공학해석을 위한 병렬처리용 컴퓨터의 응용

하 재 선*

1. 병렬컴퓨터란 무엇인가

병렬컴퓨터를 이용한 병렬처리의 궁극적인 목적은 계산시간(wall clock time)의 단축에 있다. 그러면 왜 병렬컴퓨터을 이용하는 것이 계산시간을 단축할 수 있는 지름길인지 알아본다.

1.1 병렬컴퓨터의 필요성

인간이 발견한 최대의 속도는 빛의 속도이다. 빛의 속도는 1 nanosecond당 30cm로서 이는 인간이 구현할 수 있는 컴퓨터 신호처리의 가장 빠른 속도일 것이다. 그러나, 이것은 광섬유를 통한 광속도로서 컴퓨터 패키징에 어려움이 많다. 현재 컴퓨터내부에 사용되는 신호처리의 전달용 매체는 대부분 구리선(copper wire)으로 그것을 통한 최대 전달속도는 1 nanosecond당 9cm에 불가하다. 이러한 성능을 보여주는 단일 프로세서는 매우 비싸다. 따라서 이러한 단일 프로세서에 의존한 기존의 순차식(serial) 컴퓨터로서는 그 성능의 한계가 있다. 현재 가장 빠른 순차 컴퓨터의 성능은 9GigaFLOPS를 보여주는 것으로 알려져 있다. 이러한 한계를 극복하는 유일한 선택이 병렬컴퓨터를 이용한 병렬처리(parallel processing)다.

병렬처리란 커다란 작업량을 작은 작업량들로

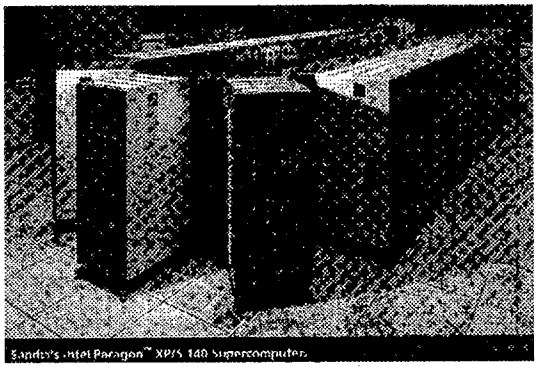


그림 1 Intel Paragon

* 대한항공 항공기술연구원

분해(decomposition)하여 여러 작업자(worker)들에게 분담을 시켜 동시(simultaneously)에 그 작업을 수행하여 소기의 결과를 성취하는 것이다. 그러한 작업자들을 병렬프로세서(parallel processor)라 칭하는데 일반적으로 중앙처리장치(central processing unit)들의 집합체로 볼 수 있다. 프로세서와 데이터의 기억장치(memory)를 합쳐 병렬요소(parallel element) 또는 병렬절점(parallel node)으로 부른다. 따라서 다수의 병렬요소들이 장착된 컴퓨터를 병렬컴퓨터라 하며, 이러한 컴퓨터를 응용하는 것을 병렬계산(parallel computing)이라 한다.

이해를 돕기 위해 다음의 예를 보자. 아주 건장한 사람이 100평에 달하는 논의 모내기를 하는데 10시간이 걸린다고 하자. 그러면 그 사람의 능력은 10평당 1시간에 모내기를 하는 것이다. 만일에 덜 건장한 사람들 10명이 10평씩 분담한다면 1시간을 약간 초과할 것이다. 물론 덜 건장한 사람들과 건장한 사람간의 능력비교에 따라 2시간 또는 3시간 등이 걸릴 수 있으나, 결론적으로 여러 사람들이 분담하면 자연히 비교적 빠른 시간에 작업량을 끝낼 수 있다. 만일에 1시간에 끝낸다면 이는 이상적인 결과이고, 이것이 병렬처리의 궁극적인 목적이다. 논의 크기가 클수록 작업분담이 절실하듯이, 작업량(work domain)이 클수록 병렬처리의 응용이 절실한 것은 당연하다.

현재 병렬컴퓨터들의 일부는 TeraFLOPS까지의 성능을 보이고 있다. 이렇게 빠른 처리를 요구하는 분야들이 많은데, 이들로서는 기후모델링(climate modeling), 유체의 난류해석(fluid turbulence), 오염확산(pollution dispersion), 해양순환(ocean circulation), 연소체계(combustion system), 구조의 충돌해석(crash dynamicsSimulation), quantum chromodynamics, semiconductor&superconductor modeling 등이 있으며, 앞으로 전 분야에 파급이 예상된다.

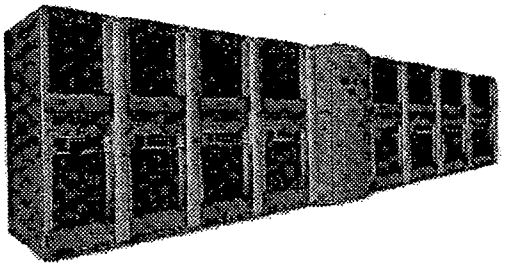


그림 2 SGI Origin 2000

1.2 병렬컴퓨터의 역사

전자식 컴퓨터의 발달과정을 살피고 특히 병렬컴퓨터들의 진화과정을 알아본다.

1.2.1 컴퓨터의 발달사

지난 50년 동안 컴퓨터는 인간의 요구에 부합하여 급속도로 발달하였다. 1950년대에 처음으로 진공튜브(vacuum tube)와 절연선을 이용한 초기 컴퓨터들이 만들어 졌다. 대표적인 컴퓨터들로, Pennsylvania 대학의 ENIAC(Electronic Numerical Integrator and Calculator), Princeton 대학의 IAS, IBM의 IBM 701 등을 들 수 있다. IAS는 Von Neumann이 제안한 설계 개념을 바탕으로 만들어졌다. 이러한 컴퓨터들의 중앙처리장치 CPU는 Fixed-Point Arithmetic(고정 소숫

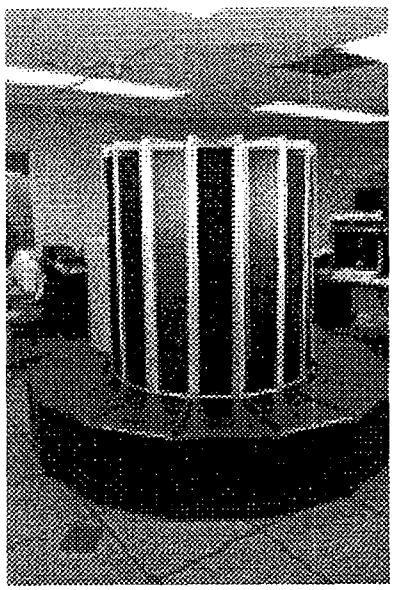


그림 3 Cray 1

점 연산)에 의한 연산이 가능하였다. 참고로 병렬컴퓨터가 아닌 보편적인(conventional) 컴퓨터를 Von Neumann 순차 컴퓨터(serial computer)라 한다. 현재 우리가 쓰고있는 개인용 컴퓨터(personal computer)가 대표적인 Von Neumann 순차 컴퓨터다. 1960년대에 들어서 트랜지스터의 기술이 나타나고, floating-point arithmetic(부동소숫점 연산)에 의한 연산이 도입되었다. 이때에 printed circuit에 의한 설계가 시작된다. 또한 FORTRAN , COBOL, ALGOL 등의 고급프로그래밍 언어들이 나타난다. 대표적인 컴퓨터들로 IBM 7090, CDC 1604, UNIVAC LARC 등이 있다. 1970년대에 들어서 Integrated Circuits(ICs)의 기술이 나타나고, multiuser, multiprogramming, time shartng OS 등의 소프트웨어의 기술이 등장한다. 또한 pipelining, cache memory 등의 기술이 도입된다. IBM 360/370, CDC 6600, PDP-8, TI-ASC 등의 대표적인 컴퓨터들이 있었다. CDC 6600는 9MegaFLOPS, CDC 7600은 40MegaFLOPS였는데, 이는 현재의 퍼스널컴퓨터들(100~300 MegaFLOPS)보다 느렸다. 1976년 Cray사는 Cray 1을 발표하였는데, 이 컴퓨터의 사양은 64 MBytes memory, 80Mhz, 130MegaFLOPS 였다. 1970년대 후반기 이후로, LSI(Large Scale Integration)/VLSI(Very Large Scale Integration)에 의한 설계, 반도체의 출현, Vector Supercomputer 등의 기술이 나왔는데, 이러한 기술에 의한 대표적인 컴퓨터들로 VAX 9000, Cray X-MP, IBM 3090, BBN TC2000 등이 있다. 이 시기에 병렬컴퓨터들이 서서히 출현하였는 데, 1968년에 최초의 병렬컴퓨터 Illiac IV가 개발된다. 1990년대에 접어들어 Vector Supercomputer에서 Parallel Supercomputer로 전환이 된다. ULSI(Ultra Large Scale Integration)/VHSIC processors들이 쓰이고, 병렬처리컴퓨터들의 전성기를 맞는다. 대표적인 컴퓨터들로서 CM-5, IBM SP1/2, Cray T3E, Intel Paragon 등이 있다.

