Hacettepe Üniversitesi
Bilgisayar Mühendisliği Bölümü
BBM442 Koşut İşlem
Genel Sınav 04-06-2014

NOT: Sadece iki soruyu yanıtlayınız. Soruların ağırlıkları eşittir.

Soru 1. "Paylaşımlı Bellekli MIMD" mimaride, MPI işlevlerini kullanan ve "Dağıtılmış Bellekli MIMD" için geliştirilmiş yazılımları çalıştırmak için bir gereksinim olduğunu varsayınız. "MPI_Scatter" işlevini "Paylaşımlı Bellekli MIMD" mimaride gerçekleştirmek için:

a) Tasarım yapınız,

b) İşlevleri C Programlama Diliyle kodlayınız.

(Not: Gerçekleştiriminizde **MPI_Barrier(MPI_Comm comm)** işlevini kullanabilirsiniz)

int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype senddatatype, void *recvbuf, int recvcount, MPI_Datatype recvdatatype, int source, MPI_Comm comm)
The source process sends a different part of the send buffer sendbuf to each processes, including itself. The data that are received are stored in recvbuf. Process i receives sendcount contiguous elements of type senddatatype starting from the i * sendcount location of the sendbuf of the source process (assuming that sendbuf is of the same type as senddatatype). MPI_Scatter must be called by all the processes with the same values for the sendcount, senddatatype, recvcount, recvdatatype, source, and comm arguments. Note again that sendcount is the number of elements sent to each individual process.

Soru 2. Aşağıda CUDA uyumlu bir grafik işlemcinin çok sayıda çekirdeğini koşut olarak kullanan bir kod kesimi verilmiştir (NVIDIA'nın CUDA teknolojisi). Bu örnekte C dizisinin her elemanı bir izlek (thread) tarafından hesaplanmaktadır. Örnek kod, dizideki eleman sayısı ile izlek sayısının eşit olmasını gerektirmektedir. Dizideki eleman sayısının izlek sayısının N katı olması durumu için `VecAdd` işlevini (kernel) yeniden kodlayınız.

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
int i = threadIdx.x;
C[i] = A[i] + B[i];
}

int main()
{
...
// Kernel invocation with N threads
VecAdd<<<1, N>>>(A, B, C);
...
}
```

Soru 3. Aşağıda Cannon matris çarpma algoritmasının ana döngüsü verilmiştir. Bu döngüde bir işlem ve iki iletişim adımı ardışık olarak gerçekleştirilmektedir. Sadece iki iletişim adımını çakıştıracak biçimde döngüyü yeniden kodlayınız.

```
MatrixMatrixMultiply(int n, double *a, double *b, double *c,
             MPI_Comm comm)
 {
      ...

      /* Get into the main computation loop */
    for (i=0; i<dims[0]; i++) {
      MatrixMultiply(nlocal, a, b, c); /*  c=c+a*b  */

      /* Shift matrix a left by one */
      MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE,
          leftrank, 1, rightrank, 1, comm_2d, &status);

      /* Shift matrix b up by one */
      MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE,
          uprank, 1, downrank, 1, comm_2d, &status);
    }
      ...
}
```

Kullanabileceğiniz MPI işlevlerinden bazıları:

```
int MPI_Init(int *argc, char ***argv)

int MPI_Finalize()

int MPI_Comm_size(MPI_Comm comm, int *size)

int MPI_Comm_rank(MPI_Comm comm, int *rank)

int MPI_Send(void *buf, int count, MPI_Datatype datatype,int dest, int
                tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype,int source, int
                tag, MPI_Comm comm, MPI_Status *status)

int MPI_Isend(void *buf, int count, MPI_Datatype datatype,

        int dest, int tag, MPI_Comm comm, MPI_Request *request)

int MPI_Irecv(void *buf, int count, MPI_Datatype datatype,

        int source, int tag, MPI_Comm comm, MPI_Request *request)

int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

int MPI_Wait(MPI_Request *request, MPI_Status *status)

int MPI_Sendrecv(void *sendbuf, int sendcount,MPI_Datatype senddatatype,
                int dest, int sendtag,void *recvbuf, int recvcount,
                MPI_Datatype recvdatatype,int source, int recvtag,
                MPI_Comm comm,MPI_Status *status)
```

Kullanabileceğiniz değişmezler: `MPI_COMM_WORLD,MPI_ANY_SOURCE,MPI_ANY_TAG`