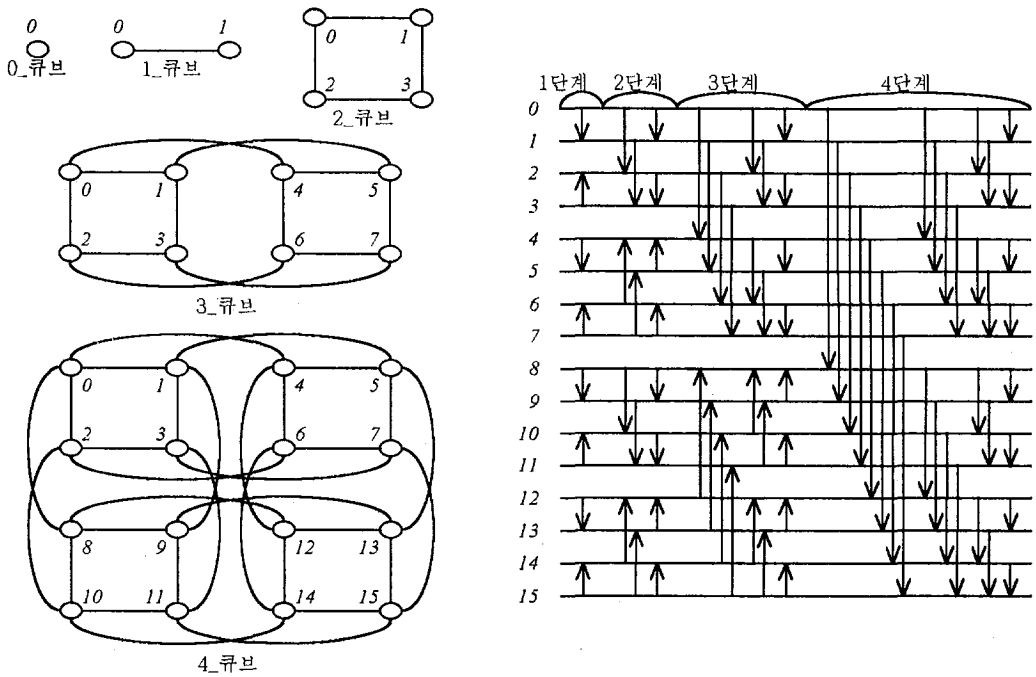
step 1: [*(0-8), (1-9), . . ., (7-15)*]

step 2: [*(0-4), (1-5), . . ., (11-15)*]

step 3: [*(0-2), (1-3), . . ., (13-15)*]

step 4: [*(0-1), (2-3), . . ., (14-15)*]

그림 2.1 과 앞서 기술한 병렬 처리 알고리즘의 단계별 step 들에 표기된 기울임체 숫자들은 하이퍼큐브 PE 들의 인식표(identifier)를 나타내며, 병렬 알고리즘의 *(i-j)*는 PE_i 와 PE_j 가



(a) 하이퍼큐브 시스템의 상호 연결 패턴 (b) Bitonic 정렬(16-데이터)

그림 2.1 하이퍼큐브 시스템의 상호 연결 패턴과 Bitonic 정렬

가지고 있는 데이터를 비교하여 작은 데이터는 PE_i 에 그리고 큰 데이터는 PE_j 에 부분 정렬함을 의미한다. 그러므로 하나의 비교-정렬쌍 *(i-j)*를 부분 정렬하기 위해서는 PE_i 와 PE_j 간에 2 회의 데이터 이동(데이터 소집 및 분산)과 1 회의 데이터 비교 과정이 필요하게 된다.병렬 알고리즘의 각 step 에 나열된 비교-정렬 쌍들은 하이퍼큐브 구조의 연결 패턴에 따라, 상호 간섭 없이 오직 1 개의 연결 Link 를 통하여 동시

에 데이터를 주고 받을 수 있다. 다시 정리하면 하이퍼큐브 구조를 갖는 시스템에 Bitonic 정렬 알고리즘을 병렬화 하여 구현할 경우 각 step에 나열된 비교-정렬 쌍들은 동시에 해당 PE끼리 데이터를 주고 받으며 병렬 처리 되는데, 이때 소요되는 시간은 2회의 데이터 이동 시간과 1회의 데이터 비교시간으로 계산할 수 있다. 따라서 주어진 하이퍼큐브 시스템에서 1-Hop 데이터 이동에 소요되는 시간을 t_r, 그리고 PE내에서 2개의 데이터를 상호 비교하는데 소요되는 시간을 t_c 라고 정의하면, 각 step별 소요시간은 $2t_r + t_c$ 가 된다. 따라서 4_큐브 시스템에서 16-데이터를 정렬하기 위하여 거쳐야 하는 4단계의 각 단계별 소요시간을 계산하면 다음과 같다.

$$\begin{aligned}
&1\text{단계} : \ (2t_r + t_c) \\
&2\text{단계} : \ 2(2t_r + t_c) \\
&3\text{단계} : \ 3(2t_r + t_c) \\
&4\text{단계} : \ 4(2t_r + t_c)
\end{aligned}$$

그러므로 전체 소요시간은 $10(2t_r + t_c)$로 계산된다.

