□ 기술해설 □

고도 병렬 계산(Massively Parallel Computation)

고려대학교 정 창 성*

목 차

1. 서 론
2. 병렬계산 모델
3.병렬성 분석
4. 병렬성 응용
4.1 동적 영상 경계선의 coloring
4.2 실시간 Shading
4.3 Convex hull
5. 결 론

1. 서 론

최근 10년 동안 컴퓨터 과학 및 여러 응용 분야에서 방대한 양의 데이타에 대한 높은 계산 빈도의 계산 및 고속도 처리를 요구함에 따라 컴퓨터 기술은 성능, 가격, 그리고 안정도 면에서 눈부신 발전을 거듭하여 왔다. 한가지 특기할 사실은 속도 면에서 Mega FLOPS (floating point operations per second)에서 Giga FLOPS까지 수행할 수 있는 수퍼 컴퓨터의 개발이라고 할 수 있다. 수퍼 컴퓨터는 빠른 계산 속도와 많은 메모리 용량을 가지고 대량의 데이타를 고속도로 처리할 수 있는 컴퓨터로서 수퍼 컴퓨터의 필요성은 컴퓨터 과학 뿐만 아니라, 물리, 화학, 기계 등의 여러 분야에서 점점 증대되고 있다. 과거 컴퓨터의 개발이 학문의 여러 분야에서 혁신적인 발전을 가져온 것은 사실이지만 처리할 수 있는 물리적인 속도의 한계에 도달하여 더 이상의 진보를 유지할 수 있는 물리적인 속도의 한계에 도달하여 더 이상의 진보를 유지할 수 없었다. 따라서 대량의 데이타를 초고속으로 처리할 수 있는 수퍼 컴퓨터의 개발이야말로 이러한 문제를 풀 수 있는 돌파구를 제시했다고 할 수 있다[1].

수퍼 컴퓨터를 개발하는 데는 일반적으로 두 가지 방법이 있다. 첫번째는 하나의 프로세서를 사용하는 순차 컴퓨터 상에서 빠른 신소재 개발 및 파이프라인 기법 등을 이용하여 속도를 향상시키는 방법이다. 그러나 순차 컴퓨터 상에서의 성능 향상은 시그널의 전달속도에 의해 결정되므로 빛의 속도(3×10^8 m/sec)의 물리적인 한계에 의해 성능이 제한 된다. 즉, silicon에서의 시그널의 전달속도는 최대 3×10^8 m/sec이므로 silicon 기술에 의해 제작된 직경이 3 cm인 chip의 경우 시그널이 전달되는데 10^{-9} sec가 걸린다. 따라서 한번의 floating-point 계산을 하는데 한번의 시그널 전달만 필요하다해도 최대 성능은 10^9 FLOPS (1 Giga FLOPS)를 초과할 수가 없다. 이와 같이 순차 컴퓨터는 가까운 장래에 물리적인 최대속도의 한계에 도달할 것으로 예상되어, 고속계산을 위한 해결책이 되지 않으리라 판단된다. 두번째 방법은 순차처리와는 달리 하나의 프로세서대신 여러개의 프로세서를 사용하여 주어진 문제를 여러개의 작은 sub-task들로 분할하여 동시에 처리함으로써 시간을 단축시키는 병렬처리이다. 이는 순차처리와는

* 종신회원

달리 하나의 프로세서의 성능에 관계없이 프로세서의 숫자를 증가시켜서 전체의 성능을 무한대로 향상시킬 수 있는 가능성을 가지고 있다. 실제로 3차원 fluid flow 계산, 복잡한 시스템의 시뮬레이션 등과 같은 과학 및 공학분야에서 하나의 프로세서가 가질 수 있는 최대 성능보다 1000배 이상의 고성능을 요구하는 문제들이 많이 있기 때문에 이들 문제들에 대해서는 병렬처리가 유일한 해결책으로 병렬처리의 중요성이 점점 증대되고 있다. 그러나 프로세서수의 증가에 따른 cost증가와 프로세서간의 communication overhead에 의한 성능저하 때문에 이전에는 Grosch 법칙, Minsky 가정 및 Amdahl 법칙에서 보는 바와 같이 병렬처리에 대한 많은 이견이 존재해왔다[1].

1) Grosch 법칙 : 하나의 프로세서의 성능은 프로세서 cost의 제곱에 비례한다. 따라서, n개의 프로세서로 이루어진 병렬처리 시스템에서는 하나의 프로세서의 n배의 cost가 들지만, 하나의 프로세서로 같은 성능을 얻기 위해서는$\sqrt{n}$배의 cost가 들기 때문에 병렬처리 시스템을 만들 필요가 없다.

2) Minsky 가정 : Communication overhead 때문에 p개의 프로세서로 이루어진 병렬시스템의 성능은 대략 $\log_2$에 비례한다.

3) Amdahl 법칙 : 병렬 알고리즘의 성능향상은 병렬처리할 수 없는 부분의 비율에 따라 제약을 받는다. 즉, T_1과 T_p를 각각 하나의 프로세서와 p개의 프로세서를 사용했을 때 걸리는 시간이고, 순차 알고리즘 중 병렬화 될 수 없는 부분의 비율을 α라고 하면,

$$T_p = \alpha T_1 + \frac{(1-\alpha)T_1}{p}$$

가 된다. 따라서 아무리 많은 프로세서를 사용하더라도 p개의 프로세서를 사용했을 때의 속도증가 S_p는

$$\lim_{p\to\infty} S_p = \lim_{p\to\infty} \frac{T_1}{T_p} = \lim_{p\to\infty} \frac{1}{\frac{1}{p}+\left(1-\frac{1}{p}\right)\alpha} = \frac{1}{\alpha}$$

가 되므로 병렬 알고리즘의 성능향상은 병렬화 될 수 없는 비율 α에 제약을 받는다는 것이다.

