Heap 병합 병렬 알고리즘

On the parallel merging algorithm

민 용 식*
(Yong Sik Min*)

요 약

본 논문은 힙을 병합시키기 위해 SIMD-SM-R(CREW-PRAM)상에서 구현되는 병렬 알고리즘을 제시하고자 한다.

병렬로 두개 힙을 병합시키기 위한 알고리즘은 두 부분으로 구성된다. 첫째 단계는 LEVEL-FIND로서 노드 p의 위치를 선정키위해 Wyllie(19)의 방법에 의하면 그 시간 복잡도가 O(log n)인데 비해 본 논문에서는 O(log(n/k))로 제시하였고, 그리고 둘째 방법으로서 두개의 subheap을 병합해서 새로운 힙을 생성시키는 방법으로서 Dekel과 Sahni(4)와 홍 영식(18)의 제시된 방법의 시간 복잡도로서 O(log n)인데 반해서 본 논문에서는 (m+1)/4 개의 프로세서를 이용해서 O(log m)으로 처리되어 병합된 결과가 max(2**(i-1), (m+1)/4)개의 프로세서를 이용한 경우의 시간 복잡도가 O(log(n/k)*log(n))를 지니는 알고리즘을 제시하여, 병렬 병합 알고리즘의 효율성인 EPU(Effective Processor Utilization)가 거의 1로서 최적인 가속화 율을 지닌 알고리즘을 제안하였다.

ABSTRACT

The purpose of this paper is to suggest and analyze the parallel algorithm for merging two heaps, on SIMD-SM-R (CREW-PRAM) parallel computer.

In order to create the parallel algorithm for merging two heaps, we have classified two subproblems. For the first method, to select node p as a LEVEL-FIND function, Wyllie(19) suggests the method with time complexity O(log n) while this thesis has O(log(n/k)). For the second method, to merge two subheap(that is, pheap and sheap), our algorithm has O(log(n/k)*log(n)) using max(2**(i-1), ⌈(m+1)/4⌉)'s processors while Dekel and Sahni(4)'s method and Hong's method(18) have O(log m). Also our parallel algorithm's EPU is close to 1 and so has an optimal speed-up ratio.

I. 서 론

Heap(8, 12)이란 다음의 특징을 지닌 트리 구조를 뜻한다. a)어떤 노드에 포함이 된 키는 이것의 children노드의 키보다 크지 않고, b)모든 leaf 노드들은 기껏해야 두개의 인접된 레벨을 가지고 그리고 마지막 레벨에 있는 모든 leaf 노드들은 가능한 왼쪽에서부터 존재케된다. Heap의 특징(8)은 Floyed 알고리즘에 의하면 선형시간내에 수행이되고, 그리고 Brown(8)에 의해 제시된 최악의 수행능력과 평균 처리시간은 heap들이 삽입과 삭제가 이루어질때 우선순위 큐를 이용하면 가장 효율적이다.

이같은 특징을 지닌 구조에서, n개의 구성요소를

*호서대학교 전자계산학과
접수일자 : 1992년 4월 30일

지닌 nheap과 k개의 구성요소를 지닌 kheap을 순차적인 알고리즘을 이용해서 병합시키는 방법을 살펴보면 다음과 같다.

첫째, 병합 시키고자 하는 두개의 heap 구조를 무시하여서 그것을 각각 독립된 노드로 간주하여 새로운 힙을 구성시키는 방법으로 복잡도가 O(n+k)이다.(8, 14)

둘째 방법은 Gonnet와 Munro(6, 8, 14)의 삽입 알고리즘에 의해 heap을 병합시키는 방법으로 복잡도가 O(k*log(log(n+k)))이다.

세째로 Sack과 Strothotte(8, 14)에 의한 방법으로 복잡도가 O(log(n)*log(k))이다. 그리고 태그 노드를 이용한 방법(14)으로 복잡도가 O(log(log(n/k)*log(k))이다.

또한 두개의 힙(n개의 구성요소를 지닌 nheap과 k개의 구성요소를 지닌 kheap)을 병렬로 병합시키기 위한 알고리즘을 구성시키기 위한 기존의 개념들과 방법에 대해서 살펴보면 다음과 같다.

첫째로, Wyllie(19)의 연결리스트에서 구성요소의 수를 계산하는방법과 unbounded모델(19)에서 O(n)개 프로세서를 이용하여서 O(log n)시간 내에 rooted 트리에서 각 정점의 레벨을 계산하는 방법이 있다.

둘째로는, Dekel과 Sahni(4)가 제시한 방법으로 SIMD 모델 상에서 이진 트리 방법에 의해서 주어진 집합에서 최소 원소를 찾는데 n/2개의 프로세서로서 O(log n)시간이 필요케 된다.

셋째로는, MIMD모델을 이용한 (18)방법에 의하면, 최소원소를 삭제하는 병렬 알고리즘은 「(n+1)/4」개의 프로세서로서 O(log n)시간 내에 처리가 가능케 하였다.

