LASOB 상에서 계산 트리 형식을 생성하기 위한 최적 병렬 알고리즘

(An Optimal Parallel Algorithm for Generating Computation Tree Form on Linear Array with Slotted Optical Buses)

김 영 학 †

(Young-Hak Kim)

요 약 최근에 전자 버스 대신에 광 버스를 사용하여 버스의 대역폭을 늘리고 하드웨어의 복잡도를 줄이기 위한 처리기 배열의 구조가 다수의 문헌에서 제안되었다. 본 논문에서는 먼저 슬롯된 광 버스를 갖는 선형 처리기 배열(LASOB) 상에서 괄호 매칭 문제에 대한 상수 시간 알고리즘을 제안한다.

다음에 이 알고리즘을 사용하여 길이 n의 대수 식이 주어지면 n개의 처리기를 갖는 LASOB 상에서 상수 시간에 계산 트리 형식을 생성하는 비용이 최적인 병렬 알고리즘을 제안한다. 아직 임의의 병렬 컴퓨터 모델에서 이 문제에 대한 상수 시간에 수행되는 비용 최적인 병렬 알고리즘은 알려지지 않고 있다.

Abstract Recently, processor arrays to enhance the bandwidth of buses and to reduce the complexity of hardwares, using optical buses instead of electronic buses, have been proposed in many literatures. In this paper, we first propose a constant-time algorithm for parentheses matching problem on a linear array with slotted optical buses (LASOB).

Then, given an algebraic expression of length n, we also propose a cost optimal parallel algorithm that constructs computational tree form in the steps of constant time on LASOB with n processors by using parentheses matching algorithm. A cost optimal parallel algorithm for this problem that runs in constant time has not yet been known on any parallel computation models.

1. 서 론

최근에 다중 처리기 시스템에서 전자 버스(electronic buses)에 의해서 연결된 처리기(processor)들 간의 상호연결을 광 버스(optical buses)로 대치한 병렬 컴퓨터 모델들이 제안되었다[1, 2]. 또한 이들을 계산 모델로 사용하여 정렬과 라루팅에 관련된 다수의 문제들에 대한 효율적인 병렬 알고리즘이 설계되었다[3, 4, 5]. 광 버스의 사용은 전자 버스를 사용할 경우 발생되는 버스 대역폭(bandwidth)의 제한, 용량적 적재(capacitive loading), 크로스 토크(cross-talk) 등의 문제를 해결해 준다[2]. 또한 양방향 전송의 특성을 갖는 전자 버스에 비해서 광 버스는 고유한 방향성을 가지며 단위 길이 당 예측되는 지연 시간을 갖는다.

광 버스의 이러한 특성은 공간적인 병렬성과 파이프라인 원리를 이용하여 한 버스 사이클 동안에 다수의 메시지들이 적재될 수 있다. 즉, 시분할 멀티플렉싱의 개념을 사용하여 한 버스에 다수의 메시지들이 적재된다. 최근에 광 버스를 사용한 다중 처리기 배열로서 RASOB[1], APPB[2], AROB[2]와 같은 병렬 구조들이 제안되었다. 이들 병렬 구조 중에서 RASOB, AROB는 RMESH[6, 7]에서 소개된 재구성 가능한 버스 개념을 사용하며, 알고리즘 수행 동안 각 처리기가 간단한 스위치 세팅에 의해서 동적으로 버스를 재구성할 수 있다.

본 논문에서는 슬롯된(slotted) 광 버스를 이용하여 구성된 선형 처리기 배열(LASOB)을 계산 모델로 사용

· 이 논문은 1999년도 한국학술진흥재단의 연구비에 의해 수행되었음.(KRF-99-003-E00377)

† 종신회원 : 금오공과대학교 컴퓨터공학부 교수
yhkim@cespc1.kumoh.ac.kr

논문접수 : 1999년 8월 17일
심사완료 : 2000년 3월 13일

하며, 이러한 구조는 다른 문헌에서 1차원 RASOB[1] 혹은 APPB[2] 등으로 언급된다. 본 논문에서는 먼저 LASOB 상에서 괄호 매칭(parentheses matching) 문제에 대한 비용 최적인 병렬 알고리즘을 설계한다. 다음에 이 알고리즘을 사용하여 길이 n의 대수 식(algebraic expression)이 주어지면 n개의 처리기를 갖는 LASOB 상에서 상수 시간에 계산 트리 형식(computational tree form)을 생성하는 비용 최적인 병렬 알고리즘을 제안한다. 이러한 문제는 컴파일러 설계와 같은 응용 분야에서 한 대수 식이 주어지면 이 식이 올바른지를 효율적으로 결정하기 위해 파싱 트리를 생성하는데 유용하게 이용된다.

공유 메모리를 갖는 PRAM 모델 상에서 이러한 문제를 해결하기 위한 다수의 병렬 알고리즘들이 제안되었다. Dekel 등은 길이 n을 갖는 대수 식이 주어질 경우에, EREW PRAM에서 n개의 처리기를 사용하여 $O(\log^2 n)$ 시간에 해결하는 알고리즘과 $n^2/\log n$ 개의 처리기를 사용한 $O(\log n)$ 시간 알고리즘을 제안하였다[8]. Bar-on 등은 CRCW PRAM에서 $n/\log n$ 개의 처리기를 사용하여 $O(\log n)$ 시간에 수행되는 병렬 알고리즘을 제안하였다[9]. 또한 Tsang 등은 Bar-on 등의 알고리즘을 EREW PRAM 모델에서 수행되도록 개선하였다[10]. 이 문제에 대한 최초의 상수 시간 알고리즘은 Chen 등에 의해서 제안되었으며, 그들은 2차원 $n \times n$ 크기를 갖는 RMESH를 사용하여 상수시간에 수행 가능한 병렬 알고리즘을 제안하였다[11].

그러나 Chen 등이 제안한 상수 시간 알고리즘은 2차원 $n \times n$ RMESH에서 수행되며 병렬 알고리즘의 비용(평가) 척도[12]인 *(수행시간×처리기수)*에서 $O(n^2)$의 복잡도를 갖는다. 위의 문제들에 대한 최적의 순차 알고리즘이 $O(n)$ 시간 복잡도 임을 감안하면 Chen 등의 알고리즘은 비용 최적이 아니다. 따라서 본 연구에서는 LASOB 상에서 병렬 알고리즘의 비용 척도에서 최적인 $O(n)$의 복잡도를 갖는 상수 시간 알고리즘을 설계한다. 현재 임의의 병렬 컴퓨터 모델 상에서 이 문제에 대한 상수 시간에 수행되는 비용 최적인 병렬 알고리즘은 알려지지 않고 있다.

본 논문의 2장에서는 계산 모델로 사용될 LASOB의 개념적 구조를 설명하고, 3장에서는 본 논문에서 고려한 문제의 해결을 위해 기본적인 도구로 사용될 괄호 매칭 문제에 대한 알고리즘을 설계한다. 4장에서는 임의의 대수 식이 주어질 때 LASOB 상에서 상수 시간에 계산 트리 형식을 생성하기 위한 상세한 알고리즘을 설명한다. 마지막으로 5장에서는 결론을 맺는다.

