泊松方程（一维、二维）的并行共轭梯度解法

物理学院 李哲 111120078

一、 问题的提出：

现有一维泊松方程问题：

$$\begin{cases} -u'' = f = \pi^2 \sin \pi x \\ x \in (0,1) \\ u(0) = u(1) = 0 \end{cases} \tag{1}$$

其解析解为

$$u = \sin \pi x \tag{2}$$

与二维泊松方程问题：

$$\begin{cases} -\Delta u = f = 8\pi^2 \cos(2\pi x)\sin(2\pi x) \\ x, y \in (0,1)^2 \\ u(x,0) = u(x,1) = 0 \\ u(0,y) = u(1,y) = \sin 2\pi y \end{cases} \tag{3}$$

其解析解为

$$u = \cos(2\pi x)\sin(2\pi y) \tag{4}$$

这里用中心差分方法对泊松方程进行离散化，并使用并行的共轭梯度法求解得到的线性方程组，将得到的结果与精确解比较。

共轭梯度法是一种收敛速度很快的方法，其基本思想是将解线性方程组 $Ax = B$ 的问题转化为求函数 $f = \frac{1}{2} x^T Ax - b^T x + c$ 极值的问题。这里要求方阵 A 是正定对称的。用共轭梯度法解泊松方程离散化导出的线性方程组，效果很好，所需的迭代次数较少。

二、算法实现

对于一维泊松方程，由二阶中心差分

$$u'' = \lim_{h \to 0} \frac{u(x-h) - 2u(x) - u(x+h)}{h^2} \tag{5}$$

设从 0 到 1 分为 t 个网格，则要求的是 $k = t - 1$ 个点上的值，分别是 $a_1 = \frac{1}{k}$，$a_2 = \frac{2}{k}$，……，$a_{k-1} = \frac{k-1}{k}$。将(5)代入(1)，可得方程

$$-u_{i-1} + 2u_i - u_{i+1} = h^2 f_i \tag{6}$$

考虑到边界条件，得到的是一组三对角方程组。在这里，将系数矩阵存储为一个 $k \times 3$ 的矩阵，并分配给每个进程，如下图：

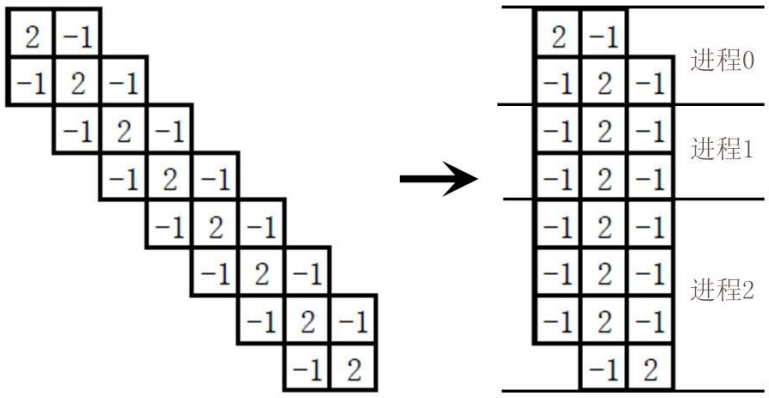

图 1 一维泊松方程的系数矩阵及分配

对应的向量的分配与系数矩阵的分配方式相同。在共轭梯度法的应用时，计算内积操作时每个进程时只使用到分配给自己的一部分向量，作完内积后汇总求和即得总的内积；计算矩阵-向量乘法操作时，每个进程获得分配给自己的一部分系数矩阵和完整的向量进行计算，将得到的向量汇总即得到完整的向量。按照图 1 所示的分配方法可以满足要求。

