并行求解三维偏微分方程
并行计算第三次上机作业

郑灵超

2016 年 11 月 30 日

目录

1 问题介绍

本次要求解的方程是一个三维的偏微分方程，其定义域为立方体

$$|x|, |y|, |z| \leq \pi$$

方程形式为

$$-\Delta u + u^3 = f, \tag{1}$$

所给的边界条件为周期边界。

2 算法介绍

此次我们采用的算法是显式差分格式，将求解区域按正方形网格剖分，即网格尺度 $h = \frac{1}{N}$, 则网格节点为 $x = ih, y = jh, z = kh$ 其迭代格式为

$$\begin{aligned} u^{(t+\Delta t)}(ih, jh, kh) =& u^{(t)}(ih, jh, kh) + \Delta t f(ih, jh, kh, t) + \frac{\Delta t}{h^2}[u^{(t)}(ih+h, jh, kh) \\ &+ u^{(t)}(ih-h, jh, kh) + u^{(t)}(ih, jh+h, kh) + u^{(t)}(ih, jh-h, kh) \\ &+ u^{(t)}(ih, jh, kh+h) + u^{(t)}(ih, jh, kh-h) - 6u^{(t)}(ih, jh, kh)] \end{aligned} \tag{2}$$

这部分数值格式的精度为 $O(\tau + h^2)$，而 CFL 条件要求

$$\frac{\tau}{h^2} \leq \frac{1}{6},$$

因此这部分数值格式的精度为 $O(h^2)$。

在上边界，即狄利克雷边界上，我们直接进行赋值；在下边界，即诺依曼边界上，我们根据 t 时刻该点的法向导数，构造出一些虚设的外部的点的函数值，再利用(2)进行迭代计算。例如在 $x + y - z = 1$ 的边界上，我们采取的格式是，

$$\begin{aligned} u^{(t+\Delta t)}(ih, jh, kh) =& u^{(t)}(ih, jh, kh) + \Delta t f(ih, jh, kh, t) + \frac{\Delta t}{h^2}[2u^{(t)}(ih-h, jh, kh) \\ &+ 2u^{(t)}(ih, jh-h, kh) + 2u^{(t)}(ih, jh, kh+h) - 6u^{(t)}(ih, jh, kh) \\ &+ 2\sqrt{3} g_{down}(ih, jh, kh, t)] \end{aligned} \tag{3}$$

3 程序分析

3.1 程序说明

我们定义了一个类 Heat, 用于求解这一类型的热方程，用户可以通过在 main.cpp 文件中修改方程的初边值条件和 CFL 条件数。网格密度 N 和计算终止时间 t_{end} 通过命令行参数读入。具体程序的结构和声明可以参见 doc 目录下的 refman.pdf 和 html 目录下的 index.html 网页。以下我们简要说明一下并行实现的算法流程:

1. 读入信息，如网格密度，计算终止时间，所用进程数目，CFL 条件数，和设置的初边值条件。

2. 由 0 号进程进行一些预处理工作：将三维的点用一个长度为 M 的 std::vector 表示，并给出将点转化为 vector 中下标的函数。并利用这个函数计算出每个编号对应的点的坐标和邻居的编号，并将这些信息发送给所需要的进程。

3. 每个进程各自的初始化工作，包括计算每个进程所包含的点的起始编号和终止编号。我们这里将所有网格均等分给每个进程，第 i 号进程的需要处理的网格编号为 $\frac{iM}{size}$ 到 $\frac{(i+1)M}{size}$。此外，每个进程需要计算自己进行迭代计算时需要相邻进程提供的数据。

4. 每个进程分别计算自己所管辖区域的 $t = 0$ 的初值情况。

5. 进行一步迭代计算。

6. 由 0 号进程收集终止时刻的所有值，合并成一个整体的向量，并返回给用户。

其中进行一步迭代计算的流程为，这里的一步迭代是每个进程分别操作的。

1. 向相邻进程收取所需要的前一时刻的数据，与自己前一时刻的数据组合起一个完整的数据集合。

2. 利用迭代格式(2)和(3)进行迭代计算。

3. 将自己所被需要的信息发送给相邻的进程。

3.2 程序评价

这次我们采用的程序的优点有：

- 将整个数据拉成一个一维连续的 vector，并等分给各个进程，负载较为均衡。

- 精确计算了计算每个点所需要的信息，并向其它进程索取，保证了信息传递没有浪费。

- 用类进行封装，用户只需要通过主函数进行修改。

此外，我认为此次写的程序还有如下不足：

- 一些基本信息的初始化工作仍然由 0 号进程完成，这儿我认为还有改进空间。

- 对于边界，我们尚未给出一个精细的处理方法。

- 采用了显式求解格式，使得时间步长受到限制。

4 数值结果

4.1 误差分析

我们选取了一个有精确解的方程进行计算，其精确解为

$$u(x,y,z,t) = \sin((x^2+y^2+z^2)t) \tag{4}$$

采取之前所介绍的方法进行计算，并将解与真实解做了误差比较，我们得到如下的结果:

网格密度 N	L2 误差	阶数	CPU 时间 (s)	CPU 时间的阶数
10	6.00e-3	/	0.0018	/
20	1.40e-3	2.09	0.3278	7.5
40	3.31e-4	2.08	13.7198	5.39
80	8.06e-5	2.04	319.70	4.54

此处我们采用的 CFL 条件数为 0.1, 计算终止时间为 1，进程数为 4.

虽然受到诺依曼边界条件的影响，我们的理论数值精度只有 1 阶，但实际计算的结果显示由 2 阶精度。

由于我们固定 CFL 条件数为 0.1，因此实际计算量为 $O(\frac{N^3}{\tau}) = O(N^5)$，与实际计算结果较为匹配。

4.2 并行效率分析

我们的测试函数仍然为(4), 对不同的进程数，得到的计算结果如下:

进程数	1	2	3	4	5	6	7	8
CPU 时间 (s)	50.8490	26.5723	17.8507	13.5908	14.0187	11.8521	10.2867	9.5289
加速比	/	1.91	2.85	3.74	3.63	4.29	4.94	5.34
效率	/	95.7%	95.0%	93.5%	72.5%	71.5%	70.6%	66.7%

此处我们采用的 CFL 条件数为 0.1，终止时间为 1，网格密度 $N = 40$.

通过这些数据，我们可以发现该程序的并行效率较高，但当核数多时效率会有所下降。经过比较，该程序最合适的进程数应设置为 4.

5 上机报告总结

此次上机作业，我们使用了 MPI 进行了第一次编程实践，对 MPI 的机理有了初步的认识，并对网格进行了一定规则的划分，使其负载尽量均衡。

此外，我们计算了精确的每个进程需要索取的数据，这是对并行传输数据的一个简单的实践。