14주차 2차시 병렬알고리즘 II

【학습목표】

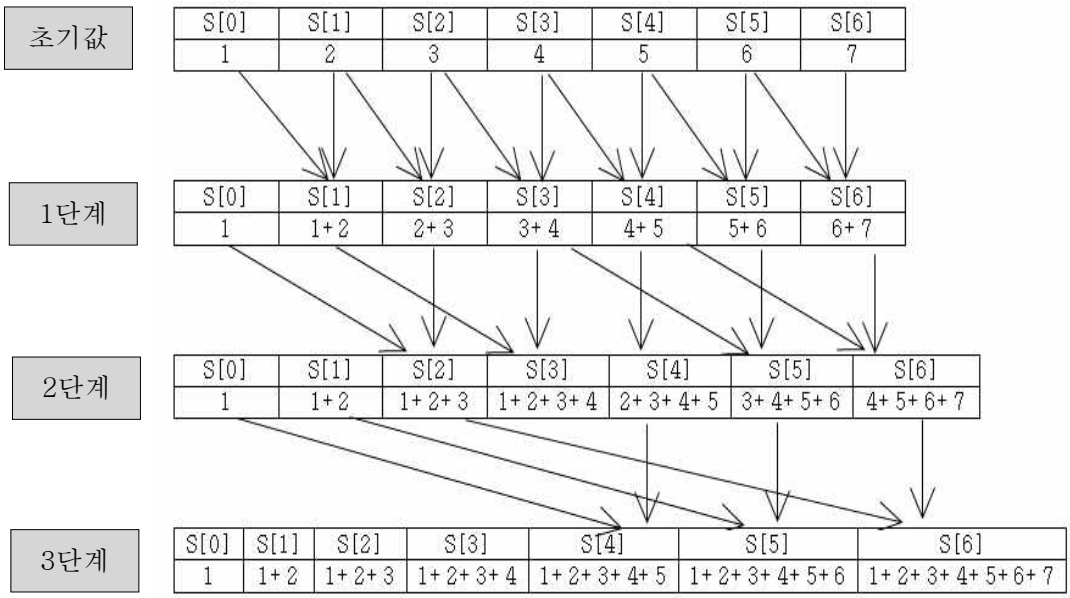
1. 병렬알고리즘의 효율성을 이해할 수 있다.
2. 병렬 합병 정렬을 이해할 수 있다.

학습내용1 : 접두부 부분 합 계산

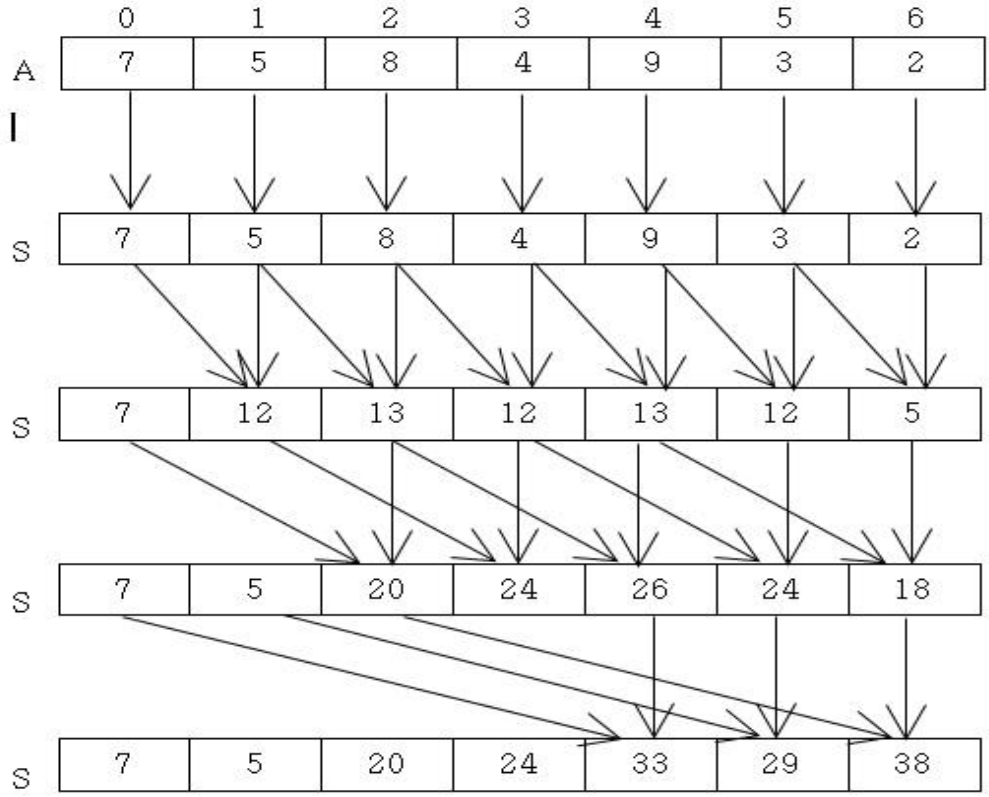
n개의 숫자가 배열 A[0]에서 A[n-1]에 기억되어 있을 때 A[0], A[0]+A[1], A[0]+A[1]+A[2],••••••••••••+A[0]+A[1]+A[2]••••••••••••+A[n-1]계산 하는 것을 접두부 부분합(Prefix Sum) 계산하기라 한다. 접두부 부분합은 동일한 메모리에서 동시에 자료를 읽지 않으므로 EREW PRAM 알고리즘으로도 생각할 수 있다.