지금까지 살펴본 4_큐브 시스템상에서 16개의 데이터를 정렬하는 과정을 일반화하여 n_큐브 시스템상에서 $N(=2^n)$개의 데이터를 정렬하는 경우를 생각해보자. 먼저 전체적으로 정렬을 완성하는데 n 개의 단계를 거쳐야 하며, 각 단계에서는 해당 큐브 별로 정렬을 수행하게 된다. 임의의 k_큐브 정렬은 k 개의 step을 지나며 완성되며, 이에 소요되는 시간 $\tau(2^k)$는 식(2-1)과 같이 계산할 수 있다.

$$\tau(2^k) = k*(2t_r + t_c) \tag{2-1}$$

따라서, $N(=2^n)$개의 데이터를 완전 정렬하는데 소요되는 시간 T(N)은 다음과 같이 계산된다.

$$\begin{aligned}
T(N) &= \tau(2^1) + \tau(2^2) + \ldots + \tau(2^k) + \ldots + \tau(2^n) \\
&= 1*(2t_r + t_c) + 2*(2t_r + t_c) + \ldots + k*(2t_r + t_c) + \ldots + n*(2t_r + t_c) \\
&= \left\{\frac{n*(n+1)}{2}\right\}*(2t_r + t_c)
\end{aligned}$$

$$= \left\{\log N * (\log N + 1)\right\} * t_1 + \left\{\frac{\log N * (\log N + 1)}{2}\right\} * t_c \qquad (2\text{-}2)$$

즉 $N(=2^n)$개의 데이터를 완전 정렬하기까지 n 단계의 부분 정렬 과정을 거쳐야 하는데, 1 단계는 1_큐브 정렬, 2 단계는 2_큐브 정렬, ..., k 단계는 k_큐브 정렬, ..., 그리고 마지막 n 단계는 n_큐브 정렬을 수행하게 되어, 전체 소요시간을 계산하면 식(2-2)와 같다.

3. 상호 연결망과 알고리즘 성능

본 절에서는 Multistage Interconnection Network(MIN)과 Mesh를 기반으로 하는 병렬 컴퓨터상에 Bitonic 알고리즘을 적용하여 성능을 분석한 기존의 연구 결과들을 참조하여 2 절에서 얻은 결과와 비교 평가한다. 나아가서 병렬 알고리즘의 효율적 운용 여부는 알고리즘을 수행하는 병렬 컴퓨터의 구조적 특성에 의해 크게 변화함을 재조명하고자 한다.

Batcher[1]와 Stone[2]은 MIN 구조를 갖는 시스템에 Bitonic 알고리즘을 적용하고, 성능을 계산하였다. 먼저 Batcher는 여러가지 Multistage Interconnection Network 중 Bitonic 비교 패턴을 효과적으로 지원할 수 있는 오메가 네트워크를 선택하여 Bitonic 정렬 알고리즘을 구현하였다. Bitonic 정렬 알고리즘의 부분 정렬 과정이 네트워크를 경유하여 이루어지게 하기 위하여 Batcher는 오메가 MIN 내부의 각 스위칭 소자에 데이터 비교 기능과 Bitonic 정렬 알고리즘의 단계별 각 step의 비교-정렬 패턴에 따라 전체 네트워크를 제어할 수 있는 제어 기능을 추가하여 Bitonic 정렬 네트워크를 제안하고, 이의 성능을 분석하였다. 오메가 MIN을 기반으로 하는 Batcher의 Bitonic 정렬 네트워크는 알고리즘을 매우 효율적으로 지원하여 $N * N$ 시스템에서 $N(=2^n)$ 개의 데이터를 정렬하는데 소요되는 시간 $T(N)_{MIN}$을 구하면 다음과 같이 계산된다.

$$T(N)_{MIN} = n^2 * t_{r(MIN)} + \left\{\frac{N * (N+1)}{2}\right\} * t_c$$

$$= (\log)^2 * t_{r(MIN)} + \left\{\frac{\log N * (\log N + 1)}{2}\right\} * t_c \qquad (3\text{-}1)$$

