선형 리스트상의 병렬 병합 알고리즘

On the Parallel Merging Algorithm for Linear Lists

閔 勇 植*

(Yong Sik Min*)

요 약

본 논문은 병렬 병합 알고리즘중에서 선형 리스트상에서의 병합시키기 위한 알고리즘을 제시하고자 하는데 있어서 다음 두가지에 그 주된 목적이 있다.

(1) 병렬 병합 알고리즘을 위한 ADT를 제시하고 그리고

(2) 선형 리스트상에서 병합되어지는 MKLP알고리즘을 제시코저 한다.

이때 MKLP 방법은 p(1<=p<=n)개의 프로세서를 이용하고 그리고 SIMD-SM(EREW-PRAM) 모델 상에 구현이 되는 병렬 병합 알고리즘을 구현코저 한다.

ABSTRACT

The purpose of this paper is classified as follows : first, to suggest the abstract data type for parallel merging algorithm and second, to suggest MLPM(Min's Linear lists Algorithm for Parallel Merge)'s algorithm which merges in linear lists.

The MLPM's method suggests the parallel linear merging algorithm which uses p(1<=p <=n)'s processors, and is applies it to SIMD-SM(EREW-PRAM) parallel computer which resloves the problem of memory conflict.

I. 서 론

최초의 디지탈 컴퓨터가 30여년전에 만들어진 이래로 이들 기계의 계산능력은 비약적으로 발전을 거듭하여 왔다. 이렇게 증강된 계산능력은 기본적으로 하드웨어의 속도에 크게 의존하여 왔다.[13] 그러나 하드웨어의 비용은 VLSI와 같은 반도체 기술의 발달로 인하여 점차로 하락이 되고 있다.[8,13] 그리고 VLSI와 같은 기술적인 혁신은 하드웨어의 비용을 더욱 더 하락시키게 될 것이며, 기술적인 면에서 더욱 더 큰 계산 능력을 얻기 위하여, 더욱 빠른 계산 능력을 지닌 컴퓨터로서 병렬 컴퓨터에 관심을 갖게 되었다.[8]

이같은 병렬 컴퓨터의 개발에 수반하여 병렬 처리 알고리즘들도 활발한 연구가 진행되고 있다. 즉, 지난 10년동안 병렬 수치 알고리즘에 대한 많은 작업들 즉, 행렬계산, 다항식의 해 구하기, 선형문제, 편미분 방정식등에 대해서 Miranker, Poole과 Voigt, Samth 그리고 Heller등이 포괄적으로 연구하였다.[13] 비수치적 병렬 알고리즘은 주로 분류와 검색에 대해서 Quinn과 Deo[13], [10]에 의해서 연구가 진행되었다.[13]

알고리즘을 설계하는 방법은 divide-and-conquer, dynamic programming, greedy method, backtracking, branch-and-bound등으로 구분[1]하고, 병렬 알고리즘

*湖西大學校 電子計算學科 助敎授

을 설계하는데는 두가지 방법[13,10] 즉, Scratch로부터 병렬 알고리즘을 설계하는 경우와 현재 존재하는 순차적 알고리즘에서 병렬 알고리즘을 구성하는 방법이 있다.

병합이란 두개의 순서화된 선형 리스트를 하나의 선형 순서화된 리스트로서 구성함을 의미한다. 이와같은 병합을 병렬로 병합 시키기 위하여 기존에 제시된 방법을 살펴 보면 다음과 같다.

순서화된 두개의 리스트를 병합하여 새로운 리스트를 형성하는 방법은 다음과 같다.

첫째로, 리스트 A에 처음 P개의 요소들로 병렬로 이진 삽입을 수행하는 Gavil방법으로 p(p<=m<=n)개 프로세서로서 수행빈도가 n / p*log (n+1)이다.

둘째, N(1<=N<=n)개의 프로세서를 이용해서 수행빈도가 n+N*log(n)인 odd-even방법이 있다.

세번째 방법으로 모든 A와 B의 N개의 서브리스트로 분할하여 분할된 서브리스트를 병합하는 방법으로서 N(n+m)개의 프로세서를 이용하여 그 복잡도가 $n+N*\log^2 n$로 수행이 된다.

