슈퍼컴퓨터 병렬처리 알고리즘

원 영 주
(육군사관학교 교수부 조교수)

1. 서 론

병렬처리 알고리즘이라 함은 일반적으로 슈퍼컴퓨터라는 하드웨어에만 다루어지는 것은 아니지만 대부분 병렬성을 지닌 슈퍼 컴퓨터의 특성을 고려할때 슈퍼컴퓨터의 범주를 기존의 상용 병렬처리 시스템과 또한 병렬성이 뛰어난 이론적인 네트워크 모델등을 통합하여 일컬을 수 있다.

여기에서도 병렬성을 근간으로 하는 여러 각도에서의 슈퍼컴퓨터에 대한 이론적인 알고리즘의 부류와 상용컴퓨터에 적용하여 사용하는 실용적인 알고리즘등을 다음과 같은 여러 종류의 taxonomy를 혼합하여 소개하겠다.

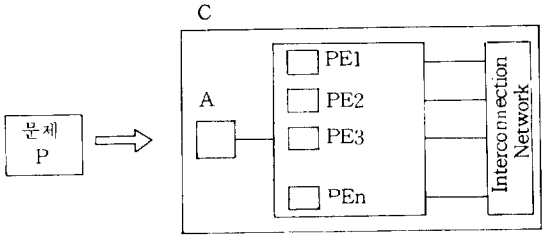
1) 프로세서수 P에 따른 분류
   -Fine grained
   -Medium grained
   -Coarse grained
2) 사용 목적의 범용성에 따른 분류
   -General Purpose
   -Semi-General Purpose
   -Special Purpose
3) 명령어의 데이타 흐름에 따른 분류
   -SISD
   -SIMD
   -MISD
4) 메모리 Read/Write 모델에 따른 분류[39]
   (SHARED MEMORY)
   -SIMDAG
   -EREW PRAM, CREW PRAM, CRCW PRAM
   -PP-RAM
   -SP-RAM
   -RP-RAM

먼저 병렬처리 알고리즘의 효율성과 성능평가 방법, 그리고 디자인 절차등의 고려사항을 개념적으로 설명하고 응용분야에 대해 알려진 병렬 알고리즘들을 소개하겠다. 응용분야로는 순차배열 알고리즘, 행렬알고리즘, 그래프 알고리즘으로 제한하여 소개하였고 응용분야에 대한 간단한 소개를 하였다. 마지막으로 병렬 알고리즘의 연구추이를 살펴보았다.

2. 병렬 알고리즘의 개념

일반적으로 주어진 문제 P를 병렬처리기 T에서 해결하고자 할 경우 먼저 P를 병렬처리가 가능부분 Pi들로 나누는 작업(Partitioning)이 필요하다.

Pi는 그 알고리즘을 실행하는 프로그램 Code를 나누고 또한 알고리즘이 처리하는 데이타를 나누는 혼합적인 의미를 가진다. 이 나누어진 Subtask는 일정한 양의 일을 수행한 다음 서로 데이타를 주고 받거나 일을 다 마쳤을 경우 종료를 알리는 적당한 방법의 동기화(Synchronization) 절차가 필요하다. 컴퓨터를 사용하는 사용자는 항상 시간 및 공간 복잡도의 측면에서 최적의 방법을 요구하기 때문에 이와 같은



분할과 제어의 문제들은 중요한 위치를 차지하고 또한 이론적인 정립이 필요한 것이다.

2.1 병렬 알고리즘의 환경

알고리즘의 효율성을 말할 때는 그 알고리즘이 적용되는 환경에 대한 많은 가정이 필요하다. 예를 들어 그림과 같은 병렬처리기 C에서 문제 P를 해결하고자 할 경우 프로그램과 데이타의 위치에 따라 최적의 병렬 처리 알고리즘은 서로 다를 수 있을 것이다.

만일 PE의 갯수가 100개이고 문제P는 100개의 Element 중 최대 값을 찾는 문제라 하자.

이때 100개의 element가 A의 local memory에 있을 경우는 100개의 element가 이미 100개의 PE에 산재해 있을 경우의 알고리즘과는 전혀 다른 형태를 가질 수 있다.

전자의 경우 대부분의 컴퓨터에서 최대값은 A자체에서 찾는 것이 훨씬 효율적이 된다. 이와같은 target computer와 문제의 초기형태등의 가정들이 알고리즘의 개발 환경으로 항상 수반되어야 한다.