$N * N$ 오메가 네트워크는 $n(=\log N)$개의 stage로 구성된다. $N(=2^n)$개의 데이터를 Bitonic 알고리즘으로 정렬하기 위하여 n 단계를 거쳐야 하며, 알고리즘의 각 단계는 정렬 네트워크의 n 개 stage를 완전히 통과하여 완성된다. 따라서 식(3-1)의 Communication routing 시간은 $n*n*t_r$로 계산된다. Stone[2]은 1-stage perfect shuffle 네트워크와 외부에 간단한 데이터 저장 버퍼를 갖는 시스템에 Bitonic 알고리즘을 적용하였다. 또한 Batcher의 정렬 네트워크에서와 같이 Perfect shuffle 네트워크 제어 기능과 내부의 각 스위칭 소자에 데이터 비교 기능을 추가하여 알고리즘을 효과적으로 지원하게 하였고, $N=2^n$개의 데이터를 정렬하는데 소요되는 시간을 계산한 결과 식(3-1)과 동일한 결과를 얻었다.

Thompson과 Kung[4] 그리고 Nassimi와 Sahni[5] 등은 Mesh 구조의 병렬 시스템에 Bitonic 정렬 알고리즘을 적용하고, 성능을 평가 하였다. Mesh 구조의 상호 연결망은 MIN이나 Hypercube의 경우와는 달리 Bitonic 정렬의 비교 패턴과 상이한 연결 구조를 가지고 있어, 알고리즘을 효과적으로 지원함에 있어 한계를 보였다. 더욱이 시스템의 크기가 커지면 PE 상호간의 평균 거리가 증가하여, Bitonic 정렬의 비교-정렬쌍 내부의 Communication routing이 복잡해지고, 이에 따른 오버헤드도 기하급수적으로 증가한다. Thompson과 Kung은 이같은 Mesh 구조의 특성을 고려하여 새로운 PE 인식표(identifier) 부여 기법인 'Shuffled row-major indexing' 기법을 제안하였다. 'Shuffled row-major indexing' 기법은 Mesh 구조에 Bitonic 정렬 알고리즘을 구현함에 있어 각 비교-정렬 쌍의 거리를 줄여주고 PE 간의 통신을 국지적 패턴으로 유도하여 상호 간섭을 배제함과 동시에 데이터 전송 거리를 최소화시키는 효과를 제공한다.

$\eta * \eta$ Mesh 시스템에 'Shuffled row-major indexing'기법을 적용하여 $N(=\eta*\eta=2^n)$개 데이터를 Bitonic 정렬할 경우에 소요되는 시간 $T(N)_{MESH}$는

$$T(N)_{MESH} = \left\{14(\eta-1)-8\log\eta\right\}*t_{r(MESH)} + \left\{\log\eta*(2*\log\eta+1)\right\}*t_c$$
$$= \left\{14*(\sqrt{N}-1)-8\log\sqrt{N}\right\}*t_{r(MESH)} + \left\{\frac{\log N*(\log N+1)}{2}\right\}*t_c \qquad (3\text{-}2)$$

로 계산된다.

표 3.1은 지금까지 기술한 MIN, Mesh, 그리고 하이퍼큐브 상호 연결망들을 기반으로 하는 시스템에 $N(=2^n)$ 개의 데이터를 Bitonic 정렬할 때 소요되는 시간을 정리,

비교하였다. 메쉬 구조는 네트워크 자체가 알고리즘을 효과적으로 지원하지 못하므로 Shuffled row-major indexing 기법을 사용하여도 전체적으로 통신에 소요되는 시간이 큰 부분을 차지하고 있다. 오메가 MIN 구조의 경우는 비교된 세 가지 상호연결망 중 Bitonic 정렬 알고리즘 운용에 가장 효율적인 네트워크로 나타나 있으나 제안된 정렬 네트워크 환경을 조성하기 위하여 각 스위칭 소자에 데이터 비교 기능과 전체 네트워크 제어 기능을 보완 해야하는 등의 요인을 내포하고 있다. 반면에 하이퍼큐브 구조는 먼저 PE 간의 상호 연결 패턴이 Bitonic 비교 패턴과 일치하여 통신에 소요되는 시간이 적을 뿐 아니라 특별한 하드웨어적 보완 없이도 알고리즘을 효과적으로 운용할 수 있어 Bitonic 정렬 알고리즘 수행에 적합한 구조임을 알 수 있다.

