**体系结构 第十章**

**陈彦帆 2018K8009918002**

1. 阻塞发送(MPI\_Send) 直到接收方接收完信息才会返回。

阻塞接收(MPI\_Recv) 直到接收完信息才会返回。

非阻塞发送(MPI\_Isend) 立即返回，但不保证消息被接收。需要后续查询(MPI\_Test)或等待(MPI\_Wait)消息被接收。

非阻塞接收(MPI\_Irecv) 立即返回，但不保证消息被接收。需要后续查询或等待消息被真正接收。

2. 归约操作：把多个元素通过适当的操作合并为一个元素。

MPI\_Reduce (sendbuf, recvbuf, count, type, op, rootID, communicator)

持有rootID的线程从所有线程中各接收长度为count，类型为type的信息，并且运行类型为op的计算，把结果放在recvbuf。

OpenMP: reduction字句：reduction (op:list)

对作用域中list中的变量作op操作的归约。

3. 栅障操作：在指定位置同步所有线程。

MPI:

int MPI\_Barrier(MPI\_Comm comm)

OpenMP:

#pragma omp barrier newline

4. 进程0发送了2个count的数据，而进程0只接收了1个count的数据，进程0被无限阻塞。

错误点在于，Bcast被写在了if条件语句内，导致对于进程0，只有发送，没有接收，进程1的Bcast也同理。Bcast应写在if语句花括号之外。

5.

#include <stdio.h>

#include <omp.h>

#define m 800

#define p 1000

#define n 1200

double A[m][p];

double B[p][n];

double C[m][n];

int main()

{

for(int i=0;i<m;i++)

for(int j=0;j<p;j++)

A[i][j] = 1.0;

for(int i=0;i<p;i++)

for(int j=0;j<n;j++)

B[i][j] = 1.0;

#pragma omp parallel for

for(int i=0;i<m;i++){

for(int k=0;k<p;k++){

double r = A[i][k];

for(int j = 0;j<n;j++){

C[i][j] += r\*B[k][j];

}

}

}

return 0;

}

6.

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define m 800

#define p 1000

#define n 1200

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);

A = (double \*)malloc(sizeof(double)\*m\*p);

B = (double \*)malloc(sizeof(double)\*p\*n);

C = (double \*)malloc(sizeof(double)\*m\*n);

line = m/num\_procs;

if(ID==0){

for( i=0;i<m;i++)

for( j=0;j<p;j++)

A[i\*p+j] = 1.0;

for( i=0;i<p;i++)

for( j=0;j<n;j++)

B[i\*n+j] = 1.0;

for(i=1;i<num\_procs;i++){

MPI\_Send(B,n\*p,MPI\_DOUBLE,i,0,MPI\_COMM\_WORLD);

MPI\_Send(A+(i-1)\*line\*p,line\*p,MPI\_DOUBLE,i,1,MPI\_COMM\_WORLD);

for(i=1;i<num\_procs;i++)

MPI\_Recv(C+(i-1)\*line\*n,line\*n,MPI\_DOUBLE,i,2,MPI\_COMM\_WORLD,&status);

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{

MPI\_Recv(B,n\*p,MPI\_DOUBLE,0,0,MPI\_COMM\_WORLD,&status);

MPI\_Recv(A+(ID-1)\*line\*p,MPI\_DOUBLE,0,1,MPI\_COMM\_WORLD,&status);

for(i=(ID-1)\*line;i<ID\*line;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];

}

MPI\_Send(C+(num\_procs-1)\*line\*n,line\*n,MPI\_DOUBLE,0,2,MPI\_COMM\_WORLD);

}

MPI\_Finalize();

