병렬 컴퓨터, 병렬 프로그래밍 기술 소개

• 이 상 호 ▌한국 IBM 기술지원센터, 정보처리 기술사 / e-mail:slee@kr.ibm.com

이 글에서는 고성능 컴퓨팅의 핵심 기술로 자리잡고 있는 병렬 컴퓨터 구조와 병렬 컴퓨팅을 위한 OpenMP, MPI 프로그래밍 기술에 대해서 살펴보고자 한다.

현재 공학적으로 접근하는 일반적인 자연현상에 대한 물리적인 지배방정식(Governing Equation)들은 대부분 밝혀져 있다. 하지만 문제는 이러한 지배방정식들이 비선형 연립 편미분 방정식으로 표현되는 경우에 수학적으로 엄밀한 해를 구하는 것이 거의 불가능하다는 것이다. 그래서 특정 조건 아래서의 수치적인 해를 구하는 수치해석(Numerical Analysis)기법이 이용된다. 주로 편미분 방정식으로 표현되는 지배방정식을 풀기 위해서 FDM(Finite Difference Method), FVM(Finite Volume Method), FEM(Finite Element Method)과 같은 수치해석기법들이 이용된다. 이런 수치해석기법들은 컴퓨터에서는 행렬을 이용한 반복적 연산으로 처리된다. 그러므로 수치해석을 위한 행렬연산을 빠르게 처리하기 위한 고성능 컴퓨터 시스템과 함께 연산의 병렬성을 이용한 병렬처리기법이 요구된다.(그림 1 참조)

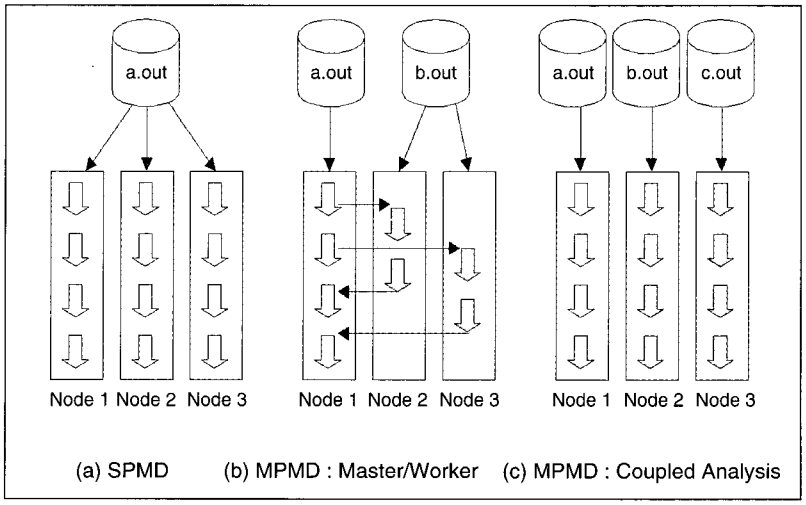


그림 1 다양한 병렬 처리 기법

고성능 컴퓨팅 시스템

최근에 수치해석기법은 구조, 유체, 소음, 충돌 해석 등과 같은 일반 적용 분야뿐만 아니라, 핵실험을 대체하기 위한 핵 시뮬레이션, 전 지구 기상 시뮬레이션, 인간의 유전자 비밀을 해독하기 위한 지놈 프로젝트 등의 초대형(Grand Challenging) 해석 업무에도 적용되고 있다. 초대형 문제를 수치해석을 통해 풀기 위해서는 현존하는 컴퓨팅 파워를 능가하는 고성능 컴퓨팅 환경이 요구된다. 예전에는 빠른 시스템을 지칭하는 데 수퍼컴퓨터라는 용어가 널리 사용되었다. 하지만 최근에는 수퍼컴퓨터라는 용어보다는

보다 포괄적인 고성능 컴퓨팅(HPC : High Performance Computing)이라는 용어를 사용하는 것이 일반적이다. 수퍼컴퓨터의 정의와 마찬가지로 고성능 컴퓨팅 시스템의 정의를 내리는 것도 쉽지는 않다. 하지만 최근에는 TOP500 리스트(http://www.top500.org)를 참조하는 경우가 많다. TOP500 리스트는 독일 만하임 대학교의 H. Meuer와 미국 테네시 주립대학교와 국립 오크리지 연구소의 J. Dongarra, 테네시 주립대학교의 E. Strohmaier가 공동 작성한 보고서이다.

