**北京邮电大学软件学院**

**2020-2021学年第一学期实验报告**

**课程名称：** 并行计算

**项目名称：** 天体运动模拟

**项目完成人：**

**指导教师：** 卢本捷

**日 期： 2020年 12 月 14 日**

1. **实验目的**

**以分治的方法设计并实现天体运动模拟。**

1. **实验内容**
2. **在二维平面上以牛顿经典力学模拟宇宙运行模型。**
3. **假设有10000个星球。**
4. **星球的质量、位置、初始速度以随机的方式生成。**
5. **每秒钟进行一次计算。**
6. **以点绘图的方式描述天体的位置及其变化。**
7. **实验环境**
8. **两台或以上的windows 或linux等。**
9. **Visual studio 2019**
10. **实验要求**
    1. **建立Barnes Hut 模型**
    2. **以天体位置为依据建立4叉树。**
    3. **计算质量与中心**
    4. **计算每个点的受力与加速度。**
    5. **更新天体位置。**
    6. **观察最后的宇宙运行情况。**
11. **MPICH 实验步骤**
12. **问题描述与分析**

N体问题是指找出已知初始位置、速度和质量的多个物体在经典力学情况下的后续运动。在本次实验中，你需要模拟N个物体在二维空间中的运动情况。通过计算每两个物体之间的相互作用力，可以确定下一个时间周期内的物体位置。

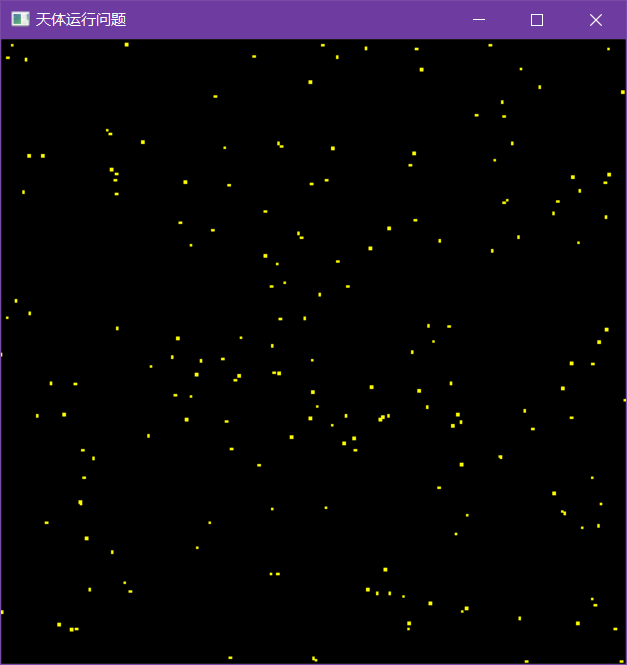
在本次实验中，N个小球在均匀分布在一个正方形的二维空间中，小球在运动时没有范围限制。每个小球间会且只会受到其他小球的引力作用。在计算作用力时，两个小球间的距离不会低于其半径之和，在其他的地方小球位置的移动不会受到其他小球的影响（即不会发生碰撞，挡住等情况）。你需要计算模拟一定时间后小球的分布情况，并通过MPI并行化计算过程。

