문제 3 (학부/대학원 공통): 입자 데이터의 병렬화

(배점: 학부 30 점 / 대학원 20 점)

문제 개요

이번 문제는 주어진 입자 데이터를 병렬환경에서 어떻게 빨리 나눌 수 있는지 알아보는 것에 중점을 두었다. 고정되어 있는 규격화된 자료인 경우 해당 데이터를 나눌 수 있는 여러 기법들이 이미 잘 알려져있지만, 공간을 자유로이 움직이는 입자를 다루는 계산의 경우 영역 분할 (Domain Decomposition)의 문제는 쉽게 다룰 수 없는 문제이다. 따라서 이 문제는 가장 간단한 입자 데이터(실수)의 영역 분할을 가장 빨리 실행할 수 있는 데에 중점을 두고 있다.

문제 내용

먼저 0과 1 사이에 10,000,000*niter 개의 실수를 무작위로 생성한다 (시드 번호는 예제에 있는 것과 동일해야 함). 그리고 이렇게 만들어진 실수의 집합을 niter개의 서로 다른 영역으로 분할한다. 여기서 niter는 실행시에 input값으로 주어진다.

그림 1 은 주어진 sequential 프로그램의 예제이다. 49-54 (f90 의 경우는 25-31) line 들에서는 무작위 수를 만들어서 메모리에 저장한다. 여기에 사용된 ran2()라는 함수는 0 과 1 사이에 무작위로 실수를 생성시킨다. 무작위 수를 생성할 때 seed 가 필요한데, 여기에서는 –1284, -1285, ..., -(1284+niter)라는 niter 개 만큼의 seed 들을 이용한다. 그리고 55-59 (32-37) line 에서는 주어진 maxnp*niter 개의 무작위 수들을 해당 영역 (이 경우 lval[0-niter-1]/lval(1:niter))에 >저장한다. 여기에서는 각각 영역의 크기는 일정하다고 가정한다. 즉 niter=4 일 경우 각각 영역의 크기는 1/4=0.25 가 된다.

병렬화 방법

주어진 시리얼 코드를 보고 이를 병렬 코드로 변환한다. 그림 2에는 병렬 프로그램의 예가 주어져 있다. main 문은 이전 시리얼 코드와 거의 동등하지만 무작위 수를 생성할 때에는 각자 rank에 주어진 것만 생성하는 것만 크게 다르다. 즉 각자의 seed 번호는 -(myid+1284) 로 주어진다. 그러면 이렇게 모든 노드에서 무작위 수가 생성되고 이를

주어진 영역 [1./nid *myid, 1./nid*(myid+1)) 안에 들어오는 수만 보유하고 다른 수들은 해당 rank로 보내는 것이다.

```c
int main(int argc, char **argv){
    long i, j;
    long np,niter,maxnp=100000000;
    float *val,**lval;

    niter = atoi(argv[1]);

    float time1, time2;
    time1 = gettime();


    val = (float*)malloc(sizeof(float)*maxnp*niter);
    lval = (float**)malloc(sizeof(float*)*niter);
    long nlval[niter];

    float step = (1.1-0.1)/(float)niter;

    for(i=0;i<niter;i++){
        lval[i] = (float*)malloc(sizeof(float)*maxnp*2);
        nlval[i] = 0;
    }
    np = 0;
    for(i=0;i<niter;i++){
        iseed = -1*(i+1284L);
        for(j=0;j<maxnp;j++){
            val[np++] = ran2(&iseed);
        }
    }
    for(i=0;i<np;i++){
        j = val[i]/step;
        *(lval[j]+nlval[j]) = val[i];
        nlval[j]++;
    }
    for(i=0;i<niter;i++){
        printf("p%d has %ld members ::: %g %g\n", (int)i,nlval[i], step*i, step*(+1));
    }
    time2 = gettime();
    printf("Wallclock time = %g second\n", (time2-time1));
}
```

```f90
      program main
      parameter(maxnp= 100000000)
      integer i,j,np,niter
      real timearray(2),time1,time2,walltime
      integer iseed
      character*80 arg
      real, allocatable, dimension(:,:) :: lval
      real, allocatable, dimension( : ) ::  val
      integer*8, allocatable, dimension( : ) ::  nlval
      real step



      call getarg(1,arg)
      read(arg,*) niter
      time1 = etime(timearray)


      step = (1.-0.)/niter
      allocate(val(1:niter*maxnp),lval(1:maxnp*2, 1:niter), nlval(1:niter))
      do i = 1, niter
         nlval(i) = 0
      enddo
      np = 1
      do i= 1, niter
          iseed = -1*(i+1283)
          do j = 1, maxnp
             val(np) = ran2(iseed)
             np = np + 1
          enddo
      enddo
      np = np - 1
      do i = 1, np
        j = val(i)/step + 1
        lval(nlval(j)+1, j) = val(i)
        nlval(j) = nlval(j) + 1
      enddo
      do i = 1, niter
      print *, 'p', (i-1), 'has ',nlval(i),'members'
      enddo

      time2 =  etime(timearray)
      walltime = time2-time1
      print *,'Wallclock time=',walltime
      stop
```