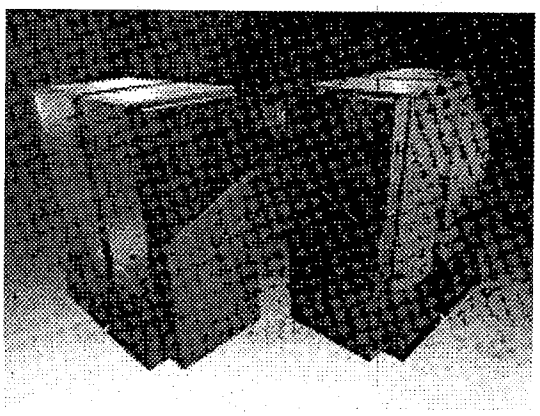


그림 4 Cray Y-MP8E

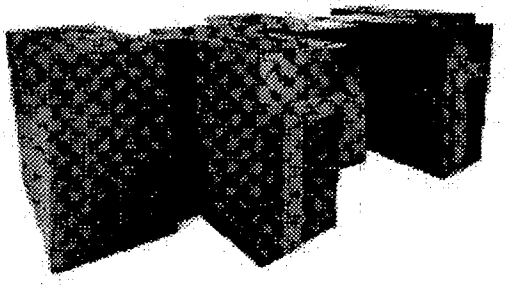


그림 5 Cray T3E

1. 2. 2 병렬컴퓨터의 발달사

병렬컴퓨터는 구조적인 차이에 따라 다양하게 분류되는데, 그 분류에 따라 빠른 속도로 연구

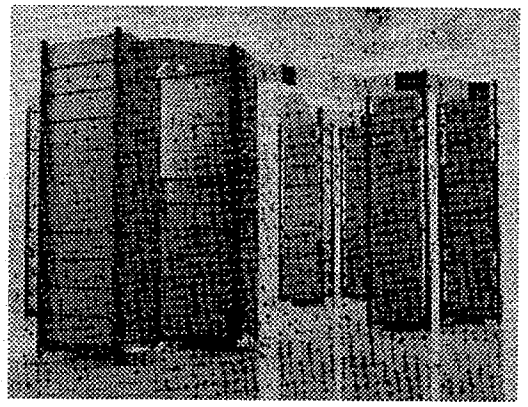


그림 6 Kendall Square KSR 1

개발되어 진화하고 있다. 최초의 병렬컴퓨터라 할 수 있는 Illiac IV는 1968년 미국 일리노이 대학에서 개발되었다. 제어장치가 하나이고 64개의 연산장치들이 2D mesh로 연결된 구조로 이루어졌다. Illiac IV는 SIMD(Single Instruction stream Multiple Data stream) 컴퓨터의 시초로 알려져 있다. 병렬요소들이 이차원 mesh로 연결되었다. 그후 SIMD에 속하는 병렬컴퓨터들은 1980년에 개발된 Goodyear MPP를 거쳐, BSP (1982), DAP 610(1980), CM-2(1990), MasPar (1990)로 이어진다. SIMD에 유사한 구조로 이루어지는 Vector Supercomputer들은 1970년에 개발된 CDC 7600을 시작으로 Cray 1(1978), CDC Cyber 205(1982), Cray Y-MP(1989), Cray MPP(1993) 등이 개발되어 왔다. 중앙처리장치, 즉 CPU가 여러 개로 설치되는 MIMD(Multiple Instruction stream Multiple Data stream) 컴퓨터는 다시 기억장치의 공유 여부에 따라 공유기억 다중프로세서(Shared Memory Multiprocessors)와 분산기억 다중컴퓨터(Distributed Memory Multicomputers)로 분류된다. 공유기억 다중프로세서는 CMU C.mmp(1972)에서 시작하여 NYU Ultracomputer(1983), Illinois Cedar(1987), BBN Butterfly(1989), KSR 1(1990), Stanford Dash (1992), Fujitsu VPP500(1992), SGI Origin 2000로 이어진다. 이에 반해 메시지 패씽에 의해 정보교환을 수행하는 분산기억 다중컴퓨터들은 Cosmic Cube(1981)를 기점으로 Intel iPSC(1983), nCube(1990), Mosaic(1992), Intel Paragon (1992), IBM SP2로 이어진다. Hypercube 연결방식으로 된 병렬컴퓨터들로서 Cosmic Cube가 최초로 개발되었으며, 그후 128개의 CPU들로 장착된 iPSC/1(8 MegaFLOPS), iPSC/2(27MegaFLOPS), iPSC/860(8 GigaFLOPS)로 이어진다. 가상공유기억 병렬컴퓨터(Virtual Shared Memory Parallel Computer)의 대표적인 병렬컴퓨터로서 Cray T3D를 들 수 있는데, 그 특징으로 DEC Alpha chips, 2048 프로세서들, Torroidal mesh, 307GigaFLOPS 등이 있으며, 현재 Cray T3E로 진화되었다. RS/6000 또는 Power PC CPU들로 장착된 IBM SP2는 한 개의 프로세서(또는 CPU)당 250MegaFLOPS의 성능을 보이며, 수백 개의 프로세서들이 장착 및 확장될 수 있다.

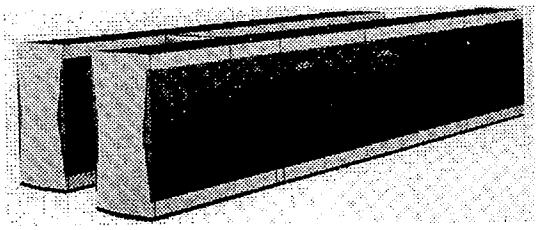


그림 7 Fujitsu VPP500

그림 8 MasPar

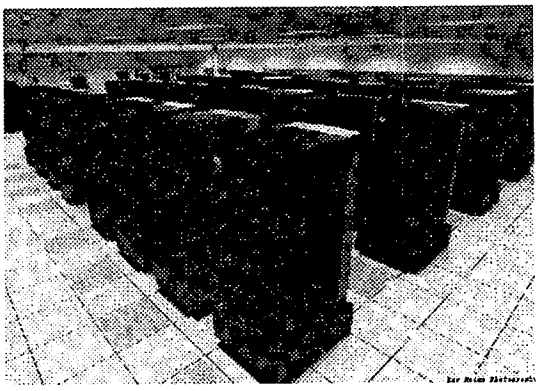


그림 9 IBM SP2

2. 병렬컴퓨터의 분류

병렬컴퓨터는 여러 기준에 따라 분류되고 있다. 프로그램을 작성함에 있어 이러한 분류에 대

한 이해는 최대의 성능(performance)을 얻기 위해 필수적이다. 다음은 이러한 분류에 대한 설명이 주어진다.

2.1 명령체계에 따른 분류

Flynn(1972)은 대부분의 컴퓨터들을 제어장치(control unit)와 데이타의 흐름에 따라 분류하는 방식을 발표하였다. 그가 분류한 방식은 네 가지로 SISD(Single Instruction Stream Single Data Stream), SIMD(Single Instruction stream /Multiple Data stream), MISD(Multiple Instruction stream/Single Data stream), MIMD (Multiple Instruction stream/Multiple Data stream)로 분류하며, 이 방식이 병렬세계(Parallel community)에서 가장 널리 통용되고 있다.

