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 12 月 10 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

$$-\pi \le x, y, z \le \pi,$$

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

直接求解这个方程较为困难，我们此处采用迭代法求解，迭代格式如下:

$$-\Delta u^{(n+1)} + \lambda u^{(n+1)} = f - (u^{(n)})^3 + \lambda u^{(n)}, \quad \lambda > 0. \tag{2}$$

迭代的初始值可以任取，比如我们取

$$u^{(0)}(x, y, z) = u_0(x, y, z) = 0. \tag{3}$$

而对于迭代格式对应的偏微分方程(2), 我们可以采用 Fourier 变换的方法来求解。

对方程(2)左右两边同时作用 Fourier 变换，

$$-\widehat{\Delta u^{(n+1)}} + \lambda \widehat{u^{(n+1)}} = \widehat{f} - \widehat{(u^{(n)})^3} + \lambda \widehat{u^{(n)}}. \tag{4}$$

在实际计算中，我们将求解区域划分为 $N \times N \times N$ 的正方体网格，对函数 u 的 Fourier 变换即为对 u_{ijk} 的三维离散 Fourier 变换。

注意到

$$-\widehat{\Delta u}_{ijk} = (i^2 + j^2 + k^2)\widehat{u}_{ijk}, \tag{5}$$

从而方程(4)可以写为

$$(i^2 + j^2 + k^2 + \lambda)\widehat{u}_{ijk} = \widehat{f} - \widehat{(u^{(n)})^3} + \lambda \widehat{u^{(n)}}, \tag{6}$$

从而我们的迭代算法如下:

1. 计算迭代初值 $u^{(0)} = u_0$，并令 $n = 0$。
2. 计算迭代格式(2)的右端项 $f - (u^{(n)})^3 + \lambda u^{(n)}$, 并做 Fourier 变换得到 $\widehat{f} - \widehat{(u^{(n)})^3} + \lambda \widehat{u^{(n)}}$。
3. 计算 $\widehat{u^{(n+1)}}$,
$$\widehat{u^{(n+1)}}_{ijk} = \frac{1}{i^2 + j^2 + k^2 + \lambda}(\widehat{f} - \widehat{(u^{(n)})^3} + \lambda \widehat{u^{(n)}}).$$
4. 利用 Fourier 逆变换计算 $u^{(n+1)}$。
5. $n = n + 1$。如果满足终止条件，则结束计算；否则回到第二步。

3 程序分析

3.1 程序实现

我们用 MPI 结合 FFTW 中一维的 DFT 函数来实现三维的 Fourier 变换。

3.1.1 进程的拓扑结构

将求解区域划分为 $N \times N \times N$ 的正方体网格，假设使用的进程数 size $= Np^3$，则每个进程的数据为 M^3，其中 $M = \frac{N}{Np}$。

我们按照 z 从小到大，y 从小到大，x 从小到大的顺序来给进程编号和对进程内的数据进行编号。则 rank$\%Np$ 表示进程在 x 方向的位置编号，(rank$/Np$)$\%Np$ 表示进程在 y 方向的位置编号，rank$/Np/Np$ 表示进程在 z 方向的位置编号。

从而，我们可以为每个方向建立一个通讯器。每个通讯器包括所有在这个方向处于同一条直线的进程，将一共 Np^3 个进程划分为 Np^2 个组，每组包含 Np 个进程。

3.1.2 三维 Fourier 变换的实现

由于三维的离散 Fourier 变换等价于三个方向的一维离散 Fourier 变换的复合，因此我们可以通过对三个方向分别进行一维 Fourier 变换来实现三维的离散 Fourier 变换。

以 x 方向 DFT 为例，$0, 1, \ldots, Np-1$ 这 Np 个进程位于同一条直线上，这些进程一共包含了 $M \times M \times N$ 的数据，共需要进行 M^2 次长度为 N 的一维 DFT。为了充分利用所有进程，我们把数据重新分配给每个进程，让每个进程执行 $\frac{M^2}{Np}$ 次 DFT，再将数据重新传递给原来的进程，从而完成了 x 方向的 DFT。