본 논문은 divide-and-conquer의 한 예인 병합에 대해서 병렬로 병합시키기 위한 ADT(Abstract Data Type)를 제안하고서 선형 순서화된 리스트 상에서의 병합하는 알고리즘을 제안함과 동시에 다른 알고리즘과 비교를 하는데 주된 목적이 있다. 본 연구에 사용된 평가 즉, 시간 복잡도는 병렬 알고리즘의 세가지 평가 방법[1,2]중 가속화 율인 T(1)(n) / T(k)(n)에서 T(k)(n)를 구하고자 한다. 즉, 여기서 T(i)(n)은 i개 프로세서를 이용해서 크기 n의 문제에 관한 시간 복잡도에서의 가장좋은 최악의 복잡도이다. 이때 프로세서들을 할당하거나, 관리하는 문제는 본 논문에서 고려하지 않았다.

본 논문의 구성은 다음과 같다. 제 2장에서는 병렬 병합을 위한 ADT와 선형순서화된 리스트상에서 병합하는 병렬 알고리즘을 제시하고자 한다. 그리고 제 3장에서는 제시된 알고리즘의 시간 복잡도, 프로세서의 수, 사용된 모델을 나열함과 동시에 타 방법론과 비교를 하였다. 끝으로 제 4장에서는 결론을 지었다.

Ⅱ. 병렬 병합 알고리즘

2.1 병렬 병합을 위한 ADT

일반적인 병렬 병합 방법은 리스트 A와 B의 크기 n, m를 병렬로 병합시키기 위해서는 입력된 리스트들을 p개의 프로세서에 할당될 크기로 분할시켜 n(1<=p<=n)개의 서브리스트로 분할시키게 된다. 이같은 서브리스트를 병합시켜 전체 병합된 결과를 얻도록 서브리스트를 결합하므로서 이루어진다. 만약 각 프로세서에 할당된 서브리스트들이 상대적으로 크기가 큰 경우에는 병렬 병합 기법 설계에서 리스트를 분할시켜서 수행을 하게 된다. 이같은 방법을 계속 적용하므로 병렬 병합 방법은 순환적 프로시듀어로서 표현된다. 즉, 더 이상의 분할없이도 병합이 이루어질 크기인 아주 작은 크기의 리스트를 분할시킨 서브리스트로 나타낸 것이다. 이같은 내용을 기본으로 하여 구현시킨 병렬 병합 ADT는 알고리즘 1과 같이 기술이 된다.

알고리즘 1 ADT for the parallel merging

```
function MERGEN(A, B)
  global n,m,p
       A(1 : n), B(1 : m)
  begin
    if SMALLSIZE(A, B)
          then MERGEN=DirectMERGEN(A, B)
          else for all i 1<=i<=p do in parallel
            Ai=Decompose(A, A1,……, Ap)
            Bi=Decompose(B, B1,……, Bp)
            allfor
     for all i 1<=i<=p do in parallel
          Si=MERGEN(Ai, Bi)
  allfor
          MERGEN=Combine(S1, ……, Si)
  end
```

SMALLSIZE(A, B)는 A와 B의 크기가 더 이상 나누어지지 않고도 병합이 이루어질 정도의 적은 크기인가를 결정하는 부울 함수값이다. 만약 크기가 적은 경우에는 그때 병합이 이루어지게 된다.(DirectMERGEN) 그렇지 않은 경우에는 A와 B를 각 프로세서에 할당되도록 A와 B의 서브리스트(Ai와 Bi)로 분할

시키기 위해서 Decompose($A,A_1,\cdots,A_p$)와 Decompose ($B,B_1,\cdots,B_p$)를 이용한다. 그리고 나누어진 서브리스트 A_i와 B_i를 병합하기 위해 S_i=MERGEN(A_i, B_i)로서 순환적으로 이용이 된다. 그리고 Combine($S_1,\cdots,S_i$)는 각 프로세서에 의해서 할당된 리스트를 병합된 결과를 구하는 것이다.

2.2 MLPM 알고리즘

MLPM알고리즘이란 두 개의 선형 순서 리스트 A와 B(|A|=n, |B|=m)

즉, $a_1<a_2<\cdots<a_n$와

$b_1<b_2<\cdots<b_m$

를 병렬로 병합해서 하나의 선형 순서 리스트 C={$c_1, c_2, \cdots, c_{n+m}$}를 얻기 위한 방법으로서 다음 세가지 단계로 수행된다.