그러나, 최근 VLSI 기술의 발달로 값싼 프로세서의 대량 제작이 가능하게 됨에 따라 같은 성능을 가진 순차 컴퓨터보다 cost-effective한 병렬 컴퓨터를 개발할 수 있게 되어, 더이상 Grosch 법칙은 성립되지 않는다. 또한 Minsky가 가정을 한 60년대 당시만해도 프로세서 갯수가 얼마 안되었고, 프로세서가 증가할 경우 overhead가 매우 높은 운영 체제를 사용했기 때문에 그와 같은 가정이 나올 수 있었다. 그러나, 최근에는 프로세서의 증가와 함께 job분배를 효율적으로 할 수 있는 운영체제의 개발과 효율적인 병렬 알고리즘이 개발됨에 따라 $\log_{2p}$보다 더 나은 성능을 보여주는 많은 병렬 프로그램이 개발되고 있기 때문에 communication overhead에 따른 성능저하는 Minsky의 가정보다는 훨씬 작다는 판단이 일반적이다. 많은 공학 및 과학의 문제들에서 Amdahl 법칙에서의 α는 문제크기 n에 따라 달라지는 함수 α(n)으로 나타낼 수 있고, 대부분의 경우 문제의 크기가 크므로 효율적인 병렬 알고리즘을 사용할 경우 $\lim_{n\to\infty}$가된다. 따라서 병렬처리를 할 경우의 속도증가는

$$\lim_{n\to\infty} S_p = \frac{p}{1+(p-1)\alpha(n)} = p$$

가 되어 Amdahl 법칙과는 달리 문제 크기가 클 때 효율적인 병렬 알고리즘에 의해 최적의 성능향상을 얻을 수 있다. 이와 같이 VLSI기술의 발달로 여러개의 프로세서를 수용할 수 있고 값싼 chip의 개발이 가능하게되어 이를 대량으로 사용한 cost/performance면에서 성능이 좋은 상업용 고도 병렬 컴퓨터가 등장하고, 순차 컴퓨터 상에서의 성능향상은 물리적인 빛의 속도 한계에 도달함에 따라 고속계산을 필요로 하는 분야에서의 병렬 처리는 매우 중요하고 필수적인 것으로 인식되고 있다. 특히 Minsky 가정이나 Amdahl 법칙의 제약에서 벗어나서 대량의 데이타에 대한 고속처리를 실현하기 위해서는 가능한 주어진 문제의 병렬성을 최대한도로 찾아내서 구현하는 고도 병렬 계산(massively parallel computation)이 매우 필요하고, 이를 위

해서는 주어진 문제에 대한 병렬성의 체계적인 분석 및 이를 바탕으로한 효율적인 병렬 알고리즘의 설계가 중요하다[2,4,5]. 본 논문에서는 병렬 컴퓨터 상에 효율적으로 구현될 수 있는 다양한 병렬성을 분석하고, 영상처리, 그래픽스, 계산기하의 간단한 문제들을 예로 들면서 병렬성을 추출해서 구현하는 방법에 대해서 기술한다.

2. 병렬계산 모델

병렬 알고리즘 설계시 주로 사용되는 대표적인 병렬구조모델은 제어방식 (control strategy)에 따라 하나의 control unit에 의해 동작되는 SIMD (Single Instruction Stream-Multiple Data Stream)와 각 프로세서가 자신의 control unit에 의해 동작되는 MIMD (Multiple Instruction Stream-Multiple Data Stream) 방식으로 나눌 수 있고, 다시 프로세서간의 연결방식에 따라서 여러 개의 프로세싱소자가 메모리를 공유하고 있는 SMM (Shared Memory Model)과 각 프로세싱 소자가 local memory를 가지고 일정한 패턴에 따라 연결된 INM (Interconnection Network Model)로 나누어진다[3]. SMM은 임의의 두개 PE사이의 데이타 교환이 O(1)에 수행될 수 있다고 가정한 이론적으로 가장 우수한 병렬계산모델로 read나 write의 access conflict에 따라 CRCW (Concurrent Read/Concurrent Write), CREW (Concurrent Read/Exclusive Write), ERCW (Exculsive Read/Concurrent Write), EREW (Exclusive Read/Exclusive Write)로 나누어진다. INM은 다시 연결망에 따라 세분되는데 대표적인 것으로 cube-class, mesh-class, tree-class가 있다. Cube connected, cube-connected cycle, PM-21, perfect shuffle 등이 cube class에, linear, ring, mesh, torus, Illiac-IV type이 mesh class에, tree, x-tree 등이 tree class에 속한다(그림 1 참조). 특히 linearly connected, mesh connected, ring connected, PM21 connected 와 같이 프로세서간에 linearly connection을 가진 병렬 computer를 linear class로 분류한다.