2. LASOB 모델

전자 버스에 비해서 광 버스를 사용한 연결은 고 집적도와 더 넓은 대역폭을 가지기 때문에, 최근에 광 버스는 다중처리기 시스템에 대한 연결 대안으로 고려되고 있다[1]. 현재 광 버스를 사용한 몇 개의 병렬 구조들이 제안되었으며, 이들을 계산 모델로 사용하여 다수의 문제들에 대한 효율적인 알고리즘이 개발되었다. 본 논문에서는 이들 병렬 구조 중에서 가장 간단한 구조를 갖는 슬롯된 광 버스를 갖는 선형 처리기 배열(LASOB)을 계산 모델로 사용한다. 본 연구에서 고려한 LASOB의 기본적인 구조는 Chen 등[1]의 1 차원 RASOB 혹은 Pavel 등[2]의 APPB와 같은 구조를 갖는다. 이러한 구조에 대한 더 상세한 설명은 이들의 논문을 참조하길 바란다.

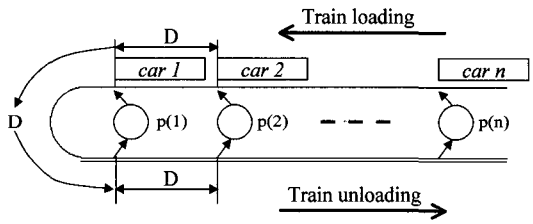

그림 1 슬롯된 광 버스를 갖는 선형 처리기 배열

그림 1은 LASOB의 기본적인 구조를 보여준다. 그림 1에서 보여진 것과 같이 각 처리기는 가장 왼쪽부터 P(1), P(2), ... , P(n) 순으로 정의되어 있고, 각 처리기 사이의 광 버스의 거리는 일정한 간격을 갖는다. 버스는 위쪽 세그멘트(train loading)와 아래쪽 세그멘트(train unloading)로 구성되며, 각 처리기는 위쪽 세그멘트에 자료를 쓰고(write) 아래쪽 세그멘트로부터 자료를 읽는다. 두 세그멘트 상에서 서로 인접한 처리기는 지속시간 D를 가지고 분할되며, 지속시간 D는 $bw+\delta$로 표현된다. 여기서 b는 비트 단위의 한 패킷의 최대길이를 의미하고, w는 한 비트가 초 단위에서 지속되는 광 펄스 폭을 의미한다. 또한 $\delta > 0$는 각 처리기 사이에 동기화를 위해 사용되는 여유시간을 의미한다.

이러한 공간적 분할은 두 세그멘트 상에서 송신과 수신 인터페이스를 사용하면 가능하고 $D \times c$의 광 화이버의 길이를 갖는다. 여기서 c는 빛의 속도를 의미한다. 한 개의 버스 사이클 동안 위쪽 세그멘트의 버스에 다수의 처리기들이 다른 시간 슬롯을 사용하여 메시지를

전송할 수 있다. 이러한 원리는 시분할 멀티플렉싱을 사용하며 광 버스가 고유한 방향성과 단위 길이 당 예측되는 지연(predictable delay) 시간을 갖기 때문에 가능하다. 그림 1에서와 같이 n car의 트레인이 위쪽 세그멘트 버스의 가장 오른쪽 끝에서 시작하였다고 가정한다. 각 car는 지속시간 D를 갖는 슬롯으로 분할되고 왼쪽으로부터 *1*에서 *n*까지의 번호를 갖는다.

가장 간단한 슬롯의 사용 예는 처리기 P(1)에 car 1, P(2)에 car 2, ..., P(i)에 car i를 할당하여 각 처리기가 해당 슬롯에 메시지를 전송하게 된다. 이러한 방법에 의한 슬롯의 할당에서 각 처리기는 한 메시지를 전송하기 위해서 정확하게 $(n-1)D$의 시간 지연을 갖게된다. 또한 다수의 처리기가 동시에 아래쪽 세그멘트 버스를 통하여 해당되는 슬롯으로부터 메시지를 읽을 수 있다. 다수의 소스와 다수의 목적지 처리기간에 메시지 전달 및 처리기들 간의 동기화를 위해서 TDSM, TDDM, 코인시던스(coincidence) 펄스, 대기 함수(waiting function) 등의 기술이 사용된다[1, 2]. 본 논문에서 고려한 LASOB는 1차원 RASOB[1]에서와 같이 대기 함수에 의해서 전체 시스템이 동기화 되었다고 가정한다.

만일 처리기 P(i)가 처리기 P(j)에 의해 전송된 메시지를 읽으려 한다면, 처리기 P(j)는 $(n+i+j-2)D$에 의해서 계산된 시간에 아래쪽 세그멘트 버스의 슬롯으로부터 메시지를 기대할 수 있다. 모든 처리기들은 동기적 제어에 의해서 작동되며 각 처리기는 특정된 시간에 따라 메시지를 주고받을 수 있다. n개의 car가 위쪽 세그멘트에서 시작하여 아래쪽 세그멘트를 통하여 전송되는 과정을 한 개의 *통신 사이클(communication cycle)*로 정의하며, $\delta=2n\tau$ 초의 시간을 갖는다. 여기서 n은 처리기의 수를 의미하고, τ는 한 메시지가 인접한 처리기를 이동하기 위한 지속시간 D를 의미한다. 또한 δ는 종점 대 종점 전파(end-to-end propagation) 시간이라 정의한다. 따라서 한 처리기가 다른 처리기와 메시지를 주고받기 위해서는 $O(\delta)$ 시간만큼을 대기해야 한다.

RMESH[6, 7]에서 가정했던 것과 같이 LASOB에서 각 처리기는 단위 시간에 산술 및 논리 연산을 수행할 수 있으며, $O(1)$ 개의 기억 장소를 갖는다. 한 통신 사이클에 필요한 시간 δ는 실제적인 크기(10,000 정도)를 갖는 처리기 수로 구성된 시스템에서 한 개의 CPU 연산에 비교되며 상수 시간으로 가정할 수 있다[2]. 따라서 알고리즘 성능 평가 시에 한 개의 내부 연산 혹은 한 개의 통신 사이클을 상수 시간에 수행되는 한 개의 단계(step)로 볼 수 있다.

3. 괄호 매칭 문제

이 장에서는 본 논문에서 고려한 문제에 대한 알고리즘 설계를 위해 주요하게 사용될 LASOB 상에서 괄호 매칭 문제에 대한 알고리즘을 설명한다. 괄호 매칭 문제는 다음과 같이 정의된다[9]. 왼쪽 괄호('(')와 오른쪽 괄호(')')로 구성된 문자열 $\sigma=x_1x_2...x_n$이 주어질 때, 왼쪽과 오른쪽 괄호의 수가 같은 개수를 갖고 각 괄호에 대한 전위 특성(prefix property)이 만족하는 지를 확인한다. 여기서 각 x_i, $1\le i\le n$은 왼쪽 혹은 오른쪽 괄호의 심벌을 의미한다. 이러한 성질을 만족하는 입력 문자열 σ는 well-formed 되었다고 정의된다. 또한 well-formed 문자열 σ가 주어질 때 각각의 왼쪽과 오른쪽 괄호의 매칭 쌍을 찾는 문제를 괄호 매칭 문제라고 한다.