# 运行结果分析

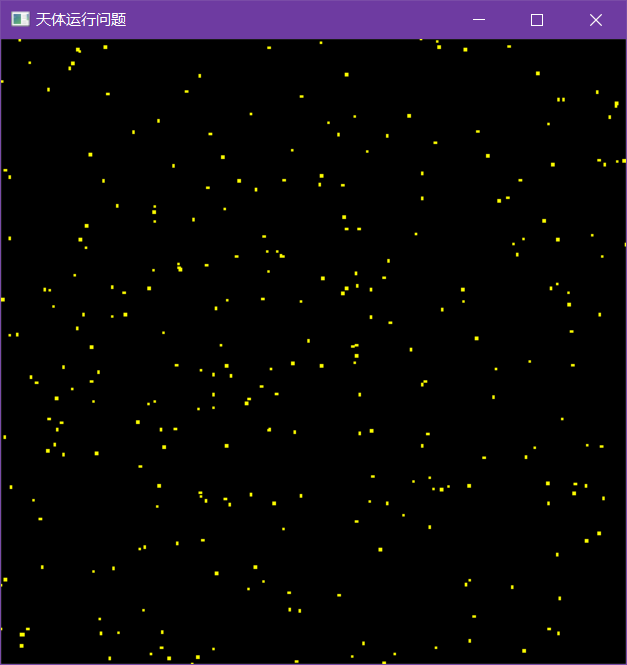
1. 初始化界面，随机生成星球的位置。

随机生成 50\*size 个点(size为进程数)，坐标随机分布在（0-200，0-200）的范围内，质量随机生成。下图为星球刚开始生成的随机点。

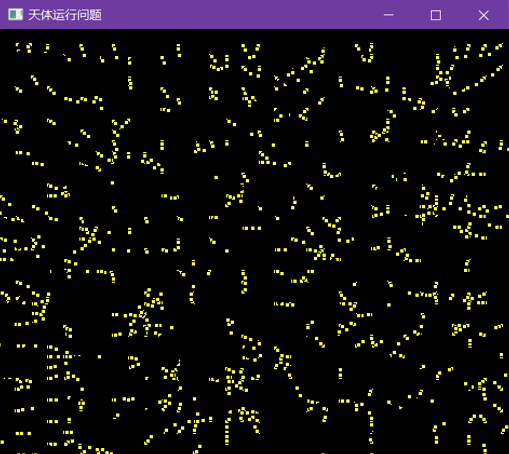
刚开始的的随机状态



在经过50此迭代后星球开始变得更密集

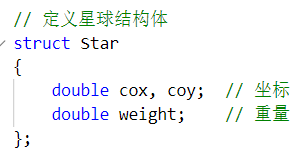


最终在500次迭代之后，聚拢的趋势变得比较明显。

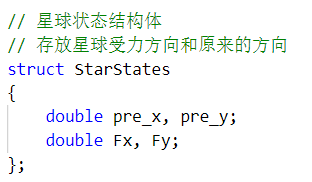


# 实现过程函数解析

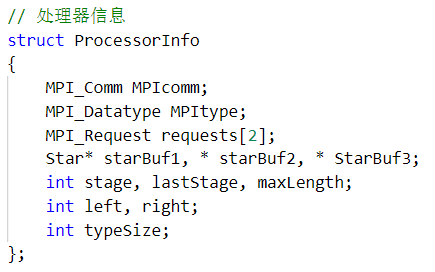
本试验采用easyX图形库实现星球图形界面



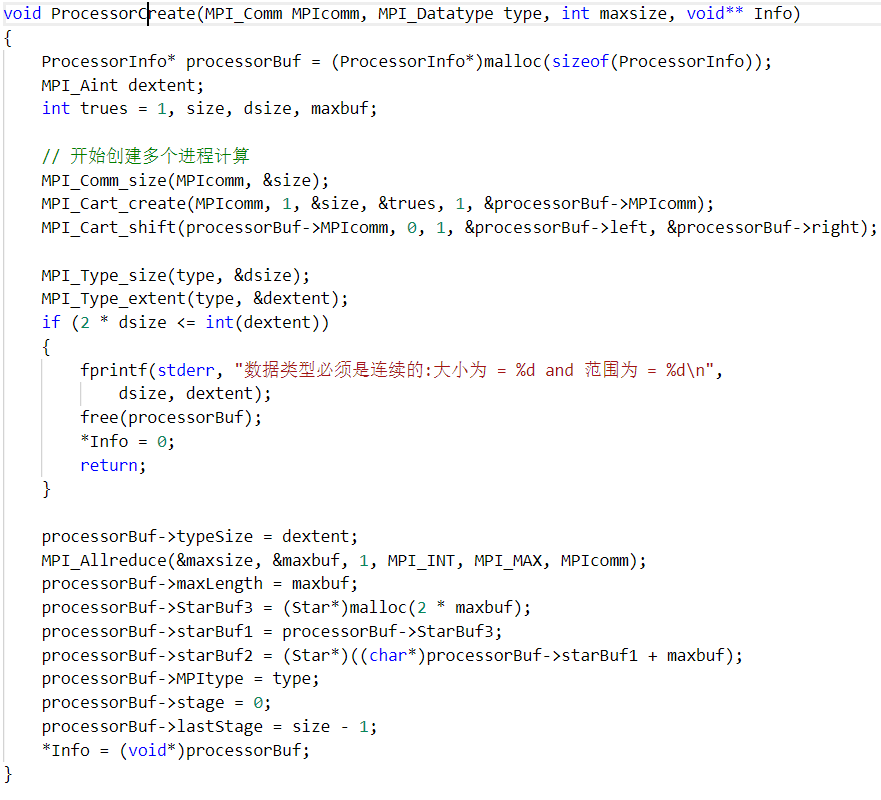
结构体定义了每个星球所包含的属性，,其中有cox、coy、weight属性，分别表示星球的坐标和质量。类型是double。



结构体定义了每个星球的受力和方向。pre\_x、pre\_y记录每一个星球在位置改变前的坐标。Fx、Fy是星球每个方向的分力大小。



processorInfo结构体包含了每个处理器的信息。每个处理器计算一组天体的数据，MPIcomm是进程管理器， MPItype包包含MPI中的一系列数据类型，requests是请求，buf1、buf2、buf3是星球缓存， stage和last\_stage是处理需要进行的阶段。



创建处理器函数，传入的参数为MPIcomm进程管理器、MPI\_Datatype数据类型，buf的最大容量maxsize，以及存储处理器信息的变量Info。

首先为指针变量\*processorBuf申请了processorInfo大小的内存空间，用来初始化处理器，最后将初始化之后的处理器赋值给Info，至此一个处理器processor初始化成功。

