한국해안·해양공학회 학술발표논문집 (2008)
2008년 9월 25일-9월 26일
제17권, pp. 40-43

최근 PC와 병렬프로그래밍
Recent Status of PC and Parallel Programming

전인식 [1]
Insik Chun [1]

1. 서 론

해양파동과 해수유동의 시·공간적 변화를 계산하는 수치해석들은 일반적으로 광대영역을 대상으로 하기 때문에 수많은 절점에서의 반복계산을 요구하는 경우가 많다. 반대로 소규모 영역에서의 유체현상을 Navier-Stokes 방정식을 풀어 정밀해석하는 경우에도 해의 안정성 및 수렴을 위하여 시간 및 공간 격자간격을 매우 조밀하게 설정해야 되는 경우가 보통이다. 이와 같은 문제들은 공히 막대한 메모리 양과 오랜 계산시간을 요구한다. 이 두 가지 문제점을 해소하기 위해서는 기 설치된 슈퍼컴퓨터를 사용하거나 어렵지만 유휴 소형컴퓨터들을 연결하는 clustering 방식을 사용할 수 있다. 후자의 경우는 일종의 분산메모리 방식으로서 MPI (Message Passing Interface) 약정에 의한 프로그래밍 기법이 확립되어 있으나 하드웨어의 구성에서 상당한 전문지식이 요구되어 일반적인 수치모델러들의 접근이 용이치 않다. 그러나 다행이도 최근에 출시되는 소형컴퓨터들은 다중 core를 탑재하고 메모리를 공유하는 식의 일종의 병열 컴퓨터로 출시되어 있다. 이들 기종들에서 공유메모리를 제어하기 위한 OpenMP (Open Multi-Processing) 약정을 이용하여 병렬프로그램을 작성하여 사용하면 수치해석의 계산시간을 현격히 단축할 수 있다. 본 고에서는 최근 소형컴퓨터를 병열컴퓨터로 사용하기 위한 하드웨어 및 소프트웨어의 구성과 그리고 간단한 병렬프로그래밍 기법에 대해서 소개하기로 한다.

2. 다중 core 컴퓨터와 OpenMP

현재 주변의 일반적인 소형컴퓨터는 대부분 32 비트용 컴퓨터이다. 이 비트 단위는 한번에 처리될 수 있는 데이터 단위를 의미한다. 32 비트용 컴퓨터는 동시에 2^{32}개, 즉, 4G 바이트의 주소를 지정할 수 있다. 반면, 64 비트용 컴퓨터는 2^{64}개, 즉, 4G 바이트x2^{32}개의 주소를 지정할 수 있기 때문에 64 비트 컴퓨터는 공학계산에 자주 발생하는 메모리 용량의 문제를 대폭 해소할 수 있다. 이와 같은 32 또는 64 비트는 CPU의 차이에 따른 것이며 응용프로그램도 출시부터 32 비트용과 64 비트용이 구분되어 있다. 32 비트용 응용 프로그램은 64 비트용 컴퓨터에서 대부분 사용가능하나 (64 비트 전용 컴퓨터는 예외) 64 비트용 프로그램은 32 비트용 컴퓨터에서 사용 불가능하다. 즉, 32 비트용 포트란 컴파일러는 64 비트용 컴퓨터에서 사용할 수 있으나 4 GB의 기억용량만 제어가능하다. 또한, OS도 32 비트용과 64 비트용이 구분되어 있다. 즉, 윈도우즈를 사용할 경우에 64 비트용 컴퓨터에는 64 비트용 윈도우즈를 사용하여야 한다.

최근에는 1개의 시스템에 여러 개의 core가 적재되어 있는 다중 프로세서 컴퓨터가 다수 출시되고 있다. 정확히는 1개의 Machine에 수개의 Package가 존재할 수 있으며 1개의 Package가 설치되어 있고 그 안에 2개의 core가 있으면 Dual core, 4개의 core가 있으면 Quad core라고 칭한다. 최근 대부분 출시되는 컴퓨터는 Dual core 또는 Quad core 컴퓨터이며 2개 이상의 package 구성이 필요할 시는 별도 조립이 요구된다.