1993년 이후 매년 6월과 11월에 Linpack 벤치마크 평가에 의한 컴퓨터들의 성능 순위를 정해서 500위까지의 고성능 시스템들에 대한 자료를 발표하는 것이다. 그러므로 TOP500 리스트를 참조하면 현재 전 세계에서 가장 빠른 500위까지의 시스템들을 쉽게 확인할 수 있다. 뿐만 아니라 연도별 고성능 시스템의 구조와 변천에 대해서도 분석할 수가 있다. TOP500 리스트의 시스템 구조를 살펴보면 MPP, NUMA 방식의 병렬 컴퓨터 시스템의 점유율이 계속 증가하는 추세를 알 수 있다. 반면 벡터 방식의 수퍼컴퓨터 시스템의 점유율은 계속 하락하고 있다. 최근의 TOP500 리스트에서 특이한 점은 상용 업무를 위한 고성능 컴퓨터 시스템들이 많이 늘어나고 있다는 것이다. 과학, 공학 분야의 수퍼컴퓨터뿐만 아니라 대규모 데이터 베이스(VLDB : Very Large DB), 인터넷 웹 서버, 통신 등의 상용 업무에도 고성능 컴퓨터들이 많이 도입되는 추세이다. 또한 최근에는 상대적으로 저렴한 비용에 고성능을 제공할 수 있는 리눅스 클러스터 시스템의 활용도 증가하고 있다.

병렬 컴퓨터 구조

지난 30 년 동안 컴퓨터 시스템의 성능은 무어의 법칙에 따라 계속해서 향상되어 왔다. 무어의 법칙은 마이크로프로세서의 집적도가 매 18 개월마다 두 배씩 성장한다는 예측 법칙이다. 1960년대에 인텔의 고든 무어 회장은 트랜지스터 회로들을 구성하는 가는 선들의 폭이 1 년에 대략 10% 씩 줄어들기 때문에 3 년마다 집적시킬 수 있는 트랜지스터 수가 4 배씩 증가하고, 그에 따라 새로운 칩들이 출현하게 될 것을 예측하였다. 매 18 개월마다 성능이 두 배씩 향상된다는 것은 3 년이면 4 배, 6 년이면 16 배, 9 년이면 64 배, 12 년이면 256 배, 15 년이면 1,024 배가 증가하는 것을 의미한다. 하지만 앞으로 반도체 소자의 특성상 집적 회로의 폭이 0.04 마이크론에 이르게 되면 소자들이 물리적으로 더 이상 자성을 보유하지 못 하게 되어 집적도의 증가가 이론상 불가능해지는 한계에 다다르게 된다. 그러므로 계속해서 단일 프로세서의 성능을 향상시킬 수는 없는 것이다. 이에 대한 대안으로 여러 프로세서들을 연결하여 전체 시스템의 성능을 향상시키는 병렬 컴퓨팅 기술이 적용되고 있다. 병렬 컴퓨터 시스템은 크게 멀티 프로세서와 멀티 컴퓨터 모델로 구분된다. 멀티 프로세서 시스템은 단일 시스템 안에 여러 개의 프로세서를 가진 시스템을 가리킨다. 공유 메모리 기법을 이용하는 SMP (Symmetric Multi Processing) 또는 NUMA(Non Uniform Memory Access) 시스템이 이에 해당한다. 멀티 컴퓨터 시스템은 독립적인 여러 대의 컴퓨터 시스템들이 고속 네트워크 또는 고속 스위치를 통해 연결된 구조를 가리킨다. 이 때 독립된 각 컴퓨터를 노드(Node)라고 한다. 클러스터나 MPP 시스템이 멀티 컴퓨터 구조에 속한다고 할 수 있다.(표 1 참조)

병렬 프로그래밍 기법

병렬 프로그래밍의 목적은 가능한 많은 프로세서 자원들을 사용하여 프로그램을 수행함으로써 프로그램의 수행 시간을 최소한으로 줄이는 것이다. 현재 병렬