2.2 병렬 알고리즘의 선택

병렬 컴퓨터라고 해서 항상 병렬 알고리즘을 사용하는 것이 좋은 것은 아니다. 즉, 주어진 환경에서 문제를 해결하기 위해 어느 병렬 알고리즘 혹은 순차 알고리즘을 선택할 것인지를 결정하는 것도 고려되어져야 한다. 다음과 같은 예를 보자.

[7]은 n vertex의 graph에서 all pairs shortest path를 찾는 시간복잡도가 $O(\log^2 n)$이며 $O\left(\frac{n^3}{\log n}\right)$ 프로세서를 사용한 병렬 알고리즘을 제안하였다.

이에 반해 잘 알려진 Floyd의 sequential 알고리즘은 $O(n^3)$의 시간 복잡도를 가졌다. 따라서 실제로 우리가 k개의 프로세서를 가졌을 경우, [7]의 알고리즘을 사용한다면 프로세서 하나가 $O\left(\frac{n^3}{k \log n}\right)$ 프로세서의 일을 수행하게 하여 실현할 수 있다. 따라서 전체의 일의 양을 (프로세서수 X시간)으로 나타내었을 때 [7]의 알고리즘은 $O(n^3 \log n)$의 양에 해당하는 일을 해야 한다.

이것은 Floyd 알고리즘의 $o(\log n)$배의 값에 해당한다. 구체적으로 4log n 배라 가정하면 R개의 프로세서를 사용했을 경우 병렬알고리즘 : Sequential알고리즘은 $\frac{4n^3 \log n}{k} : n^3$의 시간 복잡도를 갖는다.

즉, $k > 4\log n$일 경우에만 병렬처리를 함으로써 이득을 얻을 수 있다는 것이며 이것은 n=1024인 경우, 40개의 프로세서를 이용하여 병렬알고리즘을 적용한 것과, 1개의 프로세서를 이용 Sequential 알고리즘을 적용한 것이 같은 시간이 걸린다는 뜻이다. n이 커짐에 따라 k는 더 커질 것이며 이러한 경우 Sequential 알고리즘을 그대로 이용 $o\left(\frac{n^3}{k}\right)$의 시간 복잡도를 갖는 것을 사용하는 것이 훨씬 효율적이라는 것을 알 수 있다.[51] 실제로 [19]는 sequential 알고리즘을 변형하여 $k=16<40$개의 프로세서를 갖는 hypercube에서 112 vertex graph에 대해 약 10배의 speedup을 실험적으로 얻었다.

2.3 병렬 알고리즘의 디자인

다양한 병렬구조를 내포한 슈퍼컴퓨터도 일반적인 컴퓨터에서와 마찬가지로 사용자가 요구하는 궁극적인 목표는 원하는 task를 상위 레벨의 description으로 나타내기만하면 그것을 target computer에 맞게 여러 level에서의 알고리즘을 선택, 병렬구조에 맞게 변환하는 Intelligent Translator 혹은 compiler의 구현이라고 할 수 있다.

이것은 DFS (Depth First Search) 나 BFS (Breadth First Search)의 방법중 어느것을 사용할 것인가 하는 것과 같이 논리적이고 상위레벨의 선택일 수도 있으며, 혹은 array로 정의된 데이타의 특정한 연산을 할 경우 그 subarray의 길이를 어떤값으로 나누느냐 하는 것과 같은 하위 레벨의 선택일 수도 있다. 대개 이와 같은 경우는 사용자가 컴퓨터의 구조에 대한 지식을 요구하지 않으며 응용분야에 대한 새로운 병렬 알고리즘의 선택이나 구현은 연구의 대

상에서 상대적으로 멀어질 것이다.