표 3.1. 상호 연결망에 따른 알고리즘 Run-time 비교 (N=P)

상호 연결망	Run-time
오메가 MIN	$(\log N)^2 * t_{r(MIN)} + \Phi_{t_c}$
MESH	$\left\{14(\sqrt{N}-1) - 8\log\sqrt{N}\right\} * t_{r(MESH)} + \Phi_{t_c}$
하이퍼큐브	$\left\{(\log N)^2 + \log N\right\} * t_{r(HYPER)} + \Phi_{t_c}$

Φ_{t_c} 는 전체 프로세스 중 데이터들을 비교하는데 소요되는 총 시간으로 모든 시스템에서 공히 $\left[\frac{\log N * (\log N + 1)}{2}\right] * t_c$ 로 계산된다. $t_{r(MIN)}$ 은 MIN 구조의 스위칭 소자 및 소자간의 Link 를 통과하는데 소요되는 시간, $t_{r(MESH)}$ 는 MESH 네트워크의 인접 PE 간에 데이터가 이동하는데 소요되는 시간, 그리고 $t_{r(HYPER)}$ 는 하이퍼큐브 구조에서 1-Hop 데이터 이동 시간을 나타낸다.

4. 데이터가 많은 경우 Bitonic 정렬

3 절까지는 처리할 데이터의 개수($N=2^n$)와 시스템의 프로세서 개수($P=2^p$)가 동일한 경우($N=P$), 즉 각 프로세서 마다 1 개의 데이터가 배정된 상태를 가정하여 Bitonic 정렬 과정을 기술하고, 이에 소요되는 시간을 계산하였다. 그러나 일반적으로 처리될

데이터 개수가 프로세서 개수보다 많아($N > P$) 각 프로세서가 복수 개의 데이터를 보유하고 있는 경우가 빈번히 발생한다. 본 절에서는 이와같이 데이터의 개수가 프로세서의 개수보다 많은 경우($n = p + i, i = 1,2,3,\ldots$), 이들 데이터를 정렬하는 병렬 Bitonic 정렬 기법을 제안하고, 이를 하이퍼큐브 구조에 적용하여 성능을 분석하였다.

데이터가 프로세서 보다 많은 경우, 소팅 이후 정렬된 데이터 배열 형태에 따라 병렬 소팅 알고리즘의 운용 방식이 변화하게 된다. 본 연구에서는 표 4.1에 보이는 것과 같은 2가지 형태의 데이터 정렬 방식(High-order Interleaving 방식과 Low-order Interleaving 방식)을 제안하고, 이에 따른 병렬 Bitonic 정렬 알고리즘을 구현하였다. 먼저 High-order Interleaving 방식은 올림 차순 정렬의 경우 PE_0에 전체 데이터 중 2^i 개의 작은 데이터가, 그리고 PE_{P-1}에 2^i 개의 큰 데이터가 정렬하게 된다. 또한 Low-order Interleaving 방식은 프로세서 id를 PE_0에서 PE_{P-1}까지 일단 P개의 데이터를 올림 차순으로 정렬하고, 이어서 다시 PE_0에서 PE_{P-1}까지 연속 올림 차순으로 데이터를 정렬한다. 그림 4.1 은 4개의 프로세서를 갖는 하이퍼큐브 시스템에 제안된 두 가지 정렬 방식을 각각 적용하여 8개 데이터 올림 차순 정렬을 수행한 후 각 프로세서에 분배 정렬된 데이터 배열을 그려 보이고 있다.

표 4.1. 데이터 Interleaving(올림차순 정렬)

프로세서 id	소팅 후 데이터 정렬	
	High-Order Interleaving 방식	Low-Order Interleaving 방식
0	$0, 1, 2, \ldots, (N/P) - 1$	$0, P, 2P, \ldots, N - P$
1	$(N/P), (N/P)+1, \ldots, (2N/P)-1$	$1, P+1, 2P+1, \ldots, N-P+1$
2	$(2N/P), (2N/P)+1, \ldots, (3N/P)-1$	$2, P+2, 2P+2, \ldots, N-P+2$
.....	.....	.....
$P-1$	$(P-1)N/P, (P-1)N/P+1, \ldots, N-1$	$P-1, 2P-1, 3P-1, \ldots, N-1$