현재 다수의 병렬 컴퓨터 모델 상에서 괄호 매칭 문제에 대한 효율적인 병렬 알고리즘들이 제안되었다[9, 10, 11, 13, 14, 15]. Bar-on 등은 $|\sigma|=n$인 입력 문자열이 주어질 때 CREW PRAM 상에서 $O(n/\log n)$ 개의 처리기를 사용하여 $O(\log n)$ 시간에 수행되는 병렬 알고리즘을 제안하였다[9]. 후에 서로 다른 기법을 사용하여 EREW PRAM 상에서 $O(n/\log n)$ 개의 처리기를 사용하여 $O(\log n)$ 시간에 수행되는 다수의 비용 최적인 알고리즘들이 제안되었다[10, 13, 14]. 또한 이 문제에 대한 상수 시간 알고리즘은 [11, 15]에서 제안되었다. [11]에서는 2차원 $n\times n$ RMESH 상에서 상수 시간에 수행되는 비용 최적이 아닌 병렬 알고리즘을 제안하였다. [15]에서는 $O(n)$ 개의 메모리 영역을 공유하는 $O(n)$ 개의 처리기를 사용한 BSR 모델에서 상수 시간에 수행되는 병렬 알고리즘을 제안하였다.

따라서 본 논문에서는 이 문제가 이들 구조보다 더 현실적인 LASOB 상에서 상수 시간에 수행되며 비용 최적인 병렬 알고리즘을 설계한다. 괄호 매칭 문제에 대한 순차 알고리즘은 스택을 이용하면 $O(n)$ 시간에 쉽게 해결될 수 있다. 이 문제에 대한 LASOB 상에서 수행되는 병렬 알고리즘의 설계를 위해서는 위에서 고찰한 다른 연구에서와 같이 전위 합(prefix sum) 문제가 필수적으로 사용된다. 전위 합 문제는 다양한 컴퓨터 모델에서 병렬 알고리즘의 설계에 널리 알려진 기술이다. 예로써 $\sigma=((()))$의 문자열이 주어질 경우에 왼쪽 괄호의 심벌 위치에는 1을 오른쪽 괄호의 심벌 위치에는 -1을 할당하고 전위 합을 계산한다. 그러면 각각의 전

위 합의 결과가 음수가 아니고 최종 전위 합이 0이면 위에서의 정의에 따라 입력 문자열 σ는 well-formed가 된다.

다음에 well-formed인 σ에 대해서 각각의 괄호에 대한 매칭 쌍을 찾기 위해서 연속되는 왼쪽 괄호('('에 대해서는 1을 연속된 오른쪽 괄호(')')에 대해서는 -1을 할당하고 그렇지 않은 경우에 대해서는 0을 할당한다. 위의 예의 경우에 적용하면 0,1,0,0,0,-1이 되고 각 심벌의 위치에 대한 전위 합은 0,1,1,1,1,0가 된다. 여기서 각 괄호에 대한 매칭 쌍은 자신의 전위 합과 같은 값을 갖는 가장 가까운 위치의 인덱스 정보를 확인하면 쉽게 결정될 수 있다.

다음에는 지금까지 개략적으로 설명한 과정이 LASOB 상에서 어떻게 상수 시간에 구현될 수 있는 지를 상세하게 설명한다. 먼저 알고리즘 설계를 위해 사용될 기본적인 연산들을 정의한다.

보조정리 1. *1, 2, … , n*의 한 순열(permutation)을 $\pi(1), \pi(2), \cdots, \pi(n)$이라 정의한다. 그러면 LASOB 상의 모든 처리기 P(i), $1 \le i \le n$은 자신의 데이타를 처리기 P($\pi(i)$)에 상수 시간에 전송할 수 있다[1, 2].

위의 보조정리에서 각 처리기 P(i)가 자료를 전송할 목적지의 처리기에 대한 인덱스 $\pi(i)$을 알고 있다면 LASOB 구조의 기본적인 동작에 의해서 소스와 목적지 간에 1 대 1 매핑이 가능하며 한 통신 사이클에 수행가능 하다. 이러한 매핑은 TDSM 혹은 TDDM에 의해서 쉽게 해결될 수 있다.

보조정리 2. *n*개의 처리기를 갖는 LASOB 상에서 각 처리기에 한 개의 비트 정보가 저장되어 있을 때, 이들 각 비트들에 대한 전위 합은 상수 시간에 계산될 수 있다[2].

다음에는 임의의 심벌 혹은 '^'으로 구성된 길이 *n*의 리스트 $\{a_1, a_2, \cdots, a_n\}$이 주어질 때, '^'을 제외한 나머지 값들을 앞쪽부터 순서적으로 재배열하는 문제(자료 압축)를 고려한다. 예로서 {+, ^, *, (, ^, ^, 7}의 리스트가 입력으로 주어질 때, 출력 리스트는 {+, *, (, 7, ^, ^, ^}이 되도록 입력 리스트를 재배열한다. *n*개의 처리기를 갖는 LASOB 상에서 각 처리기 P(i)에 a_i가 저장되어 있을 경우 이러한 문제의 해결을 위해서 먼저 a_i가 '^' 일 경우는 0, 나머지 경우는 1로 설정한다. 다음에 보조정리 2에 의해서 전위 합을 계산하고 보조정리 1에 의해서 P(i)에 저장된 심벌을 자신의 전위 합에 해당되는 처리기로 라우트 하면, 이러한 문제는 상수 시간에 해결될 수 있다. 따라서 위의 결과를 다음 보조정리로 요약한다.

보조정리 3. 임의의 심벌 혹은 '^'으로 구성된 길이 *n*의 리스트가 주어지면, *n*개의 처리기를 갖는 LASOB 상에서 '^'을 제외한 나머지 값들을 앞쪽부터 순서적으로 재배열하는 문제는 상수 시간에 해결될 수 있다.

이제 이러한 도구를 이용하여 괄호 매칭 문제를 해결하는 방법을 설명한다. 먼저 입력 문자열 $\sigma = x_1 x_2 \ldots x_n$이 주어질 때, σ가 well-formed 되었는가를 확인하는 방법을 알아본다. 초기에 LASOB 상의 각 처리기 P(i)에 입력 문자열 x_i가 저장되어 있다고 가정한다. 각 처리기 P(i)는 x_i가 '('일 경우 1을 할당하고 ')'일 경우는 -1을 할당한다. 이러한 값들에 대한 전위 합은 보조정리 2를 다음과 같은 절차에 따라 두 번 적용하면 쉽게 해결될 수 있다. 처음에 처리기 P(i)가 -1을 가질 경우 0으로 설정하면 *n*개의 이진 값들에 대한 전위 합 문제로 정의되므로 보조정리 2를 바로 적용할 수 있다. 다음에 처리기 P(i)가 1을 가질 경우 0으로 설정하고 -1을 가질 경우 1로 설정하면 이진 값들의 전위 합 문제로 정의되므로 보조정리 2를 다시 적용할 수 있다. 각 처리기 P(i)는 첫 번째 사이클에서 계산된 전위 합과 두 번째 사이클에서 계산된 전위 합의 차를 계산하면 전체 전위 합을 상수 시간에 계산할 수 있다.