마지막으로, Yoo, Y.B(15) 방법에 의하여 힙을 생성시키는데 있어서 m/4개의 프로세서를 이용하여 O(log m)시간내에 처리를 가능케하였다.

본 논문에서는 divide-and-conquer의 한 예인 병합에 대해서 현재 존재하는 순차적인 알고리즘에서 병렬로 구현하는 방법으로서, 근본적인 데이타 형태의 하나인 priority queue(15)로서 표현하는 것중 기억구조상에 가장 좋은 방법중의 하나임과 동시에 어떤 작업을 수행하는데 있어서 효율적인 구현으로 표시되는 힙(Heap)(15)을 SIMD-SM 모델 병렬 컴퓨터에 직접 적용할 수 있는 unbounded 병렬 알고리즘을 제시함과 동시에 그것의 분석에 그 주된 목적이 있다. 그리고 병렬 알고리즘의 평가는 세가지 방법(15, 16)중 가속화 율로서 T(1)(n)/T(k)(n)에서 T(k)(n)을 구하고서 한다.

본 논문에서 제시된 구성은 다음과 같다. 제 2장에서는 unbounded모델로서 SIMD나 MIMD에서 이용할 수 있는 병렬식 구조로서 힙을 병합시키는 알고리즘을 제안하였고, 그리고 3장에서는 이같은 병렬 힙 병합 알고리즘의 시간 복잡도, 공간 복잡도, 이용된 프로세서의 수 그리고 제시된 타 방법과 비교를 하였다. 끝으로, 4장에서는 본 논문의 결론을 서술하였으며, 순차적인 힙 방법과 병렬 힙 병합 방법을 비교 설명하였다.

Ⅱ. 병렬 병합 알고리즘

2.1 크기가 같은 경우의 병합

nheap의 크기 n와 kheap의 크기 k가 같은 경우의 병합에 있어서는 순차적인 방법에서 사용되는 병합 방법과 동일하게 처리가 된다. 즉 1개의 프로세서를 이용하여서 O(log(k))시간내에 처리가 된다.((15)의 알고리즘 merge equal-perfect-heaps참조) 이 때 nheap과 kheap은 완전 힙인 경우이다.

2.2 크기가 다른 경우의 병합

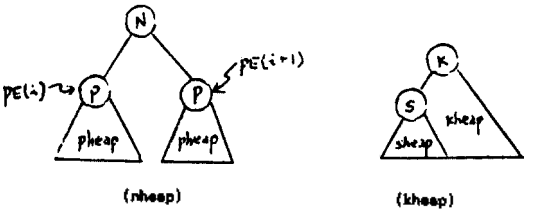
크기가 n인 nheap과 크기가 k인 kheap을 병렬로서 병합시키는 경우를 살펴보면 다음과 같다. 여기서 kheap을 nheap에 병합하게 된다. 이러한 kheap을 nheap에 병합시키는 과정을 도식화 한것이 그림 1과 같이 나타낼 수가 있으며 그 방법은 다음의 세 단계를 거쳐서 수행이 된다.

첫째로, nheap에서 병합되어질 k개의 slot(8)들의 노드에 해당되는 레벨을 정해서 그 레벨의 해당 노드를 각각의 프로세서에 할당시킨다. 둘째로는, nheap에서 각 프로세서에 할당된 nheap의 subheap인 kheap과 kheap에서 각 프로세서에 할당된 kheap의 subheap인 sheap을 병합시킨다. 이때 병합된 새로운 subheap(pheap+sheap)은 힙의 구조를 만족하게 된다. 마지막 단계로서, 두번째 단계에서 병합된 subheap의 결과를 지닌 각각의 프로세서들을 공용 기억장소에 저장된 nheap에 연결시킴과 더불어서 이같은 nheap이 힙의 조건을 만족하도록 구성시키면 된다.

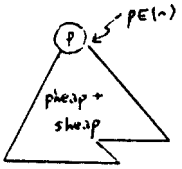
2.3 LEVEL-FIND 알고리즘

병합시키는 과정의 첫번째 단계로서 nheap에서 병합되어질 k개의 slot들의 각노드에 해당되는 레벨을

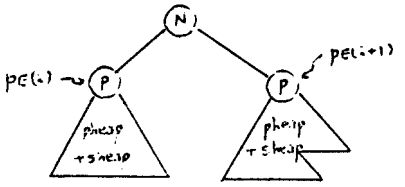
정해서 그 레벨의 해당노드인 p노드를 각각의 프로세서에 할당할 위치를 결정해야한다. 이때, 본 논문에서는 nheap의 레벨(level)을 이용해서 처리를 하였다. 그리고 nheap은 완전 힙인 경우와 불완전 힙인 경우가 존재케 되는데 이 경우를 살펴보면 다음과 같다. 완전 힙상에서의 kheap의 크기 k는 k<n으로 가정하여 처리한다.



