EREW PRAM 모델에서 특수 볼록 이분할 그래프 매칭을 위한 병렬 알고리즘

김 명 호[†] · 정 창 성[††]

요 약

본 논문에서는 특수 볼록 이분할 그래프에서 최대 매칭을 구하는 문제를 다룬다. 이 문제는 여러 가지 스케줄링 문제로 변환이 가능하다. 본 논문에서는 EREW PRAM 모델에서 최적의 시간 복잡도를 갖는 병렬 알고리즘을 제시한다. 이 병렬 알고리즘을 설계하기 위해 먼저 순차 알고리즘을 분석하여 수식을 유도하고, 이 수식에서 어떻게 병렬성을 얻을 수 있는지를 보인다. 그리고 이 수식을 기반으로 하는 병렬 알고리즘을 제시한다. 이 병렬 알고리즘은 n개의 프로세서로 구성된 EREW PRAM 모델에서 $O(\log n)$의 시간복잡도를 갖는다.

A Parallel Matching Algorithm for a Special Convex Bipartite Graph on an EREW PRAM Model

Myung-Ho Kim[†] · Chang-Sung Jung[††]

ABSTRACT

In this paper, we consider the problem of finding a maximum matching for a special convex bipartite graph. The problem arises frequently in diverse application areas such as scheduling problems. In this paper, we present an optimal parallel matching algorithm on an EREW PRAM model. We first analyze a sequential matching algorithm; then we show how to find parallelism from its inherent sequential property by deriving a formula. And we design the parallel matching algorithm using the formula. Time complexity of the algorithm is $O(\log n)$ on an EREW PRAM with n processors.

1. 서 론

볼록 이분할 그래프(Convex Bipartite Graph) G는 (A, B, E)의 3 집합으로 표현할 수 있다. 여기서, $A=\{a_0, a_1, \cdots, a_{n-1}\}$과 $B=\{b_0, b_1, \ldots, b_{m-1}\}$은 분리된 정점의 집합이고, E는 다음 조건을 만족하는 간선의 집합이다[1]:

1. $(i, j) \in E$이면, ($i \in A$이고 $j \in B$)이거나 ($i \in B$이고 $j \in A$)이다.
2. $((a_i, b_j) \in E$이고 $(a_i, b_{j+k}) \in E)$이면, $(a_i, b_{j+q}) \in E$, $1 \le q < k$이다.

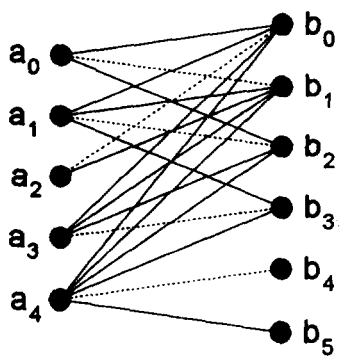
조건 1은 이분할 그래프를 뜻하고, 조건 2는 볼록 조건을 나타낸다. 볼록 이분할 그래프는 다음과 같이 나타낼 수도 있다.

※본 연구는 숭실대학교 96년도 교내학술연구비에 의해 수행
† 정 회 원:숭실대학교 컴퓨터학부
†† 정 회 원:고려대학교 전자·전기·전파공학부
논문접수:1997년 9월 1일, 심사완료:1998년 1월 16일

$$T=\{(i, s_i, h_i)|0 \le i \le n-1\}$$
$$s_i=\min\{j|(a_i, b_j) \in E\}$$
$$h_i=\max\{j|(a_i, b_j) \in E\}$$

이 표현 방법은 A의 i번째 원소 a_i에 대해서 a_i와 연결된 B의 원소 중 제일 작은 번호를 갖는 b_{s_i}와 가장 큰 번호를 갖는 b_{h_i}의 번호를 저장하는 방법으로 볼록 이분할 그래프는 볼록한 성질을 가지고 있으므로 유일하게 표현될 수 있다. 특수 볼록 이분할 그래프는 볼록 이분할 그래프 중 s_i가 모두 같은 것을 의미한다.

$F \subseteq E$이고 F의 어느 두 간선도 같은 정점을 갖지 않을 때 F를 매칭이라 한다. 최대 매칭(Maximum matching) M은 매칭 중 가장 많은 간선을 가지는 매칭 F를 의미한다. 앞으로 매칭은 최대 매칭을 의미한다. 볼록 이분할 그래프 매칭은 스케줄링 문제에 응용될 수 있다. 그림 1은 특수 볼록 이분할 그래프의 예를 보인 것이고, 점선은 이 그래프의 매칭을 나타낸 것이다. 그리고 이 그래프는 다음과 같이 나타낼 수 있다: $T=\{(0, 0, 2), (1, 0, 3), (2, 0, 1), (3, 0, 3), (4, 0, 5)\}$.