공학계산에서 사용하는 프로그램들은 대용량의 기억용량을 점할 경우 단일 프로세서에서의 순차적 (Serial) 계산방식에 의할 것 같으면 계산이 불가능하거나 가능하더라도 수 시간에서 몇 달에 걸치는 집행시간이 요구된다. 그러나, 프로그램을 병열화하여 사용하면 이와 같은 난점을 대폭 해소할 수 있다. 결국, 병열프로그램은 주로 대 기억용량 프로그램을 대상으로 하는 경우가 많다. 이와 같은 이유로 병열프로그램이 가능한 다중 프로세서 컴퓨터는 대용량의 메모리를 설치하는 쪽으로 진화하고 있으며 최근에 출시된 64 비트용 컴퓨터도 그 예이다. 32 비트 컴퓨터에서도 병열프로그램은 가능하나 앞서 언급한 것처럼 최대 4 GB의 용량의 제한을 받는다. 반면, 64비트 컴퓨터에서는 4 GBx2^{32}의 address를 지원하기 때문에 4 GB 이상의 메모리를 설치하여 사용할 수 있다.

병열프로그램을 수행할 수 있는 컴퓨터는 메

1발표자: 건국대학교 토목공학과 교수

모리를 이용하는 방식에 따라서 분산메모리를 이용하는 MPI 방식과 공유메모리를 이용하는 OpenMP 방식이 있다. 여러 대의 개인용 PC를 연결하여 사용하는 cluster 방식이 MPI 방식이며 최근의 dual core 또는 quad core 컴퓨터와 같이 여러 개의 core가 동일 메모리를 공유하는 방식이 OpenMP 방식이다.

3. 병열프로그램의 적용

여기에서는 WINDOWS 운영체제에서 OpenMP 방식을 이용한 병열 프로그램밍에 대하여 간단히 기술하기로 한다. 병열계산에서 계산을 독자적으로 수행하는 단위를 thread라고 칭한다. 통상 1개의 core안에 1개의 thread를 설정한다. 물론 1개의 core에 1개 이상의 thread를 설정할 수도 있지만 동일 core안에서는 thread들이 순차적으로 계산을 수행하기 때문에 단일 thread를 설치하는 경우에 비하여 오히려 계산속도가 떨어진다. 따라서, dual core 컴퓨터에는 2개의 thread를, quad core 컴퓨터에서는 4개의 thread를 설정하면 된다. 병열계산은 주로 반복계산을 수행하는 DO LOOP가 대상이 되는 수가 많으며 한 cycle의 계산결과가 다음 cycle의 계산에 영향을 미치지 않는다면, 즉, 각 cycle의 계산이 각각 독립적으로 이루어진다면 Loop를 여러 개의 thread로 나누어 분산처리하게 된다.

병열프로그램은 기존의 프로그램 안에 병열계산을 지시하는 지시어를 삽입하면 된다. 지시어들은 수 종류가 있으나 위에서 기술한 DO LOOP의 경우에는 간단한 몇 줄의 지시어를 삽입함으로써도 계산효율을 대폭 높일 수 있다. Table 1의 예는 FFT (Fast Fourier Transform)보다 계산시간이 훨씬 오래 걸리는 DFT (Discrete Fourier Transform)에 대하여 병열 프로그램밍을 수행한 것이다. 대상 데이터는 16,384개의 난수이며 계산을 2번 반복 수행하였다. 본 병렬프로그램은 순차적 프로그램에 단 몇 줄의 지시어만 삽입하여 작성되었으며 각 지시어들은 굵은 글씨로 표기되어 있다. 각 지시어들의 의미는 Table 2와 같다.

