基于 MPI 的相似轨迹计算

吉吉

加里顿大学计算机科学与技术学院

Email: csyiji@gmail.com

摘要：利用 Open MPI [1] 并行计算环境，实现对于给定轨迹查找与之相似的所有轨迹的实验。

关键词：并行算法，MPI，相似轨迹，LCSS

1 实验内容

1.1 背景

在用户的位置信息和移动轨迹中，蕴含丰富的用户个性信息，对用户移动轨迹研究可有效挖掘用户的行为模式和行为偏好，对智能交通、广告推送、异常轨迹检测、社交推荐等应用有着重要作用。

1.2 业务场景

分析移动对象轨迹最常用的应用之一是寻找具有相似轨迹的移动对象并进行归类。一个典型的基于移动对象轨迹的分析场景如下："对于给定的一个移动对象轨迹，查找与这条轨迹在一定相似度范围内的所有轨迹"。

1.3 相似度定义

最长公共子序列 LCSS(Longest Common Subsequence) [2] 中对两个字符串或文本的相似度度量思想：对于两个字符串 s1 和 s2，尝试找到第三个字符串 s3，若 s3 同时出现在 s1 与 s2 中且顺序相同，那么 s3 的长度越长则 s1 与 s2 越相似。

此处关注的相似度重点在于轨迹经过临近的点、经过各点的顺序两个方面。点的顺序相同的情况下，经过不同的临近点越多，相似度越高。

$$\begin{cases} 0 \quad , \quad if \quad A \quad or \quad B \quad is \quad empty. \\ 1 + LCSS_{\delta,\epsilon}(Head(A), Head(B)) \quad , \quad if \quad |a_{x,n} - b_{x,m}| < \epsilon \quad and \quad |a_{y,n} - b_{y,m}| < \epsilon \quad and \quad |n - m| \leq \delta \\ max(LCSS_{\delta,\epsilon}(Head(A), B), LCSS_{\delta,\epsilon}(A, Head(B))) \quad , \quad otherwise \end{cases} \tag{1-1}$$

公式1-1是 LCSS 算法原型公式，其主要分为三点：

- 当 A 或 B 轨迹中有一个为空，则临近点个数为 0；

- 否则，循环判断 A、B 两条轨迹中的各点之间是否满足临近点阈值，是则加 1 并递归进行；
- 否则，将计算的点去掉，分别再与另一轨迹计算是否符合临近点估计，并求两种情况的最大值。

利用 LCSS 可以求出两条轨迹中的临近点个数，然后利用公式1-2可计算出轨迹 A,B 的的相似度。

$$\frac{LCSS_{\delta,\epsilon}(A,B)\times 2}{m+n} \tag{1-2}$$

2 重点难点分析

2.1 文件数据的解析

我们实验的数据由 10000 个轨迹数据文件和 1 个索引文件。其整个执行流程如下：

- 假设目标轨迹为 A, 则先根据 A 的轨迹编号在索引文件中查询，得到行号 x 与列号 y；
- 轨迹 A 数据所在文件名为 x_data , 读取此文件第 y 行即为轨迹 A 的内容；
- 轨迹的内容结构为: 轨迹编号 + 轨迹数量 + 轨迹序列.., 读取轨迹序列存入二维数组，再进行后续处理。

2.2 LCSS 算法阈值选择

由 LCSS 算法1-2的定义说明，其中计算的判断条件在于轨迹点距离 ϵ 和轨迹点顺序差 δ 两个阈值的选择。而此两个阈值不能仅仅设为常数，它们会随着目标轨迹的变化而变化。因此：

- ϵ：选取轨迹上每两个相邻点间的欧式距离和的 $\frac{1}{k}$
- δ：选取轨迹上点数目的 $\frac{1}{k}$

2.3 相似度算法的非递归方式实现

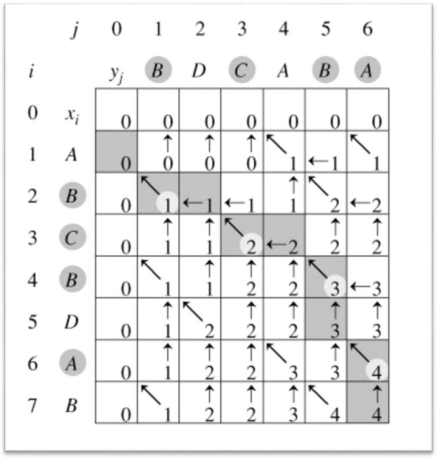

图 2-1 LCSS 非递归实现

LCSS 公式1-1 给出的是求临近点的递归实现的方式，而我们的每条轨迹点约有 60 100 个点，则任意两条轨迹计算的递归深度至少为 3600，占用的栈空间很大，因此我们采用非递归方式实现求临近点个数的算法。图2-1展示了非递归的实现方式：