첫번째 단계로 공용 기억장소에 있는 선형 순서 리스트 A를 이용해서 각각의 프로세서에 할당될 A의 서브리스트로 분할시킨다. 즉, p개의 서브 리스트{$A_1,A_2,\cdots,A_p$}로서 분할시키기 위해 p개 프로세서가 이용된다. 둘째 단계에서는 각 프로세서에 할당된 A의 서브리스트에서 최대값을 찾는다. 찾은 최대값은 공용기억장소의 MAXS에 할당시킨다. 그리고 각 프로세서는 할당된 자신의 최대값을 이용해서 공용기억장소에 존재하는 선형 순서 리스트 B의 서브리스트를 결정한다. 이때 B의 서브리스트 역시 p개의 서브리스트{$B_1,B_2,\cdots,B_p$}로 분할된다. 이때 다음 두 조건이 만족된다.

(1) $|A_i|+|B_i|=(n+m)/p$(반올림) ($1<=i<=p$)

(2) A_i U B_i에서의 모든 요소들은 A_{i+1} U B_{i+1}의 크기 보다 적다($1<=i<=p$).

마지막 세번째 단계는 각 프로세서에 할당이 되어진 두개의 선형 순서리스트의 서브 리스트 A와 B를 병합해서 리스트 C를 형성한다. 이같이 수행되는 과정을 단계별로 살펴보면 알고리즘 2와 같다.

알고리즘 2 parallel-linear-merge

// 리스트 A, B와 C는 공용기억장소에 있다 //

단계 1 :

PE(i)(i=1,2,$\cdots$,p,p>=1)에 할당되어질 선형 순서 리스트 A의 서브 리스트 A를 생성한다. 여기서 서브 리스트는 {$A_1,A_2,\cdots,A_p$}로 분할된다.

단계 2 :

(2.a) for all i $1<=i<=p$ do in parallel

PE(i)의 서브 리스트 A의 최대값을 찾는다.

allfor

(2.b) for all i $1<=i<=p$ do in parallel

PE(i)의 최대값을 이용해서 자신의 프로세서에 할당된 선형 순서 리스트 B의 서브리스트는 {$B_1,B_2,\cdots,B_k$}($1<=k<=p$)로 분할된다. 그리고 리스트 B를 k개로 분할시키기 위해서 중위수방법을 이용한다.

all for

단계 3 :

for all i $1<=i<=p$ do in parallel

PE(i)는 자신의 국지메모리에 할당된 서브 리스트 A_i와 B_i를 이전 병합방법에 의해 병합한다.

allfor

end

(1) 병렬 병합 방법

두 개의 선형 순서리스트를 병합시키는 첫번째 방법은 공용 기억장소에 있는 리스트 A의 서브 리스트 A를 각 프로세서에 할당시키기 위해 리스트 A를 p ($1<=p<=n$)개의 서브 리스트로 분할하여 각각을 프로세서에 할당시킨다. 이것은 다음과 같이 수행된다.

(1.a) for j=1 to $\lceil n/p \rceil$ do

(1.b) for all i $1<=i<=p$ do in parallel

PE(i)의 국지 메모리에 리스트 A 가운데의 $A(1+(i-1)*\lceil n/p \rceil+(j-1)$번째를 옮긴다.

allfor

endfor

[정리 1]

서브 리스트 A_i로 분할시키기 위해 $O(\lceil n/p \rceil)$로서 수행된다.

(증명)

리스트 A를 서브 리스트 A로 분할시키는 방법에서 단계 1.b의 수행 빈도는 0(1)이고, 1.a의 수행 빈도는 0(⌈n/p⌉)이다. 전체 수량빈도는 0(⌈n/p⌉)가 된다.

(2) Max-Median 알고리즘

리스트 A의 서브 리스트들이 각 프로세서에 할당된 후 두번째 단계로서 리스트 B의 서브 리스트를 결정해야 한다. 이때 먼저, 각 프로세서들이 지니고 있는 서브 리스트 A_i들의 최대값을 구하게 된다. 최대값은 A_i가 지닌 제일 마지막번째 요소의 값이다. 왜냐하면 서브 리스트 A_i는 선형 순서 서브 리스트이기 때문이다. 그리고 이 최대값에 따라 리스트 B의 서브 리스트 B_i를 결정케 된다. 즉, 각 프로세서에 해당되는 리스트 B의 서브 리스트 B_i는 앞에서 추출된 각 프로세서의 최대값으로 해당된 리스트 B의 위치를 중위수를 이용해서 결정케 된다.