2.1.1 SISD(Single Instruction Stream Single Data Stream)

일반적인 컴퓨터들은 기본적인 세가지 장치들로 구성된다. 이들은 제어장치(control unit), 연산장치(arithmetic unit), 그리고 기억장치(random access memory unit)로 이루어진다. 이러한 장치들로 이루어진 보편적(convensional)인 컴퓨터를 병렬컴퓨터(parallel computer)에 반하여, Von Neumann 식의 순차 컴퓨터(serial computer)라고 칭한다. 이러한 컴퓨터들로써 IBM-compatible PC, Apple Mac, Sun SPARC Workstation 등이 그 예다.

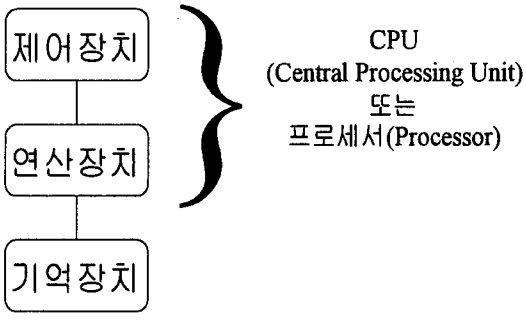


그림 10 SISD
(Single Instruction Stream Single Data Stream)

2.1.2 SIMD(Single Instruction stream/Multiple Data stream)

제어장치는 하나이나 연산장치가 여러 개로 구성된 컴퓨터가 여기에 속한다. 제어장치에서 명령이 보내지면 여러 연산장치들이 같은 명령(instruction)을 동기적으로(syncronously) 수행한다.

이러한 컴퓨터는 프로그램 작업이 수월하고 연산속도가 빠르나, 같은 작업이 연속적으로 많을 때 작업속도가 매우 비효율적으로 떨어진다. 이러한 컴퓨터들로서는 Liliac IV, Cray-2, Cray YMP, CM-2, CM-200 등이 있다.

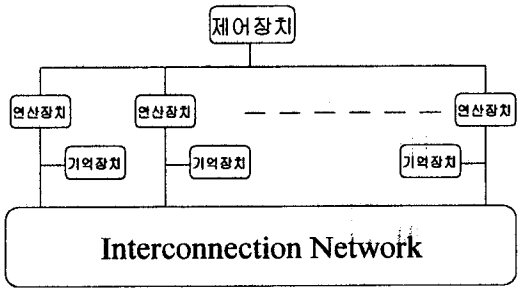


그림 11 SIMD
(Single Instruction Stream/Multiple Data stream)

2.1.3 MISD(Multiple Instruction stream/Single Data stream)

여러개의 제어장치들과 하나의 연산장치로 구성된 컴퓨터로, 실용적으로 비효율적이어서 이러한 컴퓨터는 존재하지 않다.

2.1.4 MIMD(Multiple Instruction stream/Multiple Data stream)

제어장치와 연산장치로 이루어진 마이크로프로세서를 중앙처리장치(CPU) 또는 프로세서(processor)라고 하는데, 이러한 프로세서들이 여러 개로 이루어진 컴퓨터들이 MIMD에 속한다. SIMD 컴퓨터의 프로세서들이 동기적으로 작동된다면, MIMD 컴퓨터는 프로세서들이 비동기적(asyncronously)으로 작동한다. 현재의 대부분 슈퍼컴퓨터들이 이러한 부류에 속한다. 프로그램

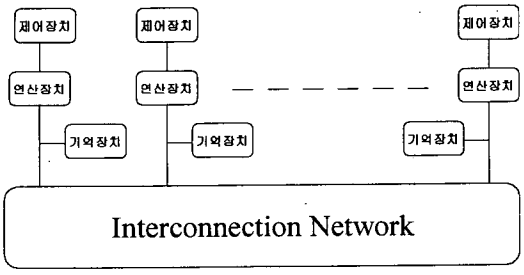


그림 12 MIMD
(Multiple Instruction Stream/Multiple Data Stream)

작성에 어려움이 있으나, 가장 효율적인 구조로 이해되고 있으며, 기억장치의 공유 여부에 따라 두가지로 다시 분류되는데, 다중프로세서 Multiprocessors)로 알려지는 Tightly Coupled Machines은 공유기억장치(Shared Memory Computer)를 갖고 있으며, 다중컴퓨터(Multicomputers)로 알려지는 Loosely Coupled Machines은 분산기억장치(Distributed Memory Computer)를 갖고 있다. 이러한 컴퓨터의 예로, 다중프로세서에 Encore, Sequent, Multimax 등이 있고, 다중컴퓨터에는 nCUBE, Intel iPSC, Cray T3E, CM-5 등이 있다.

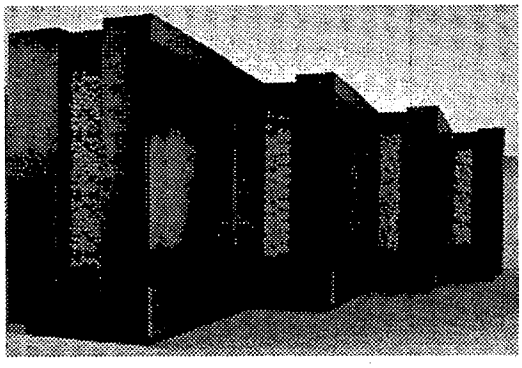


그림 13 Thinking Machines CM-5

2.2 Network에 따른 분류

프로세서들의 연결방식에 따른 분류로서, 다중프로세서들 또는 다중컴퓨터들의 성능에 결정적으로 작용되는 컴퓨터의 구조적인 분류이다. 대표적으로 완전연결방식(fully connected), mesh, ring, hypercube, fat tree 등이 있다.

2.2.1 완전연결

모든 프로세서들이 일대일로 연결된 가장 강력한 연결방식(interconnection network topology)이나, 프로세서들의 수에 기술적으로 제한이 있어, 작은 수의 프로세서들로 구성된 컴퓨터에 쓰인다.

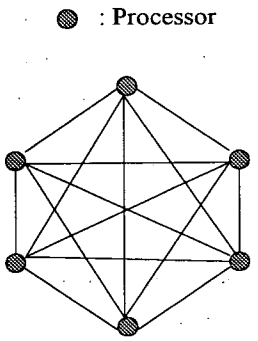


그림 14 완전연결방식

2.2.2 Mesh(Torus)

프로세서들을 격자로 배열하는 방식으로 오직 이웃장치들 간에만 정보교환이 이루어진다. 프로세서들의 수는 격자(lattice)의 차원(k)과 격자의 간격(width, w)에 의해 w^k으로 표현된다. 이러한 연결방식은 Intel Paragon, Intel Delta, Cray T3D, Cray T3E 등에서 보인다.

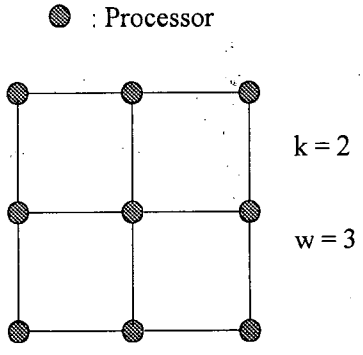


그림 15 Mesh 연결방식

2. 2. 3 Rings

Ring형식으로 연결되어, 양쪽 끝단의 프로세서들이 연결되어 있다. 일반적으로 선형배열의 단순 링(simple ring)과 각기 반대편과 활줄교환링크(chordal communication link)로 연결된 코달링(chordal ring)방식 등이 있다. KSR 1, KSR 2 등의 컴퓨터들이 이러한 연결방식을 택하였다,

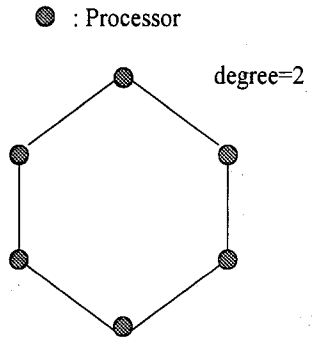


그림 16 Ring 연결방식

2. 2. 4 Hypercube

Hypercube형식으로 프로세서들이 연결된 방식으로, k차원에서 프로세서의 수는 2^k으로 결정된다. 이러한 연결방식은 Intel iPSC/860, nCUBE, SGI Origin 2000 등에서 보인다.