그러나 이러한 이상적인 시스템의 실현은 사실 거의 불가능하다. 따라서 사용자가 컴퓨터의 구조를 정확히 알고 있고, 사용하는 컴퓨터의 구조에 따라 최적의 알고리즘이 달라질 수 있는 환경에서 알고리즘을 디자인할 경우 고려되어져야 하는 것 중에 몇 가지를 소개하면 [34,35]

- 새로운 알고리즘의 개발 : 이것은 앞에서 설명한 대로 기존의 Sequential algorithm을 변형하여 병렬알고리즘을 만드는 것과, 전혀 새로운 병렬 알고리즘을 개발하는 경우로 나눌 수 있다. 이 알고리즘의 개발은 다음과 같은 세부적인 내용을 포함하기도 한다.
- 선택된 알고리즘을 target컴퓨터에 적절하게 mapping하는 방법
- target컴퓨터 topology에서 data의 효율적인 communication기법의 적용
- Static 혹은 dynamic하게 가용한 모든 computing power를 이용할 수 있도록 일을 균등히 나누는 적절한 balancing기법의 사용 등이다.

2.4 병렬 알고리즘의 성능평가

Super컴퓨터에서의 알고리즘은 그 성능의 세밀한 표현이 요구되므로 알고리즘의 시간복잡도를 단순한 big-O notation 등과 같이 나타내기에는 부적합하다. 즉, 더 자세하고 정확한 시간 복잡도의 표현이 요구된다. 이를 위해서 시간복잡도를 결정짓는 많은 인자들(parameters)의 집합으로 부터 그 알고리즘의 성능을 표현할 수 있다. 이 인자들로는 각 프로세서의 계산속도, 프로세서의 수, 상호연결망의 latency나 throughput 혹은 프로세서의 topology, 메모리 용량, I/O bandwidth등을 생각할 수 있다.

그러나 이와같은 많은 인자들로 구성된 수식으로 알고리즘의 성능을 표현하는 것은 매우 어려울 뿐 아니라 정확한 표현을 할 수 없는 경우가 대부분이다. 따라서 병렬 알고리즘의 성능은 실제로 simulation기법이 많이 사용된다. 즉 병렬처리 알고리즘에 대한 simulation을 통해서 통계적인 성능의 분석과 평가를 하는 것이다. 이것은 아직도 정립되지 않은 상태로 연구의 여지가 많은 분야이다.

병렬 알고리즘의 성능을 실험적으로 평가하는데는 대개 S/P(S : speedup, P : 프로세서수)의 값으로 나타낸다. 우리는 하나의 프로세서로 O(N)의 시간복잡도를 갖는 문제를 P개의 프로세서를 사용하여 O(N)/P 시간에 해결할 수 있는 병렬 알고리즘을 얻는다면 성공적이라 할 수 있다. 따라서 병렬 알고리즘의 효율성은 S/P의 값이 클수록 좋은데 거의 대부분 $0<S/P<1$의 값을 갖는다.

부정확하기는 하나 이론적으로 알고리즘을 분석하기 위해서는, 완전한 병렬알고리즘을 표현하기 위해서는 task를 processor에 분할하고 데이타와 communication의 분배를 인자들로 나타낼 수 있는 도구가 필요할 것이다. 또는 communication primitive (예 : SEND, RECEIVE 등)로써 병렬 알고리즘의 표현을 함으로써 LOW LEVEL 에서의 알고리즘 분석을 도울 수도 있다.

후자의 대표적인 예가 앞에서 언급한 PRAM 모델이다. 모델은 단점 즉 communication 비용을 무시했기 때문에 실현하는데 제한점이 있음에도 불구하고 널리 사용되는 이유는 다른 모델에서의 전환에 융통성(예로 mesh나 hypwrcube와 같은 bounded degree network에서 PRAM의 Simulation이 가능하다)이 있기 때문이다.

또한 Communication과 Synchronization을 machine independent 하게 표현할 수 있는 Primitive를 사용하는 것도 필요하다. [29]

3. 병렬 알고리즘의 소개

병렬 알고리즘은 실제의 병렬처리기가 만들어 지기 훨씬 이전에 연구되고 분석되어 비교적 많은 종류의 모델에 대한 결과 들이 존재한다. 여기에서는 지면의 제한상 다음의 몇가지 경우에 대해 간단히 소개하고자 한다.