(그림 1) 특수 볼록 이분할 그래프
(Fig. 1) Special Convex Bipartite Graph

볼록 이분할 그래프 매칭을 위한 많은 알고리즘이 개발되어 왔다. n을 $|A|$라 하고, m을 $|B|$라 할 때, [1]은 $O(nm)$의 시간복잡도를 갖는 순차 알고리즘을 제시하였고, [2]는 $O(n+m\,A(m))$의 시간복잡도를 갖는 빠른 순차 알고리즘을 제시하였다. 여기서 A(·)는 Ackermann 함수의 역함수이다. 특수 볼록 이분할 그래프 매칭을 위해서는 [3]에서 $O(n \log n)$인 순차 알고리즘을 개발하였다.

병렬 알고리즘으로는 [3]에서 특수 볼록 이분할 그래프 매칭을 위해 $O(n^2)$개의 프로세서를 갖는 EREW (Exclusive Read and Exclusive Write) PRAM 모델[5]에서 $O(\log n)$의 알고리즘을 제시하였고, [4]에서는 O(n)의 프로세서로 구성된 MCC(Mesh-Connected Computer)에서 $O(\sqrt{n})$ 복잡도를 가지는 병렬 알고리즘을 제시하였다.

병렬 알고리즘의 성능은 프로세서 이용의 효율성(Effectiveness of Processor Utilization: EPU)으로 평가할 수 있는데, 병렬 프로그램 PA의 EPU는 다음과 같이 정의된다[6, 7].

$$EPU=\frac{\text{순차 알고리즘의 시간 복잡도}}{PA\text{가 요구하는 프로세스 수}\times PA\text{의 시간 복잡도}}$$

여기서 $0 \le \text{EPU} \le 1$임을 알 수 있다. 그리고, EPU가 1에 가까울수록 좋은 알고리즘이고, 1이면 이 병렬 알고리즘 PA는 최적이라고 한다. 따라서 [3]에서 제시한 병렬 알고리즘은 EPU가 $1/n$이 되고, 이는 그다지 좋지 못한 알고리즘이라고 볼 수 있다. 본 논문에서는 n개의 프로세서로 구성된 EREW PRAM 모델에서 $O(\log n)$의 시간복잡도를 갖는 특수 볼록 이분할 그래프 매칭을 위한 병렬 알고리즘을 제시한다. 최적의 순차 알고리즘이 $O(n \log n)$의 시간복잡도를 가지므로, 이 알고리즘은 최적이다.

본 논문의 구성은 다음과 같다. 2절에서는 기본적인 병렬 처리 기법에 관하여 설명하고, 3절에서 병렬 알고리즘을 제시한다. 그리고 4절에서 결론을 내린다.

2. 기본적인 병렬 처리 기법

본 절에서는 본 논문에서 사용하는 용어 정의 및 병렬 처리 기법에서 대해서 설명한다. $PE[i]$는 i번째 프로세서를 나타낸다. 자연수 i에 대하여 i_b를 i를 이진화했을 때 (0부터 시작하여) b번째 하위 비트를 나타낸다고 하자. 병렬 알고리즘에서 분할 해결법을 사용하여 알고리즘을 개발할 때 블록(*block*)이라는 개념을 사용한다. 2^k 블록은 프로세서의 번호를 이진수로 표현했을 때 하위 k 비트 (i_{k-1}:0)만 다른 2^k개의 프로세서로 이루어진다. 전체 프로세서의 개수가 2^m일 때 2^k 블록은 2^{m-k}개가 존재한다. 또한 왼쪽 2^k 블록은 i_{k-1}

$=0$인 2^k 블록을 뜻하고, 오른쪽 2^k 블록은 $i_{k-1}=1$인 2^k 블록을 뜻한다. 2^{k+1} 블록은 인접한 왼쪽 2^k 블록과 오른쪽 2^k 블록을 합하여 얻어진다.