Table 1. 병렬프로그래밍의 예 (DFT 계산)

```
C  OPERATIONS USING A DFT (DISCRETE FOURIER TRANSFORMATION) ROUTINE
C
C GENERATE INPUT TIME SERIES BY RANDOM NUMBER GENERATOR
C -5<=X<=5
          PARAMETER(M=50000)
          IMPLICIT REAL*8(A-H,O-Z)
          COMMON/VARI/XAA(M),XBB(M),XA(M),XB(M)
          integer(4) :: time_begin, time_end, crate, cmax
          call system_clock(count=time_begin, count_rate=crate, count_max=cmax)
C
          OPEN(7,FILE='OUTP.DAT',STATUS='UNKNOWN')
C
          PI=4*DATAN(1.0D0)
C NDATA : NUMBER OF INPUT DATA
C NCYCLE : TOTAL NUMBER OF THE REPETITION OF THE CALCULATION.
          NDATA=16384
          NCYCLE=2
          NFOUR=0
C
C GENERATE INPUT TIME SERIES USING RANDOM NUMBER GENERATOR.
C         CALL RANDOM_SEED()
          DO 10 I=1,NDATA
          CALL RANDOM_NUMBER(X)
          XAA(I)=10*(X-0.5)
10        XBB(I)=0.0
C
C REPEAT THE CALCULATION UP TO 'NCYCLE' CYCLE
          DO 20 ICYCLE=1,NCYCLE
C
C FOURIER TRANSFORMATION OF THE INPUT TIME SERIES

          DO 30 I=1,NDATA
          XA(I)=XAA(I)
30        XB(I)=XBB(I)
C
          CALL DFT(XA,XB,NDATA,0)
C
          WRITE(*,*)'NCYCLE = ',NCYCLE,'    ICYCLE = ',ICYCLE
          WRITE(*,*)XA(NDATA-1),XB(NDATA-1)
          WRITE(*,*)XA(NDATA),XB(NDATA)
          WRITE(7,*)ICYCLE
          WRITE(7,*)XA(NDATA-1),XB(NDATA-1)
          WRITE(7,*)XA(NDATA),XB(NDATA)
20        CONTINUE
C CHECK CPU TIME
          call system_clock(count=time_end, count_rate=crate, count_max=cmax)
          print *, 'cpu time = ', (time_end - time_begin) * 1.0 / crate
C
          STOP
          END
C
```

```
C=============================================================
      SUBROUTINE DFT(XR,XI,N,ISIGN)
C=============================================================
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION XR(N),XI(N)
      COMPLEX*16 FW,FT(0:N),ARG1
C
C  ISIGN=0 : FOURIER TRANSFORM
C  ISIGN=1 : INVERSE TRANSFORM
C
C                N-1
C    F(W)=(1/N) SIGMA f(T) EXP(-2*PI*CI*W*T/N);    W=0,1,2,...,N-1
C                T=0
C
C          N-1
C    f(T)= SIGMA F(W) EXP(+2*PI*CI*W*T/N);    T=0,1,2,...,N-1
C          W=0
C
C  FREQ(W)=2*W*PI/(N*DT) ; W=0,1,2,...,(N/2)
C  FREQUENCY INTERVAL (IN HERTZ) DF=1/(N*DT)

      PI=4.0*DATAN(1.0D0)
C
!$OMP PARALLEL PRIVATE(I,J,FW,ARG,ARG1)
!$OMP DO SCHEDULE(DYNAMIC,10)
        DO 10 J=0,N-1
10    FT(J)=DCMPLX(XR(J+1),XI(J+1))

!$OMP DO SCHEDULE(DYNAMIC,10)
      DO 20 I=0,N-1
      FW=(0,0)
      DO 30 J=0,N-1
      IF(ISIGN.EQ.0) THEN
      ARG=-2.*PI*I*J/N
      ARG1=DCMPLX(0,ARG)
      FW=FW+FT(J)*CDEXP(ARG1)/N
      ELSEIF(ISIGN.EQ.1) THEN
      ARG=2.*PI*I*J/N
      ARG1=DCMPLX(0,ARG)
      FW=FW+FT(J)*CDEXP(ARG1)
      ENDIF
30    CONTINUE
      XR(I+1)=DREAL(FW)
      XI(I+1)=DIMAG(FW)
20    CONTINUE
!$OMP END PARALLEL
C
      RETURN
      END
-------------------------------------------------------------------------------
```

Table 2. 사용지시어의 의미

지시어	의 미
!$OMP PARALLEL	병열계산의 시작. 모든 지시어는 항상 !$OMP로 시작함.
PRIVATE(I,J,FW,ARG,ARG1)	I,J,FW,ARG,ARG1가 각 thread의 사적변수임을 나타냄. 만약에 공유변수 (default)로 되어 있으면 한 thread가 계산할 때마다 타 thread에 의하여 이미 계산된 이들 변수들의 값이 변하기 때문에 전체적인 계산결과가 심각한 오류에 빠지게 됨. thread간 데이터 충돌이 발생할 수 있는 변수들은 모두 PRIVATE 로 지정해야 됨.
!$OMP DO	DO LOOP의 시작
SCHEDULE(DYNAMIC,10)	각 thread가 10개씩의 LOOP를 할당받으며 계산을 먼저 끝낸 thread가 다음 10개를 선착순으로 할당받는다는 의미
!$OMP END PARALLEL	병렬계산을 종료