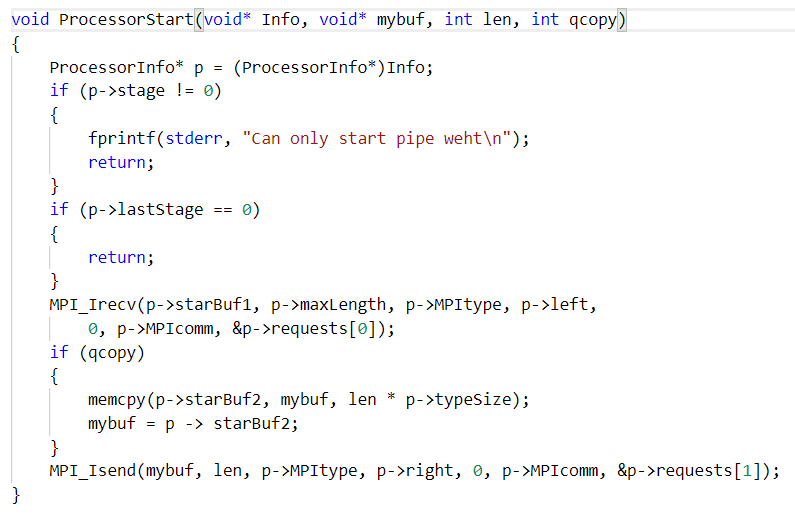
使用MPI\_Comm\_size(comm, &size)得到所有参加运算的进程的个数，并存储在变量size中。

使用MPI\_Cart\_create(comm, 1, &size, &trues, 1, &processorBuf -> MPIcomm) 创建笛卡尔拓扑，MPI提供两种拓扑，即笛卡儿拓扑和图拓扑，分别用来表示简单规则的拓扑和更通用的拓扑。对于每一维，说明进程结构是否是周期性的。 MPI\_CART\_CREATE返回一个指向新的通信域的句柄，这个句柄与笛卡尔拓扑信息相联系。如果reorder =false，那么在新的进程组中每一进程的标识数就与在旧进程组中的标识数相一致，否则，该调用会重新对进程编号，该调用得到一个ndims维的处理器阵列，每一维分别包含dims[0] dims[1] ... dims[ndims-1]个处理器。MPI\_Cart\_shift(processorBuf->MPIcomm, 0, 1, &processorBuf->left, &processorBuf->right);ProcessorBuf->MPIcomm 是带有笛卡尔结构的通信域句柄，0是direction，即需要平移的坐标维，1为disp，偏移量，processorBuf->left为源进程的卡氏坐标，processorBuf->right为目标进程的卡氏坐标。之后使用MPI\_Type\_size(type,&dsize)来获取形参type数据类型的大小。使用MPI\_Type\_extent(type, &dextent)来获取形参type的数据类型的范围，将值的大小存储在变量dextent中。

ProcessorInfo变量Info的属性typesize即dextent的值。然后使用MPI\_Allreduce(&maxsize, &maxbuf, 1, MPI\_INT, MPI\_MAX, comm)函数，所有进程都执行归约操作 相当于每一个进程都执行了一次MPI\_Reduce。

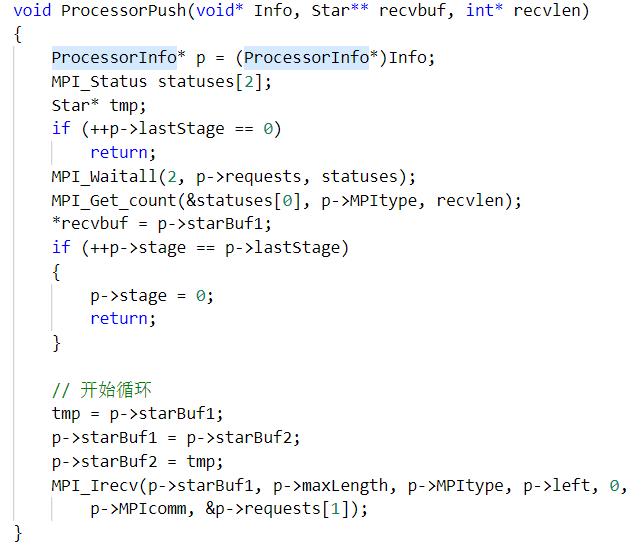
processorBuf其他属性的初始化，其中buf3的大小为maxbuf的两倍，buf1的大小等于buf的大小，buf2为3倍的maxbuf。

最后将processorBuf的内容赋值给形参Info，至此Info处理器初始化完毕。



ProcessorStart()函数用来向Info中传递数据。传递进来的参数为Info，mybuf，len，qcopy。如果Info的stage不为0，则表示处理器中为空，不再往下执行，函数返回。如果last\_stage为0表示数据传递结束，函数返回。

MPI\_Irecv(p->buf1, p->maxlen, p->type, p->left, 0, p->MPIcomm, &p->requests[0])函数用来接受数据。MPI\_Isend(mybuf, len, p->type, p->right, 0, p->MPIcomm, &p->requests[1])函数用来发送数据。