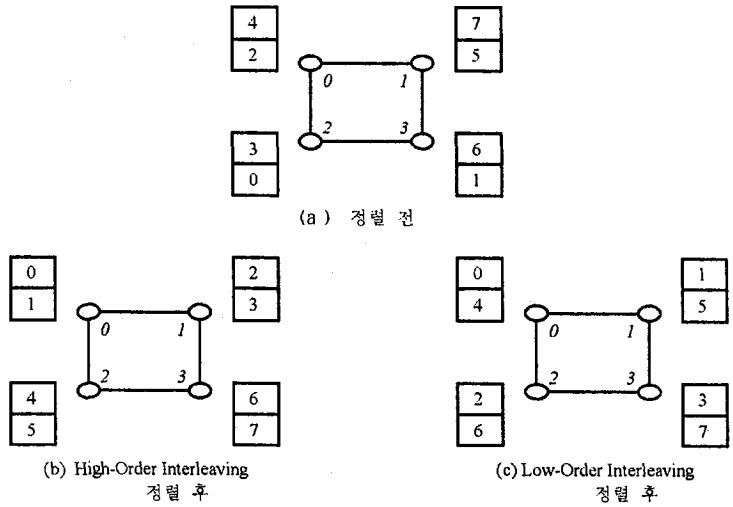


그림 4.1 데이터 정렬 방식

4.1. High-Order Interleaving(HOI) 정렬 방식

High-Order Interleaving(HOI) 정렬 방식을 적용하여 정렬이 완성되면 표 4.1에 보이는 것과 같이 어떤 프로세서가 연속된 2^i 개 데이터를 정렬된 상태로 보유하게 된다. 따라서 HOI Bitonic 정렬 알고리즘의 비교 패턴을 도식화하면 그림 2.1(b)에 보인 일반 Bitonic 정렬의 비교 패턴과 유사함을 알 수 있다. 단지 HOI Bitonic 정렬의 각 단계별 프로세스(process)가 개념적으로 프로세서간의 데이터 병합과 프로세서 내부의 데이터 정렬 과정으로 나뉘어 진행되는 점이 일반적인 Bitonic 알고리즘과 차이점으로 볼 수 있다.

그림 4.2는 $P = 4(p = 2)$, $N = 8(n = 3)$, 그리고 $i = 1$ 인 경우 HOI Bitonic 정렬 알고리즘의 비교 패턴을 나타내고, 그림 4.3은 알고리즘 실행 과정에 따른 데이터 비교-정렬 프로세스를 단계별로 보여주고 있다. 그림 4.2의 제 1 단계는 프로세서 간의 데이터 병합 과정 없이 각각 프로세서 내부의 데이터를 정렬하며, 소요시간은 t_c 이다. 프로세서 내부에서 이루어지는 데이터 정렬은 네트워크를 통한 데이터 이동 없이 프로세서 내부의 레지스터(register)에서 수행되므로 1 회의 비교 연산 시간 t_c가 소요된다. 제 2 단계는 먼저 1_큐브 병합을 수행하고 이어서 프로세서 내부의 데이터를 정렬하며, 소요시간은 $2*(2t_r + t_c)+ t_c$이다. 마지막 제 3 단계는 2_큐브 병합과 1_큐브 병합을 차례로 수행한 후 프로세서 내부의 데이터를 정렬하게 되며, 소요시간은$4*(2t_r + t_c)+ t_c$이다. 그러므로 그림 4.3에서와 같이 4 개의 프로세서를 가진 하이 퍼큐브 구조 시스템에 HOI Bitonic 정렬 알고리즘을 이용하여 8 개의 데이터를