(a)각 프로세서의 p 결정



(b)pheap과 sheap 병합



(c)병합된 결과

그림 1. 병렬 힙 병합과정

(1)nheap이 완전 힙인 경우

노드의 위치 p를 선정하는 과정에서 nheap이 완전 힙이므로 해서 항상 kheap의 노드들이 가장 왼쪽에 있는 노드에서 부터 채워지는 특성이 존재케 된다. 이같은 특성을 지닌 완전 힙에서 노드의 위치 p를 찾기 위해서는 다음 두가지 단계를 거친다.

첫째로는 각 프로세서에 노드의 위치 p를 할당시키기 위해서는 먼저 할당될 프로세서의 수를 결정해야만 할 것이다. 즉, 프로세서는 kheap의 leaf의 노드 수와 같은 것이다. 그리고 둘째로는, 결정된 프로세서 수 만큼에 해당되는 위치를 선정하면 된다. 이때 nheap이 완전 힙이므로 해서 첫번째 프로세서는 항상 nheap의 가장 왼쪽에 존재하는 leaf 또는 sub-heap을 가리키게 된다. 이와 더불어서 각각의 프로세서에 채워져야 할 slot들의 수들 역시 보관하게 된다. 이같은 slot들의 수를 보관시키기 위해서는 공용 기억 장소에 광의 변수 S(i)(i=1, 2, .., $2^{**}(l-1)$, l=kheap에서의 레벨 수 그리고 i는 프로세서 i번째 것 즉, PE(i)를 뜻한다.)를 이용한다. 이상과 같이 완전 힙의 경우 노드의 위치 p를 찾기위한 알고리즘은 다음 알고리즘 1와 같다.

```
알고리즘 1 parallel-perfect-level-find
//p : 프로세서의 수
  PE(i) : i번째 프로세서(1<=i<=p)
  S(PE(i)) : i번째 프로세서가 가지고 있는 slot의 수//
1. //프로세서 수를 결정//
  p=2**(h(kheap)-1)
2. //nheap에서 PE(i)의 해당 위치 설정//
  for all i 1<=i<=p do in parallel
    j=2**(h(nheap)-1)+i-1
    PE(i)=nheap의 j번째 위치
  allfor
3. //PE(i)가 지니는 slot 수 결정//
  for all i 1<=i<=p do in parallel
    S(PE(i))=PE(i)의 slot 수
  allfor
end
```

(정리 1)

알고리즘 parallel perfect-level-find는 O(1)시간 내에 수행이 되어진다.

(증 명)

단계 1, 2, 3 각각의 수행빈도는 명백히 O(1)이므로 알고리즘은 O(1)이다.

(2)nheap이 불완전 힙인 경우

nheap이 불완전 힙인 경우, 노드의 위치 p를 선정하기 위해서는 다음과 같은 세단계를 거쳐서 결정이 된다.

첫번째 단계로서, 각각의 프로세서에 노드의 위치 p를 할당시키기 위해서 할당될 프로세서의 수를 결정해야만 한다. 두번째 단계로서 결정된 프로세서가 자칭해야 할 곳의 위치를 결정해야만 할것이다. 이때 nheap의 높이(h(nheap))와 kheap의 높이(h(kheap))의 차를 이용해서 결정된 레벨상에 있는 각 노드에 대한 subtree가 처음 완전 이진 트리가 아닌 subtree의 각 노드를 그 위치로 한다. 그리고 이때 결정된

subtree의 크기와 slot간의 차가 1 이하가 아닌 경우에는 다시 그 하위의 subtree를 찾아서 크기와 slot간의 차가 1 이하가 되는 불완전 힙을 선택하여 이 위치를 프로세서의 첫번째에 할당하면 된다.

그리고 두번째 단계에서 결정된 위치 다음번을 둘째 프로세서에 그리고 그 다음을 세째 프로세서에 할당시키게 된다. 이때 프로세서에 할당되는 수는 첫번째 단계에서 결정된 p개의 수만큼 이용한다. 그리고 할당된 각 프로세서의 slot들의 수는 완전 힙의 경우와 같이 S(i)로 세트시킨다. 이상과 같이 nheap이 불완전 힙상에서의 p위치의 노드를 선정하기 위한 알고리즘은 다음 알고리즘 2와 같다.

알고리즘 2 parallel-nonperfect-level-find
//h(nheap) : nheap의 height
p:프로세서의 수, PE(i):i번째 프로세서(1<=i<=S(PE(i)) : i번째 프로세서가 지니는 slot수
pl : nheap에 병합될 곳의 처음위치//
1. //프로세서의 수를 결정//
p=2**(h(kheap)-1)
2. //첫번째 프로세서에 할당될 위치를 결정//
2.a level = h(nheap)-h(kheap)-1
이때 level이 0 이하인 경우는 level을 0으로 세트한다.
현재 레벨의 위치를 계산한다.(pl=2**level)
2.b 만일 nheap의 노드가 leaf가 아닌 경우
(1)nheap상의 pl의 subheap이 완전 힙인경우, 다음번의 위치 (pl=pl+1)를 선정한다. go to step 2.b
(2)LD=SIZE(pl's subheap)-SIZE(pl's slot)
(3)만일 LD<=1인 경우는, go to step 2.c
(4)2*pl로 구성된 subheap이 완전 힙인 경우는 pl=2*pl+1로 그렇지 않은 경우는 pl=2*pl로 한다.
(5)go to step 2.b
2.c PE(1)=nheap의 pl 위치
S(PE(1))=PE(1)의 slot 수
3. //두번째 프로세서에서 p번째 프로세서의 위치 할당//
for all i 2<=i<=p do in parallel
PE(i)=nheap의 pl+1-1번째 위치
S(PE(i))=PE(i)의 slot 수
allfor
end