표 1 멀티 프로세서와 멀티 컴퓨터 모델의 비교

멀티 프로세서 시스템	멀티 컴퓨터 시스템
하나의 컴퓨터	여러 대의 컴퓨터
Tightly Coupled Multiprocessor	Loosely Coupled Multiprocessor
Shared Memory, Shared Disk 모델	Shared Nothing 모델
UMA, NUMA, COMA 시스템	클러스터, MPP 구조 시스템

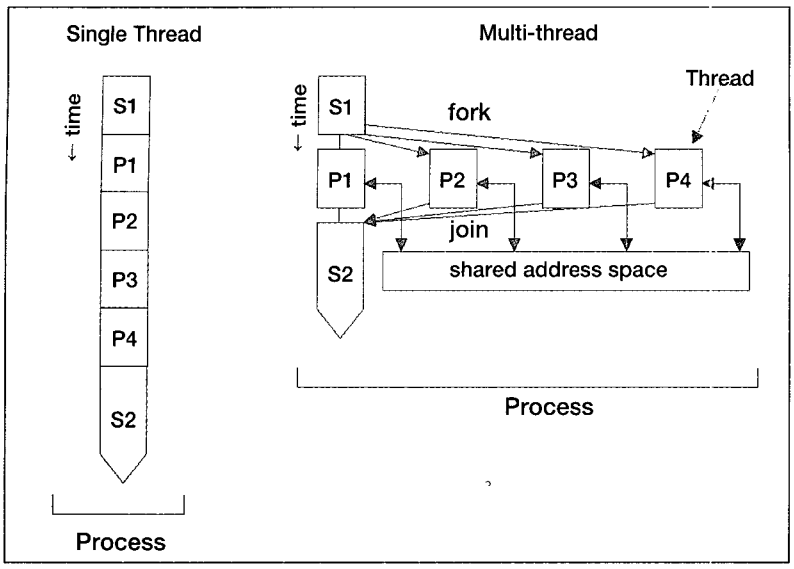


그림 2 Multi-Thread를 이용한 병렬처리 기법

프로그래밍 기법은 크게 두 가지로 구분이 된다. SMP, NUMA와 같은 공유 메모리 멀티 프로세서 시스템에서는 멀티 스레드(Multi Thread) 기법 또는 IPC 기법 등을 이용하여 병렬 프로그램을 작성하게 된다. 하지만 클러스터나 MPP 시스템과 같은 분산 메모리 멀티 컴퓨터 시스템의 경우에는 메모리를 공유하지 않고 Send, Receive와 같은 병렬 API를 통해 노드간에 데이터를 주고받는 메시지 패싱 기법을 이용하여 프로그램을 작성하게 된다.

멀티 스레드 기법과 OpenMP

SMP 구조의 시스템에 가장 적합한 모델은 멀티 스레드 기법이다. 멀티 스레드 기법을 이용하면 하나의 프로세스 안에 시스템 자원들을 공유하면서 독자적으로 수행될 수 있는 여러 스레드들을 생성할 수 있다. 그러므로 하나의 멀티 스레드 프로세스를 SMP 시스템에서 수행시키면 프로세스 안에 있는 여러 개의 스레드들이 SMP 구조의 프로세서들에 분산되어 수행될 수 있다.(그림 2 참조)

대개 POSIX 스레드 라이브러리(pthreads)가 이용된다. 하지만 스레드 라이브러리를 이용하여 직접 프로그래밍하기 위해서는 전문적인 지식이 요구된다. 그래서 일반적인 사용자들은 프로그램 안에 DO 루프와 같이 멀티 스레드 수행이 가능한 부분을 디렉티브(directive) 등을 이용하여 외부적으로 명시함으로써 컴파일러가 자동으로 멀티 스레드 프로그램을 생성하게 하는 등의 쉬운 접근 방법을 주로 이용하게 된다. 이런 용도로 널리 쓰이는 병렬 API가 OpenMP이다. OpenMP는 멀티 스레드 프로그래밍을 위한 API(Application Programming Interface)이다. Fortran, C, C++에서 쉽게 멀티 스레드 프로그램을 만들 수 있게 해 준다. OpenMP는 컴파일러 directive, 라이브러리 루틴들과 환경변수로 구성된다. OpenMP는 지난 15 년 간 SMP 기능에서 표준화되어 왔으며, Intel, HP, SGI, IBM, SUN, Compaq 등의 다양한 하드웨어 업체들과, KAI, PGI, PSR, APR, Absoft들의 여러 소프트웨어 그리고 ANSYS, Fluent, Oxford Molecular, NAG, DOE, ASCI, Dash, Livermore Software등의 어플리케이션 업체들이 지원하고 있다.