对于二维泊松方程，与一维泊松方程进行类似的离散化操作。由于题目所给的方程是一个正方形区域，因此在 x 方向和在 y 方向进行相同的划分，可得方程

$$-u_{i-1,j}-u_{i,j-1}+4u_{i,j}-u_{i+1,j}-u_{i,j+1}=h^2 f_{i,j} \tag{7}$$

取遍 i 和 j 得到一个 $k\times k$ 阶的线性方程组。为了将这个线性方程组表示为便于求解的形式，将每个方程如图 2 所示编号：

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35

图 2 线性方程组的编号

考虑到本题中的边界条件，按图 2 所示方式编号得到的线性方程组对应的系数矩阵是每行有 3～5 个元素的对角占优、正定对称的矩阵，可以用共轭梯度法求解。

线性方程组对应的向量有 $k\times k$ 个元素，为了后面理解上的方面，可以将这个向量“想象”为一个 k 阶的矩阵，第 i 行 j 列的元素对应于向量中第 $ik+j$ 个元素。将向量作类似于一维情况中的划分，但是注意到，由于问题变为二维的情况，数据量是一维情况的平方，如果再将整个向量复制到每个进程中并进行计算，通信量会非常大，在内存的使用上也可能出现问题。另外，在作矩阵向量乘法时，如果仅仅简单地按一维情况中分配系数矩阵的方法来分配

的话，在计算时还需要用到相邻进程中的数据。因此这里考虑另一种办法，即每个进程额外复制一些数据，保证每个进程里面包含了所有计算需要用到的数据来避免进程间再通信。具体如图 3 所示。

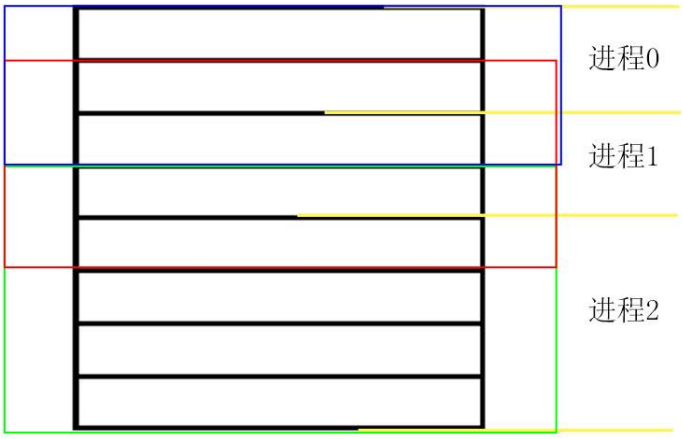

图 3

图 3 中，在矩阵-向量乘法过程中，蓝色区域对应进程 0 需要的所有元素，红色区域对应进程 1 需要的所有元素，而绿色区域对应进程 2 需要的所有元素。每个进程的矩阵-向量乘法的结果对应黄色线内的区域。

对于向量的内积操作，每个进程不需要额外的数据，数据分配方式即为“传统”的按照黄色线分配。

程序中，对于 MPI_Scatterv 和 MPI_Allgatherv 等过程，disp0[]、counts0[]数组用来存储每个进程用到的数据的偏移量和数量，而 disp[]、counts[]数组用来存储包含了额外数据的每个进程的数据的偏移量和数量。对于图 3 中的情况，有：

disp0[0]=0, disp0[1]=2, disp0[2]=4

counts[0]=2, counts0[1]=2, counts0[2]=4

disp[0]=0, disp[1]=1, disp[2]=3

counts[0]=3, counts[1]=4, counts[3]=5

由于向量中每个元素对应于离散化的区域中的每一个点，且系数矩阵只含有-1、4 这两个数值，另外系数矩阵是稀疏的，因此不需要将系数矩阵存储下来，可以直接根据相应的边界条件来求矩阵-向量乘法。然而，由于问题是二维的，有 4 个边界，处理起来比较复杂，对应于不同进程的数每个进程的任务都需要仔细考虑，另外由于每个进程获得了额外的数据，数据的编号发生改变，也需要考虑，因此程序中的 matrix_vector_product 函数考虑到了各种情况，比较复杂。