3.1 Matrix 알고리즘

가장 간단한 형태의 예로 PRAM 모델에서의 n개 element중 최대값을 O(log n)시간에 찾는 알고리즘을 그림 1에 보였다. [12] 이것과 유사하게, Matrix의 곱셈도 O(n)이나 O(log n) 시간 (n^2, n^3 프로세서를 각각이용)에 구할 수 있음을 쉽게 알 수 있다. (그밖의 Matrix multiplication 알고리즘은 그

림 2 참조)

또한 더 빠르게는 SIMDAG 모델에서 O(Log Log n)시간에 최대값을 구할 수 있다. 최근에는 점차로 상용 MIMD에 대한 알고리즘 연구가 많아지고 있다. 그 예로 Search 기술중 [30]은 Branch and bound의 대표적인 Traveling Salesperson 문제를 loosely coupled MIMD 인 CM에서 병렬처리 16개의 프로세서 사용시 S=8을 얻었다.

3.2 Sorting 알고리즘

Sorting은 컴퓨터 Software의 중요한 부분을 차지

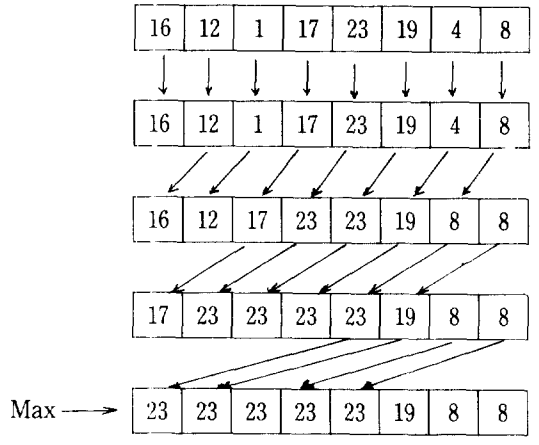


그림 1. 최대값을 찾는 과정

Model	Complexity	Processors	Reference
SIMD-MC2	$\Omega(n)$	$n*n$	[11], [49]
SIMD-CC	$\Theta(\log n)$	$n=2^{3q}$	[7]
SIMD-PS	$\Theta(\log n)$	$n=2^{3q}$	[DEK81][7]
SMM	$\Theta(\log n)$	$n^3/\log n$	[SAV81][40]
SMM	$O(n^{\log 7}/p)$	$p<n^{\log 7}/\log n$	[CHA76][5]

그림 2. 행렬곱셈 알고리즘

Model	Complexity	element	Processors	Reference
SIMD-MC2	$\theta(\sqrt{n})$	$n**2$	$n**2$	[45]
SIMD-MC2	$\theta(n+n^{2/3}\log n)$	$n**2$	$n**2$	[45]
SIMD-MC2	$\theta(n)$	$n**2$	$n**2$	[32]
SIMD-MC2	$O(n\log n)$	$n**2$	$n**2$	[36]
SMM	$O(\log n\log\log\log n)$	n	n/2	[48]
SMM	$O(\log n)$	n	nlogn	[37]
SMM	$O(\log n)$	n	$n\sqrt{n}$	[14]
Int. Net.	$\theta(\log^2 n)$	$n=2^k$	$2^{k-1}(k(k-1)+1)$	[42]
Int. Net.	$\theta(\log^2 n)$	$n=2^k$	$2^{k-2}k(k+1)$	[2]

그림 3. 순차배열 알고리즘

한다. 따라서 Sorting에 대한 많이 되어졌으며 그림 3에 대표적인 예를 보였다. 이러한 예들은 대부분 이론적인 분석을 보이지만 실제로 Sorting을 위한 Sorter는 VLSI 기술의 발달로 인해 쉽게 만들어지게 되었다. 특히 VLSI chip에서는 시간 복잡도 뿐만 아니라 그 알고리즘을 hardware로 실현했을 경우 circuit이 차지하는 공간도 고려 되어야 한다.

[46]은 존재하는 Sorting 알고리즘을 VLSI화 하였을 경우를 가정 13개의 VLSI Sorter에 대한 공간/시간의 Trade off를 분석 비교 하였다. 그밖의 MIMD에서의 Sorting은 [41,50,52] 등에서 소개 되었다.

3.3 Graph 알고리즘

그래프 알고리즘의 대표적인 예들은 그림 4를 참조하기 바란다.