OpenMP 프로그래밍 모델을 보면 Fork-Join모델로 다음 그림 3과 같이 나타낼 수 있다.

마스터 스레드가 실행을 하다

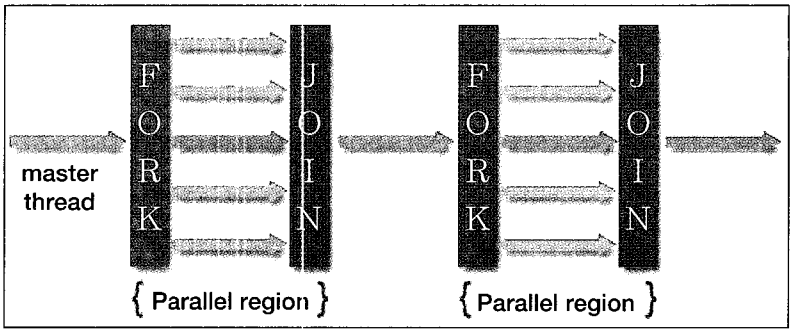
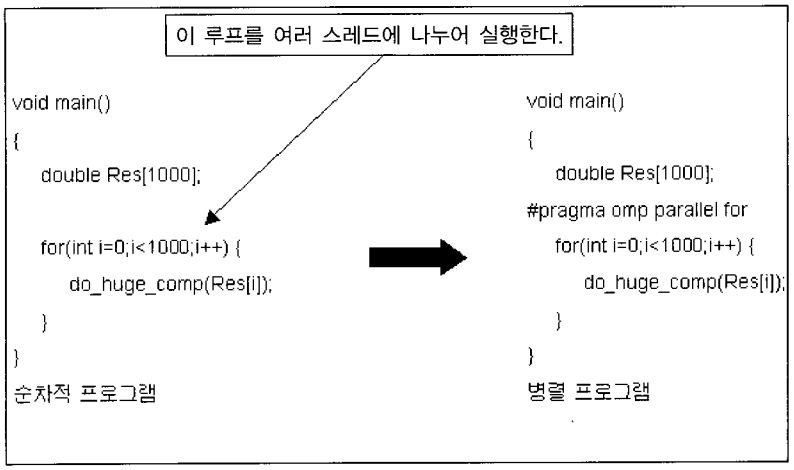


그림 3 Fork-Join 모델을 이용한 멀티 스레드 처리



프로그램 1 OpenMP를 이용한 프로그램 예

가 Do 루프 등의 병렬 처리가 가능한 부분을 만나게 되면 여러 개의 멀티 스레드로 나뉘어져 실행을 하게 되고, 루프가 끝나면 다시 마스터 스레드로 합쳐지게 된다. 또 다시 루프를 만나면 멀티 스레드로 나누어졌다가 그 후 합쳐지는 이러한 형태를 전체 프로그램에 대해서 반복하는 구조를 갖는다. OpenMP의 전형적인 사용방법을 보면 이는 보통 루프를 병렬 처리하기 위해서 사용하는데, 먼저 가장 시간이 오래 걸리는 루프를 찾아 그 부분을 여러 스레드로 나누어 실행을 하게 한다.(프로그램 1 참조)

```
#include <stdio.h>
#include <omp.h>
static long num_steps=800000000;
double step;
#define NUM_THREADS 8
void main()
{
      int i;
      double x, pi, sum=0.0;
      step=1.0/(double)num_steps;
      omp_set_num_threads(NUM_THREADS);
      # pragma omp parallel for reduction(+:sum) private(x)
      for(i=1;i<=num-steps; i++)
      {
            x=(i-0.5)*step;
            sum=sum+4.0/(1.0+x*x);
      }
      pi=step * sum;
      printf("pi=%f₩n",pi);
}
```

프로그램 2 OpenMP를 이용한 pi 계산 프로그램 예

OpenMP에 사용되는 문법은 대부분 컴파일러 direc-tive이거나 pragma이다. C에서는 pragma가 사용되고, Fortran에서는 directive가 사용된다. OpenMP 문법은 컴파일러 directive이기 때문에 OpenMP를 인식하지 못하는 컴파일러에서도 컴파일할 수 있다. 그러면 OpenMP의 간단한 예를 살펴보도록 하자. 예제는 pi를 계산하는 간단한 프로그램이다. 일반적인 순차적 프로그램 2의 예제에서 보는 것처럼 파란색으로 표시된 OpenMP 디렉티브 부분만을 추가하면 컴파일러가 병렬 처리가 가능한 실행파일을 생성하게 된다.