三、误差分析

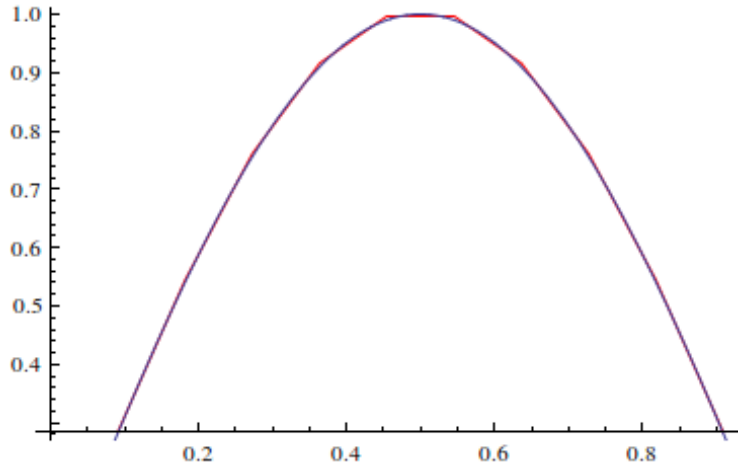

图 4　$k=10$ 时精确解与数值解对比（黑色为精确解，红色为数值解）

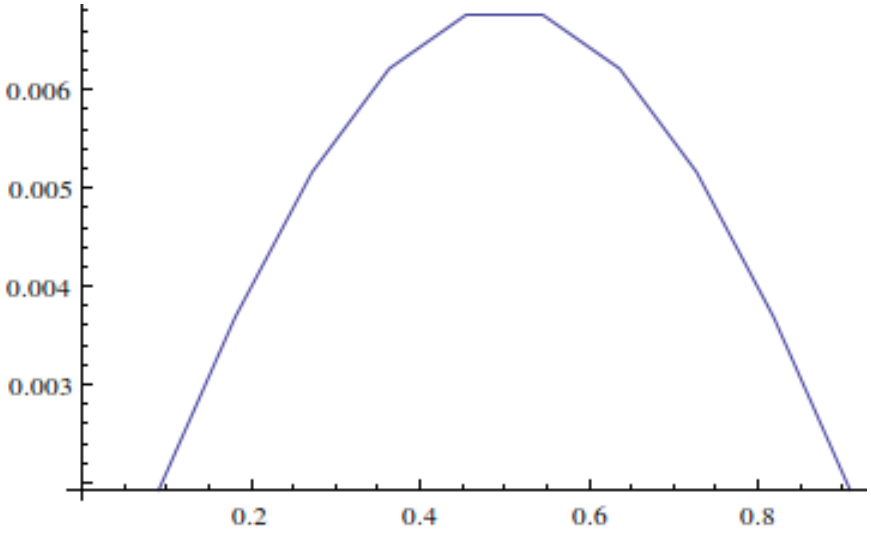

图 5　$k=10$ 时误差

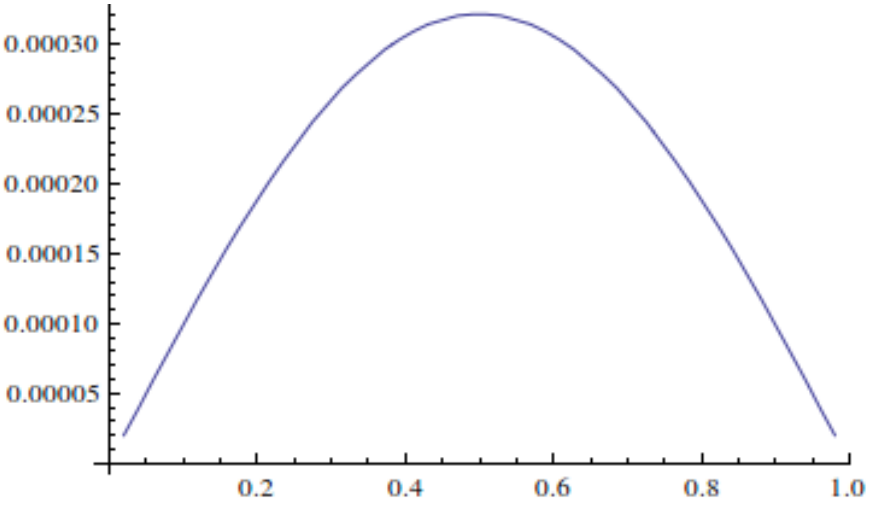