예를 들면, 선형 순서 리스트 A={1, 3, 5, 7, 8, 11, 14, 16}(|A|=8), 선형 순서 리스트 B={2, 4, 6, 9, 12, 13, 15}(|B|=7)이고, 프로세서 수가 3개이라고 가정하자. 이때 병합시키기 위한 첫번째 단계로서 각 프로세서에 할당되는 리스트 A의 서브 리스트는 그림 1(b)처럼 각 프로세서가 3개의 구성요소(⌈n/p⌉=⌈8/3⌉=3)로서 형성된다.

즉, A_1={1, 3, 5} A_2={7, 8, 11} A_3={14, 16}

그리고 두번째 단계로서 각 프로세서에 할당된 서브 리스트{A_1,A_2,A_3}의 최대값을 구한것이 그림 1(c)의 MAXS(i)와 같다. 이 최대값을 이용해서 리스트 B의 서브 리스트 B_i를 구하기 위해서 각 프로세서(PE(i))의 해당 최대값(MAXS(i), i=PE(i)의 i)으로 Sequence-Median 알고리즘(알고리즘 4)을 이용하여 수행한 결과 그림 1(d)와 같이 분할된다. 이상과 같이 수행되는 과정은 알고리즘 3과 같다.

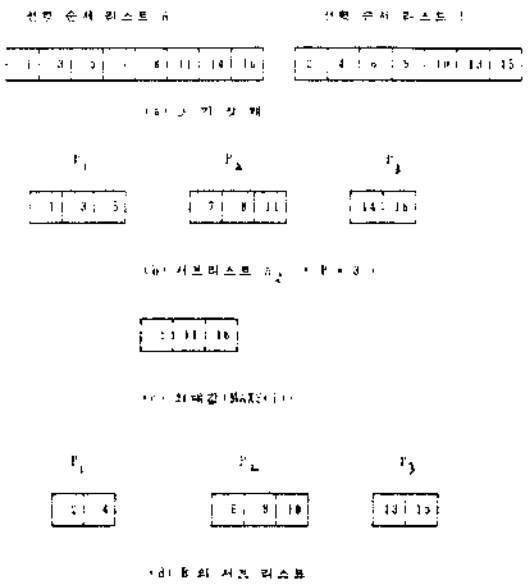

그림 1. Max-Median 방법

알고리즘 3 parallel-Max-Median

step 1 :

(1.a) for all i 1<=i<=p do in parallel

PE(i)는 자신이 지닌 서브 리스트 A중에서 제일 큰 값을 결정한다. 즉, max=A(⌈n/p⌉*(i-1))이다.

allfor

(1.b) for all i 1<=i<=p do in parallel

MAXS(i)=max

allfor

step 2 :

(2.a) for all i 1<=i<=p do in parallel

Sequence-Median(B, MAXS(i), u_{i-1}+1,m,u)

U(i)=u_{i-1}

for j=U(i-1)+1 to U(i)

PE(i)는 자신의 국지 메모리에 B(j)를 옮긴다.

endfor

if the rest of B exists

then allocate them to the last PE

allfor

end

알고리즘 4 Sequence-Median(A,X,e,f,u)

//A : 리스트, X : 찾고자하는 구성요소 값//

//e : 리스트 A의 첫 위치, f : 리스트 A의 마지막 위치//

//u : 리스트 A에서의 중위수 위치//

단계 1 :

```
    low=e
    high=f
    n=f
단계 2 :
    while n>1 do
      u=low+⌈(high-low-1)/2⌉
      w=⌊n/2⌋
      n=n-w
      if A(u)>=X
          then high=high-w
          else low=low+w
  endif
endwhile
단계 3 :
    if A(u)>X
          then u=u-1
    endif
end
```

[정리 2]

알고리즘 4 Sequence-Median의 수행 빈도는 0(log m)이다(여기서 m는 리스트 B의 크기).[8]

[정리 3]

알고리즘 3 parallel-Max-Median의 수행 빈도는 0(m/p)이다.

(증명)

알고리즘 3의 단계 1에서 1.a의 수행 빈도는 0(p), 1.b는 0(p)가 된다. 즉, 단계 1의 수행 빈도는 0(p)이다. 단계 2의 2.a에서 Sequence-Median의 수행 빈도는 정리 2처럼 0(log m)이므로 2.a의 수행 빈도는 0(log(m))이다. 그리고 2.b는 0(m/p)로 수행된다. 그러므로 단계 2의 수행 빈도는 0(max(m/p, log(m))로 된다. 알고리즘 3의 전체 수행 빈도는 0(m/p)이다.