PRAM 모델에서는 메모리를 공유하므로 왼쪽 2^k 블록과 오른쪽 2^k 블록 사이의 자료교환을 $O(1)$에 수행할 수 있다. 병렬 알고리즘에서 분할 해결법을 사용하면 다음과 같은 식의 시간복잡도를 얻을 수 있다.

$$T(n)=\begin{cases} c, & \text{if } n=1 \\ T(n/2)+MG(n), & \text{if } n>1 \end{cases}$$

여기서 c는 계산 시간이고 $MG(n)$은 부분문제의 결과를 합병하는 시간이다. c가 상수일 때 합병시간이 전체 시간복잡도를 좌우하므로 최적의 시간복잡도를 얻기 위해서는 합병시간을 최소화해야 한다. c와 $MG(n)$이 $O(1)$이면, 전체 시간복잡도 $T(n)=O(\log n)$이 된다.

본 논문에서 제시하는 병렬 알고리즘에서는 다음과 같은 잘 알려진 두 개의 기본 병렬 연산들을 사용한다.

1. 전위연산 : $\otimes$가 결합가능 연산자이고 $D=\{d_0, d_1, d_2, \cdots, d_{n-1}\}$ 리스트가 있을 때, $x_i=d_0 \otimes d_1 \otimes \ldots \otimes d_i$, $0 \le i \le n-1$를 계산하는 연산.
2. 정렬 : 모든 원소들을 특정한 값에 따라 오름차순 혹은 내림차순로 정렬하는 연산.

위의 기본 연산은 EREW PRAM 모델에서 각각 $O(n/\log n)$과 $O(n)$ 프로세서를 사용하여 $O(\log n)$에 계산할 수 있다[8, 9]. 전위합은 전위연산에서 연산자 $\otimes$ 대신 $+$를 적용하여 x_i, $0 \le i \le n-1$을 계산하는 연산이다. 랭크(Rank)는 한 리스트에서 원소들을 선택하였을 때, 선택된 각 원소에서 리스트 상에서 자신을 포함하여 자신보다 앞에 있는 원소의 수로 정의된다. 즉, 이것은 $D=\{d_0, d_1, d_2, \ldots, d_{n-1}\}$ 리스트에서 d_i가 0또는 1일 경우에 전위합을 계산하는 연산과 같다. 따라서 전위합과 랭크는 전위연산을 사용하여 구할 수 있으므로, EREW PRAM 모델에서 $O(n/\log n)$ 프로세서를 사용하여 $O(\log n)$에 계산할 수 있다.

3. 병렬 알고리즘

본 절에서는 특수 볼록 이분할 그래프를 위한 병렬 매칭 알고리즘을 제시한다. 본 알고리즘에서 사용하는 용어를 먼저 정리하면 다음과 같다.

볼록 이분할 그래프 $G=(A, B, E)$에서 간선 (a_i, b_j)가 매칭 M에 있다면, A의 a_i가 B의 b_j에 매칭되었다고 하며, b_j를 a_i의 매칭 원소라고 하고, 그 역도 성립한다. a_i가 b_j에 매칭되었다면 a_i의 매칭 값은 j라고 하고, 매칭되지 않았다면 매칭 값은 -1 이다. $[i, j]$는 B의 부분 집합 $\{b_i, b_{i+1}, \ldots, b_j\}$를 나타낸다. $[i, j]$에서 매칭 A를 찾는 문제는 $G'=(A, B', E)$에서의 매칭을 구하는 것이다. 여기서 $B'=[i, j]$이고, $[i, j]$를 A의 매칭 구간이라 한다. 매칭 구간 $[i, j]$에서의 A의 매칭 집합은 $[i, j]$에 매칭되는 A의 모든 원소를 뜻한다.

특수 볼록 이분할 그래프 $G=(A, B, E)$를 위한 순차 매칭 알고리즘 *SMatching*은 다음과 같다[3]. 여기서 $B=[u, v]$이고, 모든 a_i에 대해서 $s_i=u$이다.

Algorithm SMatching(A, u, v, MN)
입력 : (i, s_i, t_i) 형태의 집합 A와 매칭 구간$[u, v]$. $n=|A|$이며, 모든 a_i에 대해서 $s_i=u$이고 $t_i \ge u$이다. 초기치로 $MN[i]=-1$이다.
출력 : a_i의 매칭 값이 $MN[i]$에 저장된다.

```
1) Sort(A); // t_i 값에 의해서 정렬하고, 앞으로 t_i ≤ t_{i+1}
            라고 가정한다.
2) j = u − 1;
3) for i = 0 to n − 1 do
     if j ≥ v
     then return(j);
     else if (j + 1) ≤ t_i
          then begin
               j = j + 1;
               MN[i] = j;
          end
4) return(j);
END SMatching.
```

위 알고리즘은 n개의 A의 원소를 B의 $[u, v]$ 구간에 매칭시키는 알고리즘으로 매칭이 가능한 원소는 $MN[i]$에 매칭 값이 들어가고, 매칭이 안되는 것은 $MN[i]$에 -1 값을 갖는다. 매칭이 끝나면 최종 매칭 값 보다 일이 큰 값이 리턴된다. 알고리즘의 정확성은 [3]에 증명되어 있다.