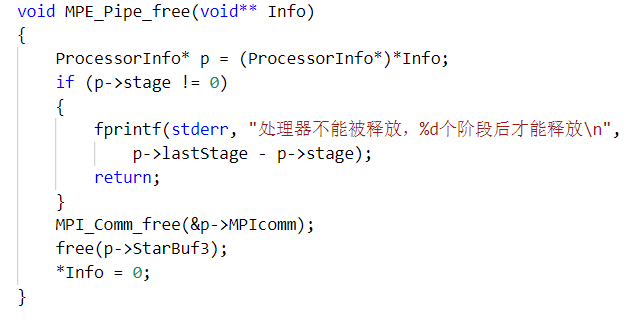


MPE\_Info\_push()函数用来创建星球群树，传入的形参为Info处理器，缓存recvbuf，接受到的数据长度recvlen。

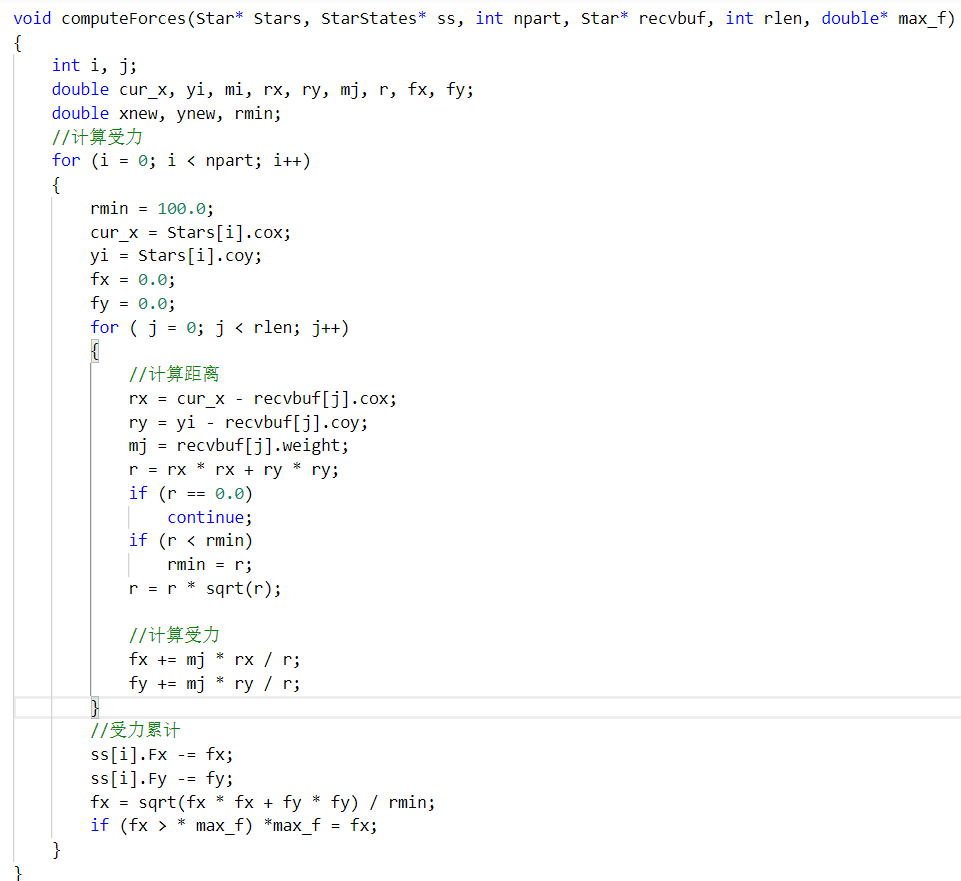
MPI\_Waitall(2, p->requests, statuses)函数必须等到非阻塞通信对象表中所有的非阻塞通信对象相应的非阻塞操作都完成后才返回，用来等待所有给定的通信结束，第一个参数为非阻塞通信对象的个数， 第二个参数array\_of\_requests非阻塞通信完成对象数组，第三个参数为array\_of\_statuses 状态数组(状态数组类型)。

MPI\_GET\_COUNT( )函数返回的是以指定的数据类型为单位，接收操作接收到的数据的个数，第一个参数status接收操作返回的状态(状态类型) ，第二个参数datatype接收操作使用的数据类型，第三个参数count 接收到的以指定的数据类型为单位的数据个数。