프로세싱소자의 갯수가 n개일 때 각 프로세싱소자는 0에서 n-1까지 index되어있고, PE[i]는 i의 index를 가진 프로세싱소자를 가리킨다. 프로세싱소자의 index를 이진법으로 나타내었을 때 cube-connected에서는 각 PE가 하나의 bit만 다른 index를 가진 모든 PE에 연결이 되어있다. Cube connected가 한 PE당 연결되는 link 숫자가 log n인데 비해 cube-connected cycle, perfect shuffle 등은 한 PE당 3개로 약간의 overhead는 있지만, cube-connected를 같은 order 내에서 simulation 할 수 있는 장점이 있다. Mesh는 각 PE가 이웃한 4개의 PE와 연결되어있고, boundary PE의 연결방식에 따라 torus, Illiac-IV type 등이 있고, 1차원 mesh의 형태가 linear, ring type이 된다. 각 프로세싱소자는

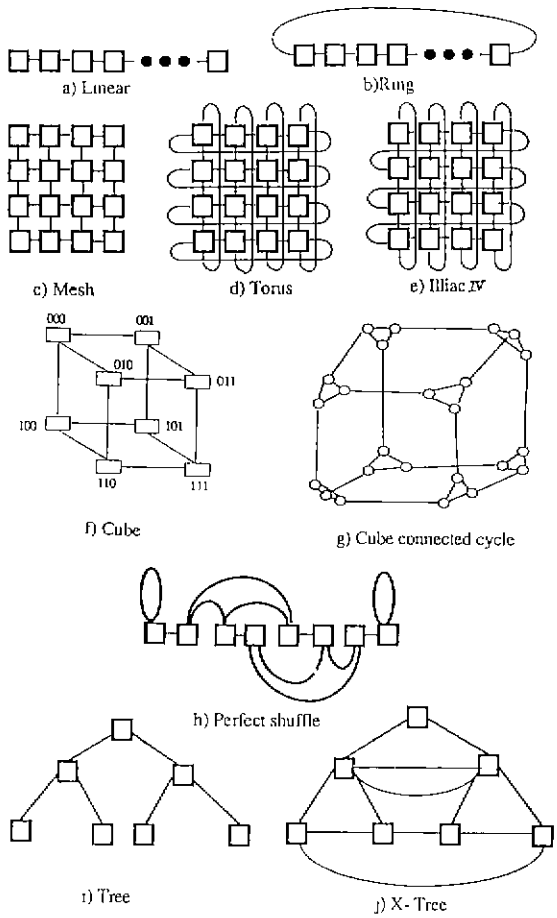


그림 1 INM 모델

하나의 arithmatic/logic 명령을 수행하거나 인접한 PE들과 데이타 교환은 O(1)에 수행될 수 있다고 가정한다.

일반적으로 대량의 프로세싱소자를 사용한 고도병렬컴퓨터에서는 fine grained parallelism을 사용하기 때문에 프로세싱소자간의 동기화가 하나의 제어장치에 의해 제어되는 SIMD방식을 많이 사용하고 프로세싱소자 (PE) 들간의 communication overhead를 줄이고 병렬성을 최대한도로 사용하기 위해 프로세싱소자들은 여러 크기의 block단위로 분할되어 subtask들을 동시에 수행하게 된다. 프로세싱소자들은 인접한 PE index를 갖는 left block과 right block의 두 block으로 나눌 수 있고, 다시 block들은 SIMD 또는 MIMD방식으로 동시에 subtask를 수행할 수 있다. 본 논문에서는 대량의 프로세싱소자를 사용하여 parallelism을 최대한도로 구현할 수 있는 고도병렬컴퓨터 상에서의 병렬 알고리즘을 다루고, 이론적인 SMM보다는 실용적인 INM 상에서 다양한 병렬성을 분석하도록한다.

3. 병렬성 분석

대량의 프로세서를 사용하여 속도를 증가시키기 위해서는 문제가 내재하고 있는 병렬성을 최대한도로 추출해서 이를 구현하는 것이 필요하다. 따라서 효율적인 병렬 알고리즘의 개발을 위해서는 주어진 문제의 병렬성을 분석하고 주어진 병렬성을 바탕으로 이에 적합한 병렬구조를 선택하여 병렬 알고리즘을 구현하게 된다. 본 절에서는 주어진 문제가 일반적으로 내재하고 있는 다양한 병렬성에 대해서 알아본다. 병렬 알고리즘 설계시 효율적으로 사용될 수 있는 병렬성은 크게 제어 병렬성(control parallelism), 데이타 병렬성(data parallelism), 알고리즘 병렬성(algorithmic parallelism) 으로 나눌 수 있다.

1) 제어 병렬성 : 제어병렬성은 수행하는 명령을 제어하는 방식에 따라 동시에 수행할 수 있는 기능을 제공하는 병렬성이다. 이는 명령을 제어하는 방식에 따라서 SIMD, MIMD, 파이프라인, 다중 흐름 병렬성으로 나누어진다. 주어진 task를 여러개의 프로세서 상에 작은 subtask로 분할하여 동시에 수행할 때 SIMD 병렬성은 모든 프로세서가 주어진 시간에 같은 명령을 수행할 수 있는 병렬성이고 MIMD 병렬성은 각 프로세서가 서로 다른 명령을 수행할 수 있는 병렬성이다. 대부분의 병렬처리시 SIMD, 또는 MIMD 병렬성 중 한가지 병렬성을 사용하거나 SIMD와 MIMD가 혼합된 병렬성을 사용한다. 파이프라인 병렬성에서는 연속된 task가 들어오는 경우 여러 개의 단계로 구성된 파이프라인을 통하여 단계별로 일련의 명령들을 연속적으로 처리함으로써 파이프라인의 각 단계가 모두 수행되는 경우 마지막 단계에서 계속적으로 결과값을 얻을 수 있기 때문에 속도를 증가시킬 수 있는 병렬성이다(그림 2 참조). 다중 흐름 병렬성은 파이프라인 병렬성과 같이 주어진 문제를 여러 단계로 분할이 곤란한 경우 시간에 따라 여러개의 stream으로 나누어서 round-robin방식으로 연속적으로 들어오는 task를 처리하는 병렬성이다(그림 3 참조).