예제 프로그램을 SMP 컴퓨터에서 직접 테스트 해보면 다음과 같다. 테스트는 375MHz POWER3 프로세서를 16 개 장착한 IBM SMP 시스템에서 수행되었다. 결과에서 보는 것처럼 간단하게 OpenMP 디렉티브만을 삽입하여 병렬처리효과를 얻게 되었다. 프로그램의 병렬 처리 효율성을 더 높이기 위해서는 다양한 튜닝이 필요하겠지만, 예제를 통해 간단하게나마 OpenMP의 성능과 그 특성을 이해했으리라 생각한다.(표 2 참조)

표 2 openMP 프로그램 테스트 결과

NUM_Threads (프로세서 수)	1	4	8	16
Elapsed Time (sec)	29.06	11.01	5.59	4.22

메시지 패싱 기법과 MPI

분산 메모리 멀티 컴퓨터 구조에서는 데이터가 노드들 간에 공유되지 않기 때문에 노드들은 필요한 데이터들을 상호 연결망을 통해 주고받을 수 있어야 한다. 그림 4에서 S1, S2는 순차적 프로그램 부분을, P1 - P4 는 병렬처리가 가능한 부분을 가리킨다. 메시지 패싱 모델에서 각 노드는 각 프로세스를 수행한다. 그리고 병렬처리 부분을 수행하기 위해서 필요한 데이터들을 서로 교환하기 위해서 각 프로세스들은 서로 통신하는 구조를 가지고 있다. 그러므로 노드 프로세스들 간의 통신 기능을 수행하는 라이브러리를 제공해 준다면 좀더 쉽게 병렬 프로그램을 코딩할 수 있게 된다. 이러한 목적으로 사용되는 것이 바로 메시지 패싱 API들이다.

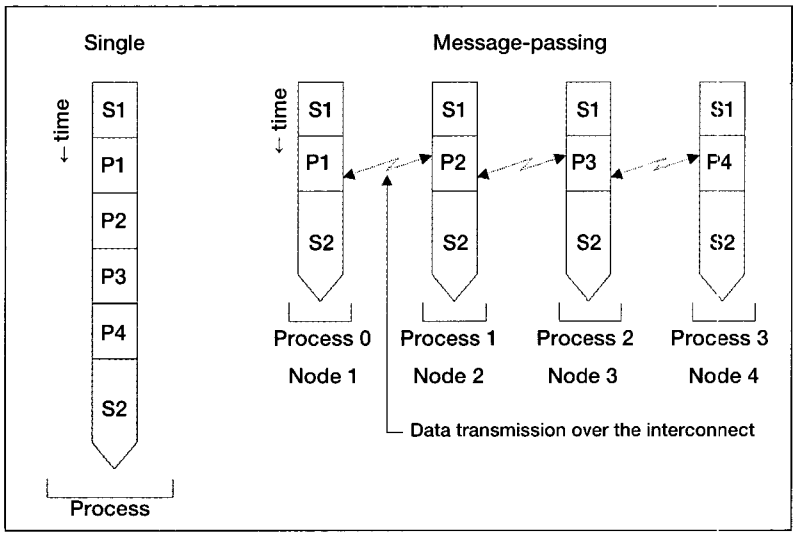


그림 4 메시지 패싱을 이용한 병렬처리 기법

```
PROGRAM main
IMPLICIT REAL*8 (a-h, o-z)
PARAMETER (n = 11)
DIMENSION a(n), b(n)
DO I = 1, n
  b(i) = i
ENDDO
DO I = 2, n-1
    a(i) = b(i-1) + b(i+1)
ENDDO
END
```