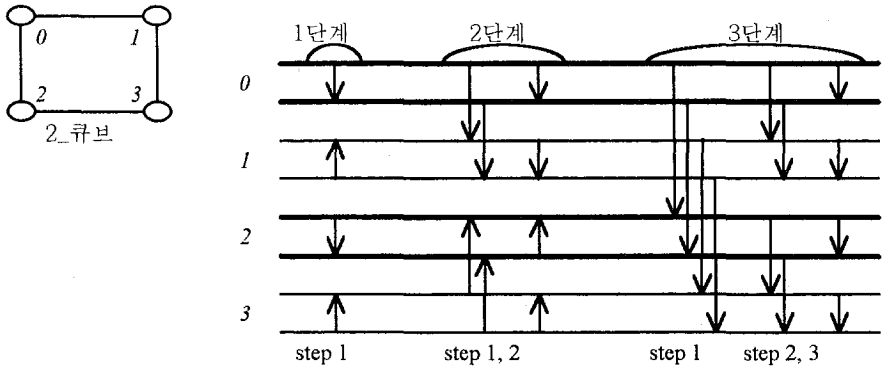


그림 4.2 HOI Bitonic 정렬의 비교 패턴

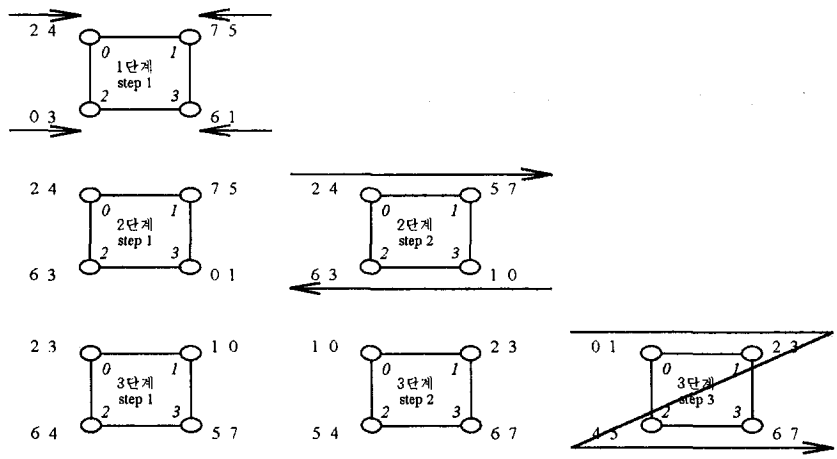


그림 4.3 HOI Bitonic 정렬

할 경우 소요되는 시간 $T_h(8,4)$는

$$T_H(8,4) = t_c + 2*(2t_r + t_c) + t_c + 4*(2t_r + t_c) + t_c$$
$$= 12t_r + 9t_c \qquad (4.1)$$

로 계산된다.

HOI Bitonic 정렬 알고리즘을 $N = 2^n, P = 2^p$, 그리고 $n = p + i, i = 1,2,3,..$ 인 경우로 일반화하여 정리하면 다음 알고리즘 I과 같다.

알고리즘 I: HOI Bitonic 정렬 알고리즘

1 단계: 프로세서 내부 2^i_데이터 정렬.

소요시간: $\frac{N}{2P} * \left\{ \frac{i*(i+1)}{2} \right\} * t_c$

2 단계:

step 1: 1_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

step2: 프로세서 내부 2^i_데이터 정렬.

소요시간: $\frac{N}{2P} * i * t_c$

3 단계:

step 1: 2_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

step 2: 1_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

step 3: 프로세서 내부 2^i_데이터 정렬

소요시간: $\frac{N}{2P} * i * t_c$

•
•

(p+1) 단계:

step 1: p_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

step 2: (p-1) _큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

•
•

step p: 1_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$.

step (p+1): 프로세서 내부 2^i_데이터 정렬.

소요시간: $\frac{N}{2P} * i * t_c$

따라서, 알고리즘 I을 수행하는데 소요되는 시간 $T_H(N,P)$는 각 단계별 소요시간을 모두 합하여

$$\begin{aligned} T_H(N,P) &= \frac{N}{P} * \{1+2+\ldots+p\} * 2t_r \\ &\quad + \left[\left\{ \frac{N}{2P} * \frac{i*(i+1)}{2} \right\} + \left\{ \frac{N}{P} * (1+2+\ldots+p) \right\} + \left\{ p * \frac{N}{2P} * i \right\} \right] * t_c \\ &= \left\{ \frac{N}{P} * \frac{p*(p+1)}{2} \right\} * 2t_r \\ &\quad + \left[\left\{ \frac{N}{2P} * \frac{i*(i+1)}{2} \right\} + \left\{ \frac{N}{2P} * p*(p+i+1) \right\} \right] * t_c \end{aligned} \quad (4\text{-}2)$$

로 계산된다.

4.2 Low-Order Interleaving(LOI) 정렬 방식

Low-Order Interleaving(LOI) 정렬 방식은 표 4.1에 보인 것처럼 정렬하고자 하는 데이터를 모든 프로세서들에 Round-Robin 형식으로 분산 정렬하는 기법이다. 따라서 LOI Bitonic 정렬 알고리즘의 비교 패턴을 개념적으로 정리하면 먼저 각 프로세서 내부의 데이터를 각각 분리시켜 $\frac{N}{P}(=2^i)$개의 p_큐브 Bitonic 부분 정렬을 완성하고, 이들을 다시 p+1, p+2, ..., p+i 큐브 Bitonic 병합을 단계별로 수행하여 전체 정렬을 이루어가게 된다.