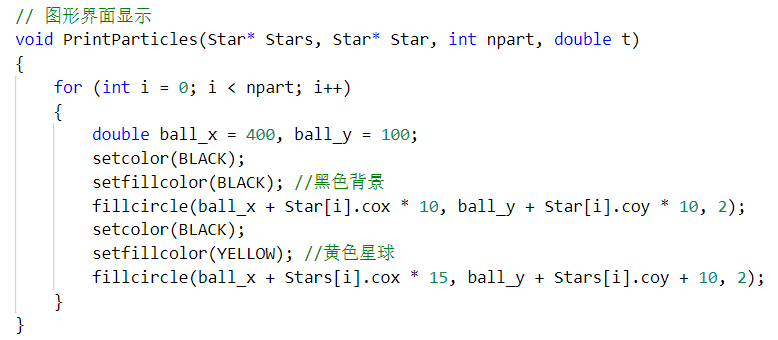
建立四叉树完毕。



MPE\_Info\_free()函数来释放处理器。



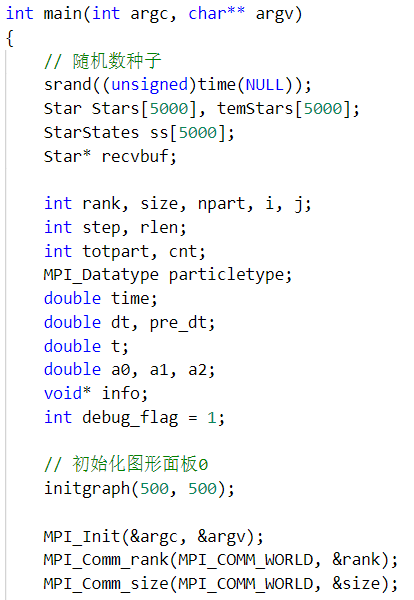
computeForces( )函数用来计算星球受力和加速度。计算x轴和y轴方向的受力和加速度。计算受力时对每个星球进行计算。计算时使用了向量的计算方式。计算结束后更新星球的受力大小。



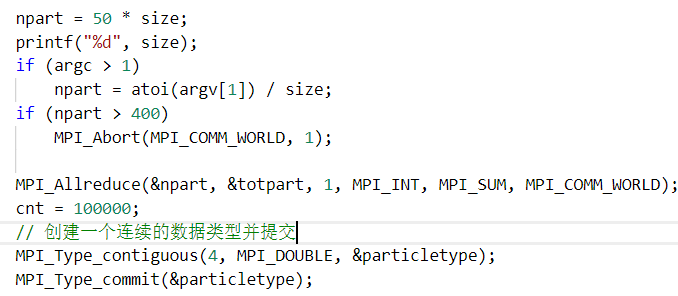
此函数是界面展示，其中展示了每个星球的位置，并将其在画布上展示出来。

黄色表示星球，黑色是背景色。

下方皆为main函数中内容

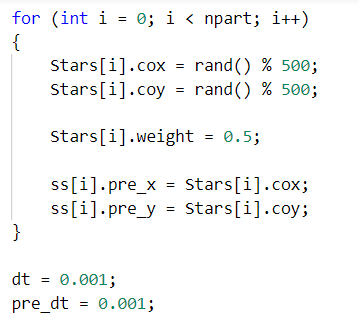


Particle particles[ ]，temParticles[ ]为每个节点中的粒子。ParticleV ss[ ]为每个粒子的速度结构体。time为计算时间。dt为计算公式中的dt，用来计算力和加速度。a0，a1，a2为星球的加速度，用来计算总的加速度。initgraph(500, 500)为初始化画布，长宽都为500。MPI\_Init(&argc, &argv)初始化进程。 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank)获取进程编号。 MPI\_Comm\_size(MPI\_COMM\_WORLD, &size)获取进程数量。



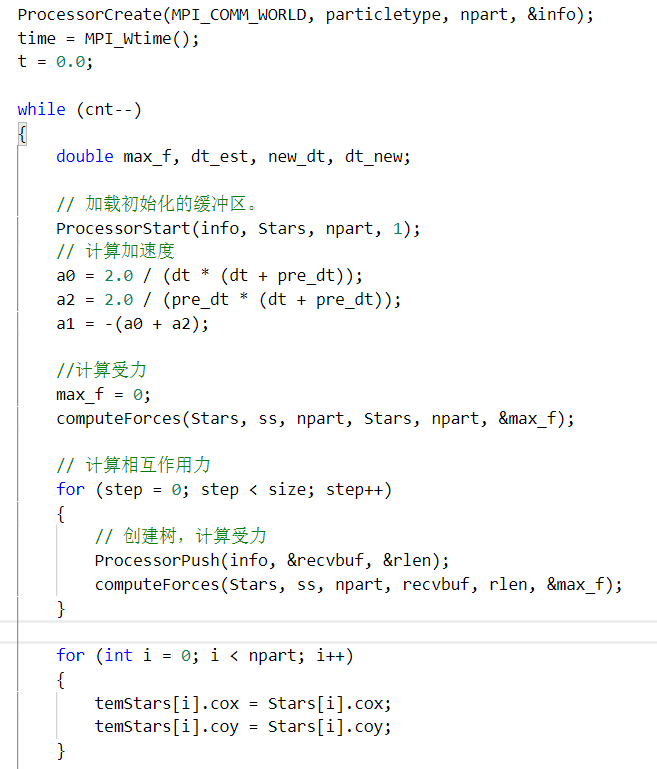
npart为总的星球数，npart=50\*size使用MPI\_Allreduce(&npart, &totpart, 1, MPI\_INT, MPI\_SUM, MPI\_COMM\_WORLD)函数将所有进程的划分相加。MPI\_Type\_contiguous(4, MPI\_DOUBLE, &particletype)用来创建

一个连续的数据类型，它得到的新类型是将一个已有的数据类型按顺序依次连续进行复制后的结果。第一个参数为count，表示复制个数(非负整数) ，第二个参数为oldtype，是旧数据类型(句柄) ，第三个参数为newtype 新数据类型(句柄)。MPI\_Type\_commit(&particletype)用来提交这个新产生的类型。

