**高性能并行计算第6次作业**

姓名：姚虎成 学号：2020317110033

**代码地址：**

MPI Greeting的C语言程序路径：/home2/2020317110033/mpi\_5/greeting.c

Pingpong的C语言程序路径：/home2/2020317110033/mpi\_5/pingpong.c

MPI 计算Pi的C语言程序路径

/home2/2020317110033/mpi\_5/mpi\_pi\_pointToPoint.c

/home2/2020317110033/mpi\_5/mpi\_pi\_point.sh

**实验结果：**

**1、MPI Greeting 程序**

使用mipcc 进行MPI程序的编译，具体代码如下：

mpicc greeting.c -o greeting

然后再使用mpirun 命令设置处理器数目，如下代码设置了4个CPU。

mpirun -np 4 ./greeting

运行结果如下图所示，第一个数字是处理器编号，第二个数字是处理器总数是4个，然后处理器的编号是从0-3，共四个处理器。

Greeting from process 0 of 4!

Greeting from process 1 of 4!

Greeting from process 2 of 4!

Greeting from process 3 of 4!

**2、MPI 点对点通信**

点对点通信采用了标准模式，两个不同的进程之间可以通过点对点通讯交换数据，一个是接收者，一个是发送者。脚本使用了MPI\_Send和MPI\_Recv,具体脚本内容见附录，通过使用以下命令编译运行。

$mpicc pingpong.c -o pingpong

$mpirun -np 2 ./pingpong

下图是该pingpong程序的运行结果，设置了两个进程，一个作为消息的发送者，一个作为消息的接收者。下面结果中，0号进程作为发送者发送了1，3，，5，7，9给1号进程，0号进程作为接收者接受了从1号进程发来的2，4，6，8，10。实现了点对点通讯。

0 sent and incremented ping\_pong\_count 1 to 1

0 received ping\_pong\_count 2 from 1

0 sent and incremented ping\_pong\_count 3 to 1

0 received ping\_pong\_count 4 from 1

0 sent and incremented ping\_pong\_count 5 to 1

0 received ping\_pong\_count 6 from 1

0 sent and incremented ping\_pong\_count 7 to 1

0 received ping\_pong\_count 8 from 1

0 sent and incremented ping\_pong\_count 9 to 1

0 received ping\_pong\_count 10 from 1

1 received ping\_pong\_count 1 from 0

1 sent and incremented ping\_pong\_count 2 to 0

1 received ping\_pong\_count 3 from 0

1 sent and incremented ping\_pong\_count 4 to 0

1 received ping\_pong\_count 5 from 0

1 sent and incremented ping\_pong\_count 6 to 0

1 received ping\_pong\_count 7 from 0

1 sent and incremented ping\_pong\_count 8 to 0

1 received ping\_pong\_count 9 from 0

1 sent and incremented ping\_pong\_count 10 to 0

**3、MPI计算Pi在不同CPU下的时间和加速比**

在计算pi的C程序时，由于计算时间过短，因此在代码实现过程中将计算step参数设置到了1亿。下表1是在不同计算规模下MPI计算Pi的时间和加速比，加速比计算使用的串行时间是之前计算的Time串行=1.670s。

可以从表中看出，随着CPU数量的增加，运行时间不断减少，加速比不断提高。MPI提升并行速度明显。

**表1 不同计算规模的时间和加速比**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CPU | 2 | 4 | 6 | 8 | 10 | 16 | 32 | 64 |
| Time（s） | 1.677 | 0.560 | 0.336 | 0.244 | 0.191 | 0.114 | 0.069 | 0.064 |
| Speedup | 0.996 | 2.983 | 4.970 | 6.834 | 8.727 | 14.613 | 24.127 | 25.977 |

下图是根据上面的表格1绘制的随着CPU数目的变化，程序运行时间的变化，以及加速比的变化。也可以直观的看出，两条折线都在CPU数目增加到最大时，趋于平稳，呈S型曲线。

**图1 CPU-Time，CPU-Speedup**

**实验分析：**

在用MPI编写greeting程序时，主要使用了几个最基本的MPI调用接口，MPI\_Init(),这是启动MPI环境的标志，标志着并行代码的开始。MPI\_COMM\_WORLD是通信子，这是MPI通信过程中不可缺少的参数，用于限定参加通信的范围，然后使用MPI\_Comm\_size(),MPI\_Comm\_rank()可以得到进程数目，以及本进程在通信空间中的编号，最后就是MPI\_Finalize()标志着并行代码的结束，后续串行代码仍然可以在主进程上运行。

