实验1 MPI

朱彤轩 191840376 2021.9.18

实验1 MPI

- 编程1
- 编程2
- 多节点多进程时间比较
  - 开方和多节点多进程比较
  - 积分多节点多进程比较
- 问题及解决方案
  - 1-运用命令行传入参数
  - 2-输出顺序错乱
  - 3-pull docker permission denied
  - 4-Docker启动ubuntu容器中使用sudo后报错

编程1

```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <mpi.h>
using namespace std;

int main(int argc, char** argv)
{
    int myid, numprocs;
    int N = atoi(argv[1]); // get N from cmd
    double totalSum=0.0, SqrtSum=0.0;
    double *data =new double[N];
    for(int i = 0; i < N; ++i){
        data[i] = i*(i+1);
    }
    //double data[]= {1,2,3,4,5,6,7,8,9,10,11};
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    --numprocs; /*数据分配时除去0号主节点*/
```

```
    if (myid == 0){ /*0号主节点，主要负责数据分发和结果收集*/
        for (int i = 0; i < N; ++i ) /*数据分发：0, */
            MPI_Send(&data[i], 1, MPI_DOUBLE, i%numprocs+1, 1 ,MPI_COMM_WORLD);
    }
    else {
        for (int i = myid-1; i < N; i=i+numprocs) /*各子节点接受数据计算开平方，本地累加
*/{
            double d;
            MPI_Recv(&d, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD, &status);
            SqrtSum+=sqrt(d);
        }
        //MPI_Reduce(&SqrtSum, &totalSum, 1, MPI_DOUBLE,MPI_SUM, 0,
MPI_COMM_WORLD);
    }
    MPI_Reduce(&SqrtSum, &totalSum, 1, MPI_DOUBLE,MPI_SUM, 0, MPI_COMM_WORLD);

    if (myid == 0){
        printf("I am process %d. SqrtSum=%f.\n", myid, totalSum);
    }
    else{
        printf("I am process %d. SqrtSum=%f.\n", myid, SqrtSum);
    }
    MPI_Finalize();
}
```

N=100时：

```
ztx@191840376:~/workspace/exp1$ mpirun -np 5 ./reduce 100
I am process 4. SqrtSum=1287.373604.
I am process 3. SqrtSum=1262.353304.
I am process 1. SqrtSum=1211.887632.
I am process 0. SqrtSum=4998.925526.
I am process 2. SqrtSum=1237.310986.
```

N=1000时：

```
ztx@191840376:~/workspace/exp1$ mpirun -np 5 ./reduce 1000
I am process 1. SqrtSum=124624.315253.
I am process 3. SqrtSum=125124.781490.
I am process 4. SqrtSum=125374.802069.
I am process 2. SqrtSum=124874.738891.
I am process 0. SqrtSum=499998.637703.
```

N=10000时：

```
ztx@191840376:~/workspace/exp1$ mpirun -np 5 ./reduce 10000
I am process 3. SqrtSum=12501249.709548.
I am process 2. SqrtSum=12498749.666921.
I am process 4. SqrtSum=12503749.730155.
I am process 1. SqrtSum=12496249.243255.
I am process 0. SqrtSum=49999998.349880.
```

编程2

```
```

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <mpi.h>

#define N 100000000
#define a 10
#define b 100
using namespace std;

int main(int argc, char** argv)
{
    int myid,numprocs;
    double local=0.0, totalSum=0.0, dx=(double)(b-a)/N; /* 小矩形宽度 */

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    --numprocs;

    if(myid==0){
        for (int source = 1; source <= numprocs; ++source) { /*结果收集*/
            double d;
            MPI_Recv(&d, 1, MPI_DOUBLE, source, 1, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            totalSum += d;
        }
    }
    else{
        for(int i=myid;i<N;i=i+numprocs) {
            double x;
            x = a + i*dx +dx/2;
            local += x*x*x*dx;
        }
        MPI_Send(&local, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD); /*本地累加结果送回主节
点*/
    }

    if(myid==0) {
        printf("I am process %d. The finalresult is %f.\n", myid, totalSum);
    }
    else{
        printf("I am process %d. My result is %f.\n", myid, local);
    }