위 알고리즘 *SMatching*은 각 $MN[i]$가 이전의 값에

영향을 받음으로 해서 전형적인 순차 알고리즘처럼 보이지만 본 논문에서는 이를 이용하여 효율적인 병렬 알고리즘을 설계할 수 있음을 보일 것이다.

A가 k개의 다른 t 값으로 이루어져 있다고 가정하고, 각각을 $h_i(0 \le i \le k-1)$로 표기하자. 또한 $h_i < h_{i+1}$, $0 \le i < k-1$이라 하자. 그러면 A를 k개의 부분 집합 $\{H_0, H_1, ..., H_{k-1}\}$로 분리할 수 있다. 여기서 H_i는 t_j값이 h_i인 a_j의 집합이다. 또한 $S_i = \bigcup_{0 \le j \le i} H_j$, $0 \le i \le k-1$이라 하고, $n_i = |H_i|$라 하자. 그리고, q_i는 SMatching을 사용하여 $[u, h_i]$에 S_i를 매칭했을 때의 최대 매칭 값이라고 하자. 그러면, q_i는 다음과 같음을 쉽게 알 수 있다.

$$
\begin{aligned}
&q_{-1} = h_{-1} = u-1\\
&q_0 = min(q_{-1} + n_0, h_0, v)\\
&q_1 = min(q_0 + n_1, h_1, v)\\
&\cdots\\
&q_i = min(q_{i-1} + n_i, h_i, v),\ 0 \le i \le k-1
\end{aligned}
$$

여기서 순환 부분을 전개하면 다음과 같은 식을 얻을 수 있다.

$$
\begin{aligned}
&q_{-1} = h_{-1}\\
&q_0 = min(q_{-1} + n_0, h_0, v)\\
&\quad = min(h_{-1} + n_0, h_0, v)\\
&q_1 = min(q_0 + n_1, h_1, v)\\
&\quad = min(h_{-1} + n_0 + n_1, h_0 + n_1, h_1, v)\\
&q_2 = min(q_1 + n_2, h_2, v)\\
&\quad = min(h_{-1} + n_0 + n_1 + n_2, h_0 + n_1 + n_2, h_1 + n_2, h_2, v)\\
&\cdots\\
&q_i = \min\left\{\min_{0 \le j \le i+1}\left(h_{j-1} + \sum_{l=j}^{i} n_l\right), v\right\},\ 0 \le i \le k-1
\end{aligned}
\tag{1}
$$

그리고, $S_i^m = \bigcup_{m \le j \le i} H_j$, $0 \le m \le k-1$로 정의하고, Q_i^m을 SMatching을 사용하여 $[h_m, h_i]$에 S_i^m을 매칭했을 때의 최대 매칭 값이라고 하자. 그러면 Q_i^m은 식 (2)와 같음을 쉽게 알 수 있다.

$$
Q_i^m = \min\left\{\min_{m \le j \le i+1}\left(h_{j-1} + \sum_{l=j}^{i} n_l\right), v\right\} \tag{2}
$$

이제, q_i는 보조정리 1에 의해서 Q_i^m을 사용하여 계산될 수 있음을 알 수 있다.

보조정리 1: $q_i = Q_i^0$.

증명: 식 (2)에서 m대신 0을 대입하면 다음과 같고 이는 식 (1)과 같다.

$$
Q_i^0 = \min\left\{\min_{0 \le j \le i+1}\left(h_{j-1} + \sum_{l=j}^{i} n_l\right), v\right\}
$$

따라서, $q_i = Q_i^0$ 이다. ■

이제 Q_i^m을 재귀적으로 표현할 수 있음을 보일 것이다. 만일 Q_i^m을 재귀적으로 풀 수 있으면, q_i는 효율적으로 계산될 것이다. 보조정리 2는 반복적인 분할 해결법을 이용하여 Q_i^m을 계산할 수 있게 수식을 유도한 것이다.