2) 데이타 병렬성 : 데이타 병렬성은 데이타를 분할하여 여러개의 프로세서가 자신에 할당된 데이타를 동시에 처리할 수 있는 병렬성으로 크게 객체 병렬성(object parallelism)과 공간 병렬성(space parallelism)으로 분류할 수 있다.

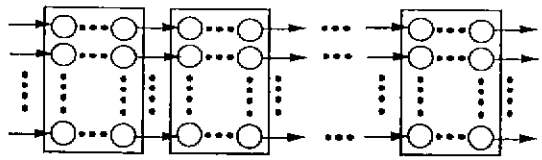


그림 2 파이프라인 병렬성

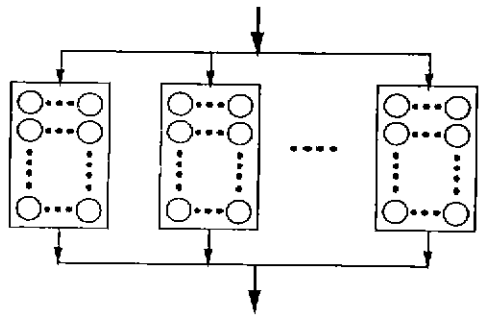


그림 3 다중 흐름 병렬성

객체 병렬성에서는 입력 데이타를 적당한 객체로 분류한 후 각각을 다른 프로세서에 할당하여 동시에 처리하는 병렬성이고, 공간 병렬성은 주어진 문제의 solution domain을 분할하여 분할된 domain을 각 프로세서에 할당하여 동시에 해를 구하는 병렬성이다.

3) 알고리즘 병렬성 : 순차 알고리즘은 일반적으로 병렬 컴퓨터상에 directly 구현될 수 없고 병렬 알고리즘은 순차 알고리즘과는 전혀 다른 형태를 가지게 된다. 즉 순차 알고리즘을 병렬 컴퓨터 상에 그대로 구현하는 경우 시간 복잡도는 순차 알고리즘과 같거나 더 많은 시간이 걸리게 된다. 주어진 문제의 병렬성은 순차 알고리즘 상에서 사용한 주요 성질에서 쉽게 찾을 수 없고 대부분 순차 알고리즘과는 달리 병렬 컴퓨터에 효율적으로 구현이 될 수 있는 새로운 algorithmic parallelism을 발견하는 것이 필요하다.

보통 주어진 문제의 병렬성은 위에서 설명한 병렬성 중 하나만 내재하고 있는 것이 아니라 여러 가지 병렬성이 복합된 hybrid parallelism의 형태를 갖고 있다. 따라서 병렬 알고리즘의 설계시 주어진 문제에서 여러가지 다양한 병렬성을 복합적으로 분석해 봄으로써 최대한도의 고도 병렬성을 추출해 낼 수가 있다.

4. 병렬성 응용

본 절에서는 많은 계산시간을 요구하는 영상처리, 그래픽스, 계산기하 분야의 간단한 문제를 예로 들면서 앞절에서 분석한 다양한 병렬성의 분석 및 구현에 대해서 설명하기로 한다.

4.1 동적 영상 경계선의 coloring

동적 영상 경계선 coloring은 동일한 배경에 대해 움직이는 간단한 물체가 있을 때 연속된 두 영상의 차이를 구해서 움직이는 방향에 따라 경계선의 color를 구하는 문제로서, 제어 병렬성의 파이프라인 병렬성과 데이타 병렬성의 공간 병렬성을 사용하여 효율적인 병렬 알고리즘을 얻을 수 있다[7].