〈 Parallel Connected Components agorithms 〉

Model	Complexity	Processors	Reference
SIMD-SM-RW	$O(\log n)$	n^4	[22]
SIMD-SM-R	$O(\log^2 n)$	n^2	[13]
SIMD-SM-R	$O(\log^2 n)$	$n(n/(\log n))$	[15]
SIMD-SM-R	$O(\log^2 n)$	$n(n/\log^2 n)$	[6]
SIMD-SM-R	$O(\log n \log d)$	$O(n^3/\log n)$	[40]
SIMD-SM-RW	$O(\log n)$	$2m$	[1]
d : diameter of the Graph ;			

〈 Parallel Biconnected Components algorithms 〉

Model	Complexity	Processors	Reference
SIMD-SM-R	$O(d \log^2 n)$	$O((n+m)/d)$	[10]
SIMD-SM-R	$O(\log^2 n)$	$O(n^3/\log n)$	[40]
SIMD-SM-R	$O(\log^2 n)$	$O(n(n/\log^2 n))$	[47]
SIMD-SM-RW	$O(\log n)$	$O(n+m)$	[44]
d : diameter of the graph, k : number of biconnected components			

〈 Parallel Minimum Spanning tree lgorithms 〉

Model	Complexity	Processors	Reference
Tree	$O(n \log n)$	$n/\log n$	[4]
MIMD-TC	$O(n^{1.5})$	$n^{0.5}$	[9]
SIMD-SM-R	$O(\log^2 n)$	$O(n^2)$	[40]
SIMD-SM-R	$O(\log^2 n)$	$n(n/\log^2 n)$	[6]
SIMD-SM-RW	$O(\log n)$	$O(n^3)$	[16]
SIMD-SM-RW	$O(\log n)$	$O(n^3)$	[17]
SIMD-SM-R	$O((m \log n)/p)$	$p \log p \leqq (m \log n)/n$	[25]

〈 Misc. graph algorithms 〉

Problem	Model	Complexity	Processors	Reference
shortest path	SIMD-CC	$\Theta(\log^2 n)$	n^3	[7]
	SIMD-PS	$\Theta(\log^2 n)$	n^3	[7]
maximum matching	SIMD-SMM	$O(\log^2 n)$	$O(n)$	[8]
planarity testing	SIMD-SM-R	$O(\log^2 n)$	$O(n^{3.29}/\log^2) n$	[18]

그림 4. 그래프 알고리즘

3.4 응용 분야 및 기본 알고리즘[43]

기본 알고리즘(Fundamental Algorithms)이란 많은 응용문제를 해결하는 데 공통적으로 사용되어지는 Subroutine이나 Library 개념의 부분 알고리즘으로 대부분 응용문제들은 이러한 FA의 집합이나 Sequence로 나타낼 수 있다.

예를 들면, Gauss elimination과 같은 알고리즘은 많은 응용분야에 사용되어지는 대표적인 FA이다.

그림 5는 슈퍼 컴퓨터에서의 많은 응용분야에 사용되어지는 FA를 나타내었다. 그림에서와 같이 하나의 응용문제는 3~10가지 정도의 FA로 구성됨을 볼 수 있다. 이것은 주어진 응용문제에 대한 대략적인 성능을 평가하는 데 이용되어질 수 있다. FA의 성능향상이 곧 많은 Application의 성능 향상이라고 할 수있으며 이것은 곧 FA에 대한 연구의 중요성을 나타낸다.

이 밖에도 MIMD나 vector 컴퓨터에서의 알고리즘 이나 실용의 예들은 [3,20,27,28] 등에서 찾아볼 수 있다.

4. 병렬알고리즘 연구 방향과 추이

최근 상용 병렬처리기의 많은 출현으로 종래 이론적인 모델에 근거한 시간 및 공간 복잡도의 분석에 기초한 알고리즘에 대한 연구는 차츰 실험적인 연구로 흐르고 있다. 이것은 단지 실험적 결과 분석만을 의미하는 것이 아니라 실험을 통한 통계적 모델의 정립과 시스템 파라미터의 유추등과 같은 광범위한 의미를 지닌다.

