分布式并行计算实验指导书

电子科技大学计算机学院

## N-Body问题并行程序设计及性能优化

## 一．实验目的

1. 使用CUDA编程环境实现 N-Body并行算法。

2. 掌握CUDA程序进行性能分析以及调优方法。

## 二．实验内容

### 1. 实验环境

计算节点配置：

CPU E5-2660 v4\*2

Nvidia K80\*2

操作系统：CentOS 7.2

CUDA：10.0

### 2. 实验内容

（1）学习和使用集群及CUDA编译环境

（2）基于CUDA实现N-Body 程序并行化

（3）N-Body并行程序的性能优化

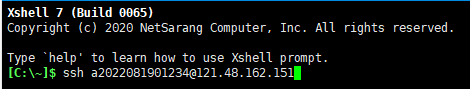
### 3. 实验步骤

（1）使用远程连接工具通过ssh方式登录集群。**注意：需要在校园网环境下才能连接！**

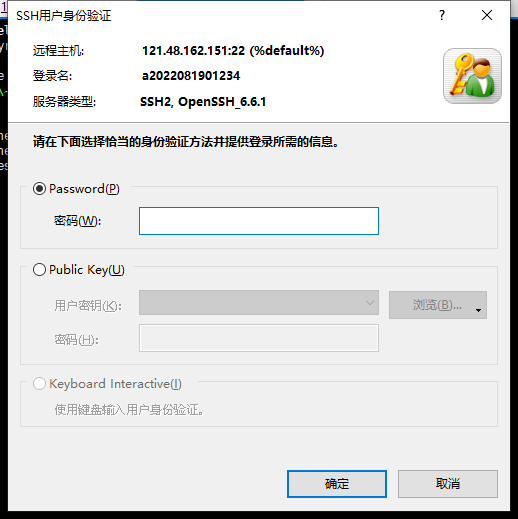
每位同学的账号为：std学号

密码：123456

以X shell 为例：



在弹出的页面输入密码：



登录成功：



（2）实验集群分为管理节点mu01和运行节点cu04、cu05。登录集群后默认进入管理节点：mu01。在此节点上传代码，修改代码。



（3）运行代码需要进入运行节点cu04或cu05。在管理节点使用如下命令进入运行节点：

ssh cu04

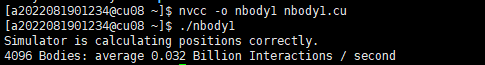


（4）进入运行节点后使用如下命令编译代码：

nvcc -o compiledFile sourceFile.cu

使用如下命令运行编译后的文件：

./compiledFile



注意点：

a. 需要在校园网环境下才能连接集群。

b. CUDA源程序后缀名为 “.cu”，编译前请注意修改源文件后缀名。

### 4. N-Body串行代码

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include "timer.h"

#include "check.h"

#include <cuda\_runtime.h>

#define SOFTENING 1e-9f

/\*

\* Each body contains x, y, and z coordinate positions,

\* as well as velocities in the x, y, and z directions.

\*/

typedef struct { float x, y, z, vx, vy, vz; } Body;

/\*

\* Do not modify this function. A constraint of this exercise is

\* that it remain a host function.

\*/

void randomizeBodies(float \*data, int n) {

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

data[i] = 2.0f \* (rand() / (float)RAND\_MAX) - 1.0f;

}

}

/\*

\* This function calculates the gravitational impact of all bodies in the system

\* on all others, but does not update their positions.

\*/

void bodyForce(Body \*p, float dt, int n) {

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

float Fx = 0.0f; float Fy = 0.0f; float Fz = 0.0f;

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

float dx = p[j].x - p[i].x;

float dy = p[j].y - p[i].y;

float dz = p[j].z - p[i].z;

float distSqr = dx\*dx + dy\*dy + dz\*dz + SOFTENING;

float invDist = rsqrtf(distSqr);

float invDist3 = invDist \* invDist \* invDist;

Fx += dx \* invDist3; Fy += dy \* invDist3; Fz += dz \* invDist3;

}

p[i].vx += dt\*Fx; p[i].vy += dt\*Fy; p[i].vz += dt\*Fz;

}

}

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

/\*

\* Do not change the value for `nBodies` here. If you would like to modify it,

\* pass values into the command line.