    MPI_Finalize();
}
```

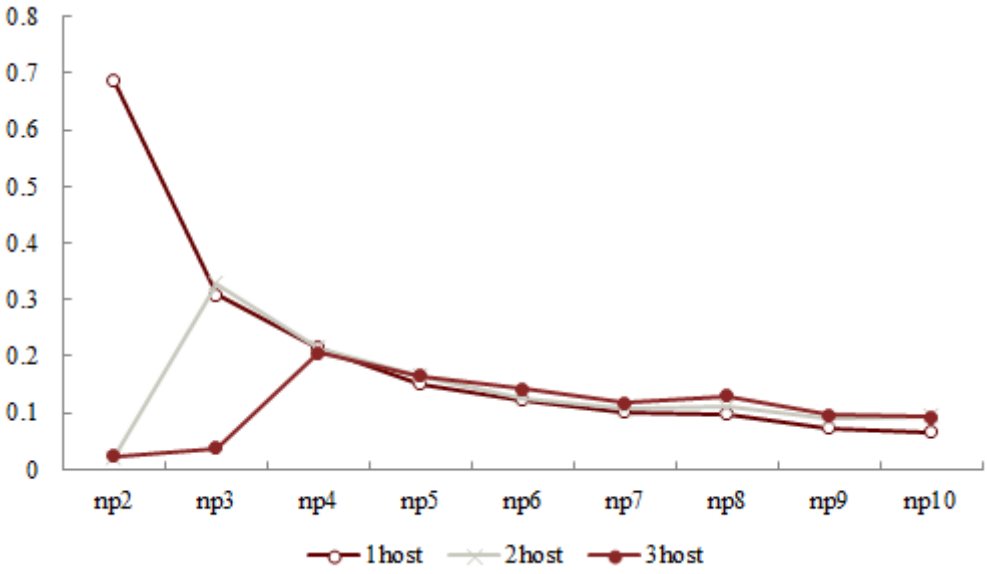
本地执行如下：

```
ztx@191840376:~/workspace/exp1$ mpirun -np 5 ./intsum
I am process 3. My result is 6249375.337162.
I am process 4. My result is 6249374.661938.
I am process 1. My result is 6249374.887612.
I am process 2. My result is 6249375.112387.
I am process 0. The finalresult is 24997499.999100.
```

多节点多进程时间比较

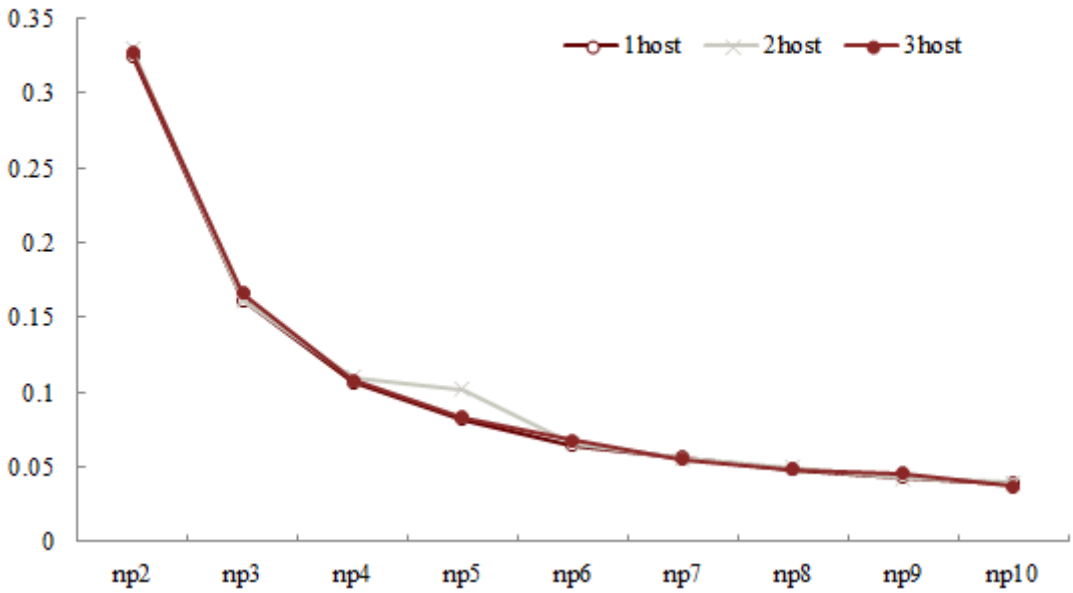
在下面的操作中，我测试了节点数目为1-3，进程数为2-10的程序的执行时间，并绘制出图像。对于计时，我选择用 `ctime` 头文件中的clock函数对主进程进行计时，根据mpi的工作原理，主进程用于首发数据，一般是最后结束的一个进程。Linux自带了 `time + 操作` 的计时方法，但是由于会记录程序加载时间，故还是选用直接在程序内部进行计时。

• 开方和多节点多进程比较



可以看出在总体而言，随着进程数的增加，程序运行的时间降低。但是在节点数为2和3的时候出现了异常，可能在进程数少的时候，节点数增加所带来的通讯代价远大于进程数和节点数增加带来的效率提升，所以出现了时间增加的情况。

• 积分多节点多进程比较



在积分程序中，选取节点的个数对于执行时间没有明显的影响，随着进程数的增加，程序运行速度有明显的提升。这可能是因为增加节点所带来的效率提升与节点之间通讯时间带来的额外花费所相抵消，故增加节点个数对于程序效率提升不明显。

问题及解决方案

1-运用命令行传入参数

需要用命令行传入编程1的参数，

2-输出顺序错乱

```
ztx@191840376:~/workspace/exp1$ mpiexec -n 5  ./reduce 11
I am process I am process 1SqrtSum=6.23607
I am process 3SqrtSum=7.69443
4SqrtSum=4.82843
I am process 2SqrtSum=7.02598
I am process 0SqrtSum=25.7849