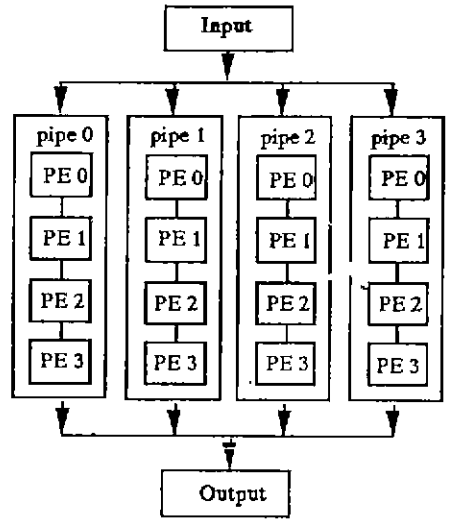


그림 4 동적 영상 경계선 coloring 병렬 계산 모델

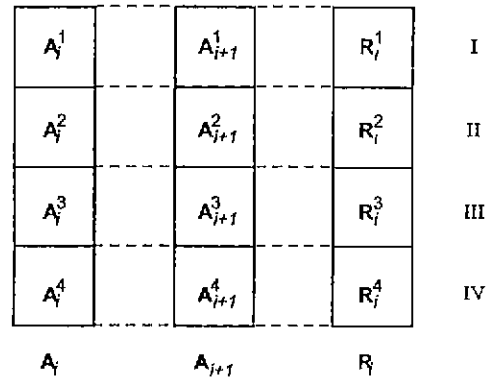


그림 5 각 pipe의 4 단계

그림 4는 동적 영상 경계선 coloring을 위한 병렬 계산 모델 예를 보여주고 있다. 4개의 프로세서로 구성된 4개의 pipe로 구성되어 있고, 각 pipe의 첫번째 프로세싱소자들은 영상을 받아들이는 입력 프로세서에, 마지막 프로세싱소자들은 coloring을 출력하는 출력 프로세서에 연결되어 있다. 입력 프로세스를 통하여 n×n 영상이 연속적으로 들어오고, 입력된 영상은 4개의 n×*n/4*영상으로 분할되어 네개의 *pipe 0*~3에 각각 입력되어 분할된 영상을 동시에 처리하는 공간 병렬성을 이용한다. 각 분할된 영상은 다시 파이프라인 병렬성을 이용하여 4단계의 파이프라인을 거치면서 바로 다음에 입력된 영

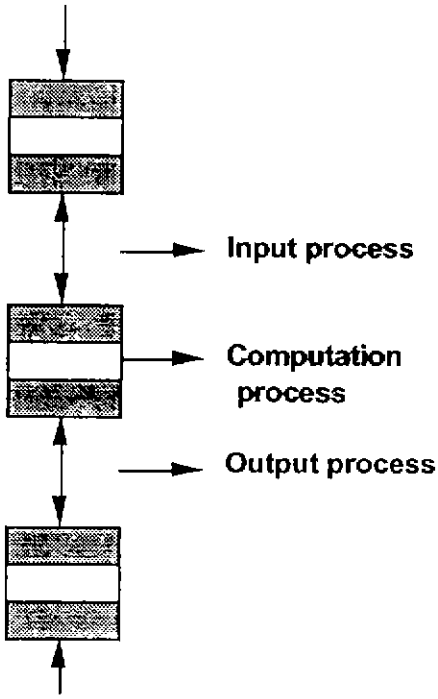


그림 6 Double buffering

상과의 차이를 구하게 된다. 그림 5와 같이 파이프라인의 각 단계에서는 n/4×n/4영상에 대한 두 영상의 차이를 구하게 되는데, 시간 t_i와 t_{i+1}에 연속적으로 각 pipe에 들어온 n×n/4영상은 각각 A_i, A_{i+1}이라 보고 A_j^1, A_j^2 A_j^3, A_j^4를 A_j가 다시 4개로 분할된 n/4×4/n 영상이라고 할 때, 각 pipe의 k번째 단계는 그림 6에서 보는 바와 같이 이중 버퍼링구조를 사용하여 k-1단계로부터 A_{i+2}^k를 입력하는 입력 프로세스, 현재 가지고 있는 A_i^k와 A_{i+1}^k의 영상차이 $R_i^k=A_i^k-A_{i+2}^k$를계산하는 computation 프로세스, k+1단계에 R_{i-2}^k와 A_{i+1}을 출력하는 출력 프로세스로 구성되어 있다. 따라서 각 단계는 영상차이를 구하는 computation 프로세서와 함께 입, 출력 프로세스를 동시에 수행함으로써 계산할 영상을 입력될 때까지 기다리지 않고 연속해서 n/4×n/4영상에 대한 차이를 계산해서 다음 단계로 넘기면, 출력 프로세서에서 4개의 n/4×n/4영상차이에 대한 최종결과를 받아서 출력한다. 그림 7은 시간에 따른 각 pipe의 단계별 computation 프로세스의 내용을 나타내고 있다. 이와 같이 동적 영상

stage \ time	t_1	t_2	t_3	t_4
I	$R_1^1=A_1-A_2$	$R_2^1=A_2-A_3$	$R_3^1=A_3-A_4$	$R_4^1=A_4-A_5$
II		$R_1^2=A_1-A_2$	$R_2^2=A_2-A_3$	$R_3^2=A_3-A_4$
III			$R_1^3=A_1-A_2$	$R_2^3=A_2-A_3$
IV				$R_1^4=A_1-A_2$

그림 7 시간에 따른 각 단계별 계산

경계선 coloring은 주어진 영상을 4개로 분할하여 4개의 pipe에서 각각 처리하는 공간 병렬성과 각 pipe를 4개의 단계로 나누고, 각 단계를 입, 출력 및 computation의 세개의 부단계로 나누는 파이프라인 병렬성을 이용하여 파이프라인이 일단 채워지면 빠른 속도로 원하는 경계선의 coloring을 구할 수 있다. (경계선의 color는 두 영상차를 가지고 쉽게 구할 수 있기 때문에 구체적인 수식은 여기서 생략한다).