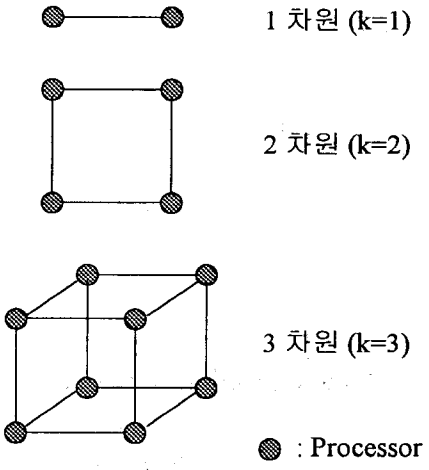


그림 17 Hypercube 연결방식

2. 3 기억장치에 따른 분류

SIMD와 MIMD는 다시 기억장치의 형태에 따라 두 가지로 크게 분류된다. 한 개의 커다란 기억장치를 버스(Bus)에 연결되어 여러 프로세서들이 공유하는 컴퓨터를 공유기억 다중프로세서(Shared Memory Multiprocessors)라 한다. 각기의 프로세서들은 별도로 공유되지 않는 작은 국부 기억장치(local memory)에 연결할 수 있다. 다른 한편으로, 공유 기억장치가 없이 각 프로세서들이 각기 별도의 국부기억장치에 연결된 컴퓨터를 분산기억 다중컴퓨터(Distributed Memory Multicomputers)라 한다.

2. 3. 1 공유기억 다중프로세서

다중연산장치들이 한 개의 공유기억장치(Global Shared Memory)에 연결되어 서로간에 정보교환을 가능하게 하는 병렬컴퓨터를 공유기억 다중프로세서라 한다. 공유기억장치와 프로세서들 간에는 버스를 통하여 연결되어, 버스의 밴드위드스(bandwidth-초당 정보교환이 가능한 정보양)에 한정된 정보 양이 교환되기에 일정 수 이상의 프로세서들만이 정보교환이 가능하다. 또한 한 프로세서가 정보를 교환하는 동안 다른 프로세서들이 그 정보를 얻으려 할 경우에는 그 프로

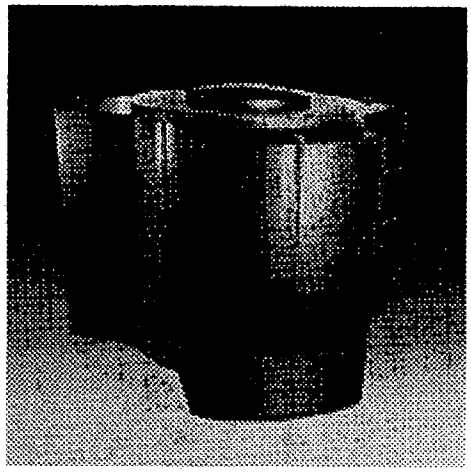


그림 18 Cray C-90

세서가 정보를 얻을 때까지 기다려야 하는데 이를 non-determinace라 한다. 따라서 이러한 병렬컴퓨터는 한정된 수의 프로세서들이 장착되어야 하기 때문에, 컴퓨터의 성능확장에 제한이 있으나, 일정수의 프로세서들이 적용될 경우에 매우 효율이 높은 성능을 얻을 수 있으며, 병렬프로그램의 작성이 용이하다. 30개 이상의 프로세서들의 장착이 어려워 일반적으로 선형증가성(scalability)의 성취가 어렵다. 따라서 테라플롭스(TeraFLOPs)에 달하는 성능을 얻기가 어렵다. 이러한 컴퓨터들로 Alliant, MasPar, AMT/DAP, CM-2, UNC, Convex, BBN, Encore, Sequent, Cray C-90, SGI Power Challenge, SGI Origin 2000, SGI Onyx, IBM SMP 등이 있다.

2.3.2 분산기억장치

다중프로세서들이 각기 자신의 국부기억장치들을 소유하여, 프로세서들간의 정보교환은 서로 연결된 내부 연결조직(interconnection network)을 통하여, 메시지패씽(message passing)으로 이루어진다. 상당수의 프로세서들이 연결방식에 따라 확장이 가능하나, 내부 연결조직의 형태와 그 성능에 직결되는 밴드위드스(bandwidth)와 레이턴스(latency)에 따라 이러한 병렬컴퓨터의 성능에 크게 좌우된다. 이러한 컴퓨터들로는 IBM SP2, nCUBE, Teradata 등이 있다.

3. 병렬처리를 위한 언어

병렬컴퓨터에서 병렬처리가 이루어지게 하는 소프트웨어는 병렬 프로그래밍 패러다임의 개념에 따라 여러 가지로 나타나는데, 여기에는 메시지 패씽 패러다임(Message Passing Paradigm), 데이터 병렬패러다임(Data Parallel Paradigm), Shared Memory Parallel Paradigm 등이 있다.

3.1 메시지 패씽 패러다임

가장 많이 쓰이는 패러다임으로서, 수많은 언어들이 개발되어 왔다. 그러나 MPI(Message Passing Interface)이전의 언어들은 병렬컴퓨터에 크게 의존하여 이동성(portability)이 없고, 표준화가 되어 있지 안았다. 이들로서, PVM, Zipcode, Parmacs, APPL, P4, Chamelon, NX, MPL, CMMD 등이 있다.

3.1.1 PVM(Parallel Virtual Machine)

PVM은 같은(homogeneous) 또는 다른 heterogeneous) 컴퓨터들을 가상적으로 하나의 병렬컴퓨터로 쓰는 것을 가능하게 하는 병렬처리용 언어다. 이것은 1989년 미국 Oak Ridge 국립연구소에서 처음으로 작성되어 가장 널리 쓰여진 message passing library이다. PVM 이전의 병렬처리용 언어들은 병렬컴퓨터에 크게 의존하여, 다른 병렬컴퓨터에 장착이 불가능하였다. 이러한

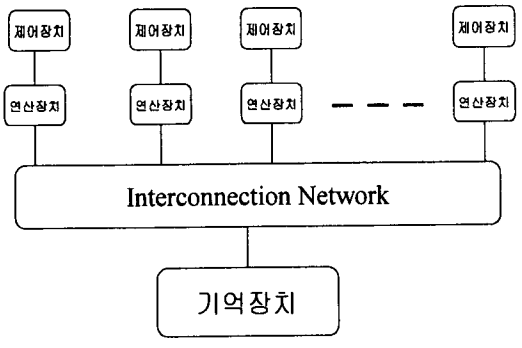


그림 19 공유기억컴퓨터
(Shared Memory Multiple Processors)

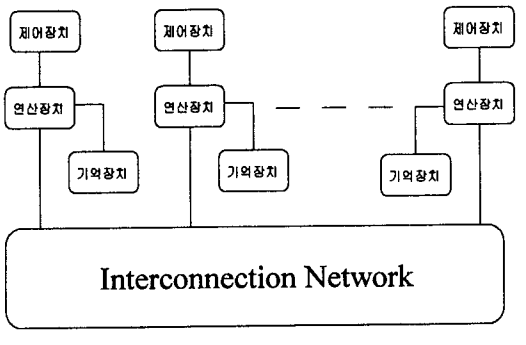


그림 20 분산기억컴퓨터
(Distributed Memory Multicomputers)

기능을 portability라고 하는데, PVM의 가장 큰 장점은 가장 용이한 portability이다. 또한 설치가 용이하고, 공개적으로 사용이 가능(Public Domain Software)하여 가장 널리 사용되어 온 Message Passing Library이나, 결정적인 단점으로 Network의 느린 속도와 제한된 Message Passing Functionality를 들 수 있다.