- 用矩阵来存储任意两点的计算结果, 计算主要遵循两个原则：
- 轨迹上两点若满足公式里的阈值范围，则将矩阵左上角数值加 1;
- 轨迹上两点若不满足阈值范围，则选择当前矩阵点的上方和左方的最大的数值作为当前值。

3 计算的技术路线

3.1 文件数据载入

实验的所有数据都存在文件中，我们共有 10000 个轨迹数据文件，每个轨迹数据文件中约有 1000 条轨迹数据，每条轨迹数据约有 60-100 个点。寻找轨迹的相似轨迹，同一时间我们只关心两条轨迹的数据，因此:

- 在加载目标轨迹时，是先从索引文件中读取到轨迹标号的行列，然后读取行对于的数据文件名，加载目标轨迹数据到内存。
- 在加载当前对比轨迹时，则直接从数据文件中读取轨迹数据，与目标轨迹进行对比计算。
- 并行计算时，在每个进程里都进行以上两个操作，不采用由主进程分发形式。

3.2 计算区域的选择

我们的实验数据太多，以至于单进程要跑完所有文件数据将耗费大量的时间，我们的 MPI 测试环境也对我们的测试进程进行了资源限制。

```
1
2 Begin PBS Prologue Wed May 24 08:23:11 CST 2017
3 Job ID:     1359.mnode.site
4 Username:   pcclass
5 Group:      root
6 Nodes:      cnode12
7 End PBS Prologue Wed May 24 08:23:11 CST 2017
8
9 mpirun: killing job...
10
11
12 Begin PBS Epilogue Wed May 24 09:23:17 CST 2017
13 Job ID:     1359.mnode.site
14 Username:   pcclass
15 Group:      root
16 Job Name:   job
17 Session:    10179
18 Limits:     neednodes=1:ppn=1,nodes=1:ppn=1,walltime=01:00:00
19 Resources:  cput=00:59:47,mem=8924kb,vmem=23580kb,walltime=01:00:01
20 Queue:      js20
21 Account:
22 Nodes:  cnode12
23 Killing leftovers...
24
25 End PBS Epilogue Wed May 24 09:23:18 CST 2017
26
```

图 3-2 MPI 测试环境的资源限制

如图3-2 第 18 行，显示整个实验环境时间限制在 3600 秒，超过这个阈值，则会出现第 9 行所示的结果，我们的任务将被杀死。因此我们进行了一些测试，如表3-1所示。

表 3-1 基本测试数据

文件总数量	进程总数量	所用总时间 (秒)
10000	20	2740
10000	1	>3600
4	4	6.56
2	2	6.67
1	1	6.7

从中可得出大约一个进行执行 1 个文件查询需要约 7 秒钟，为了我们程序可在资源限制下运行成功，我们最多可选择 500 个文件进行 MPI 实验，而 $\sqrt{500} = 22.6$ 实现的搜索区域选择正方形且边长为奇数最简洁，因此我们选择了 21×21 的矩形区域作为我们的实验目标区域。所有文件名排列矩阵如图3-3，目标轨迹所在区域的三种情况如图3-4。

100	200	300	400	500	600	...	...	9900	10000
99	199	299	399	499	599	...	...	9899	9999
...	...	...	...	...	...			...	...
...	...	...	...	...	...	...	...	...	...
6	106	206	306	406	506			9806	9906
5	105	205	305	405	505	...	...	9805	9905
4	104	204	304	404	504			9804	9904
3	103	203	303	403	503	...	...	9803	9903
2	102	202	302	402	502			9802	9902
1	101	201	301	401	501	...	...	9801	9901