4.2 실시간 Shading

본 절에서는 n개의 구가 존재하는 공간에 점광원이 일정한 궤적을 따라서 움직일 때, 실시간에 n개의 구를 shading하는 문제에 대해서 생각해보기로 한다. 광원의 위치가 주어졌을 때 구를 shading하기 위한 각 pixel intensity I는 간단한 다음 식에 의해 구해지는 것으로 가정한다.

$$I=I_ak_a+I_pk_d(N\cdot L)/R^2$$

이때 I_a와 I_p는 각각 점광원의 ambient intensity와 reflectance intensity이고, k_a와 k_d는 ambient reflectance와 directional reflectance 계수이다. N은 주어진 pixel의 normal vector이고, L과 R은 pixel과 점광원간의 vector와 거리를 각각 나타낸다. 실시간 shading문제는 다중 흐름 병렬성, 파이프라인 병렬성 및 객체 병렬성을 사용하여 효율적으로 다음과 같이 병렬처리할 수가 있다. 실시간 shading을 위한 병렬 계산 모델은 4개의 stream으로 나누어지고 각 stream은 4개의 프로세싱소자로 구성되어 있고, 각 stream에서 output을 받아서 이를 출력하는 output 프로세서로 구성되어 있다. Stream 1에

서는 주어진 시간 t에서의 점광원의 위치를 구한후, 그 위치에서의 shading을 계산하고, 마찬가지로 stream 2, 3, 4도 각각 t+Δt, t+2Δt, t+3Δt에서의 점과원의 위치를 구하고 shading을 계산한후 출력 프로세서를 통하여 stream 1에서 4까지의 결과를 차례로 출력한다. stream 1은 계산한 shading 출력이 끝나는 대로 t+4Δt에서의 shading을 계산하고, 마찬가지로 stream 2, 3, 4도 계산한 shading 출력이 끝나는 대로 t+5Δt, t+6Δt, t+7Δt의 shading을 계산한다. 즉 각 stream은 자신의 shading 계산을 마치고 출력이 끝나는 대로 3Δt 이후 shading을 구하는 출력하는 과정을 반복하게 된다. 그리고 각 stream은 4단계의 파이프라인으로 구성되어 각 단계는 n/4개의 구에 대해서만 shading하여 다음 단계로 출력하게 된다. Stream 1에서는 t, t+4Δt, t+8Δt,…에서의 shading결과가 파이프라인을 통하여 연속적으로 출력되고, 마찬가지로 stream 2에서는 t+Δt, t+5Δt, t+9Δt,…, stream 3에서는 t+2Δt, t+6Δt,…, stream 4에서는 t+3Δt, t+7Δt…에서의 shading결과를 연속적으로 출력하게 되어 t, t+Δt, t+2Δt, t+3Δt에서의 shading을 출력한후 t+4Δt, t+5Δt. t+6Δt, t+7Δt의 shading 및 이후의 shading도 시간 지체없이 실시간에 출력할 수 있다. 따라서 시간에 따라 shading을 분산하여 처리하는 다중 흐름 병렬성, 각 stream을 파이프라인으로 처리하는 파이프라인 병렬성, 각 파이프라인의 단계별로 구를 분할하는 객체 병렬성을 사용하여 shading을 고속도로 처리할 수 있다.

4.3 Convex hull

2차원 convex hull 문제는 평면상에 n개의 점의 집합 S가 주어졌을 때 이들 점들을 포함하는 최소 convex polygon의 꼭지점들을 구하는 문제로서 이를 n개의 프로세싱 소자 (PE)를 가진 mesh connected와 cube connected 병렬 컴퓨터상에서 구하는 병렬 알고리즘을 설계해 보기로 한다. 이때 효율적인 data routing 방법으로 알려진 selected data broadcasting 기법을 사용한다. Selected data broadcasting은 두개의

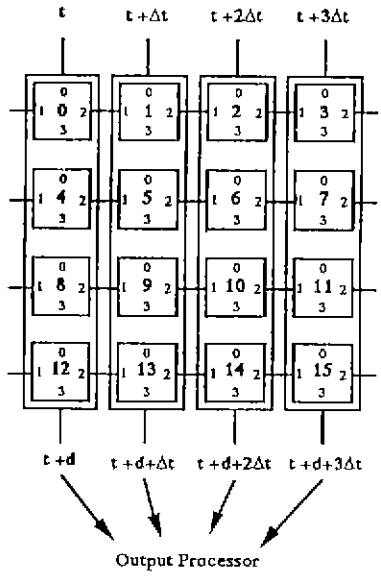


그림 8 실시간 shading 병렬 계산 모델

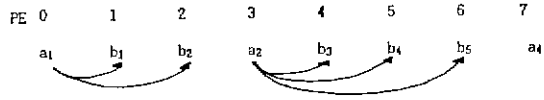
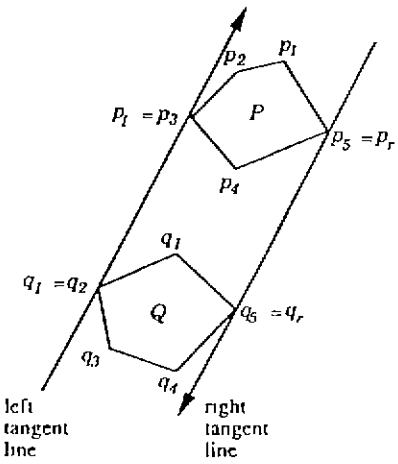


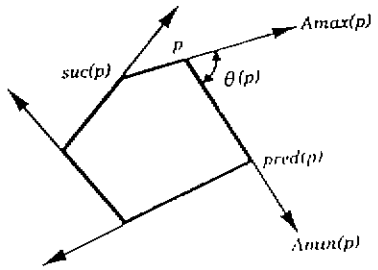
그림 9 Selected broadcasting

sorted list인 $A=(a_1, a_2, \cdots\cdots, a_p)$와 $B=(b_1, b_2, \cdots\cdots, b_q)$를 merge한 list C가 주어졌을 때 A의 각 element a_i를 $a_i \le b_j \le a_{i+1}$인 모든 b_j를 저장하는 PE에 보내는 operation으로 MCC에서 O(≈n), CCC에서 O(logn)이 걸린다[6]. C의 각 점은 각 프로세싱소자 하나에 들어 있다고 가정한다.