<그림 1> sequential 프로그램의 예제. 위쪽은 f90 이고 아래쪽은 C 코드

```c
void domaindecomp(float **ibase, size_t *mmem, float valmin, float valmax, MPI_Comm Comm){
    ......................
}
void Check(float*val, size_t np, float lvalmin, float lvalmax){
    size_t i;
    for(i=0;i<np;i++){
        if(val[i]<lvalmin || val[i]>=lvalmax){
            fprintf(stderr,"Error in Check %g : %g %g\n",val[i],lvalmin,lvalmax);
            exit(99);
        }
    }
}
int main(int argc, char **argv){
    long i, j;
    size_t np,maxnp=100000000;
    float *val,**lval,lvalmin,lvalmax;
    float time1, time2;
    int myid,nid;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &nid);
    if(nid != 2 && nid != 4 && nid != 8 && nid != 16 && nid != 32 && nid != 64){
        fprintf(stderr,"Error input number of parallel ranks\n");
        MPI_Finalize();
    }
    time1 = gettime();
    val = (float*)malloc(sizeof(float)*maxnp);
    float step = (1.-0.)/(float)nid;
    iseed = -1*(myid+1201);
    for(i=0;i<maxnp;i++){
        val[i] = ran2(&iseed);
    }
    np = maxnp;
    float valmin = 0.L;
    float valmax = 1.L;
    domaindecomp(&val, &np, valmin, valmax, MPI_COMM_WORLD);
    printf("p%d has %ld members\n", myid,np);
    MPI_Barrier(MPI_COMM_WORLD);
    time2 = gettime();
    if(myid==0) printf("Wallclock time = %g second\n", (time2-time1));
    lvalmin = 1./nid *(float)(myid);
    lvalmax = 1./nid *(float)(myid+1);
    Check(val,np,lvalmin,lvalmax);
    MPI_Finalize();
}
```