3.1.2 MPI(Message Passing Interface)

PVM에서 발견된 단점을 최대한 보완하여, Message Passing Library의 표준화로 개발된 것이 MPI(Message Passing Interface)이다. MPI의 개발 목적은 Message Passing 프로그램을 위하여, 표준화(Standard), 실용성(Praticability), 이식성(Portability), 효율성(Efficiency), 풍부한 기능성(Functionality) 등에 있다. MPI는 1990대에 들어오면서, 병렬처리용 언어의 표준화에 대한 필요성이 절실히 부각이 되어왔다. 1992년 4월에 미국 Virginia의 Williamsburg에서 분산 기억 다중컴퓨터에서 쓰이는 Message Passing 프로그램의 표준화를 위한 workshop에서 MPI의 출현이 시작된다. MPI는 주로 PVM을 비롯한 Express(ParaSoft), Zipcode(Caltech), Parmacs(Argonne 국립연구소), APPL(NASA Lewis), P4(Argonne 국립연구소), Chamelon(Argonne 국립연구소), NX(Intel), MPL(IBM), CMMD(Thinking Machine) 등과 같은 여러 병렬언어들에 뿌리를 두고 있다. 같은 해 11월 PVM의 축적된 기술을 바탕으로 Oak Ridge 국립연구소에서 MPI-1이 발표되고, 이때 학교, 컴퓨터회사, 연구소등지로부터 175명의 전문가들이 MPI Forum을 형성하여, 1994년 공식적인 일차 판(first version)이 발표되었다. MPI는 115개 이상의 루틴들을 갖고 있으며, 가장 강력한 성능과 이식성, 표준성으로 인하여, 현재 그리고 앞으로 가장 널리 쓰일 대표적인 병렬처리용 언어이다.

3.1.3 기타 언어들

앞에서 언급한 여러 언어들이 하드웨어의 개발자에 따라 개발되어 왔다. Intel이 개발한 iPSC series, paragon 등과 같은 하이퍼큐브(hipercube) 연결방식의 NX를 발표했으며, TMC(Thinking Machine Co.)는 CM-5에서 작동하는 CMMD를 발표하였다. IBM은 독자적인 MPL(Message Passing Library)을 IBM SP series을 위해 개발하였다. 이러한 언어들은 MPI로 통합하게 된다.

3.2 공유기억 패러다임

예일대학에서 개발된 Linda는 Tuple Space라는 가상 공유기억장치(Virtual Shared Memory System)를 이용한다. Tuple Space는 프로세서간의 정보교환과 싱크로나이제이션을 가능하게 해준다. 동질 또는 이질의 Unix 웍스테이션들의 네트웍을 가상 공유기억장치로 이용하기에 linda는 공유 기억 다중프로세서나 분산 기억 다중컴퓨터 모두를 플랫폼(platform)으로 가능하다. Cray T3D/T3E, DEC Alpha, Hitachi SR2201, HP 9000, IBM SP2, IBM RS/6000, SGI Onyx, Sun SPARCS 등 기종에 관계없이 공유기억 프로그래밍 패러다임으로 쓰인다. Linda는 Linda-C와 Linda-fortran으로 나누어진다. 그 외에 KSR 병렬 프로그래밍언어와 같이 특별한 이름이 없이 특정한 공유 기억 다중프로세서에서 운용되는 언어들은 많이 있어 왔다.

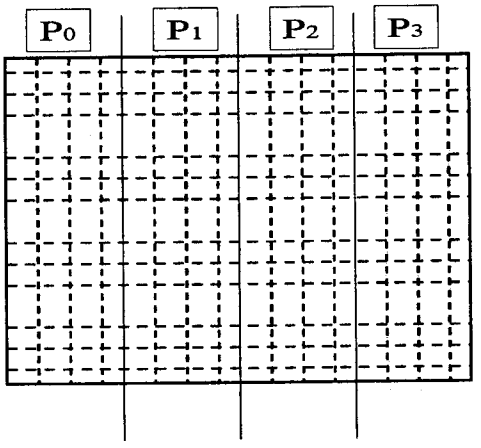


그림 21 Columnwise Block Striping

3.3 데이터 병렬 패러다임

주로 SIMD 병렬컴퓨터에서 운용되는 대표적인 언어들로 HPF(High Performance Fortran)가 있다. FORTRAN은 과학 및 공학에 가장 널리 쓰이고 있는 프로그래밍 고급 언어다. 1954년 IBM에서 John Backus의 주도하에 개발된 FORTRAN은 FORTRAN II(1958), FORTRAN IV (1966), FORTRAN 77(1978), FORTRAN 90 (1992), FORTRAN 95(1997)로 지속적으로 진화되어 왔다. 1991년 미국 Albuquerque에서 열린 Supercomputing 91은 HPF를 위한 최초의 HPF Forum이었다. 이 당시 HPF는 데이터 병렬 프로그래밍 언어들인 Thinking Machines사의 CM FORTRAN, Syracuse 대학 Fox와 Rice 대학 Kennedy에 의한 FORTRAN D, 그리고 Vienna 대학에서 개발된 Vienna FORTRAN 등을 그 모체로 갖고 있다. HPF는 Fortran 90을 2차원으로 확장시킨 데이터 병렬 프로그래밍 언어다. FORTRAN 90은 데이터 병렬 프로그래밍 개념에 근접하여 개발되었지만 병렬컴퓨터를 위한 언어는 아니다. 그 밖에 SplitC, Forge90 등이 있다.

4. 병렬컴퓨터의 성능에 대한 이해

병렬컴퓨터의 성능은 플롭스(FLOPS)의 크기로 표현되며, 병렬알고리즘의 성능은 스피드업(speed-up)과 효율(efficiency)로 표기한다.

4.1 FLOPS

플롭스는 일초당 부동 소숫점(floating point)의 연산 수를 말하는 것으로, 부동 소숫점의 연산이란 곱하기, 더하기, 빼기, 나누기를 말한다. 그러므로 1MegaFLOPS는 일초에 백만번의 더하기 또는 곱하기 등을 수행할 수 있는 성능을 말한다. 또한 100MHz는 일초당 100×106번 싸이클(Cycle)이 가능한 성능으로, 그 처리장치가 한 싸이클에 두 번의 부동 소숫점의 연산이 가능하다면, 200MegaFLOPS가 된다.

4.2 Speed-Up

스피드업이란 n개의 프로세서들이 쓰였을 때, 한 개 사용했을 때에 비하여 얼마나 빠른지를 나타내는 병렬처리 프로그램의 성능의 변수로 가장 널리 쓰인다. 한 개를 사용했을 때 걸리는 Wall Clock Time을 T_1로 표기하고, n개를 사용했을 때 걸리는 Wall Clock Time을 T_p로 표기한다면, 그 때의 스피드업은 $S_p=T_1/T_p$로 계산된다. 이러한 계산은 병렬컴퓨터에서 병렬처리 프로그램으로 얻은 방식으로 이를 알고리즘 스피드업(Algorithmic Speed-Up)이라 한다. 만일에 T1을 순차컴퓨터에서 순차처리 프로그램을 이용한 가장 빠른 시간인 경우에 이렇게 얻은 수치를 병렬 스피드업(Parallel Speed-Up)이라 한다.

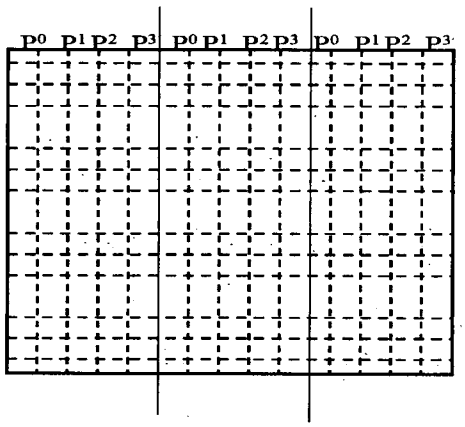


그림 22 Columnwise Cyclic Striping

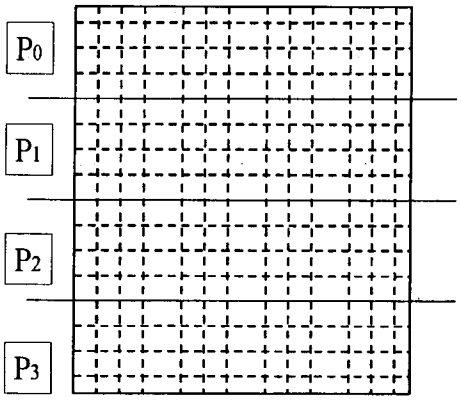


그림 23 Rowwise Block Striping

4.3 병렬효율

병렬프로그램이 얼마나 효율적인 가를 나타내는 수치로 n개의 프로세서들이 사용된 경우에 그 효율은 $E_p=S_p/n$으로 표기된다. 따라서 n개의 프로세서들을 사용하여 n만큼의 스피드업을 얻었다면 효율은 100%가 된다.