계산은 사양을 달리하는 2대의 컴퓨터에 상기 프로그램을 적재하여 순차적 계산과 병열계산을 각각 수행하였으며 그 결과를 Table 3에 제시하였다. 컴퓨터 A에서는 병렬계산이 순차적 계산결과에 비하여 약 반 정도의 계산시간이 소요되나 컴퓨터 B에서는 약 1/8 정도로 시간이 단축되었다. 물론 병열계산조차 FFT 보다는 비교되지 않을 정도로 오래 소요되나 2의 승수로 맞추기 위하여 부득이 데이터 개수를 희생시키고 싶지 않은 경우에서는 병열계산을 통하여 DFT를 사용해볼 수도 있을 것이다. 이와 같이 병열프로그램 기법에 대하여 깊이 공부하지 않아도 몇 개의 지시어만을 사용하여 DO LOOP를 간단히 병열화 함으로써 계산효율을 기할 수 있다.

Table 3. 컴퓨터 사양 및 계산소요시간의 비교

컴퓨터 구분	사 양			계산소요시간 (sec)	
	CPU clock speed	Core 개수	RAM 용량	순차적 계산	병렬 계산 (openMP)
A	3.0 GHz	2	3 GB	34.53	16.58
B	2.5 GHz	8	32 GB	36.55	4.49

Table 4는 연직막체방파제의 수리성능을 파악하기 위하여 작성한 기존의 순차적 FORTRAN 프로그램인 BUOY(DOUBLE).FOR를 병렬화한 후 그 성능을 분석한 것이다. 계산속도를 평가하기 위하여 상기 컴퓨터 A와 그리고 컴퓨터 B를 비교하였으며 컴퓨터 B에서는 할당 thread의 수를 달리 해가며 계산을 수행하였다. 표에서 순차적 계산과 1개의 core가 할당될 때는 컴퓨터 A가 B보다 약간 빠르나 core가 2개일 경우에는 컴퓨터 B가 빠름을 볼 수 있다. 컴퓨터 B에서 core의 개수가 증가하면서 계산시간이 단축되며 8개인 경우에는 순차적 계산결과보다 약 1/7정도 계산시간이 단축됨을 알 수 있다. 두 컴퓨터 간 계산속도의 변화를 Fig. 1에 비교하였다. 그림에서 core 개수 0은 순차적 계산을 의미한다. core의 개수가 증가하면서 그 효율이 감소하며 core 가 4개를 초과하면서 core 개수의 증가에 따른 계산속도의 단축정도가 점차 감소함을 볼 수 있다.

Table 4. 컴퓨터 사양 및 계산시간의 비교

구 분			컴퓨터 A	컴퓨터 B
계산유형	순차적 계산		192.9	207.1
	병렬계산 할당 core 개수에 따른 계산시간 (sec)	1	187.3	202.5
		2	126.3	101.9
		3	-	75.0
		4	-	55.8
		5	-	47.4
		6	-	40.8
		7	-	35.6
		8	-	30.5

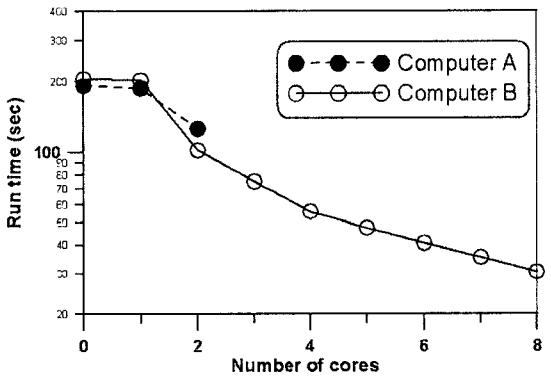

Fig. 1. 계산시간의 비교.

4. 결 론

임의 수치계산에서 OpenMP 코드를 이용하여 병렬프로그램을 작성하고 최근 출시되고 있는 dual-core 또는 quad-core 컴퓨터에서 병렬계산을 수행함으로써 계산시간을 대폭 단축할 수 있다. 더욱이 컴퓨터와 OS 및 컴파일러를 64 bit용으로 구성할 경우에는 계산속도뿐만 아니라 계산 시 소요되는 기억용량에 의한 한계도 쉽게 해결할 수 있다.

감사의 글

본 연구는 2008년도 항만리모델링 연구사업과 관련하여 이루어진 것임.