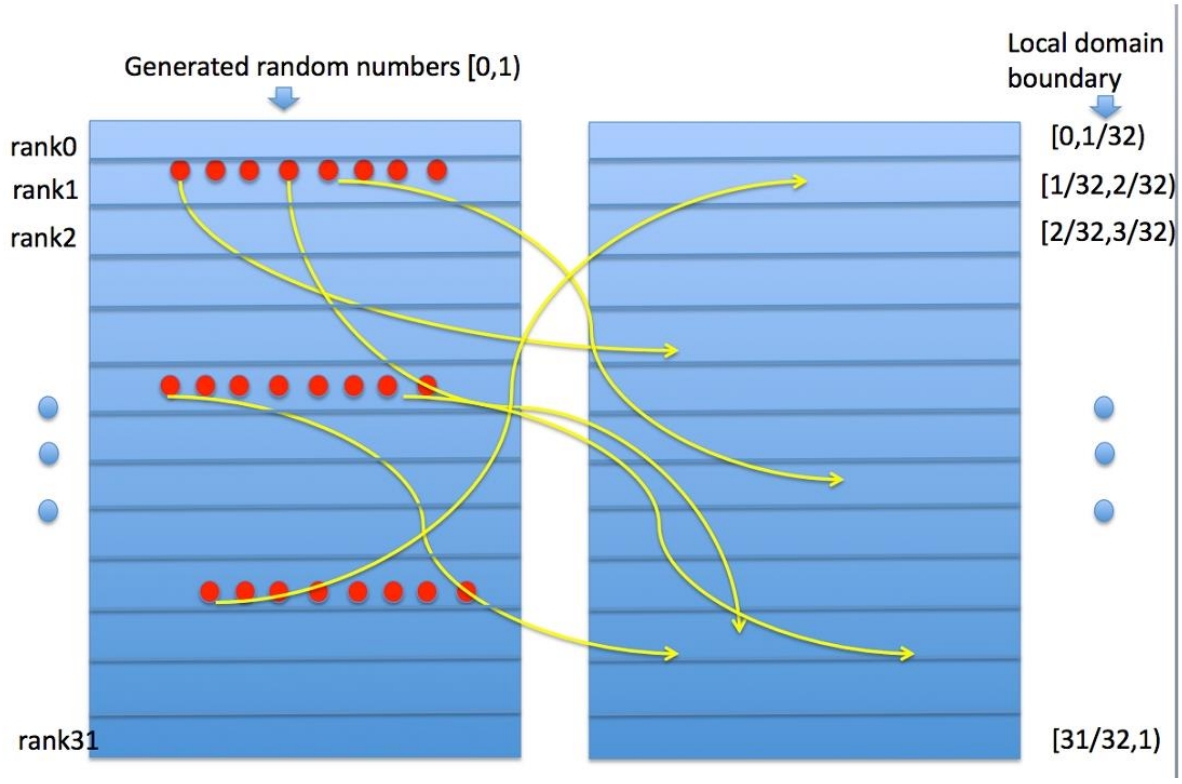
```f90
module data
   real,allocatable,dimension(:) :: base
end module data
      recursive subroutine domaindecomp(nmem, valmin, valmax, Comm)
      use data
      include 'mpif.h'
      .......
      return
      end
      program main
      use data
      include 'mpif.h'
      parameter(maxnp= 100000000)
      integer i,j
      real timearray(2),lvalmin,lvalmax
      real*8 time1,time2,walltime
      integer iseed
      character*80 arg
      integer*8 np
      real step,valmin,valmax
      integer myid,nid
      call MPI_Init(ierror)
      call MPI_Comm_size(MPI_COMM_WORLD,nid,ierror)
      call MPI_Comm_rank(MPI_COMM_WORLD,myid,ierror)
      time1 = MPI_Wtime()
      step = (1.-0.)/nid
      allocate(base(1:maxnp*3))
      iseed = -1*(myid+1284)
      do j = 1, maxnp
         base(j) = ran2(iseed)
      enddo
      np = maxnp
      valmin = 0
      valmax = 1
      call domaindecomp(np,valmin,valmax,MPI_COMM_WORLD)
      print *, 'p', myid, 'has ',np,'members'
      call MPI_Barrier(MPI_COMM_WORLD,ierror)
      time2 =  MPI_Wtime()
      walltime = time2-time1
      if(myid.eq.0) print *,'Wallclock time=',walltime
      lvalmin = 1./nid *myid
      lvalmax = 1./nid *(myid+1)
      call Check(base,np,lvalmin, lvalmax)
      call MPI_Finalize(ierror)
      stop
      end
      subroutine Check(base,np,lvalmin,lvalmax)
      integer*8 np,i
      real base(np),lvalmin,lvalmax
      do i = 1, np
         if(base(i).lt. lvalmin .or. base(i).ge.lvalmax)then
            print *,'Error in DD ', base(i),i,lvalmin,lvalmax
            stop
         endif
      enddo
      return
      end
```