由于 x 方向连续的数据在内存上也连续，但 y, z 方向不然，从而接下来执行另外两个方向的 DFT 时，需要将数据重新排列。此外，我们采用 MPI_Alltoall 函数进行同一个通讯器的所有进程间的数据传递，由于数据排列规则的问题，再传递结束之后需要进行一次数据的重新排列才能调用 FFTW 的函数进行运行。这些重新排列的细节在此略去。

3.2 程序评价

我们这次采用的程序的优点有:

- 运行时所有进程都执行 FFT，而且每个进程的任务一致，负载较为均衡。
- 只使用了存数据的内存和一个临时变量的内存，没有浪费任何内存空间。
- 传递信息时只传递了需要的信息，没有浪费。

但还有以下几点不足:

- 尝试使用 MPI_DataType 来定义列向量，从而实现快捷的信息传递和接收，但由于 MPI_Alltoall 函数需要传递给 i 号进程的信息严格位于传递给 j 号进程的信息之前

$(i < j)$，从而这个方案无法生效（或许是我程序写的不对）。从而在我们的程序中只能采用先对数据进行重新排列，再进行传递，浪费了不少的执行效率。

4 数值结果

4.1 误差和收敛性分析

我们此次试验计算了几个有真解的数值算例，

$$u(x, y, z) = \sin(x) \tag{7}$$

得到的计算结果如下：

网格密度 N	误差	CPU 时间 (s)
16	4.64e-15	0.018
32	4.64e-15	0.100
64	4.61e-15	1.391
128	4.62e-15	20.22

此处我们采用的进程数为 8，$\lambda = 10$，迭代步数为 100 步，迭代初始值 $u_0 = 0$。

对于另外一个算例，

$$u(x, y, z) = \sin(x) + 10 \tag{8}$$

经过实验发现，$\lambda = 10$ 时格式不收敛，此处我们修改 $\lambda = 1000$，以确保其收敛性。仍然采用 8 进程，迭代 100 步，得到的结果如下：

网格密度 N	误差	CPU 时间 (s)
16	1.25e-11	0.021
32	1.25e-11	0.077
64	1.25e-11	1.366
128	1.25e-11	20.22

可以发现只要参数 λ 取得足够大，就能保证格式的收敛性。而采用之前提议的格式

$$-\Delta u^{(n+1)} = f - (u^{(n)})^3 \tag{9}$$

对测试函数 $u(x, y, z) = \sin(x)$，在我个人试验中，是无法保证收敛性的。

但另一方面，参数 λ 取得过大会影响迭代的收敛速度，因此如何给出一个合适的 λ 是一个问题。我个人给了一种方案，选取右端项 f 的最大值的一个倍数，即

$$\lambda = Const * ||f||_\infty.$$

4.2 并行效率分析

为了分析并行计算的效率，我们使用了与之前不同的一台机器，以测试在 1,8,27 进程下计算所需要的时间。我们采取的测试函数的真解为

$$u(x,y,z) = 1 + \sin(x+y+z), \tag{10}$$

网格剖分密度 $N = 108$，计算迭代步数为 100，参数 $\lambda = 10$。在这些条件下，得到的计算结果如下：

进程数目	计算时间 (s)	加速比	效率
1	41.92	/	/
8	6.24	6.72	83.97%
27	4.55	8.21	34.12%

由于在初始化的时候我们申请了所有的 fftw_plan，以便重复使用，所以可以预期当迭代步数增加时，运行时间将得到一定优化，以下我们对之前相同的实验数据，将迭代步数修改为 1000，进行重新计算，结果如下：

进程数目	计算时间 (s)	加速比	效率
1	411.98	/	/
8	62.64	6.62	82.74%
27	36.60	11.26	41.69%

可以发现对 27 进程，确实随着迭代步数的增加，效率会提高。

5 上机报告总结

此次上机作业，我们针对一个偏微分方程，设计了一个迭代法求求解，再利用迭代格式的性质，采用 Fourier 变换来求解这个迭代格式。

本次的并行算法，我们采用了并行实现一维的 FFT，再利用迭一维的 FFT 来完成三维的 FFT 运算。

FFT 由于需要传递的信息量较大，本身是不适合作为并行算法的，本次实验结果也显示其并行效率不是甚高。

最后，由于原格式在测试下的不稳定性，我们对其进行了修改，最终所采用的格式为(2)。经实验表明，该格式不用去求解关于常数 C 的三次方程，而且是一个收敛的数值格式，能得到比较好的结果。