\*/

int nBodies = 2<<11;

int salt = 0;

if (argc > 1) nBodies = 2<<atoi(argv[1]);

/\*

\* This salt is for assessment reasons. Tampering with it will result in automatic failure.

\*/

if (argc > 2) salt = atoi(argv[2]);

const float dt = 0.01f; // time step

const int nIters = 10; // simulation iterations

int bytes = nBodies \* sizeof(Body);

float \*buf;

buf = (float \*)malloc(bytes);

Body \*p = (Body\*)buf;

/\*

\* As a constraint of this exercise, `randomizeBodies` must remain a host function.

\*/

randomizeBodies(buf, 6 \* nBodies); // Init pos / vel data

double totalTime = 0.0;

/\*

\* This simulation will run for 10 cycles of time, calculating gravitational

\* interaction amongst bodies, and adjusting their positions to reflect.

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Do not modify these 2 lines of code.

for (int iter = 0; iter < nIters; iter++) {

StartTimer();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* You will likely wish to refactor the work being done in `bodyForce`,

\* as well as the work to integrate the positions.

\*/

bodyForce(p, dt, nBodies); // compute interbody forces

/\*

\* This position integration cannot occur until this round of `bodyForce` has completed.

\* Also, the next round of `bodyForce` cannot begin until the integration is complete.

\*/

for (int i = 0 ; i < nBodies; i++) { // integrate position

p[i].x += p[i].vx\*dt;

p[i].y += p[i].vy\*dt;

p[i].z += p[i].vz\*dt;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Do not modify the code in this section.

const double tElapsed = GetTimer() / 1000.0;

totalTime += tElapsed;

}

double avgTime = totalTime / (double)(nIters);

float billionsOfOpsPerSecond = 1e-9 \* nBodies \* nBodies / avgTime;

#ifdef ASSESS

checkPerformance(buf, billionsOfOpsPerSecond, salt);

#else

checkAccuracy(buf, nBodies);

printf("%d Bodies: average %0.3f Billion Interactions / second\n", nBodies, billionsOfOpsPerSecond);

salt += 1;

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Feel free to modify code below.

\*/

free(buf);

}

## 三．实验要求

1、源代码中，\*\*\*\*注明不能修改的代码区间内，不允许插入/修改代码。

2、实验基础分部分：源文件命名为“nbody1.cu”。

3、性能分部分：源文件命名为“nbody2.cu”

4、需提交两个源代码和实验报告。

5、目录结构为（注意目录需要以“学号\_姓名”命名）：

学号\_姓名（目录名）

├─ nbody1.cu

├─ nbody2.cu

└─ 实验报告.docx

## 四．实验评分

CUDA实验分为两个部分：

1、基础实验分（70%）：完成串行改并行代码,代码运行结果正确。

2、性能分（30%）：根据4096粒子仿真时的实测性能进行打分。全班最高加速性能者，性能分得满分，性能最低为0分，其余同学性能得分则根据其实测性能，在全班最高加速性能A与最低性能之间进行插值。

## 实验报告

1．程序说明。说明程序的功能、结构。

2．调试说明。包括上机调试的情况、上机调试步骤、调试所遇到的问题是如何解决的，并对调试过程中的问题进行分析，对执行结果进行分析。

3．给出源程序清单并清晰注释。

4. 给出程序运行结果。

5. 详细说明优化的思路、采用的技术以及优化的效果

6. 实验报告需附上可见的手写(电子)签名。

**电 子 科 技 大 学**

**实 验 报 告**

|  |  |
| --- | --- |
| **学生姓名：** | **学 号：** |
| **一、实验室名称：** | |
| **二、实验项目名称：** | |
| **三、实验原理：** | |
| **四、实验目的：** | |
| **五、实验内容：** | |
| **六、实验器材（设备、元器件）：** | |
| **七、实验步骤及操作：** | |
| **八、实验数据及结果分析：** | |
| **九、实验结论：** | |
| **十、总结及心得体会：** | |
| **十一、对本实验过程及方法、手段的改进建议：** | |
| **报告评分：**  **指导教师签字：** | |