<그림 2> parallel 프로그램의 형태. 위쪽은 f90 이고 아래쪽은 C 코드

이 예제 프로그램의 main 부분은 되도록 훼손하지 않고 domaindecomp라는 함수(서브루틴)의 내용을 추가하여 병렬 프로그램을 완성하는 것이 이번 문제의 목표이다(그림 2에서 "……." 로 표현된 부분; C 코드의 경우 34번째 줄, f90 코드의 경우 7번째 줄). 기타 무작위 수를 생성하는 함수들은 이전 시리얼 코드를 가져다 사용한다.

추가 설명

그림 3 은 병렬화의 개념를 대략적으로 보여준다. 왼쪽 그림은 영역분할하기 전에 0 부터 1 사이의 무작위 수를 생성(rank0 에서 rank31 까지 모든 rank 에서)하는 것을 >나타낸다. 여기서 붉은 색은 개개의 무작위 수라고 생각한다. 그리고 각각의 rank 들에 local domain boundary 설정(예를 들면 rank0 는 0 에서 1/32 까지 영역을 rank31 은 31/32 에서 1 까지 영역을)한다 (오른쪽그림의 오른편에 표시). 그리고 전체 rank 들이 갖고 있는 무작위 수들 중에 해당 영역에 들어오는 수들을 MPI 통신>을 통해서 가져온다. 이러한 통신을 얼마나 효율적으로 수행할수 있는지가 이번 문제 풀이 성공의 열쇠이다.



<그림 3> 병렬화 개념도

컴파일과 실행 방법

(1) 시리얼 코드의 컴파일은 아래와 같이 수행한다.

%> gcc –o sequential.c.exe sequential.c –lm

%> gfortran –o sequential.f90.exe sequential.f90 –lm

생성된 실행파일은 아래와 같이 수행한다.

%> ./sequntial.[c 또는 f90].exe 32

(2) 그리고 병렬 코드의 컴파일은

%> mpicc –o parallel.c.exe parallel.c –lm

%> mpif90 –o parallel.f90.exe parallel.f90 –lm

와 같은 방법을 사용한다.

실행은

%> mpirun –np 32 ./parallel.c.exe

%> mpirun –np 32 ./parallel.f90.exe

등과 같이 수행할 수 있다.

실행시에 Check()이라는 함수를 통해서 영역분할된 데이터에 문제가 있을 경우 에러 메시지를 내고 종료된다. 또한 시리얼 코드의 결과와 동일한 결과가 얻어져>야 한다. 컴파일 시에 어떠한 Optimization 옵션도 사용하지 않는다. 또한 AVX 등의 벡터 연산과 관련된 어떤 옵션이나 내장함수를 사용할 수 없다.

평가 방법

(1) 학부 32 CPU, 대학원 64 CPU 를 이용해서 병렬화된 코드 수행 시간을 `time` 으로 측정. Reference 병렬 코드 대비 실행 시간 비율로 순위 결정.

(2) 실행 시간 측정은 시리얼 코드의 경우는

%> time ./sequential.c[f90].exe 32(or 64)

병렬 코드의 경우는

%> time mpirun –np 32 (or 64) ./parallel.c[f90].exe
방식으로 수행하여 최종으로 프린트된 전체 시간을 가지고 평가한다.

(3) 시리얼 코드와 병렬 코드의 실행 결과 (해당 영역의 입자 개수)는 동일해야 한다. 그리고 Check() 호출 후에 정상적으로 종료해야 한다. 본 조건이 만족되지 않는 경우 결격처리한다.