입력 문자열 σ가 well-formed 되었는지는 마지막 처리기 P(n)이 자신의 전위 합이 0인지를 확인하고 각 처리기 P(i)가 음수가 아닌 전위 합을 갖는 지를 확인하면 된다. 실제 이러한 경우는 처리기 P(n)에서 만 결정될 수 있다. 각 처리기 P(i)가 음수 값을 가질 경우는 1을 그렇지 않으면 0을 설정하여 전위 합을 계산하면, 음수가 포함될 경우 처리기 P(n)은 0이 아닌 값을 갖기 때문에 각 처리기 P(i)에서의 전위 합이 음수가 포함되었는가의 여부를 쉽게 결정할 수 있다. 따라서 다음의 정리가 증명된다.

보조정리 4. *n*개의 처리기를 갖는 LASOB 상에서 왼쪽과 오른쪽 괄호로 구성된 입력 문자열 $\sigma = x_1 x_2 \ldots x_n$이 주어질 때, σ가 well-formed 되었는가의 결정은 상수 시간에 계산될 수 있다.

다음에 입력 문자열 σ가 well-formed 되었을 경우에

각 괄호에 대한 매칭 쌍을 결정하는 과정을 설명한다. 각 괄호에 대한 매칭 쌍을 결정하는 개략적인 과정은 다음과 같다.

단계 1. $\sigma = x_1 x_2 \ldots x_n$에서 x_{i-1}과 x_i가 모두 왼쪽 괄호('((') 일 경우 w_i를 1로 설정하고, x_{i-1}과 x_i가 모두 오른쪽 괄호('))') 일 경우 w_i를 -1로 설정한다. 나머지 경우는 w_i를 0으로 설정한다.

단계 2. 각 w_i, $1 \le i \le n$에 대한 전위 합 e_i를 계산한다.

단계 3. 전위 합 e_i를 이용하여 각 x_i에 대응되는 괄호 x_j를 결정한다.

이제 위의 각 단계가 LASOB 상에서 어떤 방법에 의해서 상수 사이클에 해결될 수 있는 지를 상세하게 설명한다. 초기에 각 처리기 P(i)에 입력 심벌 x_i가 저장되어 있다고 가정한다.

[단계 1, 2] LASOB 상의 각 처리기 P(i), $1 \le i \le n-1$가 갖는 심벌 x_i를 처리기 P(i+1)에 라우트 한다. 그러면 각 처리기 P(i)는 x_{i-1}과 x_i의 값을 가지며 바로 w_i의 값을 결정할 수 있다. 각 처리기간의 자료의 라우팅은 보조정리 1에 의해서 상수 시간에 수행 가능하다. 단계 2에서 처리기 P(i)에 저장된 각 w_i에 대한 전위 합 e_i는 보조정리 2에 의해서 상수 시간에 계산될 수 있다. 위의 well-formed 인지를 결정하는 방법에서의 전위 합 계산 절차를 참조한다.

[단계 3] 각 처리기 P(i), $1 \le i \le n$가 자신의 전위 합 e_i를 LASOB의 위쪽 세그먼크의 분할된 슬롯 car i에 적재한다. 각 처리기 P(i)는 아래쪽 세그멘트를 통과하는 모든 슬롯의 데이타를 읽어서 $e_i = e_k$이고 $i < k$를 최초로 만족하는 인덱스 값 k를 저장하고, 이러한 조건을 만족하지 않으면 '^'을 저장한다. 여기서 e_k는 car k에 저장되어 있고 LASOB 상의 모든 처리기는 동기화 되어 있기 때문에 인덱스 k는 쉽게 결정될 수 있다. 위의 과정은 한 통신 사이클에 수행가능하며 각 처리기 P(i)는 k 혹은 ,^'의 값을 갖는다. 같은 방법으로 다음 통신 사이클에 각 처리기 P(i)는 자신의 전위 합 e_i를 LASOB의 위쪽 세그멘트의 분할된 슬롯 car i에 적재한다. 각 처리기 P(i)는 아래쪽 세그멘트를 통과하는 모든 슬롯의 데이타를 읽어서 $e_i = e_{k'}$이고 $i > k'$를 최초로 만족하는 인덱스 값 k'를 저장하고, 이러한 조건을 만족하지 않으면 '^'을 저장한다. 그러면 각 처리기 P(i)는 k' 혹은 '^'의 값을 갖는다.

두 사이클을 통하여 얻어진 최종결과를 정리하면 각 처리기 P(i)는 <k,^>, <^,k'>, <k,k'> 중의 한 개의 쌍을 갖는다. 그러면 P(i)는 <k,^> 혹은 <^,k'>을 가질 경우는 입력 심벌 x_i의 매칭 쌍이 각각 x_k 혹은 $x_{k'}$임을 결정할 수 있고, <k,k'>을 가질 경우는 입력 심벌 x_i가 '('이면 x_k로 ')'이면 $x_{k'}$으로 매칭 쌍을 결정할 수 있다. 실제 각 처리기 P(i)에는 자신의 괄호에 매칭되는 괄호에 대한 인덱스 값이 저장된다. 따라서 다음의 정리가 증명된다.

정리 1. n개의 처리기를 갖는 LASOB 상에서 왼쪽과 오른쪽 괄호로 구성된 입력 문자열 $\sigma = x_1 x_2 \ldots x_n$이 주어질 때, 각 괄호에 대한 매칭 쌍의 결정은 상수 시간에 계산될 수 있다.

4. 계산 트리를 구성하는 알고리즘

이 장에서는 길이 n을 갖는 대수 식이 주어질 때 $5n$개의 처리기를 갖는 LASOB 상에서 상수 시간에 계산 트리 형식을 생성하는 알고리즘을 설명하고, 실제 이 문제가 n개의 처리기를 갖는 LASOB 상에서 상수 시간에 수행 가능함을 보인다. 입력 식에 포함된 연산자는 +, -, *, ↑(승수 연산자), 괄호, 오퍼런드(상수와 변수)로 구성된다. 각 연산자의 우선 순위는 보통의 산술식에서와 같이 ↑, *, /, +, -에 따라 적용되고, ↑연산자는 오른쪽 결합(right associative)을 갖고 나머지 연산자들은 왼쪽 결합을 갖는다고 정의한다. 예로서 a+b/c/d↑e↑g+h의 식은 (a+((b/c)/(d↑(e↑g))))+h와 같은 의미를 갖는다.

본 논문에서는 이러한 문제를 해결하기 위해서 [9, 11]에서 제안된 알고리즘을 LASOB 상에서 상수 시간에 수행되도록 구현하였다. 알고리즘 설명을 위해 [11]에서 사용했던 것과 같은 대수 식인 a/b-c+d↑(e/f+g↑h)를 예로 사용한다. 또한 LASOB 상의 우리의 알고리즘 수행 후에 예제로 주어진 대수 식에 대한 최종적인 계산 트리의 형식이 그림 2와 같이 생성됨을 보인다.