프로그램 3 1차원 FDM 프로그램

현재 MPI(Message Passing Interface, http://www-unix.mcs.anl.gov/mpi/)가 업계 표준 메시지 패싱 API로 사용되고 있다. MPI가 등장하기 전에는 PVM, EXPRESS, LINDA 등이 사용되었다. 메시지 패싱 표준들은 C 언어나 포트란 언어를 위한 API들을 제공한다. 그러므로 프로그래머는 메시지 패싱 API를 이용하여 기존 C, 또는 포트란 프로그램을 병렬 처리 버전으로 변환시킬 수 있다. MPI 프로그램 3의 예를 살펴보자. 간단한 1차원 FDM 프로그램을 MPI를 이용한 병렬 프로그램으로 변환해 보기로 한다.(그림 5 참조)

FDM 프로그램은 행렬 b(n)의 값들을 더해서 행렬 a(n)의 값들을 계산하는 것이다. 이 경우에는 데이터 분할 기법을 이용하여 프로그램을 병렬 처리할 수 있다. 프로그램 4는 변환된 MPI 프로그램과 실행 예이다.

MPI_ISEND, MPI_IRECV는 각 노드에서 수행되는 MPI 프로그램들이 서로 필요로 하는 데이터들을 주고받기 위해서 사용되는 포인트-투-포인트(Point-to-Point) 메시지 패싱 라이브러리들이다. 그림 6에서 보는 바와 같이 데이터 분할로 인해 발생되는 경계값 데이터들을 서로 교환하기 위해 사용된다. 이러한 MPI 라이브러리들을 이용하여 기존의 순차적 프로그램을 병렬화시키면 그림 6에서 보는 바와 같이 데이터들이 각 프로세서에 분할되어 병렬 처리되게 된다.

MPI 프로그램을 살펴보면 기

존 프로그램보다 복잡하고 OpenMP를 이용하는 것보다 어렵다는 것을 알 수 있다. MPI를 이용하기 위해서는 메시지 패싱 기법 및 병렬 라이브러리 사용에 대한 전문지식이 필요하다. 하지만 프로세서의 확장성에 한계가 있는 SMP 시스템에서 OpenMP를 이용하는 것과 달리 수백

```
PROGRAM main
INCLUDE 'mpif.h'
IMPLICIT REAL*8 (a-h, o-z)
PARAMETER (n = 11)
DIMENSION a(n), b(n)
INTEGER istatus (MPI_STATUS_SIZE)
CALL MPI_INIT(ierr)
CALL MPI_COMM_SIZE(MPI_CO-MM_WORLD, nprocs, ierr)
CALL MPI_COMM_RANK(MPI_CO-MM_WORLD, myrank, ierr)
CALL para_range(1, n, nprocs, myrank, ista, iend)
ista2 = ista
iend1 = iend
IF (myrank == 0)                ista2 = 2
IF (myrank == nprocs - 1)       iend1 = n - 1
inext = myrank + 1
iprev = myrank - 1
IF (myrank == nprocs - 1)       inext = MPI_PROC_NULL
IF (myrank == 0)                iprev = MPI_PROC_NULL
DO I = ista, iend
      b(i) = i
ENDDO
CALL MPI_ISEND (b(iend), 1, MPI_REAL8, inext, 1 MPI_COMM,_WORLD, isend1, ierr)
CALL MPI_ISEND (b(ista), 1, MPI_REAL8, iprev, 1, MPI_COMM_WORLD, isend2, ierr)
CALL MPI_IRECV (b(ista-1), 1, MPI_REAL8, iprev, 1, MPI_COMM_WORLD, irecv1, ierr)
CALL MPI_IRECV(b(iend+1),1, MPI_REAL8, inext, 1, MPI_COMM_WORLD, irecv2, ierr)
CALL MPI_WAIT (isend1, istatus, ierr)
CALL MPI_WAIT (isend2, istatus, ierr)
CALL MPI_WAIT (irecv1, istatus, ierr)
CALL MPI_WAIT (irecv2, istatus, ierr)
DO I = ista2, iend1
   a(i) = b(i-1) + b(i+1)
ENDDO
CALL MPI_FINALIZE (ierr)
END
```