1. 접두부 부분합 구하는 CREW PRAM 알고리즘

숫자가 배열 A[0]에서 A[n-1]에 기억되어 있을 때 접두부 부분합을 구해서 m번째 부분합인 A[0]+A[1]+A[2]••••••••••••+A[m-1]을 S[m-1]에 저장 하려고 한다. 알고리즘은 logn 단계로 구성 된다.



- 각 단계별로 S[i]에 계산되는 값

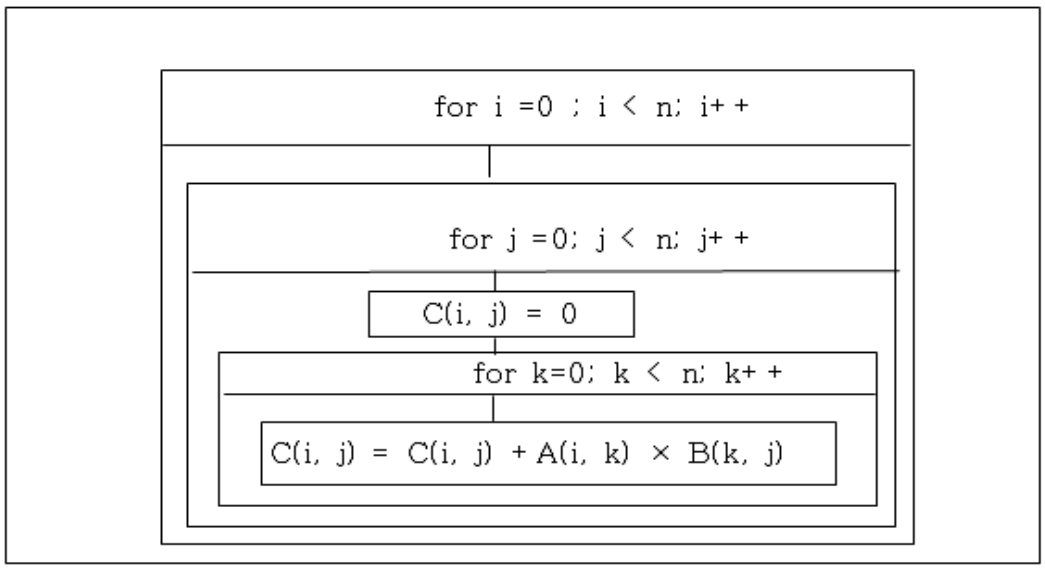


- 접두부 부분합을 구하는 과정

[위의 그림]을 자세히 살펴보면 다음과 같다.
(1) 1단계 PE(M)는 A[M]+A[m-1]의 값을 계산하여 S[m]에 저장한다.
(2) 2단계 PE(M)는 S[m]+S[m-2], A[m]+A[m-1]+A[m-2]+A[m-3]의 4개의 순자의 합을 계산하여 S[m]에 저장한다.
(3) k단계 PE(M)는 S[m]+S{m-2^{k-1}}즉 2^k개 즉 A[m]+••••••+A[m-2^k+1]의 값을 계산하여 S[m]에 저장한다.