보조정리 2: m과 m_1이 $0 \le m \le m_1 \le k-1$이고, $m_1 \le i \le k-1$이라 할 때 Q_i^m은 다음과 같다:

$$
Q_i^m = \min\left(Q_{m_1-1}^m + \sum_{l=m_1}^{i} n_l, Q_i^{m_1}, v\right). \tag{3}
$$

증명: 식 (2)를 $m \le m_1 \le k-1$인 m_1에 대해 두 부분으로 나누면 다음과 같은 식을 얻을 수 있다.

$$
\begin{aligned}
Q_i^m = \min\Bigg[&\min\left\{\min_{m \le j \le m_1}\left(h_{j-1} + \sum_{l=j}^{i} n_l\right), v\right\},\\
&\min\left\{\min_{m_1 \le j \le i+1}\left(h_{j-1} + \sum_{l=j}^{i} n_l\right), v\right\}\Bigg].
\end{aligned}
$$

위 식의 $\min\left\{\min_{m \le j \le m_1}\left(h_{j-1} + \sum_{l=j}^{i} n_l\right), v\right\}$에서 $\sum_{l=m_1}^{i} n_l$을 밖으로 빼내고, 두 번째 식 $\min\left\{\min_{m_1 \le j \le i+1}\left(h_{j-1} + \sum_{l=j}^{i} n_l\right), v\right\}$를 $Q_i^{m_1}$으로 대치하면 다음과 같은 식을 얻을 수 있다.

$$
\begin{aligned}
Q_i^m &= \min\left\{\min_{m \le j \le m_1}\left(h_{j-1} + \sum_{l=j}^{m_1-1} n_l\right) + \sum_{l=m_1}^{i} n_l, Q_i^{m_1}, v\right\}\\
&= \min\left(Q_{m_1-1}^m + \sum_{l=m_1}^{i} n_l, Q_i^{m_1}, v\right).
\end{aligned}
$$

■

보조정리 1과 2로부터 다음 보조정리 3을 얻을 수 있다.

보조정리 3: $0 \le m \le k-1$인 m에 대해서 q_i는 다음과 같다:

$$q_i = \min\left\{Q^0_{m-1} + \sum_{l=m}^{i} n_l,\ Q^m_i, v\right\}.$$

만일 q_i를 알 수 있다면 H_i의 원소 a_j의 매칭 값은 H_i에서의 랭크를 계산하여 쉽게 계산할 수 있다.

따라서, 매칭을 구하기 위한 주된 일은 q_i를 병렬로 계산하는 것이다. q_i는 보조정리 3에 의해서 분할 해결법을 사용하여 다음과 같이 순환적으로 계산할 수 있다. 먼저 Q^0_i, $0 \le i \le m-1$과 Q^0_i, $m \le i \le k-1$을 각각 왼쪽과 오른쪽 블록에서 동시에 계산한다. 그러면 왼쪽 블록의 Q^0_i, $0 \le i \le m-1$은 q_i와 같고, 오른쪽 블록의 Q^m_i, $m \le i \le k-1$은 보조정리 3에 의해서 $\sum_{l=m}^{i} n_l$를 계산하므로서 얻을 수 있다. 자세한 병렬 알고리즘 PMatching은 다음과 같다. 알고리즘의 각 문장 뒤의 조건문은 그 조건이 만족되는 문장만 수행됨을 의미한다.

Parallel Algorithm PMatching (A, u, v, MN)

입력: (i, s_i, t_i) 형태의 집합 A와 매칭 구간 $[u, v]$. $n = |A|$이며, 모든 a_i에 대해서 $s_i = u$이고 $t_i \ge u$이다. 초기치로 $MN(i) = -1$이다.

출력: a_i의 매칭 값이 $MN(i)$에 저장된다.

1) a_i, $0 \le i \le n-1$을 t_i의 오름차순으로 정렬한다.

// 앞에서 설명한 것과 같이 A는 t_i의 값에 따라 $\{H_0, H_1, \ldots, H_{k-1}\}$로 분할된다. 여기서 H_i는 t_j값이 h_i인 a_j의 집합이다. H_i에서 제일 작은 인덱스를 갖는 원소를 대표 원소라고 하자.

2) q_i, $0 \le i \le k-1$을 다음과 같이 계산한다.

2.1) 모든 H_i, $0 \le i \le k-1$의 대표 원소들을 연속된 기억 장소로 옮기고, n_i를 계산한다. $s = \lceil \log k \rceil$라고 하면 모아진 레코드는 2^s 블록에 저장된다.

2.2) $p^i_i = n_i$, $Q^i_i = \min(h_{i-1} + n_i, h_i, v)$, $QR_i = Q^i_i$, $pr_i = n_i$, $0 \le i \le s-1$.