(정리 2)
p 위치의 노드를 찾기 위한 알고리즘 2의 프로세서의 수는 O(2**(l-1))이다. (여기서 l≥1)
(증 명)
프로세서의 수를 결정하기 위해서는 알고리즘 2의 단계 1에서와 같이 p 2**(h(kheap)-1)의 명령에 의해서 수행이 된다. 이때 kheap의 높이는 레벨에 대응된다. 그러므로 2**(h(kheap)-1)은 kheap에 존재하는 한 레벨 l의 최대 노드 수 인 2**(l-1)이다.

(정리 3)
알고리즘 parallel-nonperfect-level-find을 수행하기 위한 시간 복잡도는 O(log(n/k))이다.
(증 명)
알고리즘 2에서 단계 1의 수행빈도는 O(1)이고 단계 3의 수행빈도 역시 O(1)이다. 단계 2에서 2.a는 nheap의 높이에서 kheap의 높이를 빼는 경우로서 수행빈도는 O(1)이고 그리고 2.b에서는 nheap상에서 slot들의 각 노드의 위치 p를 결정한다. 그러므로 nheap의 각 노드에서 p위치에 이르는 경로가 결정되는 과정은 log(n)-log(k) = log(n/k)이다. 그리고 2.c의 수행 빈도는 O(1)이다. 그러므로 단계 2의 수행빈도는 O(log(n/k))이다. 즉, 알고리즘 2의 전체 수행빈도는 O(log(n/k))이다.

2.4 병합 알고리즘

병렬 병합알고리즘을 기술하는 경우에 있어서 앞 절에서는 첫 번째 단계인 각 레벨을 프로세서에 할당시키는 부분이었다. 본 절에서는 할당된 프로세서를 이용하여서 kheap의 subheap인 sheap과 nheap의 subheap인 pheap이 병합되어지는 알고리즘을 제시하고자 한다.

(1)할당 알고리즘

각 프로세서는 공용 기억장소 nheap에서 subheap의 근에 해당되는 곳을지칭하는 알고리즘이 앞 절의 알고리즘 1,2에 제시되어져 있다. 그러면 이러한 알고리즘을 이용하여서 국지메모리에 이동시키기 위해서는 다음의 명령에 의해서 수행이 되어진다.

for all PE(i) 1<=i<=p do in parallel
while nheap's last node do
nheap(PE(i)의 위치)로 구성된 subheap을 각

프로세서의 pheap에 옮긴다.
endwhile
allfor

이렇게 구성된 pheap과 병합될 sheap을 선택하기 위해서 역시 공용 기억장소에 있는 kheap을 각 프로세서의 국지메모리에 해당되는 slot수 만큼 옮겨야 한다. 이같이 수행하기 위해서 각 프로세서들은 자기 자신이전에 존재하는 프로세서들이 유지하고 있는 slot들의 합에 의해서 그 합이 나타내는 번째를 kheap의 위치로 결정한다. 그리고 결정된 kheap의 위치에서 부터 해당 프로세서가 지닌 slot 수만큼을 자신의 프로세서의 국지메모리에 이동시켜서 구성시킨 것이 실제로 병합된 sheap으로 생성이 된다.

예를들면, 그림 2에서 원으로 표시된 노드는 내부노드이고, 사각형으로 표시된 노드를 slot들이라고 가정하자. 그리고 각 프로세서들은 자기자신이 가지고 있는 slot들의 수는 S(i)에 저장이 되어져 있다. 그리고 해당 프로세서가 가르키는 kheap의 해당 위치를 결정하기위해서 X(i)(i는 프로세서의 i번째)를 이용하여서 kheap의 해당 위치를 선정케 된다. 이때 X(i)의 초기값으로 1이 세트되어져 있다.

만약 S(i)={2,2,1}인 경우라고 가정하면 그때 S(i)와 X(i)의 해당되는 값을 나타낸 것이 그림 2와 같다. 즉, 첫번째 프로세서 PE(1)의 slot 수인 S(1)=2이고 그리고 첫번째 프로세서가 kheap에서 지칭하는 위치 즉, X(1)=1이므로 첫번째 프로세서는 kheap의 첫번째 위치(X(1)=1)에서 slot수만큼 즉 2개(S(1)=2)를 지칭한다. 그리고, 두번째 프로세서 PE(2)의 slot 수는 S(2)=2이고 두번째 프로세서가 kheap에서 지칭하는 해당위치는 첫번째 프로세서의 slot 수인 S(2)=2와의 합 즉, X(2)=2(S(1))+1=3으로서 kheap의 3번째 위치에서 두번째 프로세서가 지닌 slot수 즉 2(S(2)=2)이므로 2개를 지칭한다.