ztx@191840376:~/workspace/exp1$
```

使用c++的cout进行输出，可以看出输出时候有些句子是错乱的。推测是因为cout的 `<<` 操作是链式操作，其实是分开的，由于缓冲区的存在导致多个进程的输出混在一起。

后改用c语言中的printf方式避免这种情况。

3-pull docker permission denied

```
ztx@191840376:~/workspace/exp1$ docker pull ubuntu:16.04
Got permission denied while trying to connect to the Docker daemon socket at
unix:///var/run/docker.sock: Post
http://%2Fvar%2Frun%2Fdocker.sock/v1.39/images/create?fromImage=ubuntu&tag=16.04:
dial unix /var/run/docker.sock: connect: permission denied
```

docker安装完成，一般用户没有权限启动docker服务，只能通过sudo来通过root用户权限来启动docker，此时对于一般用户而言，需要执行docker ps或者docker images命令查看容器或者镜像提示如题所示的错误。

把普通用户加入到docker组中，执行：

```
ztx@191840376:~/workspace/exp1$ sudo gpasswd -a $USER docker
Adding user ztx to group docker
ztx@191840376:~/workspace/exp1$ newgrp docker
```

在进行 `docker pull ubuntu:16.04` 可以进行docker的安装。

4-Docker启动ubuntu容器中使用sudo后报错

```
ztx@191840376:~/workspace/exp1$ docker exec -it origin16 bash
root@origin16:/# sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak
bash: sudo: command not found
```

需更新下软件源

apt-get update

apt-get install sudo