5. 병렬처리 프로그램을 위한 전략

병렬처리 알고리즘을 세움에 있어 가장 중요한 것이 프로세서들간의 Load Balancing과 Communication Minimization이다.

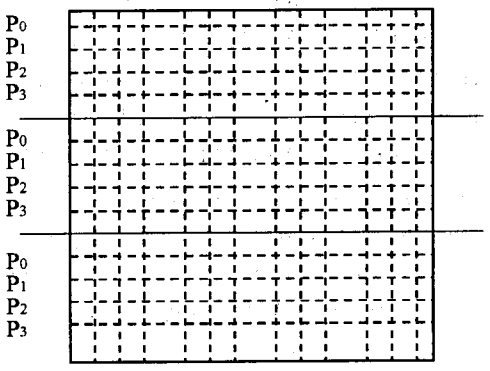


그림 24 Rowwise Cyclic Striping

5.1 Load Balancing

병렬알고리즘의 성능을 나타내는 가장 대표적인 값이 Speed-Up이다. 그런데 높은 값의 Speed-Up에 제한을 주는 요인 중의 하나가 가장 느린 프로세서의 속도다. 만일 어떤 프로세서가 작업량이 많아 Syncronization Point에서 작업을 계속 수행할 때, 이미 작업을 마친 다른 프로세서들은 기다려야 하는 데, 이러한 현상을 Bottleneck이라 한다. 따라서 전체적인 성능은 크게 떨어지는 데, 이러한 현상은 프로세서들에게 작업량을 균일하게 분배하지 못한 데서 비롯된다. 작업량을 균일하게 분배하는 것을 Load Balancing이라 한다. Load Balancing은 앞서 Domain Decomposition에 의해 이루어진다.

5.2 Communication Minimization

Load Balancing과 더불어 Speed-Up에 가장 영향을 미치는 요인이 프로세서들간에 communication이다. communication에 참여하는 프로세서들은 그 시간에 계산을 할 수 없다. 따라서 communication을 가장 최소화하도록 병렬처리 알고리즘을 세워야 한다.

5.3 Matrix Decomposition

매트릭스 연산에서 각 프로세서들에게 部매트릭스(submatrix)의 할당(mapping)은 병렬처리에서 필수이다. 따라서 부매트릭스의 산출 전략은 매트릭스 연산에 앞서 반드시 필요하다. 이러한 작업을 Partitioning이라 한다. Partitioning에는 대체로 네가지를 들 수 있은 데, Columnwise/Rowwise Block Striping, Columnwise/Rowwise Cyclic Striping, Block Checkerboard Partitioning, Cyclic Checkerboard Partitioning 등이 있다.

5.4 Domain Decomposition

유한요소법(Finite Element Method)에 의한 구조해석에 병렬처리의 적용은 직관적이다. 선형해석을 고려할 때, 각 요소의 강성계수 매트릭스(Elemental Stiffness Matrix)는 요소의 위치에 상관없이 산출되기에 각 요소에 할당(mapping)된 프로세서들이 동시에(simultaneously) 계산할 수 있다. 그런데 계산된 각 요소의 강성계수 매트릭스들은 전체 강성계수 매트릭스(Global Stiffness Matrix)을 얻기위해 합산(assembly)되어야 하는데, 여기서 프로세서들간에 communication이 요구된다. 따라서 프로세서들간에 communication의 최소화와 균일한 Load Balancing을 얻기 위해 Domain Decomposition이 전제로 주어진다.

대표적인 Domain Decomposition 방법으로 Greedy Method, RSB(Recursive Spectral Bisection) Method, RCB(Recursive Coordinate Bisection) Method, Maladroit Method 등이 있다.

6. 수치해석에서 병렬처리

본 장에서는 대형 선형방정식에 적용한 병렬처리 알고리즘을 간략히 소개하고 그 결과를 토론

한다. 대형 선형방정식은 매트릭스 형태로 Ax=b로 표현된다. 여기서 매트릭스 A와 매트릭스 b가 주어지는데, 매트릭스 A는 n×n 계수들(coefficients)로 이루어지고, b는 n×1로 구성된다. 이때 변수 매트릭스 x는 일반적으로 두가지 방법에 의해 해를 찾는데, 첫째는 매트릭스 A의 계수(Off diagonal coefficients)들이 영(zero)값을 갖는 sparse 매트릭스인 경우에는 반복방법(Iterative method)이 사용된다. 이러한 방법으로 가장 기본적인 Jacobi's Iterative Method, 그리고 구조해석의 유한요소법의 코드들이 가장 널리 쓰는 Preconditioned Conjugate Gradient Method 등이 있다. 이에 반해 매트릭스 A가 작은 규모이고 Non-Sparse인 경우에 직접제거방법(Directive Elimination Method)가 쓰인다. 이러한 대표적인 알고리즘으로 Gauss's Elimination Method가 있으며, 그 파생된 이론들로 Doolittle Method, Crout Method, 그리고 유한요소법에서 가장 널리 쓰이는 Cholesky Method가 있다.

$$\mathbf{Ax} = \mathbf{b} \qquad \textit{Equation}$$

$$\mathbf{x} = \mathbf{x}^{o} \qquad \textit{Initialization}$$

Iteration k

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} + (\mathbf{b} - \mathbf{A}\mathbf{x}^{k}) / \mathbf{A}_{\mathbf{d}}$$

Tolerance checking

$$\mathbf{x}^{k+1} = \mathbf{x}^{k}$$

$$\left\| \mathbf{x}^{k+1} - \mathbf{x}^{k} \right\| \le \varepsilon \qquad \text{Tolerance}$$

그림 25 Iacobi iteration algorithm

6.1 Jacobi's Iteration Method

이 방법은 가장 이상적인 병렬처리 알고리즘이다. 반복 횟수 i+1에서 새로운 값 x^{i+1}는 이전 단계, 즉 반복 횟수 i에서의 값 x^{i}에만 의존한다. 완전한 양함수(explicit function)형태로서 병렬로 계산이 가능하다. 병렬처리 알고리즘은 다음과 같다.

p개의 프로세서들이 쓰인다고 하자. 매트릭스 A, 매트릭스 b, 그리고 x^{i}를 열방향으로 p개로 분할한다. 그러면, 매트릭스 A의 p개 서브매트릭스들, 즉 A_1, A_2, ..., A_k,..., A_p들은 n/p×n 매트릭스가 된다. 마찬가지로 매트릭스 b도 p개의 서브매트릭스들, 즉 b_1, b_2, ..., b_k, ..., b_p들은 n/p×1 매트릭스가 된다. 따라서 각기 프로세서들은 각기 할당(mapping)된 서브매트릭스에 대한 jacobi 알고리즘을 수행한다. 예로 k번째 프로세서는 $x_k^{i+1} = x_k^{i} + (b_k - A_k x^{i}) / A_D$를 수행한다. 여기서 벡터 x^{i}는 MPI-ALLGATHER를 이용하여 x_k^{i}로부터 구한다. i번째 반복에서, 각 프로세서는 norm을 계산한다. 이 단계에서 syncronization point가 필요하다. 즉 모든 프로세서들은 각기 계산된 Norm 값은 MPI-ALLREDUCE에 의한 總合을 구한다. 주어진 허용치(tolerance)에 만족이 안되면 다시 위의 과정을 계속 반복한다. 최종적으로 허용치에 적합한 Norm이 확보되면, MPI-GATHER에 의해 주프로세서(primary processor)가 최종 해를 산출한다.