그림 4.4는 그림 4.2에서와 같이 $P=4(p=2)$, $N=8(n=3)$, 그리고 i=1인 경우 LOI Bitonic 정렬 알고리즘의 비교 패턴을 나타내며, 그림 4.5는 알고리즘 실행 과정에 따른 데이터 이동 경로를 단계별로 보여주고 있다. 그림 4.4의 제 1 단계는 각 프로세서 내부의 2개 데이터를 분리하여 전체 시스템에 개념상 2개의 2_큐브{(0-a, 1-a, 2-a, 3-a), (0-b, 1-b, 2-b, 3-b)}Bitonic 정렬을 수행하는 단계로, 소요시간은 $2*(1+2)*(2t_r+t_c)$이다. 제 2 단계는 제 1 단계에서 생성된 2개의 2_큐브를 병합하는 단계로 각 프로세서 내부에 존재하는 2개의 데이터를 정렬하고, 이어서 2_큐

브 병합과 1_큐브 병합을 차례로 실행하게 된다. 따라서 제 2 단계 수행에 소요되는 시간은 $t_c + 2*2*(2t_r + t_c)$이다. 그러므로 그림 4.5 와 같이 4 개의 프로세서를 가진 하이퍼큐브 구조의 시스템에서 LOI Bitonic 정렬 알고리즘을 이용하여 8 개의 데이터를 정렬 할 경우 소요되는 시간 $T_L(8,4)$는

$$\begin{aligned} T_L(8,4) &= 2*(1+2)*(2t_r + t_c) + t_c + 2*2(2t_r + t_c) \\ &= 20t_r + 11t_c \end{aligned} \tag{4.3}$$

로 계산된다.

LOI Bitonic 정렬 알고리즘을 $N = 2^n$, $P = 2^p$, 그리고 $n = p + i, i = 1,2,3,\ldots$ 인 경우로 일반화하여 정리하면 알고리즘 II 와 같다.

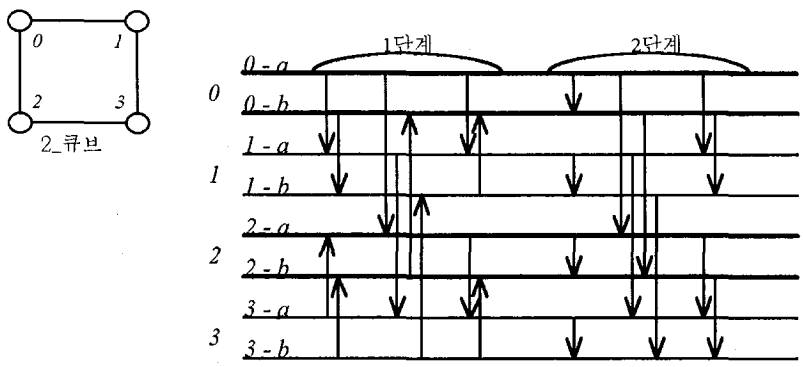


그림 4.4 LOI Bitonic 정렬의 비교 패턴

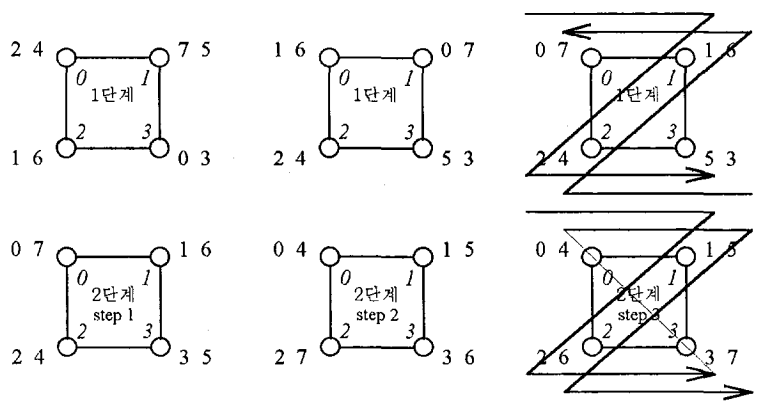


그림 4.5 LOI Bitonic 정렬

알고리즘 II: LOI Bitonic 정렬 알고리즘

1 단계: 2^i 개의 p_큐브를 각각 오름차순 및 내림차순으로 정렬.

소요시간: $\left\{\frac{N}{P} * \frac{p*(p+1)}{2}\right\} * (2t_r + t_c)$

2 단계: Bitonic p_큐브 쌍 병합-정렬

step 1: 프로세서 내부 2-데이터 쌍 정렬.

소요시간: $\frac{N}{2P} * t_c$

step 2: p_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

•
•

step p+1: 1_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

3 단계: Bitonic (p+1)_큐브 쌍 병합-정렬.

step 1: 프로세서 내부 4-데이터 쌍 정렬.

소요시간: $\frac{N}{2P} * 2 * t_c$

step 2: p_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

•
•

step p+1: 1_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

•
•

(i+1)단계: Bitonic (p+i-1)_큐브 쌍 병합-정렬.

step 1: 프로세서 내부의 2^i - 데이터 쌍 정렬.

소요시간: $\frac{N}{2P} * i * t_c$

step 2: p_큐브 병합.

소요시간: $\frac{N}{P} * (2t_r + t_c)$

•
•

step $p+i$: 1_큐브 병합.