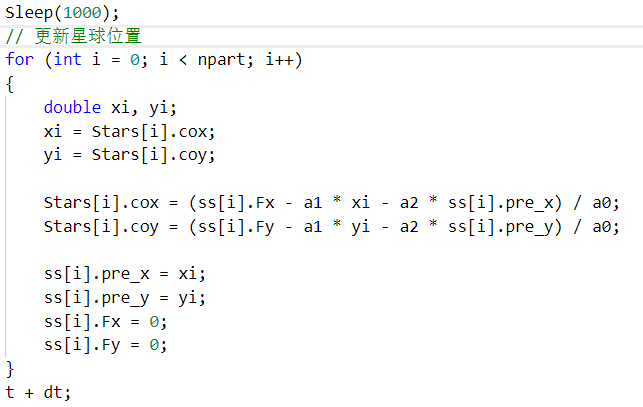


初始化星球的位置、质量和受力，位置为随机数来生成，范围为0~500。

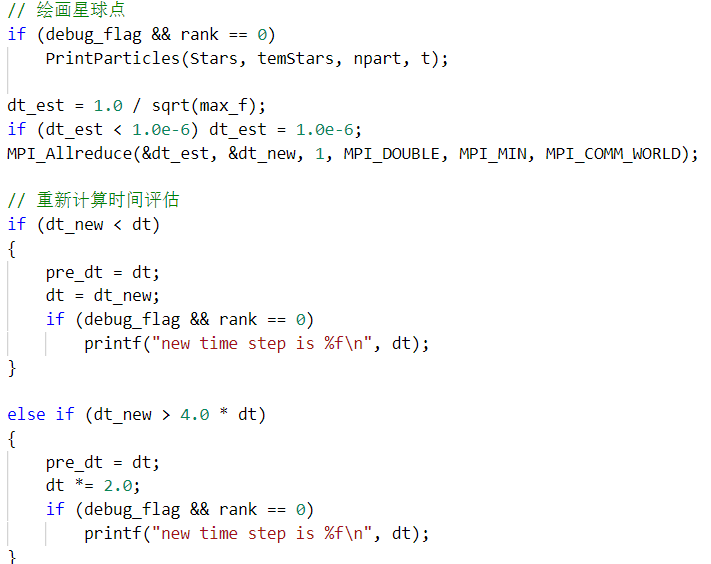
星球的质量皆初始化为0.55，ss中的位置初始化为Stars的位置，并且受力初始化为0。dt、pre\_dt为0.001。



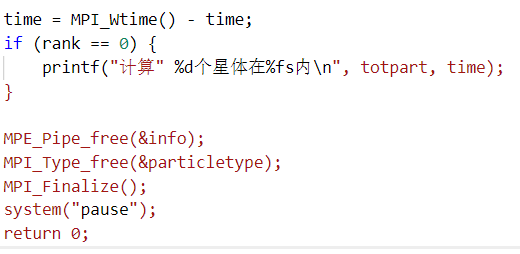
创建处理器，并记录当前时间。循环开启处理器，并计算每个星球的加速度和受力。如果size>1则创建树，并且计算受力。



Sleep(1000), 1s更新一次星球的位置。



将星球显示在画布上。重新计算时间评估。



计算运行时间time，最后释放处理器的信息和星球的类型，程序结束。

1. **调试心得**

通过本次实验，我对使用mpi库的基础函数变的更加熟练，对并行计算的基本思想有了更深入的了解，并且学会了如何使用分治的思想模拟天体运行。通过计算星球的受力、速度等影响因素，将星球之间的运动过程使用easyX图形库展现了出来，并且最终将程序运行成功，并对实验结果进行了一系列的分析。

1. **实验源码**
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <time.h>
5. #include <math.h>
6. #include <mpi.h>
7. #include<graphics.h>

