第十章作业

钟赟 2016K8009915009

1. **请介绍 MPI 中阻塞发送 MPI_Send/阻塞接收 MPI_Recv 与非阻塞发送 MPI_Isend/非阻塞接收 MPI_Irecv 的区别。**

阻塞发送：消息已经发送后，即到达消息缓冲区后，才能执行接下来的语句。

阻塞接收：进程处于阻塞状态，直到在消息缓冲区接收到消息后，才能执行下一语句。

非阻塞发送：发送原语通知系统将要发送的消息在消息缓冲区后，即可返回，无需等到消息从本地送出即可执行下一语句。

非阻塞接收：接收原语无需等待消息缓冲区是否已经接收到发送的消息，直接执行下一语句。

阻塞通信和非阻塞通信均属于异步通信机制，阻塞通信机制中保证了缓冲区等资源的可用，非阻塞通信不能保证缓冲区等资源的可用。

2. **请介绍什么是规约（Reduce）操作，MPI 和 OpenMP 中分别采用何种函数或子句来实现规约操作。**

规约操作：数据规约指通过规约函数将复杂的数据分成一批较小的数据。

MPI 中通过一下函数实现规约操作：

```
int MPI_Reduce(                 /*在每个进程上都有一组输入元素，输出规约的结果*/
    void *          input_data,  /*指向发送消息的内存块的指针 */
    void *          output_data, /*指向接收（输出）消息的内存块的指针 */
    int             count,       /*数据量*/
    MPI_Datatype datatype,       /*数据类型*/
    MPI_Op          operator,    /*规约操作，包含 12 种常用操作：加减乘除、逻辑操作等*/
    int             dest,        /*要接收（输出）消息的进程的进程号*/
    MPI_Comm        comm         /*通信器，指定通信范围*/
);
int MPI_Allreduce(
    void *          send_data,
    void *          recv_data,
    int             count,
    MPI_Datatype datatype,
    MPI_Op          op,
    MPI_Comm        communicator
);      /*与 MPI_Reduce 基本相同，但是将规约结果分发到所有进程，因此不需要根进程标识*/
```

OpenMP 中通过 reduction 子句实现规约操作，规约操作包括加、减、乘、与(and)、或(or)、相等(eqv)、不相等(neqv)、最大(max)、最小(min)等，其格式为

reduction(reduction-identifier:list)

3. **请介绍什么是栅障（Barrier）操作，MPI 和 OpenMP 中分别采用何种函数或者命令来实现栅障。**

栅障操作：栅障是一类同步屏障指令，作用在与保证操作的相对顺序。设置某个操作点(即栅障)，使线程必须等待其他线程都到达该栅障之后才可以进行之后的操作。

MPI 通过函数 MPI_Barrier(MPI_Comm comm) 实现栅障操作。

OpenMP 中通过 barrier 编译制导语句实现栅障操作，语句格式如下：

#pagma omp atomic newline

4. 下面的 MPI 程序片段是否正确？请说明理由。假定只有 2 个进程正在运行且 mypid 为每个进程的进程号。

```
If(mypid==0){
        MPI_Bcast(buf0,count,type,0,comm,ierr);
        MPI_Send(buf1,count,type,1,tag,comm,ierr);
}else{
        MPI_Rcev(buf1,count,type,0,tag,comm,ierr);
        MPI_Bcast(buf0,count,type,0,comm,ierr);
}
```

不正确。

MPI_Bcast(buffer,count,datatype,root,comm) 是从一个序列号为 root 的进程将一条消息广播发送到组内的所有进程，不可以将 MPI_Bcast 等同为多个 MPI_Send 语句。因此，不能用 MPI_Recv 和 MPI_Bcast 匹配。
集体通信中，这两个进程应按照相同的顺序进行，MPI_Bcast 函数应该处于所有进程的执行流程中，且对于所有进程来说，广播进程和其他进程之间的发送/接收缓冲区名字是一致的。

5. 矩阵乘法是数值计算中的重要运算。假设有一个 m×p 的矩阵 A，还有一个 p×n 的矩阵 B。令 C 为矩阵 A 与 B 的乘积，即 C=AB。C_{ij} 表示矩阵在(i, j)位置处的值，则 0≤i≤m-1，0≤j≤n-1。请采用 OpenMP，将矩阵 C 的计算并行化。假设矩阵在存储器中按行存放。

```
#include<stdio.h>
#include<stdlib.h>
#include<omp.h>

//suppose Matrix A and B are known
double A[m][p], B[p][n];
double C[m][n];
int main()
{
    int i, j, k;
    #pragma omp parallel for shared (A, B, C), private(i, j, k)
    for(i = 0; i < m; i ++)
        for(j = 0; j < n; i ++){
            C[i][j] = 0;
            for(k = 0; k < p; k ++)
                C[i][j] = A[i][k] * B[k][j];
        }
    return 0;
}
```