학습내용2 : 행렬의 곱셈

n×n 행렬 A, B, C, 가 주어졌을 A행렬과 B행렬을 곱하여 C행렬에 기억시키고자 한다. 행렬의 곱셈식의 알고리즘은 다음과 같다.



[행렬의 곱셈 알고리즘]

* 행렬의 곱셈 알고리즘을 보면(n=3이라 가정)

1	2	3
4	5	6
7	8	9

A행렬

×

9	8	7
6	5	4
3	2	1

B행렬

=

30	24	18
84	69	54
138	114	90

C행렬

C(0,0)=C(0,0)+A(0,0)*B(0,0) 계산값 9
C(0,0)=C(0,0)+A(0,1)*B(1,0) 계산값 12
C(0,0)=C(0,0)+A(0.2)*B(2,0) 계산값 9 C(0,0) = 9 +12 + 9

C(0,1)=C(0,1)+A(0,0)*B(0,1) 계산값 8
C(0,1)=C(0,1)+A(0,1)*B(1,1) 계산값 10
C(0,1)=C(0,1)+A(0,2)*B(2,1) 계산값 6 C(0,1) = 8 +10 + 6
C(0,2)=C(0,2)+A(0,0)*B(0,2) 계산값 7
C(0,2)=C(0,2)+A(0,1)*B(1,2) 계산값 8
C(0,2)=C(0,2)+A(0,2)*B(2,2) 계산값 3 C(0,2) = 7 + 8 + 3
와 같이 반복 계산을 한다.

```
for (i = 0; i < n; i++)
 for (j = 0; j < n; j++)
  C(i, j) = 0;
   for (k = 0; k < n; k++)
    C(i, j) = C(i, j) + A(i, k) * B(k, j);
```

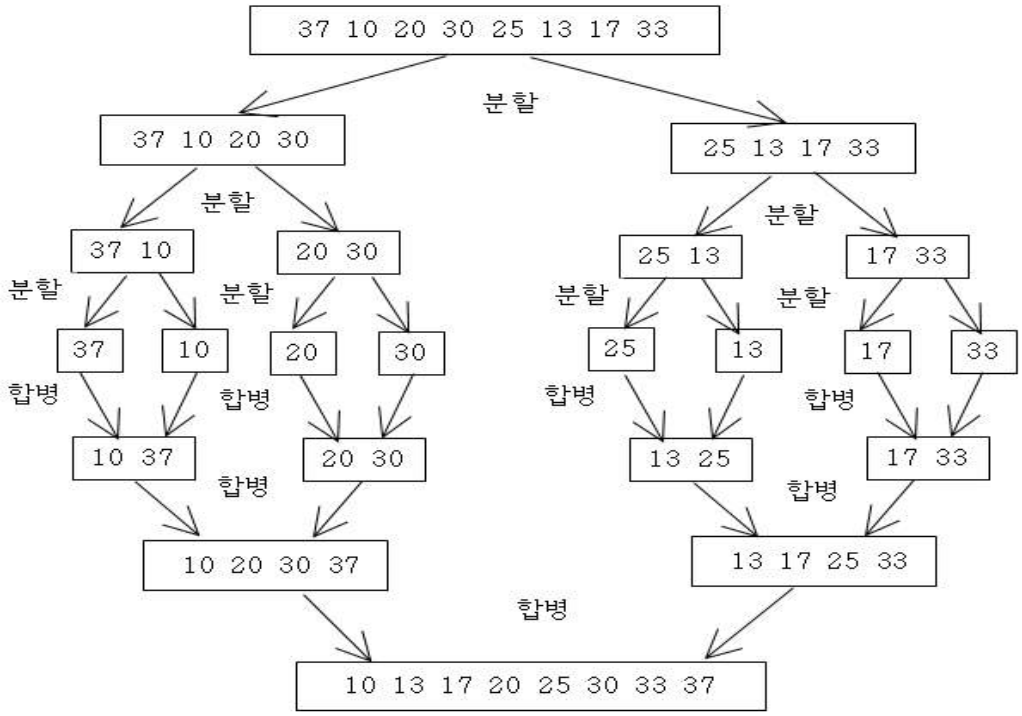
[n*n 행렬의 곱셈]