// $p^b_a = \sum_{i=a}^{b} n_i$, QR_i: 현재 블록에서 제일 큰 Q 값, pr_i: 현재 블록에서 제일 큰 p값

2.3) k를 1에서 s까지 증가시키면서 각 2^k 블록에서 다음을 병렬로 반복 수행한다.

// l_1, l_2: 왼쪽 2^{k-1} 블록에서 제일 작은 인덱스와 제일 큰 인덱스,

// r_1, r_2: 오른쪽 2^{k-1} 블록에서 제일 작은 인덱스와 제일 큰 인덱스

2.3.1) $Q_i = \min(QR_{i-2^{k-1}} + p^{r_1}_i, Q^{r_1}_i)$, $r_1 \le i \le r_2$.

2.3.2) $p_i = p^{r_1}_i + pr_{i-2^{k-1}}$, $r_1 \le i \le r_2$.

2.3.3) $QR_i = \min(QR_i + pr_{i+2^{k-1}}, QR_{i+2^{k-1}})$, $l_1 \le i \le l_2$,

$QR_i = \min(QR_{i-2^{k-1}} + pr_i, QR_i)$, $r_1 \le i \le r_2$,

$pr_i = pr_i + pr_{i+2^{k-1}}$, $l_1 \le i \le l_2$,

$pr_i = pr_{i-2^{k-1}} + pr_i$, $r_1 \le i \le r_2$.

3) 매칭 값을 찾기 위해 H_i에서 a_j를 가지는 모든 프로세서는 다음을 수행한다.

3.1) r_j, $0 \le j \le n-1$과 w_i, $0 \le i \le k-1$을 구한다.

3.2) 각 H_i, $0 \le i \le k-1$의 원소 a_j를 위해 $r_j < w_i$이면, $MN(i)$을 $q_{i-1} + r_j + 1$로 하고, 아니면 -1로 한다.

END PMatching.

위 알고리즘을 자세히 설명하면 다음과 같다. 이 알고리즘은 보조정리 3을 기반으로 설계된 것이다. 보조정리 3을 적용하기 위해서는 먼저 입력으로 들어온 집합 A를 H_i, $0 \le i \le k-1$, 집합으로 분할하는 것이 필요하다. 따라서 1)단계에서는 집합 A를 H_i의 집합으로 분리하기 위해 정렬을 수행한다.

2)단계에서는 q_i, $0 \le i \le k-1$을 계산한다. q_i, $0 \le i \le k-1$을 계산하기 위해서는 먼저 각 H_j, $0 \le j \le i$의 크기 n_j가 필요하다. 따라서 2.1)단계에서 n_i, $0 \le i \le k-1$을 계산한다. n_i는 H_i와 H_{i+1}의 대표원소의 인덱스 차를 계산하면 알 수 있으므로, 2.1)단계와 같이 모든 대표원소를 한곳으로 모은 후 바로 인접한 대표원소의 인덱스를 참조하면 된다. 2.2)단계는 초기화 작업을 수행하는 것이다. 보조정리 3을 이용하여 q_i, $0 \le i \le k-1$을 계산하기 위해서는 각 블록에서 n_i의 부분합과 인접한 왼쪽 블록의 최대 Q 값을 알아야 한다. 또한 인접한 왼쪽과 오른쪽 블록이 합병되면, n_i의 부분합이 바뀌게 된다. 그런데 기존의 왼쪽 블록에서의 부분합은 영향을 받지 않고, 오른쪽 블록의

부분합만 바뀐다는 것을 쉽게 알 수 있다. 그리고 그 값은 왼쪽 블록에서의 n_i의 합에 자신의 부분합을 더하면 된다는 것을 알 수 있다. 따라서 본 알고리즘에서는 p_i, pr_i, QR_i을 사용하여, 각 블록에서의 n_i 부분합, 각 블록의 전체 n_i 합, 각 블록에서의 최대 Q 값을 유지한다. 그리고 각 변수의 초기값은 2.2)단계와 같음을 쉽게 알 수 있다.