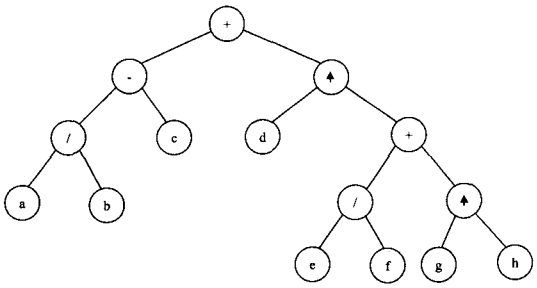

그림 2 예제 입력 대수 식에 대한 계산 트리 형식

괄호로 둘러 쌓이지 않은 모든 연산자들이 같은 순위를 갖는 식을 단순 식(simple expression)으로 정의한다. 예로서 a+(b/c)-d는 단순 식이지만 a+b/c-d는 단순 식이 아니다. 단순 식 $S = E_0 op_1 E_1 op_2 E_2 \ldots op_k E_k$이 주어졌다고 가정한다. 여기서 각 E_i는 오퍼런드 혹은 괄호 쌍에 의해서 정의된 단순 서브 식을 의미하고, op_i는 같은 우선 순위를 갖는 연산자를 의미한다. 각 연산자는 왼쪽(오른쪽) 결합을 갖고 각 E_i의 트리 형식에 대한 루트 r_i가 이미 알려졌다고 가정할 때, S의 계산 트리 형식은 아래와 같은 절차에 의해서 생성될 수 있다.

- op_k (op_1)이 루트가 된다.
- 각 op_i, ($2 \le i \le k$)의 왼쪽과 오른쪽 자식은 각각 op_{i-1}(r_{i-1})과 r_i(op_{i+1})이 된다.
- op_1(op_k)의 왼쪽과 오른쪽 자식은 각각 r_0 (r_{k-1})과 r_1(r_k)가 된다.

따라서 주어진 대수 식이 단순 식으로 변환되면 위와 같은 절차에 의해서 계산 트리를 생성할 수 있다. 본 논문에서는 주어진 대수 식에 대한 계산 트리를 생성하기 위해서 [9]에서 제안된 아래와 같은 절차를 LASOB 상에 구현한다. 또한 [11]에서는 [9]에서 제안된 PRAM 알고리즘을 $n \times n$ RMESH에서 상수 시간에 수행되도록 구현하였다.

단계 1. Knuth의 괄호 삽입 절차[16]를 사용하여 입력 식이 단순 식이 되도록 변형한다. 그러면 괄호의 쌍에 의해서 포함된 모든 서브 식은 단순 식이 된다.

단계 2. 단계 1에서 생성된 식에서 모든 괄호의 매칭 쌍을 결정한다.

단계 3. 매칭된 괄호의 쌍에 포함된 단순 서브 식의 루트를 결정하고 각 연산자의 왼쪽과 오른쪽 자식을 결정한다.

이제 위의 각 단계가 LASOB 상에서 어떻게 구현되는 가를 상세하게 설명한다. 먼저 이 문제의 해결을 위해서 $5n$개의 처리기를 갖는 LASOB를 고려하고, 초기에 길이 n인 입력 대수 식이 가장 왼쪽 n개의 처리기들에 연속적으로 저장되어 있다고 가정한다. 그러면 입력 식 $\sigma = x_1 x_2 \cdots x_n$이 주어지면 각 처리기 P(i), $1 \le i \le n$는 x_i의 값을 갖는다. 여기서 x_i는 상수, 변수, 연산자, 괄호 등으로 구성된다. 다음은 위의 각 단계가 LASOB 상에서 상수 시간에 수행되는 상세한 과정을 설명한다.

[단계 1] 괄호의 삽입

주어진 입력 식에 Knuth의 괄호 삽입 알고리즘을 적용하여 괄호의 쌍에 포함된 모든 서브 식이 단순 식이 되도록 한다. Knuth의 괄호 삽입 절차는 아래와 같다.

- x_i가 '+' 혹은 '-'이면 '))x_i((' 이 되도록 괄호를 삽입한다.
- x_i가 '*' 혹은 '/'이면 ')x_i(' 이 되도록 괄호를 삽입한다.
- x_i가 '(' 혹은 ')'이면 각각 '((x_i' 혹은 'x_i))'이 되도록 괄호를 삽입한다.
- 가장 왼쪽과 오른쪽에 각각 두 개의 왼쪽 괄호와 오른쪽 괄호를 추가한다.

위의 절차를 관찰하면 추가적인 괄호 삽입에 의해 각 x_i에 대해서 최대 5개의 폭이 증가된다. 따라서 우리는 이러한 문제를 쉽게 설명하기 위해서 $5n$개의 처리기를 갖는 LASOB를 사용한다. 또한 일관성을 유지하기 위해서 괄호 삽입 후에 폭이 5이하인 형태는 뒤쪽에 공란(^)을 삽입한다. 예로서 ')x_i('은 ')x_i(^^', '((x_i'은 '((x_i^^'과 같은 형식으로 표현하고 이를 $x_i^1 x_i^2 x_i^3 x_i^4 x_i^5$로 정의한다. 초기에 각 처리기 P(i), $1 \le i \le n$는 x_i를 가지고 있으므로 위의 규칙을 바로 적용할 수 있으며, 특별히 처리기 P(1)과 P(n)은 마지막 규칙을 다시 적용한다. 즉, 처리기 P(1)은 두 개의 왼쪽 괄호를 추가하고 P(n)은 두 개의 오른쪽 괄호를 추가한다.

그러면 각 처리기 P(i), $1 \le i \le n$는 x_i를 포함하여 $x_i^1 x_i^2 x_i^3 x_i^4 x_i^5$의 형식을 갖는 5개의 심벌로 구성된다. 이

제 가장 왼쪽 n개의 각 처리기에 5개의 값으로 표현되어 저장된 이러한 정보를 $5n$개의 처리기에 분산하여 저장되도록 한다. 즉 처리기 P(i), $1 \le i \le n$에 저장된 $x_i^1 x_i^2 x_i^3 x_i^4 x_i^5$에서 각 x_i^k를 처리기 P($5(i-1)+k$)에 라우트 한다. 이와 같은 자료의 라우팅은 보조정리 1에 의해서 해결될 수 있으며, 5회의 통신 사이클을 수행하면 모든 정보들이 $5n$개의 처리기에 분산되어 차례로 저장된다. 본 논문의 예로 사용된 식에 위의 규칙을 적용하면 아래와 같은 형식으로 $5n$개의 처리기에 차례로 저장된다. 여기서 각 심벌은 한 개의 처리기에 저장되어 있다는 것을 주지한다.

((a^^)/(^^b^^^^))-((c^^^^))+((d^^^^ ↑ ^^^^(((^^e^^^^)/(^^f^^^^))+((g^^^^ ↑ ^^^^h^^^^)))))

위의 결과를 관찰하면 5개의 폭을 일관적으로 늘린 이유 때문에 '^'을 갖는 불필요한 부분이 생성된다. 따라서 '^'을 제외한 나머지 값들을 가장 왼쪽 처리기부터 차례로 저장해야 한다. 이러한 과정은 보조정리 3에 의해서 상수 시간에 해결될 수 있다. 그러므로 단계 1은 상수 시간에 수행되며 매칭된 괄호의 쌍에 포함되는 모든 서브 식은 well-formed 인 단순 식으로 변형된다. 단계 1을 수행한 후에는 가장 왼쪽 처리기부터 아래와 같은 형태의 값들이 저장된다.