위 그림은 병렬컴퓨터 IBM SP2와 MPI를 사용하여 Speedup Diagram을 얻었다. 사용된 프로세서들의 수는 40개이고, 매트릭스 A의 크기는 8000×8000이다. 옆 그림에서 보여주듯 40개의 프로세서들을 사용하였을 때 Speedup 값은 34.4였다. 반복횟수는 6이였다. 한개의 프로세서가 해를 얻는 데 172초가 소요된 반면, 40개의 프로

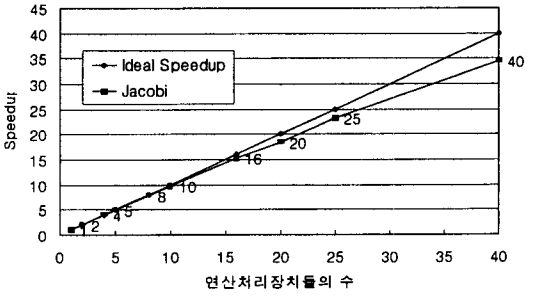


그림 26 Jacobi Speed-Up Diagram

세서들이 5초가 소요된다. 그러므로 이 알고리즘의 효율(efficiency)은 86%가 된다.

$$\mathbf{Ax} = \mathbf{b} \qquad \textit{Equation}$$

$$\mathbf{p}^o = \mathbf{r}^o = \mathbf{b} - \mathbf{Ax}^o \qquad \textit{Initialization}$$

Iteration k

$$\mathbf{u}^k = \mathbf{Ap}^k$$

$$\alpha^k = \frac{\mathbf{r}^k \cdot \mathbf{r}^k}{\mathbf{p}^k \cdot \mathbf{u}^k}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{p}^k \qquad \textit{Tolerance checking}$$

$$\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha^k \mathbf{u}^k$$

$$\beta^k = \frac{\mathbf{r}^{k+1} \cdot \mathbf{r}^{k+1}}{\mathbf{r}^k \cdot \mathbf{r}^k}$$

$$\mathbf{p}^{k+1} = \mathbf{r}^{k+1} + \beta^k \mathbf{p}^k$$

그림 27 Conjugate Gradient Speed-Up Diagram

6.2 Conjugate Gradient Method

Conjugate Gradient Method은 반복방법의 하나로 적은 반복횟수에 얻고자하는 해에 가장 빨리 수렴하기에 가장 널리 쓰이는 알고리즘이다. 앞서 Jacobi 병렬 알고리즘에서와 같이 매트릭스들, p, r, b, u 등을 n/p로 블록분할이 가능하다. 각 프로세서들에게 할당(mapping)하여, 병렬로 매트릭스 계산을 수행한다. 여기서 벡터들, p&u는 MPI-ALLGATHER에 의한 collective communication이 요구된다. 마찬가지로 MPI-ALLREDUCE에 의한 norm의 합산이 필요하고, 최종적으로 MPI-GATHER에 의한 최종 해를 구한다.

위 그림은 병렬컴퓨터 IBM SP2와 MPI를 사용하여 Speedup Diagram을 얻었다. 사용된 프로세서들의 수는 40개이고, 앞의 Jacobi Algorithm에 사용된 같은 크기 매트릭스 A와 b를 사용하였다. 옆 그림에서 보여주듯 40개의 프로세서들을 사용하였을 때 Speedup 값은 33.1였다. 반복횟수는 3이였다. 한 개의 프로세서가 해를 얻는데 86.1초가 소요된 반면, 40개의 프로세서들이 3초가 소요된다. 그러므로 이 알고리즘의 효율(efficiency)은 82.7%가 된다.

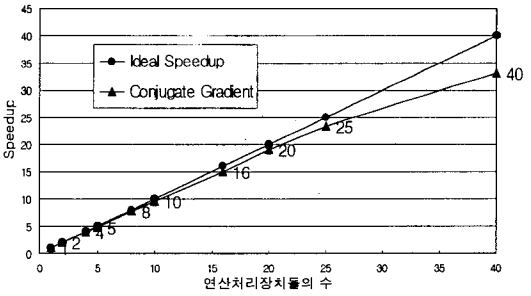


그림 28 Conjugate Gradient Speedup Diagram

6.3 Choleski's Decomposition Method

반복방법에 반한 방법으로서, 직접적으로 해를 구하는 방법을 직접방법(Direct Method)이라 한다. 가장 대표적인 방법이 Gauss Elimination Method이다. 여기서 파생된 직접방법들로서, Doolittle Method, Crout Methoid, Choleski's Decomposition Method 등이 있다.

매트릭스 A는 대칭성에 상관없이 하부매트릭스(lower matrix) L과 상부매트릭스(upper matrix) U로 분할이 된다. 하부매트릭스(lower matrix) L의 대각성분들이 1값을 갖게 하는 알고리즘을 Doolittle Method라 하고, 상부매트릭스(lower matrix) U의 대각성분들이 1값을 갖게 하는 알고리즘을 Crout Method라 한다. 만일에 매트릭스 A가 대칭인 경우 1값이 아닌 대각성분값을 갖고 비대각성분의 값들은 영 값을 갖는 또

$$U^*_{ij} = K_{ij} - \sum_{k=\max(r_i, r_j)}^{i-1} U_{ki} U^*_{kj}$$

$$U_{ij} = U^*_{ij} / D_{ij} \qquad \text{for } r_{ij} \le i < j;\ j = 2,\ldots,n$$

$$D_{ij} = K_{ij} - \sum_{k=r_j}^{j-1} U_{ki} U^*_{kj} \qquad \text{for } j = 1,\ldots,n$$

그림 29 Choleski Decomposition Speed-Up Diagram

다른 매트릭스 D를 얻고, 하부매트릭스(lower matrix) L와 부매트릭스(lower matrix) U를 같게 하는 알고리즘이 Choleski's Decomposition Method이다. 유한요소법을 이용한 선형해석은 구조물의 강성매트릭스 K가 Positive Definite System이므로 대칭이다. 따라서 선형해석의 선형방정식의 해는 대부분 Choleski's Decomposition Method을 이용한다.

위 그림 29에서 보여주는 병렬처리 알고리즘은 Shared Memory Multiprocessors에 적용될 수 있는 것이다. 매트릭스 K는 Choleski's Decomposition Method에 의해 LDLT 또는 UDUT로 분할된다. 그림 30은 Choleski Decomposition에 적용된 병렬처리 알고리즘의 스피드업 결과를 보여준다. 사용된 매트릭스 K는 n×n의 대칭 매트릭스로서 n의 값이 1000으로부터 6000까지 크기를 변화시키고, 매트릭스의 밴드위드스를 300으로 일정하게 잡았다. 32개 프로세서가 장착된 공유 기억 다중프로세서인 KSR 2에서 계산되었다. 16개의 프로세서가 사용될 때 약 14의 스피드업을 얻는다.

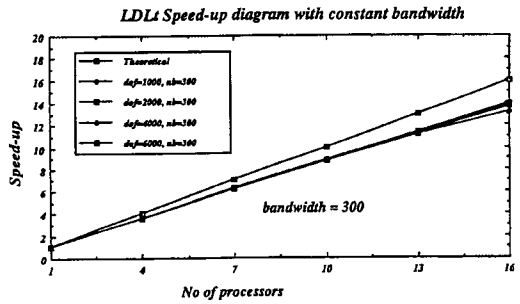
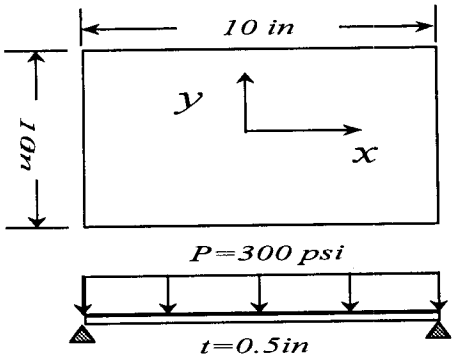


그림 30 Choleski Decomposition Speed-up Diagram

7. 구조해석에서 병렬처리

본 절에서 병렬처리를 이용하여 유한요소법에 적용한 구조해석을 간략히 소개한다. 여기서 얻은 결과들은 MPI로 만들어진 병렬처리 프로그램(GT-PARADYN)을 미국 Maui High Performcnce Computing Center의 IBM SP2 wide nodes에 적용하여 얻었다. IBM SP2는 분산 기억 다중컴퓨터이다. IBM RS/6000 프로세서로 만들어진 wide node는 66 MHz Clock Speed를 갖으며, node들간에 연결된 high performance switch의 latency는 39.2μsec이고 bandwidth는 35.6 MB/sec이다. 각 node들의 memory는 256 MB이다.