역시 세번째 프로세서 PE(3)는 slot수가 S(3)=1이고 X(3)=2(S(1)+2(S(2))+1=5 즉 kheap의 5번째에서 세번째 프로세서가 지닌 slot 수(S(3)=1)만큼 즉, 1개가 지칭된다. 이상과 같이 수행되어지는 알고리즘은 알고리즘 3과 같다.

알고리즘 3 parallel-assign
X(1..n)
//X : 해당 프로세서가 가르키는 kheap의 해당위치를 결정하는 것으로 배열의 초기값은 전부 1이 세트되어져 있다.//
for all PE(i) 1<=i<=p do in parallel
1. //각 프로세서가 kheap에서의 해당 위치를 결정//
1.a for all i 1<=i<=p do in parallel
sum=0
1.b for j=1 to (i-1) do
sum=sum+S(j-1)
endfor
X(i)=X(i)+sum
allfor
2. //결정된 kheap의 위치에서 해당되는 값만큼 이동시킴//
2.a for all i 1<=i<=p do in parallel
2.b for j=X(i) to (X(i)+S(i)-1) do
kheap의 j번째 위치를 해당 PE(i)의 sheap에 옮김
endfor
allfor

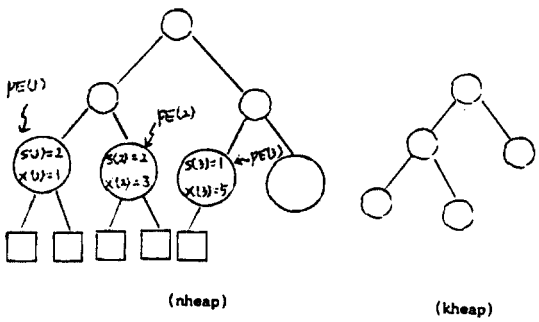


그림 2. 두개의 힙(nheap과 kheap)

(정리 4)
위의 알고리즘은 O(max(p, 한 프로세서가 지닌 slot의 수)로서 수행이 되어진다.

(증 명)
이 알고리즘의 복잡도는 단계 1의 1.a에서 O(1)이고, 1.b에서는 p개 프로세서를 이용해서 p-1개 만큼 수행이 되므로 O(p)이다. 그러므로 단계 1의 복잡도는 O(p)이다. 단계 2의 2.a에서 O(1)이고, 2.b에서는 kheap의 해당 위치에서 slot 수 만큼 이동이 생기므로 O(한개 프로세서가 지닌 slot의 수)이다. 그러므로 단계 2의 복잡도는 O(한개 프로세서가 지닌 slot의 수)이다. 이때 단계 1의 p개인 프로세서의 수와 단계 2의 한개 프로세서가 지닌 slot수 중에 큰값이 선택되므로, 알고리즘의 복잡도는 max

(p, 한 프로세서가 지닌 slot의 수)가 된다.

(2)병합 알고리즘 기법

각 프로세서가 자신의 국지 메모리에 두개의 힙(pheap과 sheap)을 하나로 만들기 위한 알고리즘은 다음과 같다.

```
for all i 1<=i<=p do in parallel
1. if SIZE(pheap) > SIZE(sheap)
     then newroot ={last element in pheap}
          exchange(pheap, sheap)
     else newroot ={last element in sheap}
2. distribute pheap to temporary location t
3. place newroot at pt
4. copy t to leftson of newheap(pt)
5. copy sheap to rightson of newheap(pt)
6. sift up(newheap)
allfor
```

이 같은 알고리즘은 두개의 힙 즉, pheap과 sheap을 병합 시키기 위해서는 두개의 힙 가운데서 크기가 큰 힙의 마지막 노드를 두 힙의 새로운 힙으로 구성한다. 그리고 새로이 형성된 힙은 힙 구조를 만족해야 하므로 이때 sift-up함수에 의해서 힙 구조를 생성시키면 된다. 이같이 새로이 병합된 힙을 공용 기억장소에 기억되어져 있는 nheap의 해당 위치로 옮기면, 전체가 병합된 형태의 결과를 유지하게 된다. 그러나 이 경우에 새로이 병합된 힙의 근 노드의 값이 변경 되는 경우에 있어서 공용 기억장소에 있는 nheap에 옮겨진 상태에서의 nheap은 힙 구조를 만족시키지 못하는 경우가 발생된다. 그러면 nheap은 힙으로서의 구성 요건이 상실되므로 해서 요구된 결과가 만들어 지지 않는다.