((a)/(b))-((c))+((d ↑(((e)/(f))+((g ↑h)))))

[단계 2] 괄호 매칭과 불필요한 괄호의 제거

이 단계에서는 단계 1의 결과를 이용하여 모든 괄호에 대한 매칭 쌍을 결정하고 불필요한 괄호를 제거한다. 먼저 모든 괄호에 대한 매칭 쌍은 정리 1에 의해서 결정될 수 있으며, 괄호를 갖는 처리기 P(i)는 자신의 괄호에 매칭되는 괄호의 위치를 지역변수 $m(i)$에 저장한다. 불필요한 괄호를 제거하는 규칙은 다음과 같다.

- ((서브식)) 형태의 괄호를 가질 경우 '^(서브식)^' 형식으로 변형한다.
- (오퍼런드) 형태의 괄호를 가질 경우 '^오퍼런드^' 형식으로 변형한다.

단계 1에서 '^'을 제외하고 정돈된 식을 E라 정의하고 각각을 $e_1 e_2 \cdots e_{|E|}$로 표현한다. 먼저 ((서브식)) 형태를 '^(서브식)^' 형식으로 변환하는 과정을 설명한다. 보조정리 1에 의해서 각 처리기 P(i), $2 \le i \le |E|$가 자신의 $m(i)$와 e_i의 복사 본을 P(i-1)에 라우트 한다. 그러면 모든 처리기 P(i), $1 \le i \le |E|$는 e_i, e_{i+1}, $m(i)$, $m(i+1)$의 값을 가지므로 이 규칙을 바로 적용할 수 있다. 각 처리기 P(i)는 만일 e_i와 e_{i+1}이 '(('이고 $m(i+1)$이 $m(i)-1$의 값과 동일하면 e_i를 '^'으로 설정한다. 그러면 첫 번째 경우의 불필요한 왼쪽 괄호는 모두 '^'으로 대치될 수 있으며, 같은 방법을 적용하면 불필요한 오른쪽 괄호 역시 '^'으로 대치할 수 있다.

다음에는 두 번째 경우, '(오퍼런드)' 형식을 '^오퍼런드^'로 변환하는 과정을 설명한다. 이러한 경우에서 모든 처리기 P(i), $1 \le i \le |E|$는 e_i가 '('이고 $m(i)$가 $i+2$와 같거나 e_i가 ')'이고 $m(i)$가 $i-2$와 같을 경우에 e_i를 '^'으로 설정한다. 이러한 계산은 자료 라우팅 없이 처리기 P(i)가 갖고 있는 정보만으로 충분하므로 상수 시간에 수행 가능하다.

위의 두 규칙을 적용하면 $e_1 e_2 \cdots e_{|E|}$에서 불필요한 괄호를 갖는 모든 e_i들은 '^'으로 대치된다. 따라서 보조정리 3에 의해서 '^'을 제외한 모든 값들을 가장 왼쪽 처리기부터 차례로 저장하고, 이러한 식을 E′으로 정의하며 각각을 $e'_1 e'_2 \cdots e'_{|E'|}$으로 표현한다. 또한 정리 1에 의해서 모든 괄호에 대한 매칭 쌍을 다시 결정하고, 각 처리기 P(i), $1 \le i \le |E'|$는 e'_i가 '(' 혹은 ')'일 경우에 그에 매칭 되는 괄호의 인덱스 값을 지역변수 $m'(i)$에 저장한다. 단계 1의 예제로부터 단계 2를 수행한 후의 결과는 아래의 식과 같다.

((a/b)-c+(d ↑((e/f)+(g ↑h))))

[단계 3] 계산 트리 형식의 생성

이 단계에서는 각 연산자에 대한 왼쪽과 오른쪽 자식을 결정하여 이진 트리 형태를 갖는 계산 트리 형식을 생성한다. 즉, 단계 2에서 계산되어 차례로 나열된 $e'_1 e'_2 \cdots e'_{|E'|}$에서 연산자를 갖는 각 e'_i의 왼쪽과 오른쪽 자식을 결정해야 한다. Lchild(i)와 Rchild(i)를 각각 연산자를 갖는 e'_i의 왼쪽과 오른쪽 자식으로 정의한다. 단계 2에서 생성된 식 E′에서 괄호의 쌍에 의해 포함된 모든 서브 식은 이미 단순 식으로 구성되었으므로 각 연산자에 대한 왼쪽과 오른쪽 자식의 결정은 이 장의 초기에 개략적으로 설명했던 절차를 따른다. 계산 트리를 생성하기 위해서 아래와 같은 두 개의 절차를 사용하며, 후에 각 절차를 LASOB 상에서 구현하는 상세한 방법을 설명한다.

- E′에서 괄호의 쌍으로 구성된 모든 단순 서브 식들에 대한 루트를 결정한다.
- E′에서 각 연산자에 대한 왼쪽과 오른쪽 자식을 결정한다.

단계 3.1 단순 서브 식들의 루트를 결정.

먼저 $5n$ 개의 처리기를 갖는 LASOB 상에서 E′에 포함된 모든 단순 서브 식들에 대한 루트를 결정하는 방법을 설명한다. 실제로는 |E′| 개의 처리기들만이 사용된다.

(1) 처리기 P(i), $1 \le i \le |E'|$가 e'_i의 값이 '('인 경우 e'_i와 $e'_{m'(i)}$의 괄호의 쌍에 포함된 단순 서브 식의 첫 번째 연산자와 그 인덱스를 결정한다. 만일 e'_{i+1}이 오퍼런드 이면 왼쪽 괄호 e'_i의 첫 번째 연산자는 e'_{i+2}이 되고 그 인덱스는 $i+2$가 되며, 그렇지 않은 경우는 $e'_{m'(i+1)+1}$이 되고 그 인덱스는 $m'(i+1)+1$이 된다. 따라서 e'_i가 '('을 갖는 각 처리기 P(i), $1 \le i \le |E'|$가 첫 번째 연산자를 상수 시간에 결정하기 위해서는 e'_{i+1}, e'_{i+2}, $e'_{m'(i+1)+1}$, $m'(i+1)$의 값들을 가져야 한다. 다음과 같은 몇 번의 자료 라우팅에 의해 각 P(i)는 위의 정보들을 가질 수 있다.

(1.1) 모든 처리기 P(i), $2 \le i \le |E'|$가 e'_i와 $m'(i)$의 값을 P(i-1)에 라우트 하고, e'_i가 ')'을 갖는 처리기 P(i), $1 \le i \le |E'|$가 바로 전에 전송 받은 e'_{i+1}을 처리기 P(m′(i))에 라우트 한다.

(1.2) e'_i가 오퍼런드를 갖는 처리기 P(i)가 e'_{i+1}의 값을 P(i-1)에 라우트 한다.

(1.3) e'_i가 '('을 갖는 처리기 P(i)가 $e'_{m'(i)+1}$의 값을 P(i-1)에 라우트 한다.