프로그램 4 MPI를 이용한 1차원 FDM 프로그램의 변환

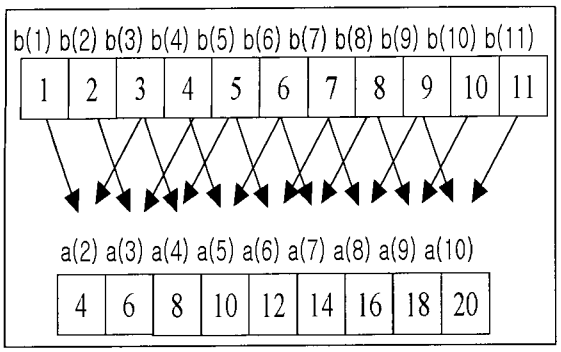


그림 5 1차원 FDM 프로그램의 실행 예

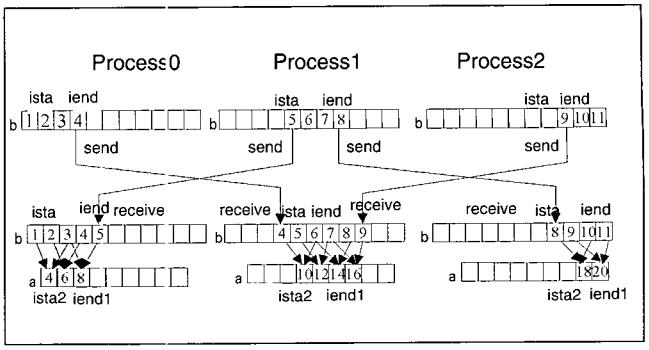


그림 6 1차원 FDM 프로그램의 실행 예(MPI 버전)

개의 프로세서를 연결할 수 있는 분산 메모리 구조의 멀티 컴퓨터 시스템을 사용하면서 병렬처리 효과가 높은 프로그램을 개발하기 위해서는 프로그램의 알고리듬을 잘 아는 개발자가 직접 MPI를 이용하여 최적화된 병렬 코드를 개발하는 것이 필요하다. OpenMP와 마찬가지로 병렬처리가 잘 되는 부분, 일반적으로 전형적인 행렬 연산과 같은 부분만을 MPI를 이용해서 처리할 수 있기 때문에 앞의 간단한 예제를 복잡한 프로그램에도 적용할 수 있다. 예제는 1차원 FDM에서 행렬의 크기가 작은 경우지만, 2차원, 3차원 FDM에서 행렬의 크기가 커지게 되면 MPI 병렬 처리를 통해 프로그램의 수행시간을 크게 단축시킬 수 있다. 이와 같은 MPI를 이용한 다양한 프로그램들이 많이 등장하고 있다. 상용 소프트웨어 업체들도 MPI를 이용한 수치해석 프로그램들을 내놓고 있다.

맺음말

이상에서 병렬 컴퓨터 구조에 대해 살펴보았다. 예전과 달리 지금은 병렬 컴퓨터 시스템들이 보편적으로 널리 사용되고 있다. 학교, 연구소, 산업체 등지에서 병렬처리기법을 적용하는 사례가 늘고 있다. 그러므로 병렬 컴퓨터 시스템과 병렬 프로그래밍 기법에 대해서 관심을 가지고 살펴보는 것이 필요하다. 병렬처리에 도전해보고자 하는 분들은 먼저 OpenMP를 시도하는 것이 쉬울 것이다. 본격적으로 병렬처리를 시도하고자 하는 분들은 MPI에 대해 관심을 가지는 것이 필요하다.

기계용어 해설

활성비등핵(Active Nucleation Site)

비등 열전달시 가열면상에 기포가 생성되는 위치, 단위 면적에 활성비등핵의 수를 활성비등핵 밀도라는 용어로 표현하기도 하며, 주로 표면이 거친 정도와 표면 가공방법에 따라 활성비등핵 밀도가 결정된다. 경우에 따라서는 특정 유체와 표면의 조합이 활성비등핵 밀도에 영향을 주는 경우도 있다. 활성비등핵의 수가 증가하면 열전달량이 증가하며, 주변 액체를 교란시키는 부차적인 효과로 열전달을 한층 향상시키게 된다.

원추형 다이(Conical Dies)

원추형 다이란 압출금형에 사용되는 다이의 형상이 원추의 형상으로 되어 있는 것을 말한다. 일반적인 금속의 압출에서 평면형 다이를 사용할 경우 유동금속이 부동영역(DMZ : dead metal zone)을 형성하며, 압출하중이 높아지는 경향이 있으나, 금속에 따라 적정한 원추각을 갖는 원추형 다이를 사용할 경우 금속유동을 원활하게 하며, 압출하중을 감소시킬 수 있다.