이러한 문제를 해결하기 위해서 본 논문에서는 국지 변수 cv를 이용하여 pheap과 sheap을 병합시켜 새로이 형성된 힙의 근 노드 값이 sheap의 값으로 변경되는 경우에 이 국지 변수 cv에 1을 세트하고 그렇지 않은 경우에는 0으로 세트시키게 된다. 그러면 각 프로세서가 가르키는 nheap의 해당 위치에서 nheap의 근 노드에 이르는 곳까지의 힙을 새로이 구성시키면 된다. 그리고 이때 각 프로세서는 다시 힙구조를 생성키 위해서 sift-up 함수를 수행한다. 이때 다시 근 노드의 값이 변화가 되는 경우에 국지변수 cv값이 다시 1로 세트되고 그렇지 않은 경우는 0으로 세트가 된다. 이렇게해서 모든 프로세서의 국지 변수 cv가 0으로 되는 경우에 최종의 nheap은 완전한 힙으로 생성된다. 이것에 대한 알고리즘은 다음 알고리즘 4와 같다. 이러한 형태의 그림은 다음 그림 3과 같다. 이 기서 점선 부분은 새로이 병합된 노드(kheap)을 뜻한다. 그리고 사각형의 값은 근의 값이 변경되는 경우를 뜻한다. 즉, 0인 경우는 불변경, 1인 경우는 값이 변경된 경우를 의미한다.

알고리즘 4 병합알고리즘 기법

```
1. n = ⌊log SIZE(병합후 nheap)⌋
2. for l = n down to 1 do
     k = 2**(l-1)
     s = 0
3. m  min(⌊SIZE(병합후 nheap)/2⌋, 2*k-1)
4. for all j k<=j<=m do in parallel
     p = 2*j
     if(p < SIZE(병합후 nheap)) and nheap(p)
     > nheap(p+1) then p = p+1
     if nheap(p) < nheap(j)
       then interchange(nheap(p), nheap(j))
     lock s
     s -> s+1
     unlocks
   allfor
5. while s < (m-k+1) do
     wait
   endwhile
  endfor
6. for all i 1<=i<=p do in parallel
     sift up(PE(i)'s subheap)
     if root node exchange, then cv = 1 else cv  0
```

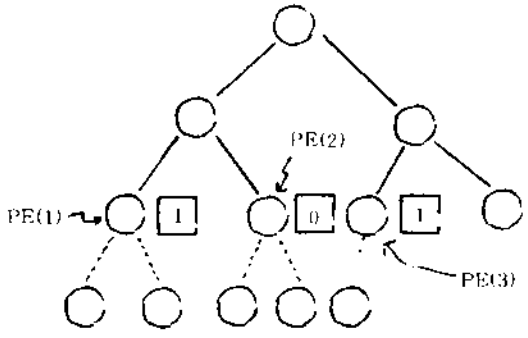


그림 3. nheap의 중간과정

```
  allfor
7 for all cv in PE(i) = 0 do in parallel
    return(nheap)
  allfor
end
```

병합 시키기위한 알고리즘 4를 수행하기 위해서 필요로 하는 프로세서의 수를계산 하면 다음 정리 5와 같다.

(정리 5)

알고리즘 4를 수행하는 프로세서의 수는 ⌈(m+1)/4⌉이다. 여기서 m은 새로이 병합된후의 nheap의 원소의 갯수 즉, n+k이다.

(증 명)

병합된 후의 nheap의 원소의 갯수는 m(=n+k)이고, 높이는 l(=h(nheap)+1)이다. 그리고 이 힙에 속하는 레벨 j에서의 최대 노드 수는 $2^{**}(j-1)=2^{**}l-1 \Leftarrow m$이다. 병렬처리를 위한 최대 프로세서의 수는 레벨이 l-1인 노드의 수로서 $2^{**}(l-2)$개의 프로세서가 필요하며, $2^{**}(l-2) <= ⌈(m+1)/4⌉$가 된다. 따라서, 알고리즘 3의 프로세서의 수는 ⌈(m+1)/4⌉이다.

(정리 6)

알고리즘 4의 시간 복잡도는 O((log(n/k))*log(n))이다.

(증 명)

알고리즘 4의 수행에 있어서, 단계 4의 수행 빈도는 O(1)이고, 단계 2-5은 O(log(n))이다. 그리고 단계 6은 sift-up 함수에 의하면 log(n)이 되므로 해당되는 subheap은 pheap과 sheap이므로 그 수행 빈도는 O(log(p+s))가 된다. 그리고 단계 1-6은 모든 프로세서의 subheap의 근 노드값의 변화가 없어야 하므로 즉, cv=0가 되어야 한다. 이것은 nheap의 근 노드에서 해당 프로세서가 지칭된 곳 즉 p노드에 이르기까지이다. 이것에 대한 수행빈도는 O(log(n/k))임을 정리 3의 단계 2에서 살펴보았다. 그러므로 단계 1-6의 수행빈도는 O(log(n/k))가 된다. 그러므로 본 알고리즘의 복잡도는 O(log(n/k))*log(n))로서 수행이 된다.