현재 많은 연구의 대상이 되고 있는 분야를 정리해 보면

- 좀더 근본적인 병렬 알고리즘의 Paradigm을 위한 이론적인 하드웨어 모델과 알고리즘 분석 tool의 개발을 연구하고 있으며 가장 연구의 여지가 많은 분야이다.
- 병렬처리기의 환경에 좌우되지 않는 상위 level의 (Search strategy 혹은 Balancing strategy) 알고리즘 정립
- 수치적인 계산의 Pipelined Vector Computer를 개선, Symbolic 연산을 위한 슈터 컴퓨터 구조

APPLICATION	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Structural analysis	x		x			x								
Weather simulation	x	x			x		x	x		x				
Particle transport											x	x	x	
Circuit simulation			x	x						x				
Computational fluid dynamics	x	x	x			x	x	x		x				
Molecular dynamics										x	x			
Wave propagation	x	x			x		x		x	x	x		x	
Reservoir simulation	x	x	x		x					x	x	x		
Nuclear reactor plant simulation	x	x	x		x	x				x	x	x		
Radiation shielding analysis		x	x							x				
Plasma simulation							x	x		x	x			
Quantum Chemistry	x	x			x									
Semiconductor device simulation	x	x	x		x			x	x	x	x	x	x	
Seismic analysis	x	x	x							x				
Nuclear safety analysis	x	x	x		x	x				x				
Nuclear reactor core analysis	x	x	x							x				
Gas dynamics	x	x	x							x				
Polymer simulation										x	x			
Enconometric modelling	x	x	x	x	x					x	x	x		
Electromagnetic field analysis	x		x					x	x			x		

FUNDAMENTAL ALGORITHMS :

1. Direct linear equation solvers
2. Iterative linear equation solvers
3. Sparse matrix methods
4. Least squares solvers
5. Non-linear equation solvers
6. Eigenvalue determination
7. Fast Fourier transforms
8. Poisson equation solvers
9. Preconditioning(conjugate gradient)
10. ODE solvers(stiff and non/stiff)
11. Monte Carlo methods
12. Numerical integration
13. Polynomial interpolation
14. Integral transforms

그림 5. 응용분야와 FA(Reprinted from [43])

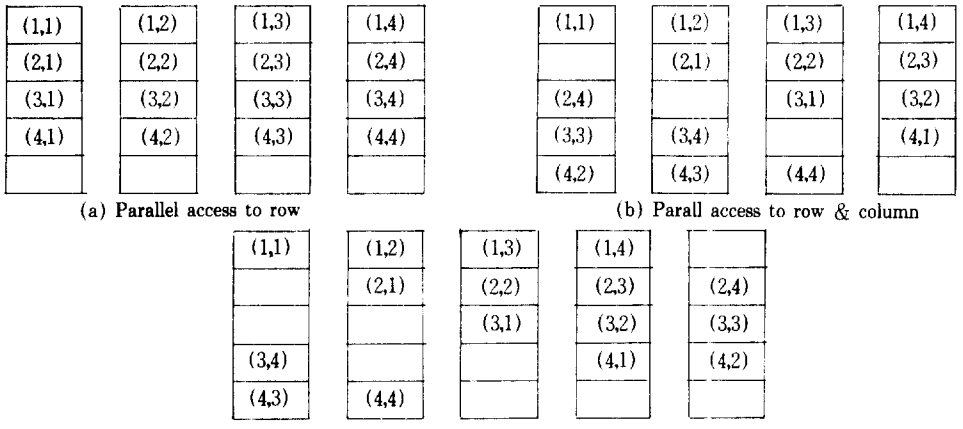


그림 6. 행렬 저장방법

의 개발에 따라 이를 효율적으로 사용하는 알고리즘 연구

- 각기의 슈퍼컴퓨터 하드웨어와 직접 관련되는 데이타 구조에 대한 연구 그림 6은 Pipelined Vector Computer의 4개의 Banked Memory 구조에 4X4 Matrix를 저장하는 세가지 경우를 나타낸다.

[21]이 지적한 대로 Matrix를 사용하는 많은 응용문제가 Matrix의 row와 column, 혹은 Diagonal의 동시 access를 요구함을 생각할 때 병렬처리기에서 그림과 같은 데이타 구조의 다양한 변환이 필요하다.

- 병렬 처리기를 효율적으로 사용하기 위한 실시간 Load Balancing 기법과 Communication 알고리즘 연구
- VLSI를 기초로 한(예 : systolic, array processor) 모델에서의 hardware 알고리즘 연구 [23, 24] 등으로 요약할 수 있다.

물론 상위 레벨의 알고리즘 연구는 많은 어려움이 예상되지만 병렬처리기의 성패를 좌우하는 중요한 역할을 할 것임에 틀림없다. 또한 하위레벨의 병렬성 연구도 hardware의 기술과 새로운 구조적 요구의 출현으로 계속 그 폭이 넓어질 것이다.