2.3)은 분할 해결법을 이용하여 q_i, $0 \le i \le k-1$을 실제로 계산하는 단계이다. 이 단계는 2.3.1)부터 2.3.3)까지 블록을 증가시키면서 반복 수행한다. $(k-1)$번째 반복 후의 인접한 왼쪽 2^{k-1} 블록과 오른쪽 2^{k-1} 블록을 구성하는 원소들의 인덱스 범위가 각각 $l_1 \le i \le l_2$, $r_1 \le j \le r_2$라고 하자. 이것은 정의에 의해 $l_2 = l_1 + 2^{k-1} - 1$, $r_1 = l_2 + 1$, $r_2 = r_1 + 2^{k-1} - 1$이다. 또한 정의에 의해 왼쪽과 오른쪽 2^{k-1} 블록의 Q는 각각 $Q_i^{l_1}$, $l_1 \le i \le l_2$와 $Q_j^{r_1}$, $r_1 \le j \le r_2$이다. 그리고 $(k-1)$번째 반복 후에, 왼쪽 (혹은 오른쪽) 2^{k-1} 블록은 $Q_i^{l_1}$ (혹은 $Q_i^{r_1}$)와 부분 합 $p_i^{l_1} = \sum_{j=l_1}^{i} n_j$(혹은 $p_i^{r_1} = \sum_{j=r_1}^{i} n_j$)를 가지고 있다고 하자. 이때 k번째 반복에서의 Q_i와 p_i, $l_1 \le i \le r_2$는 다음과 같이 왼쪽과 오른쪽 2^{k-1} 블록의 결과를 사용하여 계산할 수 있다. 여기서 Q_i는 명확히 쓰면 $Q_i^{l_1}$이지만 왼쪽 2^{k-1} 블록의 결과와의 표기상의 혼동을 피하기 위해서 Q_i로 표기하였다. 2^k 블록의 Q_i와 p_i, $l_1 \le i \le l_2$는 왼쪽 2^{k-1}에서 구한 $Q_i^{l_1}$과 $p_i^{l_1}$, $l_1 \le i \le l_2$와 같음을 쉽게 알 수 있다. 그러나 오른쪽 2^{k-1} 블록의 것은 그렇지 않으므로 더 계산을 해야한다. 이것은 보조정리 3을 이용하면 다음과 같이 쉽게 구할 수 있다. 즉 2^k 블록의 Q_i와 p_i, $r_1 \le i \le r_2$는 각각 $\min(Q_{l_2}^{l_1} + p_i^{r_1}, Q_i^{r_1})$과 $p_{l_2}^{l_1} + p_i^{r_1}$이다. 그런데 $Q_{l_2}^{l_1}$은 $QR_{i-2^{k-1}}$이고, $p_{l_2}^{l_1}$은 $pr_{i-2^{k-1}}$이므로, 2.3.1)과 2.3.2) 단계와 같이 수행하면 각각 Q_i와 p_i를 구할 수 있다. 2.3)단계의 마지막 단계는 다음 반복을 위해 QR_i와 pr_i를 조정하는 것으로 그 값은 각 PE[i]가 왼쪽 2^{k-1} 블록에 속하는지 오른쪽 2^{k-1} 블록에 속하는지에 따라 달라지며, 각 경우에 따라 2.3.3)과 같이 계산된다.

2.2)의 반복 수행이 끝나면, Q_i^0, $0 \le i \le s-1$이 계산이 되고, 이것은 보조정리 1에 의해서 q_i이다. 이제 각 매칭 값은 q_i를 사용하여 3)단계에 있는 것처럼 쉽게 구할 수 있다. $w_i = q_i - q_{i-1}$라 하고, r_j를 H_i에서 a_j보다 이전에 있는 원소의 수라고 하자. H_i와 q_i의 정의에 의해서 H_i에서 a_j의 매칭 값은 $q_{i-1}+1$과 q_i 사이에 있다. 따라서 H_i 중에서 첫 w_i개의 원소만 $[q_{i-1}+1, q_i]$에 매칭된다. 즉, a_j의 매칭 값은 $r_j < w_i$이면 $q_{i-1} + r_j + 1$이다. 이 계산과정이 3.1)과 3.2) 단계에 있다.

정리 1:특수 블록 이분할 그래프의 매칭은 EREW PRAM 모델에서 $O(\log n)$에 수행될 수 있다.