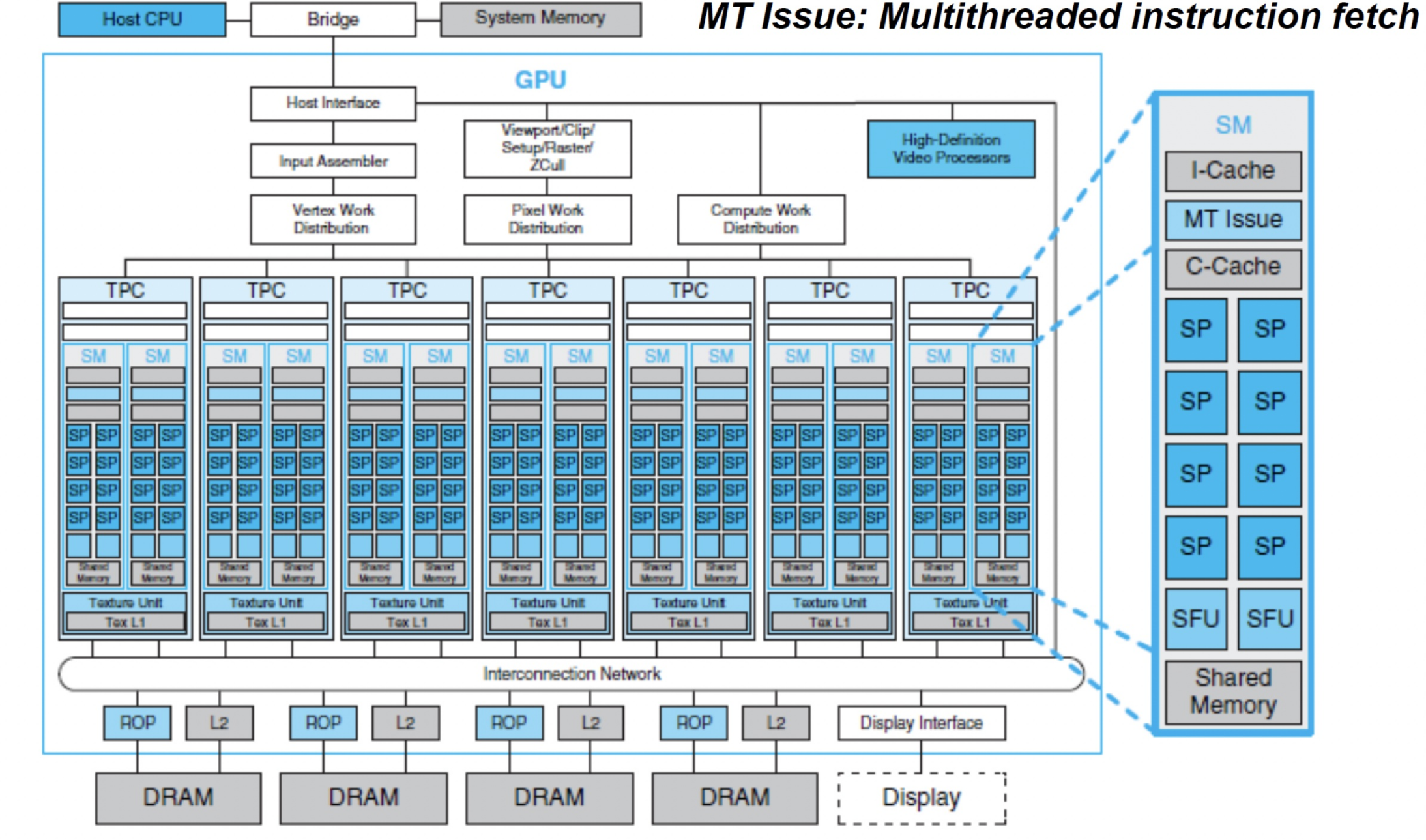
return 0;

}

比较：OpenMP程序是共享内存模型，适合单个机器的多线程编程；MPI程序通过消息传送数据，适合多台机器的并行计算。

从编程的复杂度考虑，OpenMP编程更简单。

7. Tesla架构：



GPU内部的存储层次有register，shared memory，local memory，constant memory, texture memory，global memory。

GPU的结构层次分为Grid->Block->Thread。每个thread有自己的一份register和local memory。同一个block中的每个thread有共享的一份share memory。此外，每个Grid中的Thread共享一份global memory、constant memory和texture memory。