마지막인 세번째 단계에서는 각 프로세서에 할당된 서브리스트 A와 B를 이진 병합 방법에 의해서 병합하면 알고리즘은 끝난다. 이상의 병합 과정을 알고리즘 5와 같다.

```
알고리즘 5 parellel-sublist-merge
step 1 :
  for all i 1<=i<=p do in parallel
    PE(i) having received the quadruple(a,b,c,d)
    //a,b는 리스트 A의 색인//
    //c,d는 리스트 B의 색인//
    (1.a) w=1+((i-1)*⌈n/p⌉+(c-1))
    (1.b) z=min{i*⌈n/p⌉+d,(n+m)}
    (1.c) BINARY-MERGE(A(a,b), B(c,d), C(w,z
    ))
  allfor
end
```

[정리 4]

procedure BINARY-MERGE를 수행하기 위한 빈도는 0(log n)이다. 이때 n는 구성요소의 수이다.[8]

[정리 5]

알고리즘 parallel-sublist-merge의 수행 빈도는 0(m/p *log(n/p))로 된다.

(증명)

알고리즘 5의 단계 1에서의 1.a 수행 빈도는 0(1), 1.b는 0(1), 1.c는 A의 구성요소가 ⌈n/p⌉, B의 구성요소가 m/p이므로 이 두 서브리스트를 이진 병합하면 (정리 4참조) 수행 빈도는 0(m/p*log(n/p))로 된다. 그러므로 이 알고리즘의 전체 수행 빈도는 0(m/p*log(n/p))가 된다.

Ⅲ. 병렬 병합 알고리즘의 비교및 분석

본 절에서는 MLPM 알고리즘에서 수행된 시간 복잡도, 공간 복잡도 그리고 프로세서 수를 분석하고, 타 방법들과 비교 분석하였다.

병렬로 병합시키기 위한 시간 복잡도는 다음과 같다. A의 서브 리스트를 분할하는 경우의 복잡도는 0(⌈n/p⌉)이고, 둘째 단계에서 서브 리스트 A의 최대값과 이것으로 서브 리스트 B를 구하는 방법은 0(m/p), 마지막 단계에서 서브 리스트를 병합시키는 복잡도가 0(m/p*log(n/p))이다. 그러므로 병렬로

표 1. 타 알고리즘과 비교

	사용된 모델	프로세서 수	시간 복잡도	기본 개념
Gavil의 방법	SIMD-SM-R	p(p<=m<=n)	n/n* log(n+1)	리스트 A의 처음 p개 요소들로서 병렬로 이진 삽입을 수행하여 병합
odd-even method	SIMD-SM	N(1<=N<=n)	n+N* log n	1. A와 B의 N개 서브리스트로 분할 2. 분할된 서브리스트 병합시켜 V에 놓는다. 3. PE(i)는 병합해서 C에 삽입
KLA의 방법	SIMD-SM	N(1<=N<=n+m)	n+N*log²n	1. A와 B의 N개 서브리스트로 분할 2. 분할된 서브리스트 A와 B를 병합
MLPM 알고리즘	SIMD-SM	p(1<=p<=n)	m/p* log(n/p)	1. A의 p개 서브리스트 분할 2. B의 p개 서브리스트 분할(Max-Median방법이용) 3. 분할된 서브리스트 A와 B를 병합

병합시키기 위한 전체 시간복잡도는 0(m/p*log(n/p))이다.

필요로 하는 공간 영역은 다음과 같다. 첫째 공용기억장소에 기억되어져 있는 리스트 A, B, C의 크기 n, m, n+m를 생각할 수 있다. 그리고 둘째로 MAXS의 크기는 ⌈n/p⌉이고 U역시 ⌈n/p⌉이다. 그리고 각 프로세서의 국지 메모리에는 서브 리스트 A와 B인(n+m)/p개의 영역이 존재한다. 이때 프로세서 수 만큼이므로 p*(n+m)/p개의 영역이 필요하다. 그러므로 전체 공간 복잡도는 3*(n+m)+2*⌈n/p⌉가 필요하다. MLPM 알고리즘에 사용되는 프로세서는 p(1<=p<=n)개다.

병렬로 병합시키기 위한 방법들과 본 논문에서 제시한 MLPM알고리즘에 대한 컴퓨터 모델, 프로세서 수, 시간 복잡도, 기본 개념등을 비교한 표가 표1과 같다. 여기서 공간 복잡도는 비교대상에서 제외시켰다. 표 1에서 보듯이 제시된 알고리즘의 경우에서 m는 리스트 B의 크기를 나타내는데, 이것이 리스트 A의 크기인 n와 같은 경우에는 다른 방법과 시간 복잡도 면에서 보다 좋음을 나타내고 있다. 만약 m<n인 경우에는 다른 방법에 비해 상당한 축약이 있음을 보여주고 있다. 그리고 프로세서의 수(N와 p)에서도 AKL이 사용한 수 N(n+m)보다 훨씬 적은 p(1<=p<=n)개 이다.