위의 알고리즘을 수행 하면 n회의 곱셈과 n회의 덧셈을 실행하므로 실행시간은 O(n)이다. 행렬의 곱셈을 순차적으로 수행하면 $O(n^3)$이 됨으로 이 알고리즘은 계산상의 낭비가 거의 없다 할 수 있다. 그러나 병렬 알고리즘의 효율성을 이야기할 때 그 병렬 알고리즘의 실행 시간 과 사용된 프로세서 수와의 곱한 계산값을 가장 빠른 순차 알고리즘의 시간 복잡도와 비교해 볼 때 최적이라 할 수는 없다. 왜냐하면 순차 알고리즘의 실행 시간은 $O(n^3)$보다 작기 때문이다. 그러므로 프로세의수를 n^3개를 사용하면 O(log n)시간이 걸리는 병렬 알고리즘을 구할 수 있게 된다. 행렬의 곱셈에서 하나의 프로세서로 계산을 했기 때문에 O(n)의 시간이 소요되었지만 n개의 프로세서로 나누어서 처리하게 되면 실행시간을 O(log n) 으로 줄일 수 있게 된다.

학습내용3 : 병렬 합병 정렬

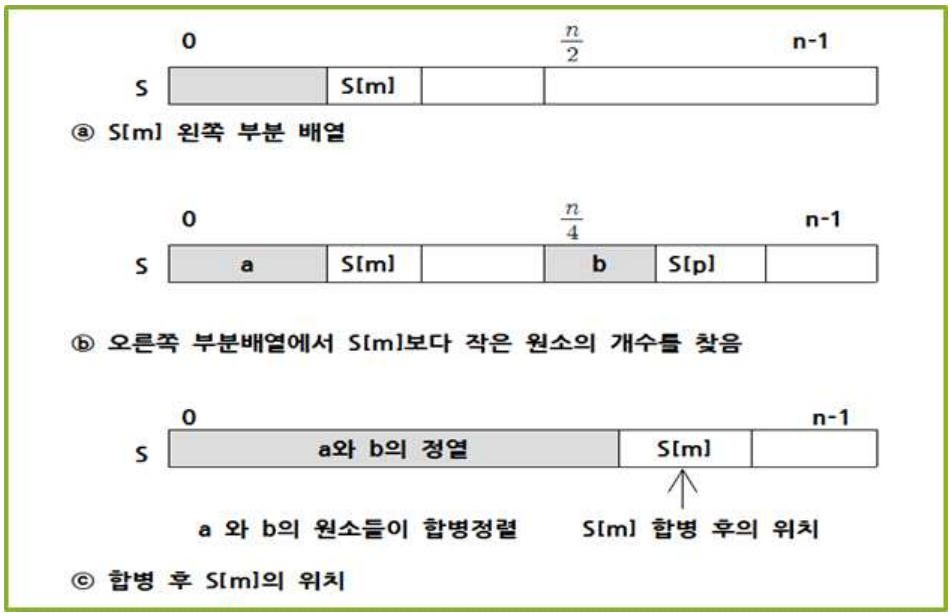
병렬 합병 정렬은 앞에서 배운 순차 합병 정렬을 이용하여 순차 합병 정렬 알고리즘을 병렬화 하여 여러 개의 합병을 동시에 진행시켜 빠르게 병합할 수 있는 알고리즘을 의미 한다.
순차 합병 정렬 알고리즘은 다음과 같이 진행 된다.

1. 분할 : 정렬 하려는 배열을 n/2개의 부분배열로 분할한다.
2. 정복 : 부분배열을 정렬 한다. (반복되면 재귀호출을 한다.)
3. 통합 : 정렬된 두 개의 배열을 합병 하여 하나의 정렬된 배열로 만든다.