Convex hull을 구하는 방법은 많지만 순차 알고리즘과 병렬 알고리즘의 비교를 쉽게 하기 위해 기본적으로 divide-and-conquer방식을 사용한다. 순차 알고리즘에서는 divide-and-conquer방식을 사용하여, 우선 점들을 y value에 따라 sorting한 후, 점들을 수평선에 의해 분할된 두개의 집합 P와 Q로 분할한다(그림 10참조). 이때 P의 점들의 y value는 Q보다 크다. 각 P와 Q에 대해 convex hull CH(P)와 CH(Q)를 구한 후, 이들을 merge하여 최종 convex hull CH(S)를 구하게 된다. Merge process는 CH(P)와 CH(Q)간의 tangent line을 구한 후 내부 점들을



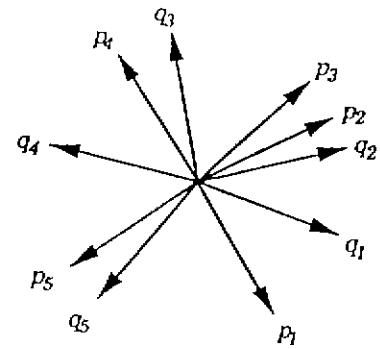
a) P and Q



b) θ(p)

그림 10 Convex P and Q

제거하며 구할 수 있다. 순차 알고리즘을 수행하는데 걸리는 시간은 $T(n)=2T(n/2)+M(n)$으로, M(n)은 merge하는데 걸리는 시간이다. Merge process에서 tangent line은 임의의 두점을 잇는 직선에서 시작하여 CH(P)와 CH(Q)의 꼭지점을 따라 traverse하면서 구하게 되므로 O(n)의 시간이 걸리고, 최종시간은 $T(n)=O(n\log n)$이 된다. 병렬 컴퓨터상에서는 각 PE에 1개의 점이 들어 있다고 가정하고, 우선 S를 예비단계로 sort하면 left block과 right block에서 각각 CH(P)와 CH(Q)를 동시에 recursive하게 구한 후 이들을 merge한다. 그러나, 순차 알고리즘에서의 merge방법을 그대로 사용하면 꼭지점을 traverse하기 위한 점들간의 communication overhead때문에 순차 알고리즘에서의 merge시간보다 더 많은 시간이 필요하게 된다. 따라서, 순차 알고리즘과는 달리 데이타들 간의 routing을 효율적으로 수행할 수 있는 새로운 algorithmic parallelism을 발견하는 것이 필요하다. 우선 tangent line을 찾기 위해 순차 알고리즘과 같이 점들을 traverse하는 대신 tangent line이 지날 수 있는 모든 가능한 점들이 쌍(pair) 집합 M을 병렬로 구할 수 있다면 그 중에서 tangent line을 쉽게 찾아낼 수 있으므로 M을 병렬로 계산할 수 있는 방법을 생각해 보도록 한다.



PE	0	1	2	3	4	5	6	7	8	9
v	q_2	p_2	p_3	q_3	p_4	q_4	p_5	q_5	p_1	q_1

$M_{p_1} = \{(p_1, q_1), (p_1, q_2)\}$ $M_{q_1} = \phi$
$M_{p_2} = \phi$ $M_{q_2} = \{(p_2, q_2), (p_3, q_2)\}$
$M_{p_3} = \{(p_3, q_3)\}$ $M_{q_3} = \{(p_4, q_3)\}$
$M_{p_4} = \{(p_4, q_4)\}$ $M_{q_4} = \{(p_5, q_4)\}$
$M_{p_5} = \{(p_5, q_5)\}$ $M_{q_5} = \{(p_1, q_5)\}$

그림 11 M 계산

Convex polygon 각 점 p에서 tangent line이 놓일 수 있는 범위는 그림 10의 (b)에서 보는 바와 같이 counterclockwise 방향으로 점을 배열할 때의 이전의 점 pred(p)와 이후의 점 suc(p)와 이루는 각 Amin(p)와 Amax(p)사이의 범위 θ(p)가 된다. 따라서 각각 P와 Q에 속한 임의의 두점 p, q에 대해 $\theta(p)\cap\theta(q)$이고 p, q를 잇는 직선이 $\theta(p)\cap\theta(q)$사이에 놓이면 p, q간에 tangent line이 존재하는 것을 쉽게 알 수 있다. 따라서 $\theta(p)\cap\theta(q)\not\equiv 0$인 모든 (p, q)의 쌍을 구할 수 있다면 그 중에서 tangent line이 지나는 두 점 p, q를 구할 수 있게 된다.