$E = 10^7\ psi \qquad E_p = 0.0 psi$

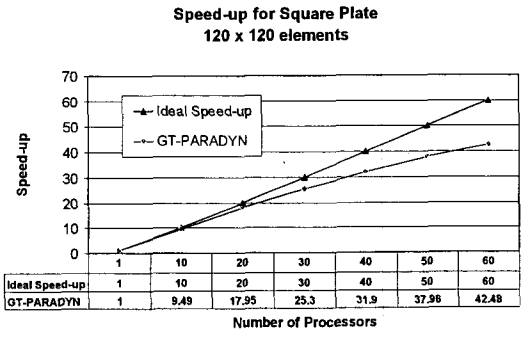
$\nu = 0.3 \qquad \sigma_y = 30{,}000 psi$

$\rho = 2.588 \times 10^{-4}\ lb - sec^2 / in^4$

그림 31 분포하중을 받는 단순지지의 평판

7.1 평판의 과도 동적 해석

단순지지의 상태의 평판이 일정한 균일 하중을 받을 때 평판의 비선형 과도 동적 거동을 계산(Transient Dynamic Analysis)하였다. 계산된 평판의 거동시간은 1.2 millisecond이고, 1,500번의 Time Step Integration이 수행되었다. 평판의 대칭성 때문에 평판의 quadrant를 120×120 유한요소로 모델링하였으며, 요소의 비선형 내력은 Belytschko-Lin-Tsay Dhell 이론과 Jaumann Stress Rate에 의해 계산하였다. 변위를 얻기 위해 Explicit Time Integration 절차가 적용되었다. 과대하중으로 인한 소성 변형은 Von Mises 항복이론에 따르는 재료로 모델링하여 내력을 적분해 나갔다. 이러한 이론들을 근거로 만들어진 병렬처리용 프로그램 GT-PARADYN에 의해 해석이 수행되었다. 그림 32는 프로세서의 수가 60



	1	10	20	30	40	50	60
Ideal Speed-up	1	10	20	30	40	50	60
GT-PARADYN	1	9.49	17.95	25.3	31.9	37.96	42.48

그림 32 평판의 과도 동적 해석의 Speed-Up Diagram

개까지 증가에 따른 스피드업을 보여준다. 60개 프로세서들이 계산에 참여했을 때 스피드업은 42.48이였다. 이 는 약 70%의 효율을 보여준다. 이러한 의미는 한 개의 프로세서가 중간 지점의 변위를 계산하는데 66,682 seconds가 소요되었으나, 60개의 프로세서를 이용하여 1,570 seconds가 소요된다는 것이다.

7.2 충돌해석

다음은 Pipe Whip 문제를 다룬다. Pipe Whip 현상은 원자력 산업에서 매우 중요시 다루는 문제다. 파이프 안으로 흐르는 유체의 갑작스런 온도 변화와 속도에 의해 파이프의 whip이 일어나 두 개의 pipe들이 서로 상대방을 마주보며 일정 속도를 유지하며 충돌하여 변형되는 거동을 예측하는 문제다. 이러한 문제는 물체의 충돌에너지가 물체를 구성하는 재료의 변형을 유발한다. 충돌속도가 어느 정도이상이 접촉면에 재료들이 소성변형을 경험하게 된다. 이러한 문제를 contact-impact 문제라 한다. 접촉면(contact surface)은 접촉력(contact force)의 크기에 따라 변화하는 Moving Boundary Surface이다. 대표적인 Contact-Impact 이론인 Master-Slave Slideline 이론을 근거로 개발된 병렬처리 알고리즘으로 작성된 병렬 프로그램 GT-PARADYN에 의해 스피드업을 얻었다. 유한요소는 Belytschko-Lin-Tsay Shell 이론이 적용되었고 Explicit Time Integration Scheme에 의해 변위가 산출된다. 각 파이프는 3,200개 4개의 절점을 갖는 사각의 shell 요소로 이루어진다. 일정한 크기의 Time Step Size는 10^{-6} second이며 1,700번 계산을 수행하여 1.7millisecond 동안의 변형 거동을 예측한다. 각 파이프의 속도는 20m/sec이다. 그림 34는 프로세서의 수가 10개까지 증가에 따른 스피드업을 보여준다. 10개의 IBM SP2 wide node들이 사용되었다. 한 개의 프로세서가 계산하는데 11,136초가 소요된 반면, 10개의 프로세서들이 사용될 때 2,287초가 소요되었다. 따라서 10개의 프로세서들이 계산에 참여하였을 때 스피드업은 4.88이다. 이렇게 낮은 스피드업의 값은 접촉력을 계산하는 contact search 부분에 대한 병렬처리의 어려움에서 비롯된다. 따라서 더 좋은 병렬처리를 위한 알고리즘의 개발이 요구된다.

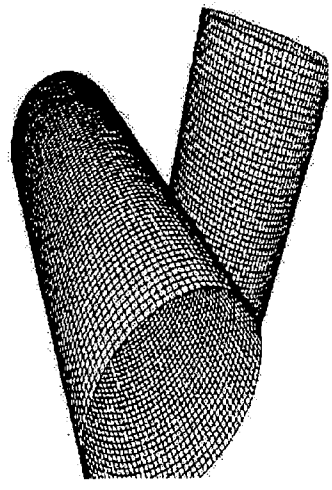
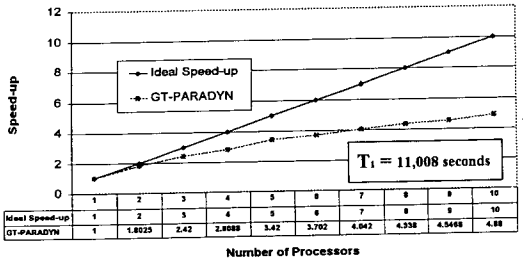


그림 33 Pipe Whip 문제

8. 맺음말

앞 절에서 언급한 바와 같이 병렬처리의 궁극적인 목적은 계산시간(wall clock time)의 단축이다. 계산시간의 단축은 CPU 시간의 상당히 크게 요구되는 작업들(computational intensive works)의 생산성의 향상에 결정적인 요인이다. 이



	1	2	3	4	5	6	7	8	9	10
Ideal Speed-up	1	2	3	4	5	6	7	8	9	10
GT-PARADYN	1	1.8025	2.42	2.8088	3.42	3.702	4.042	4.338	4.5468	4.88

그림 34 Pipe 충돌해석의 Speed-Up Diagram

러한 계산시간의 단축을 얻을 수 있는 유일한 선택은 병렬처리임을 다시 한번 주장하고자 한다.

본 논고에서 병렬컴퓨터로 진화되기까지의 전자식 컴퓨터의 역사적 발달과정과 병렬컴퓨터의 필요성을 논하였고, 병렬처리의 응용에 전제하여 요구되는 컴퓨터 구조적 이해와 병렬 알고리즘의 개발에 수반되는 전략적 이론들을 토론하였다. 그리고 수치해석과 구조해석에 적용되는 병렬처리의 예들을 간략히 소개하였다.

현재 국내에 도입된 병렬컴퓨터의 현황을 간략히 보면, 시스템 공학 연구소가 128개의 프로세서들이 장착된 Cray T3E가 있으며, 48개의 프로세서들이 장착된 SUN사의 Ultra HPC 1000이 내무부에서 사용중이다. 또한 LG전자가 40개의 Ultra HPC 1000, 동명 정보대학교가 27개의 IBM SP2SC, 삼성종합기술원이 256개의 Intel Paragon, 서울대학교가 40개의 IBM SP2, 현대자동차가 32개의 IBM SP2 등이 설치되어 사용 중에 있다.

병렬컴퓨터는 여러 개의 프로세서들이 장착되어 있기에 일반적인 순차적인 작업과 병렬처리를 위한 작업이 동시에 가능하다. 병렬컴퓨터의 사용에 있어 궁극적인 이득은 병렬처리의 활용에서 온다 따라서 여러 분야로 병렬처리의 응용에 대한 관심이 확산되길 기대한다.