5. 결 론

슈퍼 컴퓨터의 범주에 속하는 다양한 병렬 처리기에 대한 병렬 알고리즘이 지니는 기본적 의미와 특성 및 알고리즘 디자인시의 고려사항 등을 소개하였다. 또한 많은 병렬처리 모델에서의 개발된 알고리즘들을 간단하게 소개하고 앞으로의 연구분야를 정리해 보았다. 이러한 병렬알고리즘은 본지에서 소개된 하드웨어 기술이나 컴퓨터 구조에 관한 연구와

병행하여 연구되어져 그 현실성을 지니는 것이 중요하다 하겠다.

참 고 문 헌

[1] B. Awerbuch and etc al., "Finding Euler circuits in logarithmic parallel time," Proc. of the 16th Annu. ACM Symp. Theory Comp. pp. 249-257

[2] K.E.Batcher, "Sorting network and their applications," AFIP proc. 32, pp. 307-314

[3] R.G.Babb, Programming Parallel Processors, Addision-wesley, 1988

[4] J.L.Bentley, "A parallel algorithm for constructing minimum spanning trees," J. Algorithms 1(1), pp. 51-59.

[5] A. Chandram "Maximal parallelism in matrix multiplication," IBM Tech. Report RC 6193, Sep. 1976

[6] F.Y.Chin and etcal. "Efficient Parallel algorithms for Some Graph Problems," Comm. ACM 22 (9), pp. 659-665

[7] E.Dekel, D.Nassimi and S.Sahni, "Parallel matrix and graph algorithm," SIAM J.Comput., 10(4), pp. 657-675

[8] E.Dekel and S.Sahni, "A Parallel matching algorithm for convex bipartite graph and applications to scheduling," Jour. of Parallel and Distributed Compt.1, 1984, pp. 185-205

[9] N.Deo and Y.B.Yoo, "Parallel algorithms for the minimum spanning tree problem," Proc. of the 1981 ICPP, pp. 188-189

[10] D.M.Eckstein, "BFS and biconnectivity," TR 79-11, Dept, of Csci, Iowa state Univ. Ames, Iowa

[11] W.M.Gentleman, "Some complexity results for matrix computations on parallel processors," J. Assoc. Comput. Math., 25(1978), pp. 112-115

[12] W.D.Hillis and G.L.Steele, Jr., "Data Parallel algorithm," Comm. of the ACM, Dec. 1986, vol. 29, no. 12, pp. 170-1183

[13] D.S.Hirshberg, "Parallel algorithms for the transitive closure and the connected component problems," Proc. of the 8th Annu. ACM Symp. Theory Compt. pp. 55-57

[14] D.S.Hirschberg, "Fast parallel sorting algorithms," Commun. ACM 21, pp. 657-661

[15] D.S.Hirshberg, A.K. Chandra and D.V.Sarwate, "Computing connected components on parallel computers," Comm. ACM 22(8), pp. 461-464

[16] D.S.Hirshberg, "Parallel graph algorithms Without memory conflicts," Proc. of the 20th Allerton Conf., pp. 257-263

[17] D.S.Hirshberg and D.J. Volper, "A parallel Solution for the minimum spanning tree problem," Proc. of the 17th Annu. conf. Inf. Scc. Syst. pp. 680-884

[18] J.Ja'Ja' and J.simon, "parallel algorithms in graph thery : Planarity testing," SIAM J. Comput. 11(2) p. 314-328

[19] J.Jeng and S.Sahni, "All paris shortest paths on a hypercube multiprocessor," Proc. of ICPP. Aug. 1987

[20] A.H.Karp, "Programming for Parallelism," IEEE Computer, 1987. pp. 43-57

[21] D.J.Kuck, "Parallel Processing on ordinary Programs," Advances in Computers, 15, Academic Process, 1976, pp. 119-179

[22] L.Kucera, "Parallel computation and conflicts in memory access," Inf. Process. Lett. 14(2), pp. 93-96

[23] S.Y.Kung, "On Supercomputing with Systolic/wavefront array processors," Proc. IEEE, vol. 72, 1982, pp. 867-884