10. // 定义星球结构体
11. **struct** Star
12. {
13. **double** cox, coy;  // 坐标
14. **double** weight;    // 重量
15. };
17. // 星球状态结构体
18. // 存放星球受力方向和原来的方向
19. **struct** StarStates
20. {
21. **double** pre\_x, pre\_y;
22. **double** Fx, Fy;
23. };
25. // 处理器信息
26. **struct** ProcessorInfo
27. {
28. MPI\_Comm MPIcomm;
29. MPI\_Datatype MPItype;
30. MPI\_Request requests[2];
31. Star\* starBuf1, \* starBuf2, \* StarBuf3;
32. **int** stage, lastStage, maxLength;
33. **int** left, right;
34. **int** typeSize;
35. };
37. **void** ProcessorCreate(MPI\_Comm MPIcomm, MPI\_Datatype type, **int** maxsize, **void**\*\* Info)
38. {
39. ProcessorInfo\* processorBuf = (ProcessorInfo\*)malloc(**sizeof**(ProcessorInfo));
40. MPI\_Aint dextent;
41. **int** trues = 1, size, dsize, maxbuf;
43. // 开始创建多个进程计算
44. MPI\_Comm\_size(MPIcomm, &size);
45. MPI\_Cart\_create(MPIcomm, 1, &size, &trues, 1, &processorBuf->MPIcomm);
46. MPI\_Cart\_shift(processorBuf->MPIcomm, 0, 1, &processorBuf->left, &processorBuf->right);
48. MPI\_Type\_size(type, &dsize);
49. MPI\_Type\_extent(type, &dextent);
50. **if** (2 \* dsize <= **int**(dextent))
51. {
52. fprintf(stderr, "数据类型必须是连续的:大小为 = %d and 范围为 = %d\n",
53. dsize, dextent);
54. free(processorBuf);
55. \*Info = 0;
56. **return**;
57. }
59. processorBuf->typeSize = dextent;
60. MPI\_Allreduce(&maxsize, &maxbuf, 1, MPI\_INT, MPI\_MAX, MPIcomm);
61. processorBuf->maxLength = maxbuf;
62. processorBuf->StarBuf3 = (Star\*)malloc(2 \* maxbuf);
63. processorBuf->starBuf1 = processorBuf->StarBuf3;
64. processorBuf->starBuf2 = (Star\*)((**char**\*)processorBuf->starBuf1 + maxbuf);
65. processorBuf->MPItype = type;
66. processorBuf->stage = 0;
67. processorBuf->lastStage = size - 1;
68. \*Info = (**void**\*)processorBuf;
69. }
71. **void** ProcessorStart(**void**\* Info, **void**\* mybuf, **int** len, **int** qcopy)
72. {
73. ProcessorInfo\* p = (ProcessorInfo\*)Info;
74. **if** (p->stage != 0)
75. {
76. fprintf(stderr, "Can only start pipe weht\n");
77. **return**;
78. }
79. **if** (p->lastStage == 0)
80. {
81. **return**;
82. }
83. MPI\_Irecv(p->starBuf1, p->maxLength, p->MPItype, p->left,
84. 0, p->MPIcomm, &p->requests[0]);
85. **if** (qcopy)
86. {
87. memcpy(p->starBuf2, mybuf, len \* p->typeSize);
88. mybuf = p -> starBuf2;
89. }
90. MPI\_Isend(mybuf, len, p->MPItype, p->right, 0, p->MPIcomm, &p->requests[1]);
91. }
93. **void** ProcessorPush(**void**\* Info, Star\*\* recvbuf, **int**\* recvlen)
94. {
95. ProcessorInfo\* p = (ProcessorInfo\*)Info;
96. MPI\_Status statuses[2];
97. Star\* tmp;
98. **if** (++p->lastStage == 0)
99. **return**;
100. MPI\_Waitall(2, p->requests, statuses);
101. MPI\_Get\_count(&statuses[0], p->MPItype, recvlen);
102. \*recvbuf = p->starBuf1;
103. **if** (++p->stage == p->lastStage)
104. {
105. p->stage = 0;
106. **return**;
107. }
109. // 开始循环
110. tmp = p->starBuf1;
111. p->starBuf1 = p->starBuf2;
112. p->starBuf2 = tmp;
113. MPI\_Irecv(p->starBuf1, p->maxLength, p->MPItype, p->left, 0,
114. p->MPIcomm, &p->requests[1]);
115. }
117. **void** MPE\_Pipe\_free(**void**\*\* Info)
118. {
119. ProcessorInfo\* p = (ProcessorInfo\*)\*Info;
120. **if** (p->stage != 0)
121. {
122. fprintf(stderr, "处理器不能被释放，%d个阶段后才能释放\n",
123. p->lastStage - p->stage);
124. **return**;
125. }
126. MPI\_Comm\_free(&p->MPIcomm);
127. free(p->StarBuf3);
128. \*Info = 0;
129. }
131. **void** computeForces(Star\* Stars, StarStates\* ss, **int** npart, Star\* recvbuf, **int** rlen, **double**\* max\_f)
132. {
133. **int** i, j;
134. **double** cur\_x, yi, mi, rx, ry, mj, r, fx, fy;
135. **double** xnew, ynew, rmin;
136. //计算受力
137. **for** (i = 0; i < npart; i++)
138. {
139. rmin = 100.0;
140. cur\_x = Stars[i].cox;
141. yi = Stars[i].coy;
142. fx = 0.0;
143. fy = 0.0;
144. **for** ( j = 0; j < rlen; j++)
145. {
146. //计算距离
147. rx = cur\_x - recvbuf[j].cox;
148. ry = yi - recvbuf[j].coy;
149. mj = recvbuf[j].weight;
150. r = rx \* rx + ry \* ry;
151. **if** (r == 0.0)
152. **continue**;
153. **if** (r < rmin)
154. rmin = r;
155. r = r \* sqrt(r);
157. //计算受力
158. fx += mj \* rx / r;
159. fy += mj \* ry / r;
160. }
161. //受力累计
162. ss[i].Fx -= fx;
163. ss[i].Fy -= fy;
164. fx = sqrt(fx \* fx + fy \* fy) / rmin;
165. **if** (fx > \* max\_f) \*max\_f = fx;
166. }
167. }
169. // 图形界面显示
170. **void** PrintParticles(Star\* Stars, Star\* Star, **int** npart, **double** t)
171. {
172. **for** (**int** i = 0; i < npart; i++)
173. {
174. **double** ball\_x = 400, ball\_y = 100;
175. setcolor(BLACK);
176. setfillcolor(BLACK); //黑色背景
177. fillcircle(ball\_x + Star[i].cox \* 10, ball\_y + Star[i].coy \* 10, 2);
178. setcolor(BLACK);
179. setfillcolor(YELLOW); //黄色星球
180. fillcircle(ball\_x + Stars[i].cox \* 15, ball\_y + Stars[i].coy + 10, 2);
181. }
182. }
184. **int** main(**int** argc, **char**\*\* argv)
185. {
186. // 随机数种子
187. srand((unsigned)time(NULL));
188. Star Stars[5000], temStars[5000];
189. StarStates ss[5000];
190. Star\* recvbuf;
192. **int** rank, size, npart, i, j;
193. **int** step, rlen;
194. **int** totpart, cnt;
195. MPI\_Datatype particletype;
196. **double** time;
197. **double** dt, pre\_dt;
198. **double** t;
199. **double** a0, a1, a2;
200. **void**\* info;
201. **int** debug\_flag = 1;
203. // 初始化图形面板0
204. initgraph(500, 500);
206. MPI\_Init(&argc, &argv);
207. MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);
208. MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