Ⅲ. 알고리즘 분석과 비교

(1)프로세서 수

병렬로 두개의 힙을 병합시키 위해서 사용되는 프로세서의 수를 먼저 살펴보면 다음과 같다. 먼저, 병합의 첫번째 단계인 LEVEL-FIND를 하기 위해 필요로 하는 프로세서의 수는 정리 2에서와 같이 $O(2^{**}(l-1))$ 즉, 레벨 l에서의 최대 노드수)가 되고, 할당시키는 경우에 있어서도 역시 마찬가지가 된다. 그리고 실제 병합이 수행되는 알고리즘에서 사용되는 프로세서의 수는 정리 4에서와 같이 ⌈(m+1)/4⌉로서 존재한다. 그러므로 본 논문에서 존재하는 프로세서의 갯수는 O(max($2^{**}(l-1)$, ⌈(m+1)/4⌉이다.

(2)시간 복잡도

본 논문에서 사용되어지는 시간 복잡도를 살펴보면 다음과 같다. 병합의 첫번째 단계인 각 프로세서에 할당할 p 노드의 위치를 결정하기 위한 알고리즘으로서 nheap이 완전 힙인 경우는 O(1)이고, 불완전 힙인 경우는 O(log(n/k))이다. 그리고 병합이전에 각 프로세서의 국지메모리에 병합될 두개의 subheap을 할당시키기 위한 시간 복잡도는 O(max(p, 한 프로세서가 지닌 slot의 수))이다. 그리고 병합시키는 복잡도는 O((log(n/k)*log(n))이다. 그러므로 본 논문에서 제시된 시간 복잡도는 O((log(n/k)*log(n))이다.

(3)공간 복잡도

병렬로 병합시키기 위해서 필요한 공간 영역을 살펴보면 다음과 같다. 첫째로 공용기억장소에서 사용되는 영역으로서 nheap의 크기 n와 kheap의 크기 k를 생각할 수가 있다. 즉, O(n+k)임을 알 수가 있다. 둘째로서 각 프로세서가 지닌 국지메모리의 영역은 pheap의 크기 p와 sheap의 크기 s로 이루어 졌다. 즉, 각 프로세서의 국지 메모리는 O(p+s)임을 알 수가 있다. 그러므로 병렬 알고리즘의 전체 공간 복잡도는 O(n+k+프로세서 수*(s+p))이다.

(4)타 알고리즘과 비교

본 논문에서 제시된 알고리즘을 생성키 위해서 기존에 제시된 기본개념들과 부분별로 비교해 보면 다음과 같다.

첫째, 본 논문에서는 p노드의 위치를 선정키 위해서 사용되는 LEVEL-FIND 알고리즘과 이 알고리즘을 생성키위해서 사용된 Wyllie(19)의 방법과 비교해 보면 다음 표 1과 같다. 다음 표 1에서도 볼 수 있

듯이 프로세서 수에서 상당한 축약이 있음과 더불어서 사용 시간 복잡도 역시 상당한 개선이 있었음을 알 수가 있다.

표 1. Wyllie(20)방법과의 비교

	모 델	프로세서 수	복잡도
Wyllie 방법(19)	unbounded	n	O(log n)
본 논문의 방법	unbounded	2**(i-1)	O(log(n/k))

표 2. Dekel & Sahni(4)과 홍 영식(18)방법의 비교

	모 델	프로세서 수	복잡도
Dekel & Sahni(4)	SIMD	n/2	O(log n)
홍 영식(18)의 방법	MIMD	⌈(n+1)/4⌉	O(log n)
본 논문의 방법	unbounded	⌈(m+1)/4⌉	O(log m)

그리고 두개의 subheap을 병합시켜서 공용 기억장소에 존재하는 nheap에 kheap이 병합된 형태를 지닌 힙으로서 생성시키기 위해서 사용된 Dekel과 Sahni(4)가 제시한 이진트리 구조를 이용하여 힙의 조건을 만족시키게끔 만들었다. 이 두 방법을 비교한 결과 똑같은 모델을 이용해서 시간 복잡도는 동일한 조건하에서 프로세서의 수에서 상당한 진전이 있음을 알 수가 있었다.

Ⅳ. 결 론

본 논문에서는 unbounded모델상에서 힙을 병렬로 구현시키는 알고리즘을 제시하고 제시된 알고리즘의 효율성을 살펴보면 다음과 같다.

병렬 알고리즘에서의 p개의 프로세서를 이용해서 얻은 수행빈도는 O(log(n/k)*log(n))으로서 1개의 프로세서는 1/p개로 하여 얻을 수 있다. 즉, O((1/p)*(log(n/k)*log(n))이다. 그리고 순차적 방법에서의 가장 빠른 알고리즘(14)은 O(log(log(n/k))*log(k))이므로 병렬 알고리즘의 효율성 EPU=(log(log(n/k))*log(k))/((1/p)*(log(n/k)*log(n))*p)=1임을 알수가 있다. 즉, EPU가 1인 경우는 가장 우수한 알고리즘(10)으로서 최적 가속화율에 대응되게 된다. 그러므로 본 논문의 알고리즘은 거의 최적인 알고리즘을 의미하게 된다. 이것을 가속화율(10, 15)로 S(K)(n)으로 계산한 결과도 같음을 의미하게 된다. 이같은 내용은 표 3에서 참조해 보면 알수가 있다.