在使用pingpong测试时，我们得到的结果是从0号进程发数据到1号进程，通过MPI\_Send和MPI\_Recv进行标准模式的发送和接受，MPI\_Send和MPI\_Recv函数都是阻塞的函数，他们会一直阻塞调用者，直到数据完成，阻塞型的函数操作不当很容易引起死锁。在MPI中也还有其他方式进行数据的传输和接收，如进行MPI\_Wait和MPI\_Test。

在使用MPI计算Pi的时候，从图1我们发现，在两个进程的时候，时间和串行时间接近，这可能是因为我们在编写MPI程序时，逻辑是如果两个进程，我们让主进程去做加和，其余进程做积分。所以其实两个进程时也就基本上没有加速。到了CPU达到最大的64，发现程序运行时间和加速比都有趋于平稳的趋势，这可能是我们的集群由于是docker技术虚拟出来的，并不是实际的物理机器，所以导致没有64个核，或者同一时间段还有别的同学在提交任务占用了一定的资源，所以到CPU逐渐增加到后面，运行时间和加速比逐渐趋于平稳。

**附录：**

1、MPI greeting的C语言程序代码

#include <stdio.h>

#include <string.h>

#include <mpi.h>

const int MAX\_STRING = 100;

int main(int argc, char\* argv[]){

char greeting[MAX\_STRING];

double time;

int comm\_sz;

int my\_rank;

int q;

time=MPI\_Wtime();

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &comm\_sz);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);

if(my\_rank != 0) {

sprintf(greeting, "Greeting from process %d of %d!",my\_rank,comm\_sz);

MPI\_Send(greeting, strlen(greeting)+1, MPI\_CHAR, 0, 0, MPI\_COMM\_WORLD);

}

else {

printf("Greeting from process %d of %d!\n",my\_rank, comm\_sz);

for(q = 1; q< comm\_sz;q++){

MPI\_Recv(greeting, MAX\_STRING, MPI\_CHAR, q,0,MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("%s\n",greeting);

}

}

MPI\_Finalize();

time=MPI\_Wtime()-time;

printf("parallel time:%.8f",time);

return 0;

}

2、MPI 点对点通信Pingpang代码

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char\*\* argv){

const int PING\_PONG\_LIMIT=10;

MPI\_Init(NULL,NULL);

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&world\_rank);

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD,&world\_size);

if (world\_size != 2){

fprintf(stderr,"World size must be two for %s\n",argv[0]);

MPI\_Abort(MPI\_COMM\_WORLD,1);

}

int ping\_pong\_count = 0;

int partner\_rank = (world\_rank + 1)%2;

while (ping\_pong\_count < PING\_PONG\_LIMIT){

if (world\_rank == ping\_pong\_count % 2){

ping\_pong\_count++;

MPI\_Send(&ping\_pong\_count,1,MPI\_INT,partner\_rank,0,MPI\_COMM\_WORLD);

printf("%d sent and incremented ping\_pong\_count %d to %d\n",world\_rank,ping\_pong\_count,partner\_rank);

}

else {

MPI\_Recv(&ping\_pong\_count,1,MPI\_INT,partner\_rank,0,MPI\_COMM\_WORLD,MPI\_STATUS\_IGNORE);

printf("%d received ping\_pong\_count %d from %d\n",world\_rank,ping\_pong\_count,partner\_rank);

}

}

MPI\_Finalize();

}

3、MPI计算Pi的C语言程序代码

#include <stdio.h>

#include <mpi.h>

#define N 100000000

double step = 1.0 / N;

double calculate\_pi(int id,int n){

double x,sum = 0.0;

int i;

for(i=id; i<N; i=i+n-1){

x =(i-0.5)\*step;

sum+=4.0/(1.0+x\*x);

}

return sum;

}

main(int argc, char \*argv[])

{

double pi=0.0;

double sum;

double start,end;

int id,n,source;

MPI\_Status status;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &id);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &n);

start=MPI\_Wtime();

if(id!=0){

sum=calculate\_pi(id,n);

MPI\_Send(&sum, 1, MPI\_DOUBLE, 0, 6, MPI\_COMM\_WORLD);

}else{

for (source=1;source<n;source++){

MPI\_Recv(&sum, 1, MPI\_DOUBLE, source, 6, MPI\_COMM\_WORLD, &status);

pi+=sum\*step;

}end=MPI\_Wtime();

printf("The result of pi is %lf\t Runnning time is %lf\n",pi,id,end-start);

}

MPI\_Finalize();

}