211. npart = 50 \* size;
212. printf("%d", size);
213. **if** (argc > 1)
214. npart = atoi(argv[1]) / size;
215. **if** (npart > 400)
216. MPI\_Abort(MPI\_COMM\_WORLD, 1);
218. MPI\_Allreduce(&npart, &totpart, 1, MPI\_INT, MPI\_SUM, MPI\_COMM\_WORLD);
219. cnt = 100000;
220. // 创建一个连续的数据类型并提交
221. MPI\_Type\_contiguous(4, MPI\_DOUBLE, &particletype);
222. MPI\_Type\_commit(&particletype);
224. **for** (**int** i = 0; i < npart; i++)
225. {
226. Stars[i].cox = rand() % 500;
227. Stars[i].coy = rand() % 500;
229. Stars[i].weight = 0.5;
231. ss[i].pre\_x = Stars[i].cox;
232. ss[i].pre\_y = Stars[i].coy;
233. }
235. dt = 0.001;
236. pre\_dt = 0.001;
238. ProcessorCreate(MPI\_COMM\_WORLD, particletype, npart, &info);
239. time = MPI\_Wtime();
240. t = 0.0;
242. **while** (cnt--)
243. {
244. **double** max\_f, dt\_est, new\_dt, dt\_new;
246. // 加载初始化的缓冲区。
247. ProcessorStart(info, Stars, npart, 1);
248. // 计算加速度
249. a0 = 2.0 / (dt \* (dt + pre\_dt));
250. a2 = 2.0 / (pre\_dt \* (dt + pre\_dt));
251. a1 = -(a0 + a2);
253. //计算受力
254. max\_f = 0;
255. computeForces(Stars, ss, npart, Stars, npart, &max\_f);
257. // 计算相互作用力
258. **for** (step = 0; step < size; step++)
259. {
260. // 创建树，计算受力
261. ProcessorPush(info, &recvbuf, &rlen);
262. computeForces(Stars, ss, npart, recvbuf, rlen, &max\_f);
263. }
265. **for** (**int** i = 0; i < npart; i++)
266. {
267. temStars[i].cox = Stars[i].cox;
268. temStars[i].coy = Stars[i].coy;
269. }
271. Sleep(1000);
272. // 更新星球位置
273. **for** (**int** i = 0; i < npart; i++)
274. {
275. **double** xi, yi;
276. xi = Stars[i].cox;
277. yi = Stars[i].coy;
279. Stars[i].cox = (ss[i].Fx - a1 \* xi - a2 \* ss[i].pre\_x) / a0;
280. Stars[i].coy = (ss[i].Fy - a1 \* yi - a2 \* ss[i].pre\_y) / a0;
282. ss[i].pre\_x = xi;
283. ss[i].pre\_y = yi;
284. ss[i].Fx = 0;
285. ss[i].Fy = 0;
286. }
287. t + dt;
289. // 绘画星球点
290. **if** (debug\_flag && rank == 0)
291. PrintParticles(Stars, temStars, npart, t);
293. dt\_est = 1.0 / sqrt(max\_f);
294. **if** (dt\_est < 1.0e-6) dt\_est = 1.0e-6;
295. MPI\_Allreduce(&dt\_est, &dt\_new, 1, MPI\_DOUBLE, MPI\_MIN, MPI\_COMM\_WORLD);
297. // 重新计算时间评估
298. **if** (dt\_new < dt)
299. {
300. pre\_dt = dt;
301. dt = dt\_new;
302. **if** (debug\_flag && rank == 0)
303. printf("new time step is %f\n", dt);
304. }
306. **else** **if** (dt\_new > 4.0 \* dt)
307. {
308. pre\_dt = dt;
309. dt \*= 2.0;
310. **if** (debug\_flag && rank == 0)
311. printf("new time step is %f\n", dt);
312. }
313. }
315. time = MPI\_Wtime() - time;
316. **if** (rank == 0) {
317. printf("计算" %d个星体在%fs内\n", totpart, time);
318. }
320. MPE\_Pipe\_free(&info);
321. MPI\_Type\_free(&particletype);
322. MPI\_Finalize();
323. system("pause");
324. **return** 0;
325. }