증명:특수 블록 이분할 그래프의 매칭은 병렬 알고리즘 PMatching을 사용하여 얻을 수 있고, PMatching의 정확성은 보조정리 1, 보조정리 2, 보조정리 3과 위의 설명에 의해서 알 수 있다. PMatching의 시간복잡도는 다음과 같이 계산될 수 있다. 1)단계는 정렬을 수행하는 것이므로 $O(\log n)$에 수행된다. 2.1)은 전위연산을 사용하여 $O(\log n)$에 수행된다. 2.2)는 초기화하는 단계이므로 $O(1)$에 수행된다. 2.3)은 분할 해결법을 사용하는 것으로 $T(n) = T(n/2) + MG(n)$ 공식을 사용하여 계산될 수 있다. 그런데, 2.3.1)부터 2.3.3)까지는 $O(1)$에 수행되므로 $T(n) = T(n/2) + O(1)$으로 쓸 수 있고, 이것은 $O(\log n)$이다. 3)은 전위연산을 사용하면 되므로 $O(\log n)$에 수행된다. 따라서 전체 시간복잡도는 $O(\log n)$이다. ■

4. 결 론

특수 블록 이분할 그래프를 위한 매칭은 스케줄링 문제에 응용될 수 있는 문제로 이전에 많은 알고리즘들이 개발되어 왔다. 그러나 병렬 알고리즘으로는 많이 개발되어 있지 않고, 이전의 EREW PRAM 모델에서 제시된 병렬 알고리즘은 최적의 시간복잡도를 가지고 있지 않다. 따라서 본 논문에서는 EREW PRAM 모델에서 최적의 시간복잡도를 갖는 병렬 매칭 알고리즘을 제안하였다. 이 알고리즘을 위해 먼저 매칭을 위한 최적 순차 알고리즘에서 매칭을 구하는 수식을 유도하였으며, 이 수식을 분할 해결기법에 적합하도록 변형하였다. 본 논문에서 제시하는 알고리즘은 유도된 수식을 이용하여 개발되었다. 본 알고리즘은 n개의 프로세서로 구성된 EREW PRAM 모델에서 $O(\log n)$ 시간복잡도를 가진다. 이는 특수 블록 이분할 그래프의 순차 알고리즘이 $O(n \log n)$ 시간복잡도를 가지므로, 최적의 프로세서 효율성을 가진다. 따라

서 본 논문에서 제시한 알고리즘은 최적이다.

참 고 문 헌

[1] Glover, F., "Maximum Matching in a Convex Bipartite Graph," *Naval Res. Logist. Quart. 14*, (1967), pp. 313-316.

[2] Lipski, W., Jr., and F. P. Preparata, "Efficient Algorithms for Finding Maximum Matchings in Convex Bipartite Graphs and Related Problems," *Acta Inform. 15*, pp. 329-346, 1981.

[3] Deker, E. and S. Sahni, "Parallel Scheduling Algorithm for Convex," *Operations Research*, vol. 31, No. 1, pp. 24-49, 1983.

[4] Kim, M. H. and C. S. Jung, "An Optimal Parallel Matching Algorithm for a Convex Bipartite Graph on a Mesh-connected Computer," *Journal of Parallel Algorithms and Applications, Vol. 6*, (1995), pp. 129-141.

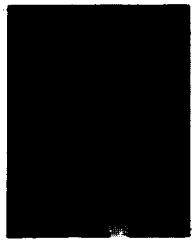
[5] Siegel, H. J., "A Model of SIMD Machines and a Comparison of Various Interconnection Network," *IEEE Transaction on Computer 28*, pp. 907-917, 1979.

[6] Akl, S. G., *The Design and Analysis of Parallel Algorithms*, Prentice-Hall International Editions, 1989.

[7] JáJá, J., *An Introduction to Parallel Algorithms*, Addison Wesley, 1992.

[8] Ladner, R. E. and M. J. Fisher, "Parallel Prefix Computation," JACM, 27, pp. 831-838, 1980.

[9] Cole, R., "Parallel Merge Sort," *SIAM Journal of Computing, Vol. 17*, No. 4, pp. 770-785, 1988.



김 명 호

1989년 숭실대학교 전자계산학과 학사.
1991년 포항공과대학교 전자계산학과 석사.
1995년 포항공과대학교 전자계산학과 박사.
1995년 2월~1995년 8월 학국전자통신연구소 선임연구원.
1995년 9월~현재 숭실대학교 컴퓨터학부 조교수.
관심분야: 분산처리, 가상병렬컴퓨터, 병렬 알고리즘, 병렬처리, 병렬구조.

정 창 성

1981년 서울대학교 전기공학과 학사.
1985년 Northwestern University 전자계산학과 석사.
1987년 Northwestern University 전자계산학과 박사.
1987년~1991년 포항공과대학교 전자계산학과 조교수.
1992년~현재 고려대학교 전자 전기 전파 공학부 부교수.
1992년~현재 Journal of Parallel Algorithms and Applications 편집위원.
관심분야: 병렬 알고리즘, 병렬처리, 병렬구조.