Computer Architecture: Parallel Programming **Fall 2024**

Homework 10 — Novmember 25

Lecturer: Hu Weiwu *Completed by: 2022K8009929010 Zhang Jiawei*

10.1

在阻塞方式中, 必须等到消息从本地送出/本地接收到消息之后才可以执行后续的语句, 保证了缓冲区等资源可再用; 对于非阻塞方式, 无须等到消息从本地送出/本地接收到消息就可执行后续的语句, 从而允许通信和计算的重叠, 但非阻塞调用的返回并不保证资源的可再用性。

10.2

归约操作是一种将集合中的所有元素通过某种方式组合起来以生成单一结果的操作，如求和、求积、求最大值、求最小值等。

在 MPI 中，归约操作由 `MPI_Reduce` 和 `MPI_Allreduce` 来实现，其中 `MPI_Reduce` 将每个进程的数据发送到根进程，根进程对所有数据进行归约操作，然后将结果发送回每个进程，而 `MPI_Allreduce` 还将结果发送回每个进程。

在 OpenMP 中，归约操作由 `reduction(reduction-identifier:list)` 来实现，其中 `reduction-identifier` 是归约操作的标识符，`list` 是需要进行归约操作的变量列表。

10.3

栅障操作是一种同步机制，用于协调多个进程或线程，使它们在某个点上等待，直到所有参与的进程或线程都到达该点后才能继续执行。

在 MPI 中，栅障操作由 `MPI_Barrier` 来实现，该函数会阻塞调用进程，直到所有进程都调用了 `MPI_Barrier` 函数。

在 OpenMP 中，栅障操作由 `#pragma omp barrier newline` 来实现，该指令会阻塞调用线程，直到所有线程都到达该点。

10.4

这个程序片段是正确的。进程 0 先广播 `buf0`，然后阻塞发送 `buf1`；进程 1 先阻塞接收 `buf1`，然后广播 `buf0`。程序中发送接收配对，也没有集体通信函数的死锁问题，因此程序是正确的。

10.5

所写程序如下：

```
1   void matrix_multiply(int m, int p, int n, double **A, double **B, double **C) {
2      int i, j, k;
3
4      #pragma omp parallel for private(i, j, k) shared(A, B, C)
5      for (i = 0; i < m; i++) {
6         for (j = 0; j < n; j++) {
7            C[i][j] = 0.0;
8            for (k = 0; k < p; k++)
9               C[i][j] += A[i][k] * B[k][j];
10        }
```

```
    }
  }
```

其中 #pragma omp parallel for private(i, j, k) shared(A, B, C) 指定了并行执行的循环，private(i, j, k) 指定了循环变量 i, j, k 为私有变量，shared(A, B, C) 指定了矩阵 A, B, C 为共享变量。

10.6

所写程序如下：

```
void matrix_multiply(int m, int p, int n, double *A, double *B, double *C, int
    size) {
   int i, j, k;
   int rows_per_proc = m / size;
   double *local_A = (double *)malloc(rows_per_proc * p * sizeof(double));
   double *local_C = (double *)malloc(rows_per_proc * n * sizeof(double));

   // Scatter rows of A to all processes
   MPI_Scatter(A, rows_per_proc * p, MPI_DOUBLE, local_A, rows_per_proc * p,
       MPI_DOUBLE, 0, MPI_COMM_WORLD);

   // Broadcast B to all processes
   MPI_Bcast(B, p * n, MPI_DOUBLE, 0, MPI_COMM_WORLD);

   // Perform local computation
   for (i = 0; i < rows_per_proc; i++) {
      for (j = 0; j < n; j++) {
         local_C[i * n + j] = 0.0;
         for (k = 0; k < p; k++)
            local_C[i * n + j] += local_A[i * p + k] * B[k * n + j];
      }
   }

   // Gather results from all processes
   MPI_Gather(local_C, rows_per_proc * n, MPI_DOUBLE, C, rows_per_proc * n,
       MPI_DOUBLE, 0, MPI_COMM_WORLD);

   free(local_A);
   free(local_C);
}
```

程序中首先计算每个进程需要处理的行数，然后使用 MPI_Scatter 将矩阵 A 的行分发给各个进程，使用 MPI_Bcast 广播矩阵 B，然后在各个进程上进行局部计算，最后使用 MPI_Gather 将各个进程的计算结果收集到进程 0。

比较来说，MPI 的实现更加灵活，可以更好地控制数据的分发和收集，而 OpenMP 的实现更加简单，但是只能在单个节点上并行计算。

10.7

GPU 存储层次示意图如下：

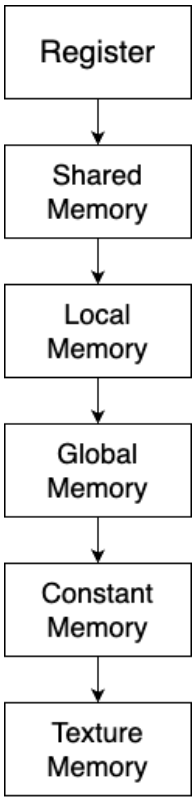

图 **10.1.** GPU 存储层次示意图

其中，寄存器是 GPU 中最快的存储器，用于存储线程的局部变量；共享内存是 GPU 中的共享存储器，用于存储线程块的共享变量；本地内存是 GPU 中的全局存储器，用于存储线程块的全局变量；全局内存是 GPU 中的全局存储器，用于存储所有线程的全局变量；常量内存是 GPU 中的全局存储器，用于存储只读数据；纹理内存是 GPU 中的全局存储器，用于存储纹理数据。