$\theta(p)\cap\theta(q)\not\equiv 0$인 모든 (p, q)의 쌍을 병렬로 구하는 방법은 그림 10의 (b)에 대한 예를 들면서 그림 9에서 설명하기로 한다. 우선 각 PE에

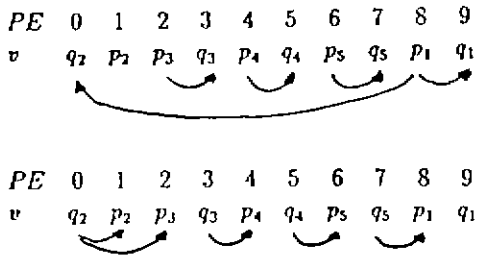


그림 12 Data routing

들어 있는 p와 q의 꼭지점은 Amin에 따라 그림 11과 같이 sort한 후 P의 각 점 p_i에 대해 p_i와 suc(p_i)사이에 있는 Q의 점들과의 쌍의 집합 M_{p_i}를 구하고 Q의 각 점 q_i에 대해 q_i와 suc(q_i) 사이에 있는 P의 점들과의 쌍의 집합 M_{q_i}를 구한다면 $\theta(p) \cap \theta[(q)] \neq 0$인 모든 쌍을 구할 수 있다는 것을 쉽게 알 수 있다. 따라서, merge step 이전에 이미 sort되어 있는 P와 Q에 있는 점을 merge한 후 그림 10과 같이 P의 각 점 p를 p와 suc(p)사이에 있는 Q의 점들을 가지고 있는 PE에 보내고, 마찬가지로 Q의 각 점 q를 q와 suc(q)사이에 있는 P의 점들을 가지로 있는 PE에 보낼 수 있으며 tangent line이 지날 수 있는 쌍의 집합을 구할 수 있는데 이는 바로 앞에서 설명한 selected broadcasting방법과 일치하므로 이를 이용하여 직접 구할 수 있다. 따라서, merge시간 M(n)은 MCC에서 $O(\sqrt{n})$, CCC에서 O(log n)으로 전체적으로 걸리는 시간은 $T(n)=T(n/2)+M(n)$이므로 MCC에서는 $O(\sqrt{n})$, CCC에서는 $O(\log^2 n)$이 걸리게 된다. 이와 같이 효율적인 routing기법을 이용하여 tangent line이 지날 수 있는 (p, q)의 쌍을 병렬로 하는 새로운 algorithmic parallelism의 발견이 효율적인 병렬 알고리즘을 개발하는데 필수적이다.

5.결 론

순차 컴퓨터의 성능제약에 따라 대량의 데이타의 고속처리를 필요로하는 분야에서의 병렬처리는 필수적인 것으로 인식되고 있다. VLSI 기술의 발달로 performance/cost의 성능이 좋은 상업용 컴퓨터가 등장하게 되면서, 병렬처리의 큰 걸림돌이 되고 있는 communication overhead 및 Amdahl 법칙의 제약에서 벗어나기 위해서는 효율적인 병렬 알고리즘의 개발이 매우 중요하고, 이를 위해 주어진 문제가 내재하고 있는 최대한도의 병렬성을 추출하고 구현하는 고도 병렬 계산 기법의 개발이 필요하다. 본 논문에서는 이를 위해 제어 병렬성, 데이타 병렬성, 알고리즘 병렬성 등의 다양한 병렬성 등에 대해 알아보았으며, 실제로 주어진 문제의 병렬성은 응용 예에서 보는 바와 같이 대체로 여러 병렬성이 결합된 복합 병렬성을 내재하고 있다. 동적 영상의 경계선 coloring에서는 데이타 병렬성의 공간 병렬성과 제어 병렬성의 파이프라인 병렬성을 이용하였고 실시간 shading에서는 제어 병렬성의 다중 흐름 병렬성, 파이프라인 병렬성과 데이타 병렬성의 객체 병렬성을 사용하였으며, convex hull 계산에서는 알고리즘 병렬성을 사용하여 효율적인 병렬 알고리즘을 계산하였다. 대부분의 문제에서는 순차 알고리즘과는 다른 알고리즘 병렬성을 발견해내는 것이 필수적이나, 이는 application specific하기 때문에 일반적인 알고리즘 병렬성을 단적으로 분석하기는 매우 어렵다. 그러나, convex hull 예제에서 보는 바와 같이 주어진 병렬구조에서 프로세서간의 communication overhead를 최대한도로 줄이는 기존의 효율적인 routing 방법을 사용할 수 있는 병렬성질을 찾아내는데 주의를 기울이면 효율적인 병렬 알고리즘의 설계에 도움이 되리라고 생각한다.

참고문헌

[1] A. L. Decegama, "The Technology of Parallel Processing," Prentice Hall, 1989.

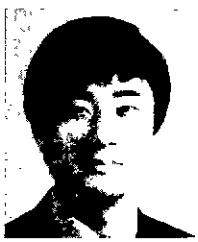
[2] S. G. Akl, "The Design and Analysis of Parallel Algorithms," Prentice Hall, 1989.

[3] H. J. Siegel, "A Model of SIMD Machines and a Comparison of Various Interconnection Networks," IEEE Trans. Comput. C-28, 1979, pp. 907~917.

[4] F. T. Leighton, "Intro. to Parallel Algorithms and Architectures," Morgan Kaugmann Publi-

sher, 1992.
[5] T. L. Freemann and C. Philips, "Parallel Numerical Algorithms," Prentice Hall, 1992.
[6] C. S. Jeong, and J. J. Choi. "Parallel Enclosing Rectangle on SIMD machines," Parallel Computing, 1992, pp. 221~229.
[7] 정창성 외 4인, "고도 병렬 컴퓨터 구조 : TIME," ADD 연구보고서, 1992.

정 창 성



1981 서울대학교 전기과 졸업
1985 Northwestern University M.S.
1987 Northwestern University Ph.D.
1987 ~1992 포항대학교 전자공학과 조교수
1992 ~현재 고려대학교 전자공학과 부교수
1992 ~현재 Editorical Review Board, Parallel algorithms and applications
관심 분야 : 병렬처리