图 3-3 所有文件排列矩阵

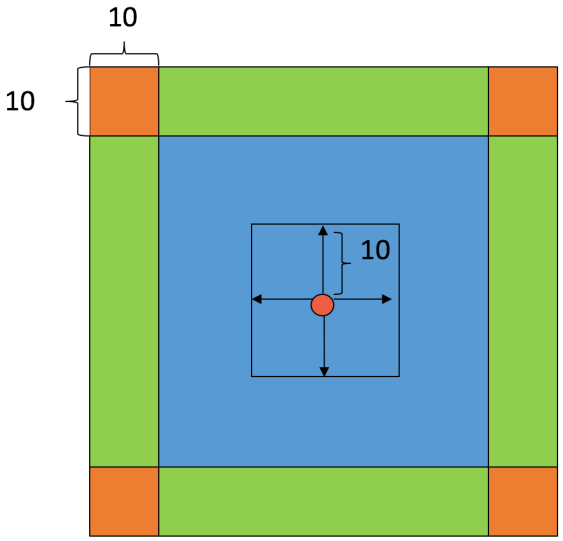

图 3-4 目标搜索区域的三块区域

3.3 分片方式

分片方式采用均分形式，我们在确定了目标搜索区域后，会将 21×21 的矩阵存到一维数组中，将进程总数量作为递增间隔，每个进程相应地处理各自的一个文件名为等差数列的文件组。

3.4 通信内容

在各个进程处理完自己对应的文件组后，会获取到相似轨迹的轨迹编号数组，以 0 号进程作为主进程，其他进程将计算得到的轨迹编号组发送到 0 号进程进行汇总。主要用到 MPI 并行环境中的 MPI_Send 和 MPI_Recv 接口。

4 实验结果及分析

4.1 实验设置

- MPI 环境设置:*source/public/home/pcclass/openmpi.sh*
- 传入测试数据 (10000 个数据文件和 1 个索引文件):*/public/home/pcclass/*20164227019*/track/cluster*。
- 实现 MPI 接口的 C 语言源程序:*/public/home/pcclass/*20164227019*/lcsstrack.c*

4.2 实验步骤

- 切换到实验目录:*cd /public/home/pcclass/*20164227019*/*
- 编译基于 MPI 接口的 C 语言源程序:*mpicc lcsstrack.c*, 生成 *a.out* 可执行文件
- 主机测试：*mpirun −np 4 ./a.out*, 数字 4 可更改为 1-4 中的任意数字，表示进程数量。
- 多核服务器运行：*qsub hello.pbs*, 这里的 hello.pbs 里默认配置了 10 个进程进程计算
- 自定义进程数运行:*vi hello.pbs*，修改 $node2 = 5 : ppn = 2$ 中的数字 5,2 和最后一行的 *./a.out* 前的数字 10, 保证前两数相乘等于最后一个数。则最后一个数据即为实际运行的进程数。
- *test*01 和 *test*02 两个文件目录，已经配置好了此实验的各个进程配置文件,*hello.pbs* 中后缀即为所配置的进程数量，用户只需 *qsubhellox.pbs* 即可用相应进程进程测试，最终会在 *job* 文件中得到结果，结果形式如图4-5。

```
Process  0 says trackid:   6815950
Process  0 says trackid:   6909964
Process  0 says trackid:   8242506
Process  0 says trackid:   8341763
Process  0 says trackid:   8392059
Process  0 says trackid:   8425059
Process  0 says trackid:   8440857
Process  0 says trackid:   8450635
Process  0 says trackid:   8564971
Process  0 says trackid:   8575765
Process  0 says trackid:   8757339
Process  0 says trackid:   8781298
Process  0 says trackid:   8867601
Process  0 says trackid:   8899056
Process  0 says trackid:   9334284
Duration of process 0 sec= 606,usec= 287657,musec= 606.287657.
^MTotal find 65

Begin PBS Epilogue Thu Jun  1 19:17:34 CST 2017
Job ID:         1420.mnode.site
Username:       pcclass
Group:          root
Job Name:       jiyijob
Session:        3254
Limits:         neednodes=2:ppn=2,nodes=2:ppn=2,walltime=01:00:00
Resources:      cput=00:20:04,mem=14188kb,vmem=166568kb,walltime=00:10:08
Queue:          js20
Account:
Nodes:  cnode10 cnode9
Killing leftovers...

End PBS Epilogue Thu Jun  1 19:17:34 CST 2017
```

图 4-5 实验结果形式

4.3 实验结果

我们测试两种形式的结果，一种是轨迹阈值很小时，多个进程间通信数量较少的情况下如表；另一种是轨迹阈值相对较大时，各个进程间通信数量相对较多的情况如表。得到的加速比结果近似于 17。

表 4-2 实验结果数据

(a) 通信数量较少

进程总数量	时间 (秒)	比值
1	2412	1
2	1207	1.99
4	606	3.53
6	407	5.26
8	305	7.02
10	245	8.74
12	204	10.50
14	180	11.9
16	156	13.73
18	138	15.52
20	126	17