- 순차 합병 정렬 알고리즘

이 알고리즘을 병렬화 하여 n/2개의 원소를 갖는 배열의 합병을 O(log n)시간에 수행하고 여러 개의 합병을 동시에 진행함으로써 $O(\log^2 n)$의 시간 복잡도를 갖는 병렬 알고리즘이 된다. 병렬 합병 알고리즘을 구상 해보면 n/2개의 원소를 갖는 두 부분 배열을 합병하는데 n개의 프로세서가 사용된다고 가정한다. 각 PE(i)는 S[i]의 합병된 후의 배열 내에서의 위치를 찾는데 사용된다.



- 병렬 합병 정렬 알고리즘의 골격

병렬 합병 정렬 알고리즘의 골격은 순차 합병 알고리즘과 유사하다. [그림 8-13]에서 첫 번째 단계는 각 부분 배열의 크기를 1로 가정하고 인접한 두 부분씩 합병 한다면 이 들은 모두 한꺼번에 병렬로 처리 할 수 있게 된다. 두 번째 단계에서 두 부분 배열의 크기가 2가되며 여기서도 n/4개의 독립적인 합병이 병렬로 처리 가능하게 된다. 이러한 과정을 계속 진행할 때 log n번째의 단계가 되고 이 알고리즘이 진행됨으로 정렬 알고리즘의 전체 실행시간은 $O(\log^2 n)$이 된다.

【학습정리】

1. 접두부 부분합 계산
- n개의 숫자가 배열 A[0]에서 A[n-1]에 기억되어 있을 때 A[0], A[0]+A[1], A[0]+A[1]+A[2],••••••••••••+A[0]+A[1]+A[2]••••••••••••+A[n-1]계산 하는 것을 접두부 부분합(Prefix Sum) 계산하기라 한다. 접두부 부분합은 동일한 메모리에서 동시에 자료를 읽지 않으므로 EREW PRAM 알고리즘으로도 생각할 수 있다.

- 접두부 부분합 구하는 CREW PRAM 알고리즘
: 숫자가 배열 A[0]에서 A[n-1]에 기억되어 있을 때 접두부 부분합을 구해서 m번째 부분합인 A[0]+A[1]+A[2]••••••••••••+A[m-1]을 S[m-1]에 저장 하려고 한다. 알고리즘은 lgn 단계로 구성 된다.

2. 행렬의 곱셈

행렬의 곱셈 알고리즘을 수행 하면 n회의 곱셈과 n회의 덧셈을 실행하므로 실행시간은 O(n)이다. 행렬의 곱셈을 순차적으로 수행하면 $O(n^3)$이 됨으로 이 알고리즘은 계산상의 낭비가 거의 없다 할 수 있다. 그러나 병렬 알고리즘의 효율성을 이야기할 때 그 병렬 알고리즘의 실행 시간 과 사용된 프로세서 수와의 곱한 계산값을 가장 빠른 순차 알고리즘의 시간 복잡도와 비교해 볼 때 최적이라 할 수는 없다. 왜냐하면 순차 알고리즘의 실행 시간은 $O(n^3)$보다 작기 때문이다. 그러므로 프로세의수를 n^3개를 사용하면 O(log n)시간이 걸리는 병렬 알고리즘을 구할 수 있게 된다. 행렬의 곱셈에서 하나의 프로세서로 계산을 했기 때문에 O(n)의 시간이 소요되었지만 n개의 프로세서로 나누어서 처리하게 되면 실행시간을 O(log n) 으로 줄일 수 있다.

3. 병렬 합병 정렬

병렬 합병 정렬 알고리즘의 골격은 순차 합병 알고리즘과 유사하다. 병렬 합병 정렬 알고리즘에서 첫 번째 단계는 각 부분 배열의 크기를 1로 가정하고 인접한 두 부분씩 합병 한다면 이 들은 모두 한꺼번에 병렬로 처리 할 수 있게 된다. 두 번째 단계에서 두 부분 배열의 크기가 2가되며 여기서도 n/4개의 독립적인 합병이 병렬로 처리 가능하게 된다. 이러한 과정을 계속 진행할 때 log n번째의 단계가 되고 이 알고리즘이 진행됨으로 정렬 알고리즘의 전체 실행시간은 $O(\log^2 n)$이 된다.