소요시간: $\frac{N}{P}*(2t_r+t_c)$

따라서, 알고리즘 II를 수행하는데 소요되는 시간 $T_L(N,P)$는

$$T_L(N,P)=\left[\left\{\frac{N}{P}*\frac{p*(p+1)}{2}\right\}+\frac{N}{P}*p*i\right]*2t_r + \left[\left\{\frac{N}{P}*\frac{p*(p+1)}{2}\right\}+\left\{\frac{N}{2P}*\frac{i*(i+1)}{2}\right\}+\left\{\frac{N}{P}*p*i\right\}\right]*t_c \quad (4\text{-}4)$$

로 계산된다.

본 절에서 기술한 High-Order Interleaving(HOI) 방식과 Low-Order Interleaving(LOI) 방식은 하이퍼큐브 구조는 물론 다양한 병렬 컴퓨터 환경에서 Bitonic 정렬 알고리즘을 이용하여 많은 양의 데이터를 정렬하고자 할 때 적용이 용이하도록 개발된 기법이다. HOI 방식은 알고리즘이 비교적 단순할 뿐 아니라, 알고리즘 수행에 소요되는 시간도 LOI 방식과 비교하여 경제적인 것으로 나타나, 성능면에서 우수한 방식임을 보였다. 그러나 정렬 알고리즘의 응용 분야와 정렬 후 요구되는 데이터 배열 방식에 따라 정렬 기법 선택을 달리할 수 있다.

5. 결 론

하이퍼큐브 구조는 Bitonic 정렬의 단계별 비교 패턴과 일치하여, 알고리즘의 Mapping이 용이할 뿐 아니라, 알고리즘을 효율적으로 지원 할 수 있는 상호 연결망을 가지고 있다. 본 연구에서는 하이퍼큐브 시스템에 Bitonic 정렬 알고리즘을 적용하여, 알고리즘의 비교-정렬 진행과정을 단계별로 분석하고, 데이터 정렬에 소요되는 시간을 계산하였다. 나아가서 오메가 MIN과 MESH 구조 등을 상호 연결망으로 갖는 시스템에 같은 알고리즘을 적용하고, 성능을 비교 분석하여 하이퍼큐브 구조가 Bitonic 정렬 알고리즘을 효과적으로 지원하는 구조임을 입증하고, 또한 병렬 알고리즘의 성능이 적용 대상 시스템의 구조에 따라 대폭 변화할 수 있음을 재조명하였다.

프로세서 개수가 한정된 시스템에서 대규모 데이터를 정렬하고자 할 경우 네트워크를 통한 데이터 이동량이 증가하여 데이터 충돌 현상등 전체 성능을 저하시키는 요인이 발생하게 된다. 본 연구에서 제안한 두 가지 병렬 Bitonic 정렬 알고리즘(High-Order Interleaving(HOI) 정렬 기법과 Low-Order Interleaving(LOI) 정렬 기법)은 하이퍼큐브 구조에서 다량의 데이터를 정렬 할 때 네트워크상에서의 데이터 충돌을 피하고 프로세서의 이용률을 높여 효과적인 데이터 정렬을 수행 하게한다. 제안된 알고리즘을 하이퍼큐브 시스템에 적용하여 소요시간을 계산한 결과 HOI 정렬 기법이 성능면에서 비교적 우수한 것으로 판명되었다. 그러나 알고리즘 선택은 응용 분야에서 요구하는 정렬 후 데이터 배열 방식에 따라 결정되어야 한다.

참 고 문 헌

[1] K. E. Batcher, "Sorting Networks and their Applications", *Proc. AFIPS 1968 SJCC*, Vol. 32, AFIPS press, Montvale, NJ., pp307-314.

[2] H. S. Stone, "Parallel Processing with the Perfect Shuffle", *IEEE Trans. on Computers*, Vol. C-20, Feb. 1971, pp153-161.

[3] D. E. Knuth, "The Art of Computer Programming", Vol. 3, *Addison-Wesley*, 1973.

[4] C. D. Thompson and H. T. Kung, "Sorting on Mesh Connected Parallel Computer", *Communications of the ACM*, Apr. 1977, pp263-271.

[5] D. Nassimi and S. Sahni, "Bitonic Sort on a Mesh-Connected Parallel Computer", *IEEE Trans. on Computers*, 1979, pp2-7.

[6] M. K. Yang and C. R. Das, "Evaluation of a Parallel Branch-and-Bound Algorithm on a Class of Multiprocessors", *IEEE Trans. on Parallel and Distributed Systems*, Vol 5, Number 1, Jan. 1994, pp74-86.