[24] S.Y.Kung, VLSI Array Processors, Prentice-Hall Inc, 1988

[25] S.C.Kwan and W.L.Ruzzo, "Adaptive parallel algorithms for finding minimum spanning trees," Proc. 184 ICPP, pp. 439-443

[26] S.Lakshmivarahan, S.K.Dhall and L.L.Miller, "Parallel Sorting Algorithms," IEEE Advanceds in Computers, vol. 23, 1984, pp. 295-354

[27] C.Lazou, Supercomputers and their use, Oxford Science Pub., 1986

[28] O.A.McBryan and E.F.Van Oe Velde, "Hypercube algorithms and implementations," SIAM J.Sci. Stat. Comp. vol.8, no.2, March 1987, pp. 227-287

[29] R.Mill, "Writing SIMD Algorithms," Proc. of the 1985 IEEE Int. Conf. on Computer Design : VLSI in Computers, pp. 122-125

[30] J.Mohan, "Experience with two parallel programs solving the salesman problem," proc. of the 1983 ICPP, pp. 191-193

[31] A. Moitra and S.S. Iyenger, "Parallel algorithms for a class of Computational problems a survey," Advances in Computers, June 1986

[32] D.Nassimi and S.Sahni, "Bitonic sort on a mesh connected parallel computer," IEEE Trans. on Compt. vol. C-27. no.1. 1979

[33] D.Nassimi and S.Sahni, "Finding connected components and connected ones on a Mesh -connected Parallel Computer," SIAM J. Comp., 9. 1980, pp. 744-757

[34] L.M.Li, T.C.King and P.Prins, "Parallel algorithm design considerations for hypercube multiprocessor," Proc. of the ICPP, 1987, pp. 717-720

[35] D.M.Nicol and J.H.Saltz, "Principles for problem aggregation and assignment in medium scale multiprocessors," ICASE TR 87-39, August 1987

[36] S.E.Orcutt, "Computer organization and algorithms for very high speed computations," Ph.D dissertation, Standford Univ. 1974.

[37] F.P.Preparata, "New parallel sorting schemes," IEEE trans. Comput. C-27. pp. 669 -673

[38] M.J.Quinn and N.Deo, "Parallel graph algorithms," ACM comput. Surv. 16(3), 1984, pp. 319 -348

[39] M.J.Quinn, Designing efficient algorithms for parallel computers, McGraw Hill, 1987

[40] C.Savage and J.Ja'Ja', "Fast, efficent parallel algorithms for some graph problems," SIAM J. Comput. 10(4), pp. 681-691

[41] S.R.Seidel and L.R.Ziegler, "Sorting on hypercubes," Proc. of the 2nd Conf. on Hypercube multiprocessors, Knoxville, Kentuckey

[42] H.S.Stone, "Parallel processing with perfect shuffle." IEEE Trans. Comput. C-20, pp. 153 -161

[43] Supercomputing Research Center, "Research and Technology Issues in Supercomputing," Univ. of Minnesota, Computer Science Dept. TR87-7

[44] R.E.Tarjan and U.Vishkin, "Finding biconnected components and computing tree functions in logarithmic parallel time," Proc. of the 25th Annu. Symp. Fundations of Comput. Sci. pp. 12 -20

[45] C.D.Thompson and H.T.Kung, "Sorting on a mesh connected parallel computer," Comm. ACM20, pp. 263-271

[46] C.D.Thompson, "the VLSI complexity of slorting," IEEE Trans. on Computers, vol. C-32, no. 12, Dec 1983, pp. 1171-1184

[47] Y.H.Tsin and F.Y.Chin, "Efficient parallel algorithms for a class of graph theoretic problems," SIAM J. Comput. 13(3), pp. 580-599

[48] L.G.Valiant, "Parallelism in comparison problems," SIAM J. comput. 4, pp. 348-355

[49] F.VanScoy, "Parallel algorithms in cellular spaces," Ph.D dissertation, Univ. of Virginia, 1976

[50] B.Wagar, "Hyperquick sort," Proc. of the 2nd conf. on Hypercube multiprocessors, Knoxville, Kentuckey

[51] Y.Won, S.Ranka and S.Sahni, "Programming a hypercube multicomputer," IEEE Software, Sep. 1988, pp. 69-77.

[52] Y.Won and S.Sahni, "A balanced bin sort for hypercube multicomputers," Jour. of supercomputing, 1988, pp. 435-448