여기서 각 괄호에 매칭 되는 쌍은 단계 2에서 이미 결정되었고 유일한 값을 갖기 때문에 자료의 라우팅은 보조정리 1에 의해서 가능하고, 5번의 통신 사이클을 수행하면 이러한 자료의 라우팅이 완료된다. 마지막으로 e'_i가 '('을 갖는 처리기 P(i)가 e'_{i+1}이 오퍼런드 이면 e'_i의 첫 번째 연산자인 FOP_i를 e'_{i+2}로 설정하고 그 인덱스 값인 FF_i를 $i+2$로 설정한다. 그렇지 않을 경우에 처리기 P(i)는 FOP_i를 $e'_{m'(i+1)+1}$으로 설정하고 FF_i를 $m'(i+1)+1$로 설정한다.

(2) e'_i가 ')'인 경우 처리기 P(i)는 $e'_{m'(i)}$와 e'_i에 포함된 서브 식에 대한 마지막 연산자 LOP_i와 그 인덱스인 LL_i를 결정한다. 이것은 (1)에서의 절차와 유사한 방법을 사용하면 상수 시간에 쉽게 해결될 수 있다.

(3) 다음에는 e_i와 $e_{m'(i)}$에 의해 매칭 되는 괄호의 쌍에 포함된 서브 식의 루트 R_i를 결정한다. 각 e'_i, $1 \le i \le |E'|$가 '('이고 FOP_i가 오른쪽 결합을 갖는 연산자일 경우 처리기 P(i)가 FF_i 값을 처리기 P(m′(i)에 전송하고, 처리기 P(i)와 P(m′(i))는 각각 R_i와 $R_{m'(i)}$를 FF_i의 값으로 설정한다. 같은 방법으로 각 e'_i의 값이 ')'이고 LOP_i가 왼쪽 결합을 갖는 연산자일 경우 처리기 P(i)가 LL_i의 값을 P(m′(i))에 라우트 하고, 처리기 P(i)와 P(m′(i))는 각각 R_i와 $R_{m'(i)}$를 LL_i의 값으로 설정한다. 이러한 과정은 보조정리 1에 의해서 두 번의 통신 사이클에 자료의 전송이 가능하며, 처리기 P(i)와 P(m′(i))는 R_i와 $R_{m'(i)}$의 값을 바로 계산할 수 있다. 표 1은 지금까지 설명한 과정을 완료한 후의 결과를 보여준다.

단계 3.2 각 연산자의 자식을 결정.

표 1과 같이 계산된 자료를 이용하여 다음과 같은 방법에 의해 E′에 포함된 각 연산자의 왼쪽과 오른쪽 자식을 결정한다.

(1) 왼쪽 결합을 갖는 연산자('+', '−', '*', '/') e'_i의 왼쪽 자식 Lchild(i)를 아래와 같은 규칙에 따라 결정한다.

- e'_{i-1}이 오퍼런드이고 e'_{i-2}가 '('일 경우 Lchild(i)는 $i-1$로 설정한다.
- e'_{i-1}이 오퍼런드이고 e'_{i-2}가 연산자일 경우 Lchild(i)는 $i-2$로 설정한다.
- e'_{i-1}이 ')'이고 $e'_{m'(i-1)-1}$이 '('일 경우 Lchild(i)는 R_{i-1}로 설정한다.
- e'_{i-1}이 ')'이고 $e'_{m'(i-1)-1}$이 연산자일 경우 Lchild(i)는 $m'(i-1)-1$로 설정한다.

위의 규칙을 직접 적용하기 위해 왼쪽 결합을 갖는 연산자 e'_i, $1 \le i \le |E'|$를 갖는 처리기 P(i)는 e'_{i-1}, e'_{i-2}, $e'_{m'(i-1)-1}$, $m'(i-1)$, R_{i-1}의 값들을 가지고 있어야 한다. 이러한 형태의 자료 수집은 보조정리 1에 의해서 쉽게 해결할 수 있다. 모든 처리기 P(i), $1 \le i \le |E'|-1$가 e'_i와 $m'(i)$를 처리기 P(i+1)에 전송하고, 다시 처리기 P(i)가 바로 전에 전송 받은 e'_{i-1}를 P(i+1)에 전송한다. 그러면 각 처리기는 P(i)는 e'_{i-1}, e'_{i-2}, $m'(i-1)$의 값들을 갖는다.

다음에 e'_i가 '('을 갖는 처리기 P(i)가 e'_{i-1}을 처리기 P(m′(i))에 전송하고, ')'을 갖는 처리기 P(i)가

표 1 매칭 되는 괄호의 쌍에 포함된 단순 식의 루트 결정 예

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
e'_i	(	(	a	/	b	)	-	c	+	(	d	↑	(	(	e	/	f	)	+	(	g	↑	h	)	)	)	)
R_i	9	4				4				12			19	16				16		22				22	19	12	9

P(m'(i))로 전송 받은 $e'_{m'(i)-1}$의 값과 R_i의 값을 처리기 P(i+1)에 전송한다. 그러면 연산자를 갖는 각 처리기 P(i)는 위의 규칙을 적용하는데 필요한 모든 자료인 e'_{i-1}, e'_{i-2}, $e'_{m'(i-1)-1}$, $m'(i-1)$, R_{i-1}의 값들을 갖는다. 위와 같은 자료의 라우팅은 보조정리 1에 의해서 상수 시간에 수행 가능하며, 왼쪽 결합 연산자를 갖는 각 처리기 P(i)는 바로 Lchild(i)를 계산할 수 있다.

(2) 왼쪽 결합을 갖는 연산자 e'_i의 오른쪽 자식 Rchild(i)는 만일 e'_{i+1}이 오퍼런드 이면 Rchild(i)는 $i+1$로 그렇지 않으면 R_{i+1}로 설정한다. 모든 처리기 P(i), $2 \le i \le |E'|$가 e'_i와 R_i를 처리기 P(i-1)에 전송하면 처리기 P(i)는 e'_{i+1}와 R_{i+1}의 값을 알고 있으므로 오른쪽 자식 Rchild(i)를 바로 결정할 수 있다. 보조정리 1에 의해서 이러한 자료의 라우팅은 상수 시간에 수행 가능하다.

(3) 마지막으로 오른쪽 결합을 갖는 연산자('↑') e'_i의 Lchild(i)와 Rchild(i)를 결정한다. e'_i의 Lchild(i)는 (2)와 유사한 방법으로 Rchild(i)는 (1)과 유사한 방법으로 결정할 수 있다.

표 2는 지금까지 설명한 과정을 완료한 후의 결과를 보여주며, 표 2에 의해서 생성된 계산 트리 형식을 도식한 예가 그림 2에 보여졌다.

이 장에서 설명한 LASOB 상에서 상수시간에 수행되는 알고리즘은 [9, 11]에서 제안된 알고리즘의 아이디어를 기본으로 한다. 그러나 기존 연구에서는 입력 크기에 선형으로 비례하는 처리기를 사용하여 최적으로 수행되는 결과를 제시하지 못하고 있다. 이들 연구 모두 선형 처리기 배열 상에서 계산 트리를 생성하기 위해 괄호 매칭 문제를 필수적으로 이용한다. 현재 PRAM 혹은 RMESH에서 입력의 크기와 같은 처리기를 사용하여 괄호 매칭 문제를 상수 시간에 해결할 수 있는 방법은 알려지지 않고 있다.