(b) 通信数量较多

进程总数量	时间 (秒)	比值
1	2734	1
2	1369	1.99
4	689	3.96
6	459	5.95
8	347	7.87
10	281	9.72
12	231	11.8
14	223	12.2
16	176	15.5
18	158	17.3
20	126	17.6

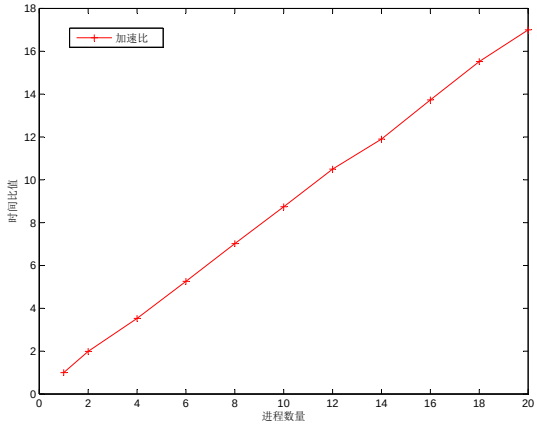

图 4-6 通信较少

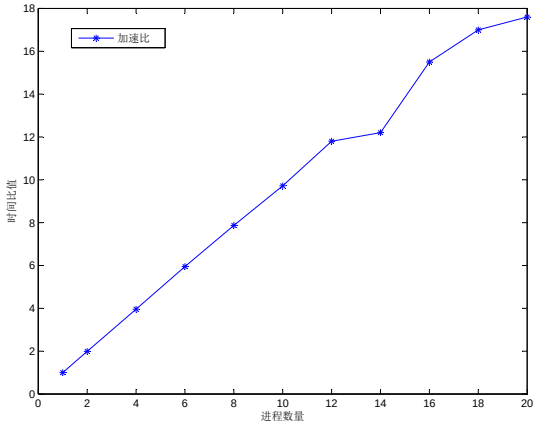

图 4-7 通信较多

4.4 实验分析

由上面的数据可得到此实验的加速比趋近于 17，这是一个非常高的结果了。我前后又进行了多次的检查和测试，结果也是基本相同。主要原因如下

- 本实验将文件读取的时间也计算入内了。由于我们的实验数据太多 (约 13GB)，我们不能将所有数据一次性载入内存。因此我们采用的方式是动态载入当前需要对比的轨迹数据。我们的数据文件中每一行都正好是一条轨迹数据，于是我们每读取一条轨迹后就将此轨迹与目标轨迹进行对比，是则记录轨迹编号，否则继续读取下一条数据。

- 最后获得相似轨迹的通信数量远远小于计算的轨迹数量，在上面的较多通信的实验中，最终计算得到的相似数据有约 13772 条，而我们的送计算区域为 $441 \times 1000 = 441000$ 条轨迹数据，约是总计算量的 $\frac{1}{40}$，通信数量相对较少，导致通信时间几乎忽略不计。我们想将最终获得的数据再调更多，但我们已经将相似度降低到 0.2 的几乎最低的程度了，几乎达到相似轨迹总量的极限了。
- 本实验并行计算方式几乎是独立的，仅在最终结果汇总的时候进行少量的数据通信，相当于规模的平均分配计算，得到的加速比相对较高。

5 提交的文件结构

在用 pcclass 用户名登录实验服务器后，在 20164227019 文件夹中存储的就是我实验所需要的材料：

- 实验数据:*track*/*cluster*/ 目录下的所有文件。
- 源码文件:*lcsstrack.c* 实现 MPI 接口的 C 语言源代码。
- MPI 环境脚本:*openmpienv.sh* 加载 MPI 库环境。
- 并行任务提交脚本:*hello.pbs* 里面默认配置了 10 个进程计算。
- 通信较少的实验数据:*test*01 里面存储了 1 个进程和 20 以内的偶数进程计算的结果。
- 通信较多的实验数据:*test*02 里面存储了 1 个进程和 20 以内的偶数进程计算的结果。
- 其他测试数据：*track*/*test* $*$ / 每个 test 目录下为自己进程实验的其他试验性数据。

6 参考文献

参考文献

[1] Open Source High Performance Computing. `https://www.open-mpi.org/`, 2006.

[2] Vlachos M, Gunopoulos D, and Kollios G. Discovering Similar Multidimensional Trajectories. In *International Conference on Data Engineering. IEEE Computer Society*, page 673, 2002.