Ⅳ. 결 론

본 논문은 공용 기억장치를 지닌 병렬 컴퓨터에서 divide-and-conquer의 형태인 병합에 관한 것으로, 병렬 병합에 대한 Abstract Data Type와 선형 순서화된 리스트에서의 병합(MLPM 알고리즘)을 제안하였다.

선형 순서화된 두 개의 리스트를 병합하기 위한 효율적인 알고리즘으로 Max-Median방법을 이용하여 처리하였다. 이때 사용된 프로세서는 p(1<=p<=n)개로서 그 수행 빈도가 0(m/p*log(n/p))이다. 이 방법은 AKL이 제시한 방법에서 사용된 프로세서 수가 n+m를 n개로 줄임과 동시에 m<n인 경우에 시간 복잡도면에서도 상당히 축약시켰다.

앞으로의 과제는 선형 리스트에 이용된 계산 모델은 SIMD로 구현시켰으나, 이 알고리즘들을 MIMD모델 상에서도 적용시킬 수 있도록 하는 것이 남은 과제이다.

참 고 문 헌

1. AKL, Selim G., "The Design and analysis of parallel algorithms", prentice-hall, 1989.
2. C. P. Kruskal, "Searching, Merging and Sorting in Parallelcomputation", IEEE Trans. on computer, Vol. c-32, No. 10, pp.942-946 oct., 1983.
3. David Nassimi and Sartaj Shani, "Data Broadcasting in SIMD Computers", IEEE Transactions on computers,

Vol. c-30, No. 2, Feb., 1981.
4. G. H. Gonnet, "Handbook of Algorithms and Data Strutures", Addison-Weseley, 1984.
5. Hwang F. K. and S. Lin, "A simple algorithms for merging two disjoint linearly-ordered sets", SIAM J. Computing, No. 1, pp.31-39, 1972.
6. Lydia Kronsjo, "Computational Complexity of Sequential and Parallel Algorithms", John Wiley & Sons, 1985.
7. M. B. Brown, R. E. Tarjan, "A Fast Merging Algorithm", Journal of ACM, Vol.26, No. 2, pp.211-226, april 1979.
8. Moitra, Abha and Iyengar, S. S., "Parallel algorithms for some computational problems", Advances in computers, Vol.26, pp.94-149, Academic press, 1987.
9. N. Gehani and A. D. McGettrick, "Concurrent programming", Addison Wesley, 1988.
10. Quinn, M. J. and Deo, N., "Parallel graph algorithms", Computing Surveys, Vol. 16, No. 3, pp.319-348, Sept., 1984.
11. Ralph Duncan, "A survey of Parallel Computer Architecture", IEEE Computer Society, Vol. 23, No. 2, pp.5-17, Feb., 1980.
12. Selim Akl and N. Santoro, "Optimal parallel merging without memory conflicts", Technical Report, SCR-TR-75, School of computer scence Caleton Univ., May. 1985.
13. Yoo, Year Back, "Parallel processing for some network optimization problems", ph.D dissertation, Washington state Univ., 1983.
14. W. D. Hillis and G. L. Steele, JR, "Data parallel algorithms", Communication of ACM, Vol. No. 12, pp.1170-1183, Dec., 1986.
15. 민용식, "선형 리스트와 힙에 관한 병렬 병합 알고리즘", 광운대학교 대학원 박사학위 청구 논문, Feb., 1991.
16. 한탁돈, "The Design and Analysis of parallel algorithms", 한국정보과학회 병렬처리 시스템 학술 강연회 발표집, 1989.

▲閔 勇 植(종신회원) 1958年 1月 19日生

1981年：光云大學校 電子計算學科 卒業
1983年：光云大學校 大學院 電子計算學科 卒業(理學碩士)
1984年～1987年：松源實業專門大學校 電子計算學科 專任講師
1991年：光云大學校 大學院 電子計算學科 卒業(理事博士)
1987年～現在：湖西大學校 電子計算學科 助教授

※主關心分野는 Sequential and Parallel 알고리즘의 설계및 분석, 컴퓨터 그래픽스등이다.