본 연구에서는 먼저 LASOB 모델의 독특한 성질인 시분할 멀티플렉싱 기술을 이용하여, 기존 연구에서 해결하지 못한 괄호 매칭 문제를 선형 처리기 배열 상에서 상수 시간에 해결하였다. 또한 Knuth의 괄호 삽입 알고리즘 적용 후에 발생하는 불필요한 괄호를 제거하고, 괄호 매칭 쌍을 바로 적용하게 하기 위해 선형 처리기 배열 상에서 자료를 재배열하는 문제를 상수 시간에 해결하였다. 따라서 이러한 문제의 해결이 선형 처리기 배열을 갖는 LASOB 상에서 계산 트리를 상수시간에 생성하기 위한 기본적인 도구가 되었다.

지금까지 설명한 계산 트리를 생성하기 위한 본 논문의 알고리즘은 $5n$개의 처리기를 갖는 LASOB 상에 수행되었다. [1]에서는 1차원 RASOB(LASOB) 상에서 n개의 입력에 대해 $p(<n)$ 개의 처리기를 가질 경우 $O(n/p)$ 단계에 정렬 문제를 해결할 수 있음을 논의하였다. 여기서 제안된 결과에 따르면 $5n$개의 처리기를 갖는 LASOB 상에서 수행되는 우리의 알고리즘은 n개의 처리기를 갖는 LASOB 상에서 상수 시간에 쉽게 시뮬

표 2 각 연산자의 왼쪽과 오른쪽 자식을 결정한 예

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
e'_i	(	(	a	/	b	)	-	c	+	(	d	↑	(	(	e	/	f	)	+	(	g	↑	h	)	)	)	)
Lchild(i)				3			4		7			11				15			16			21					
Rchild(i)				5			8		12			19				17			22			23					

레이트 될 수 있다. LASOB 상의 각 처리기에는 입력 크기에 관계없이 상수 개의 정보만이 저장되어 있으므로, 한 개의 처리기가 5개의 처리기에서 발생되는 정보를 유지할 수 있다. 또한 한 개의 처리기에서는 5개의 처리기에서 일어난 일들을 상수시간에 쉽게 시뮬레이트 할 수 있다. 그러므로 지금까지 설명한 결과에 의해 다음의 정리가 증명된다.

정리 2. 길이 n의 대수 식이 주어지면 이 식에 대한 계산 트리 형식은 n개의 처리기를 갖는 LASOB 상에서 상수 시간에 구성될 수 있다.

5. 결 론

본 논문에서는 LASOB를 계산 모델로 사용하여 먼저 괄호 매칭 문제에 대한 상수 시간 알고리즘을 설계하였다. 다음에 이 알고리즘을 사용하여 길이 n인 대수 식이 주어질 경우 이 대수 식을 n개의 처리기를 갖는 LASOB 상에서 상수 시간에 계산 트리 형식으로 변환하는 비용 최적인 병렬 알고리즘을 제안하였다. 이 문제에 대한 병렬 알고리즘은 이미 다수의 문헌에서 제안되었지만 아직 상수 시간에 수행되는 비용 최적인 알고리즘은 제안되지 않고 있다. 본 논문에서 고려한 LASOB의 구조는 전자 버스 대신에 광 버스를 사용하여 버스의 대역폭을 늘리고 하드웨어의 복잡도를 줄인 병렬 컴퓨터 구조로서, 현재 다른 병렬 컴퓨터에 비해서 더 현실성 있는 모델로 고려되고 있다.

참 고 문 헌

[1] M. Hamdi, C. Qiao, Y. Pan, and J. Tong, "Communication-efficient sorting algorithms on reconfigurable array of processors with slotted optical buses," *J. Parallel Distribut. Comput. 57*, pp. 166-187, 1999.

[2] S. Pavel and S. G. Akl, "On the power of arrays with reconfigurable optical buses," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications*, vol. III, 1443-1454, 1996.

[3] S. Pavel and S. G. Akl, "Integer sorting and routing in arrays with reconfigurable optical buses," *Proc. Int'l Conf. on Parallel Processing*, vol. II, pp. 90-94, 1996.

[4] Z. Guo, "Sorting on array processors with pipelined buses," *Proc. Int'l Conf. on Parallel Processing*, vol. III, pp. 289-292, 1992.

[5] S. Rajsekaran and S. Sahni, "Sorting, selection and routing on the array with reconfigurable optical buses," Technical Report, Department of Computer and Information Science, University of Florida, 1996.

[6] R. Miller, V. K. Prasanna-Kumar, D. Reisis, and Q. F. Stout, "Parallel computations on reconfigurable meshes," *IEEE Trans. Comput.*, vol. 42, no. 6, pp. 678-692, Jun. 1994.

[7] H. M. Alnuweiri, "Parallel constant-time connectivity algorithms on a reconfigurable network of processors," *IEEE Trans. Parallel Distribut. Systems*, vol. 6, no. 1, pp. 105-110, Jan. 1995.

[8] E. Dekel and S. Sahni, "Parallel generation of postfix and tree forms," *ACM Trans. Programming Languages and Systems 5*, pp. 300-317, 1983.

[9] I. Bar-On and U. Vishkin, "Optimal parallel generation of a computation tree form," *ACM Trans. Programming Languages and Systems 7*, pp. 384-357, 1985.

[10] W. W. Tsang, T. W. Lam, and F. Y. L. Chin, "An optimal EREW parallel algorithm for parenthesis matching," *Proc. Int'l Conf. on Parallel Processing*, vol. III, pp. 185-192, 1989.

[11] G. H. Chen, S. Olariu, J. L. Schwing, B. F. Wang, and J. Zhang, "Constant-time tree algorithms on reconfigurable meshes on size n×n," *J. Parallel Distribut. Comput. 26*, pp. 137-150, 1995.

[12] S. G. Akl, The design and analysis of parallel algorithms, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[13] L. Bergogne and C. Cerin, "A new parallel algorithm for the parentheses matching problem," *Proc. AIZU Int'l Symposium on Parallel Algorithms/ Architecture Synthesis*, pp. 364-369, 1997.

[14] K. Diks and W. Rytter, "On optimal parallel computations for sequence of brackets, " Theoretical Computer Science, vol 87, pp. 251-262, Sep. 1991.

[15] I. Stojmenovic, "Constant time BSR solutions to parenthesis matching, tree decoding and tree reconstruction from its traversals," *IEEE Trans. Parallel Distribut. Systems*, vol. 7, no. 2, pp. 216-224 , Feb. 1996.

[16] D. E. Knuth, The art of computer programming: Fundamental algorithms, Addison-Wesley, Reading, MA, 1973.

김 영 학

1984년 금오공과대학교 전자공학과(전자계산기공학전공) 졸업. 1989년 서강대학교 대학원 전자계산학과(공학석사). 1997년 서강대학교 대학원 전자계산학과(공학박사). 1989년 ~ 1997년 해군사관학교 재직(해군장교). 1998년 ~ 1999년 국립여수대학교 멀티미디어학부 교수. 1999년 ~ 현재 금오공과대학교 컴퓨터공학부 교수. 관심분야는 병렬알고리즘, 분산 및 병렬처리 등