6. 请采用 MPI 将上题中的矩阵 C 的计算并行化，并比较 OpenMP 与 MPI 并行程序的特点。

```
#include<stido.h>
#include "mpi.h"
int main(int argc, char **argv){
    double *A, *B, *C;
    int i, j, k;
    int ID, num_procs, line;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &ID);
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    //分配数据空间
    A = (double *)malloc(sizeof(double)*m*p);
    B = (double *)malloc(sizeof(double)*p*n);
    C = (double *)malloc(sizeof(double)*m*n);
    line = n/num_procs;
    if(ID==0){
        //初始化数据组
        for(i = 0; i < m; i ++)
            for(j = 0; j < p; i ++)
                A[i * n + j] = 1.0;
        for(i = 0; i < p; i ++)
            for(j = 0; j < n; i ++)
                B[i * p + j] = 1.0;
        //将 A、B 矩阵的相应数据发送给从进程
        for(i = 1; i < num_procs; i ++){
```

```
                MPI_Send(B, n*n, MPI_DOUBLE, i,0, MPI_COMM_WORLD);
                MPI_Send(A+(i-1)*line*n,line*n,MPI_DOUBLE,i,1,MPI_COMM_WORLD);
            }
            //接收从进程的计算结果
            for(i = (num_procs-1)*line; i < m; i ++)
                for(j = 0; j < n; j ++){
                    C[i*n+j] = 0;
                    for(k = 0; k < p; k ++)
                        C[i*n+j] += A[i*p+k]*B[k*n+j];
                }
        }else{//其他进程接收数据，计算结果，发送给主进程
            MPI_Recv(B,m*n,MPI_DOUBLE,0,0,MPI_COMM_WORLD, &status);
            MPI_Recv(A+(ID-1)*line*n,line*n,MPI_DOUBLE,0,1,MPI_COMM_WORLD, &status);
            for(i = (ID-1)*line; i < ID*line; i ++)
                for(j = 0; j < m; j++){
                    C[i*n+j] = 0;
                    for(k = 0; k < m; k ++)
                        C[i*m+j] += A[i*p+k]*B[k*n+j];
                }
            MPI_Send(C+(num_procs-1)*line*n,line*n,MPI_DOUBLE,0,2,MPI_COMM_WORLD);
        }
        MPI_FINALIZE();
        return 0;
}
```

7. 分析一款 GPU 的存储层次。

以 NVIDIA GeForce GTX780 为例，分析 GPU 的存储层次，示意图如下：

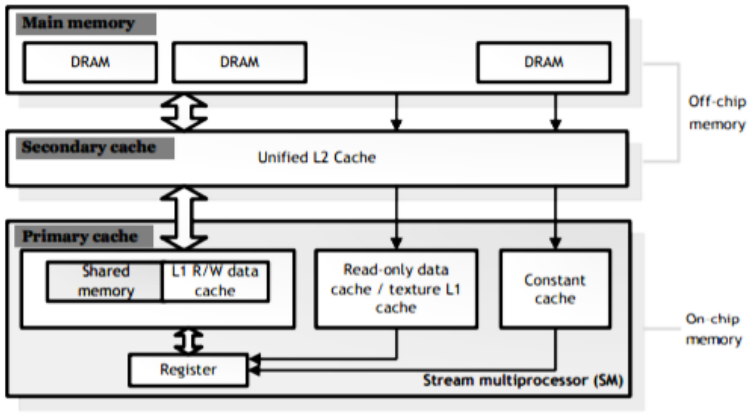

Fig. 1. Memory hierarchy of the GeForce GTX780 (Kepler).

GPU 内存空间包括 6 中类型：寄存器、恒定内存、共享内存、纹理存储器、本地内存和全局内存。GTX780 存储结构中包含片外全局存储器，通过 PCIe 总线和 CPU 间进行数据传输。片上的计算核通过两级纹理缓存和常量缓存来访问只读纹理存储器和常量寄存器。SM(Stream Multiprocessor)包含片上共享存储器和寄存器。共享存储器被动态地划分给进程块，寄存器被动态的划分给 SM 上的活跃进程块，再静态地分配给给定线程。如果分配给线程的寄存器不够，数据会溢出到全局存储器中。

GPU GTX780 采取多级存储层次，有利于实现应用的并行性。通过有效地利用片上各级存储层次，可以到达各级存储的负载均衡，在一定程度上避免了全局存储器和处理器之间的带宽的限制。

(Source Web: https://arxiv.org/ [PDF]*Dissecting GPU Memory Hierarchy through Microbenchmarking*.)