이상과 같이 본 논문에서는 이용된 자료구조는 배열을 기본으로 하여 구현을 시켰다. 그러나 이 알고리즘을 포인터를 기본으로 한 방법을 구현시키는 것이 앞으로의 과제이다.

참 고 문 헌

1. Abha Moita and S. S. Iyengar, "Parallel algorithms for some computational problems," Advances in computers, Vol. 26, pp.94-149, Academic press, 1987.
2. C. J. H. McDIAMID, B. A. Reed, "Building Heaps Fast," Journal of Algorithms, Vol.3, No.3, pp. 352 365, Sept., 1989.
3. David Nassimi and Sartaj Sahni, "Data Broadcasting in SIMD Computers," IEEE Transactions on computers, Vol.c 30, No.2, Feb., 1981.
4. Dekel, E. and Sahni, S., "Binary trees and parallel scheduling algorithms," IEEE Trnas. on Computer, Vol.C-32, No.3, pp.307 315, Mar., 1983.

표 3. 순차적 알고리즘과 병렬 알고리즘

		순차적 알고리즘		병렬 알고리즘		
		공 간 복 잡 도	시 간 복 잡 도	프 로 세 서 수	공 간 복 잡 도	시 간 복 잡 도
완전힙	크기가 같은 경우	2*n	O(log (n))	1	2*n	O(log (n))
	크기가 다른 경우	n+k+log (n/k)	O(log(log (n/k)*log(k))	max(2**(i-1), ⌈(m+1)/4⌉	n+k+프로세서 수*(s+p)	O(log(n/k) *log (n))
불완전 힙의 경우		n+k+log (n/k)+2**i	O(log(log(n/k) *log(k))			

5. Ernst E. Doberkat, "Deleting the root of a heap," Acta Informatica, 17, pp.245-265, 1982.
6. G.H. Gonnet and J.I. Munro, "Heaps on heaps," SIAM Journal of Computing, Vol.15, No.4, pp. 964-971, Dec., 1986.
7. J. T. Stasko and J. S. Vitter, "Pairing Heaps : Experiments and Analysis," Communication of ACM, Vol.30, No.3, pp.234-249, March, 1987.
8. Jorg-R Sack and Thomas Strothotte, "An algorithm for merging heaps," Tech. Report, SCR-TR-43, School of computer science, Carleton University, April, 1984.
9. M. A. Weiss and J. K. Navlakha, "The Distribution of keys in a Binary Heap," Proceedings for Workshop WADS '89, pp.510 516, Springer-Verlag, Aug., 1989.
10. Selim G. Akl, "The Design and Analysis of parallel algorithms," prentice-hall, 1989.
11. W. D. Hillis and G. L. Steele, JR, "Data parallel algorithms," Communication of ACM, Vol.29, No. 12, pp.1170-1183, Dec., 1986.
12. Williams, J.W.J., "Algorithm 232, Heapsort," Communication of ACM, Vol.7, No.6, pp.374-348, June, 1964.
13. Yoo, Year Back, "Parallel processing for some network optimization problems," Ph.D dissertation, Washington state Univ., 1983.
14. 김경태, 민용식, "Heap을 병합시키기 위한 빠른 알고리즘," 한국정보과학회, Vol.14, No.4, Dec., 1987.
15. 민용식, "선형 리스트와 힙에 관한 병렬 병합 알고리즘," 광운대학교 대학원 박사학위 청구 논문, Feb. 1991.
16. 민용식, "선형 리스트상의 병렬 병합 알고리즘," 한국음향학회지, Vol.10, No.5, pp.20-26, Oct. 1991.
17. 한탁돈, "The Design and Analysis of parallel algorithms," 한국정보과학회 병렬처리 시스템 학술 강연회 발표집, 1989.
18. 홍영식, 조유근, "다수의 프로세서를 가진 컴퓨터 시스템에 있어서 priority queue를 이용한 병행 스케쥬링 알고리즘," 한국정보과학회, Vol.12, No.2, pp.163-168, May, 1985.
19. 김태남, "On some computational structures for parallel graph algorithm," KAIST 박사학위 청구논문, Feb. 1986.

▲민 용 식(Yong Sik Min) 1958年 1月 19日生.

1981년 : 광운대학교 전자계산학과 졸업(이학사)
1984년 : 광운대학교 대학원 전자계산학과 졸업(이학석사)
1991년 : 광운대학교 대학원 전자계산학과 졸업(이학박사)
1984년~1987년 : 송원실업전문대학 전자계산학과 선임강사
1987년~현재 : 호선대학교 전자계산학과 부교수
1993~현재 : 미국 Louisiana State University 교환교수