图 6　$k=50$ 时误差

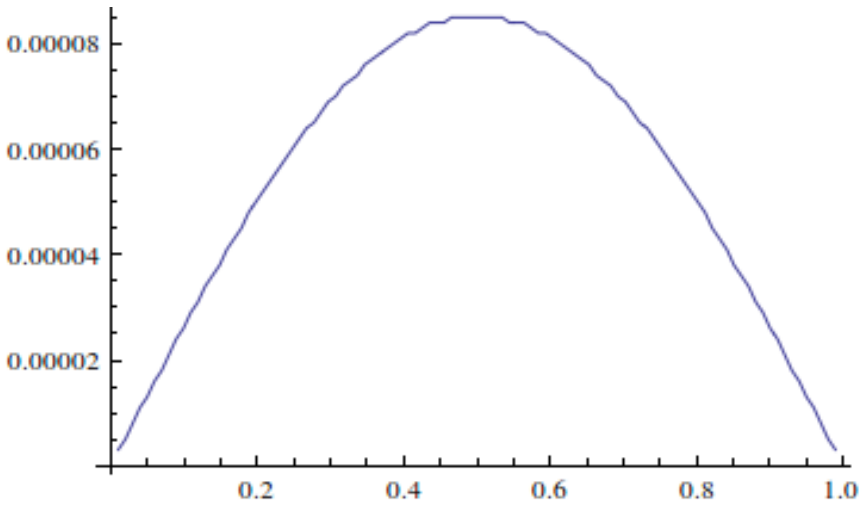

图 7 $k=100$ 时误差

二维泊松方程：

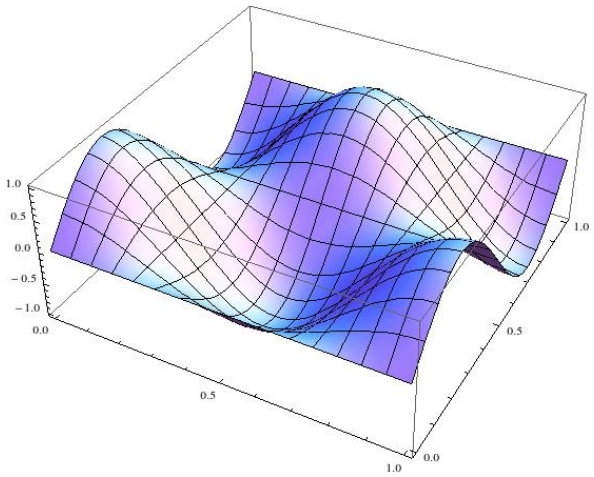

图 8 精确解

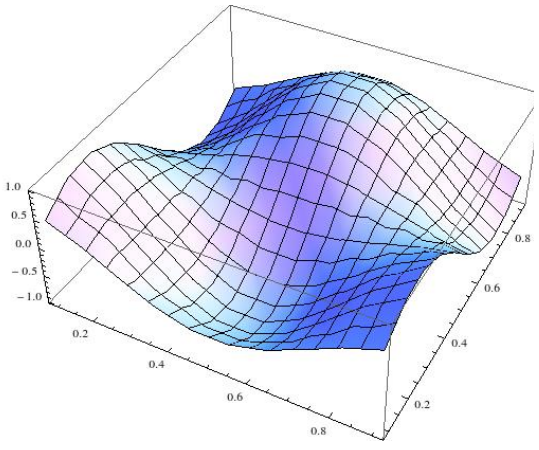

图 9 $k=10$ 时的数值解

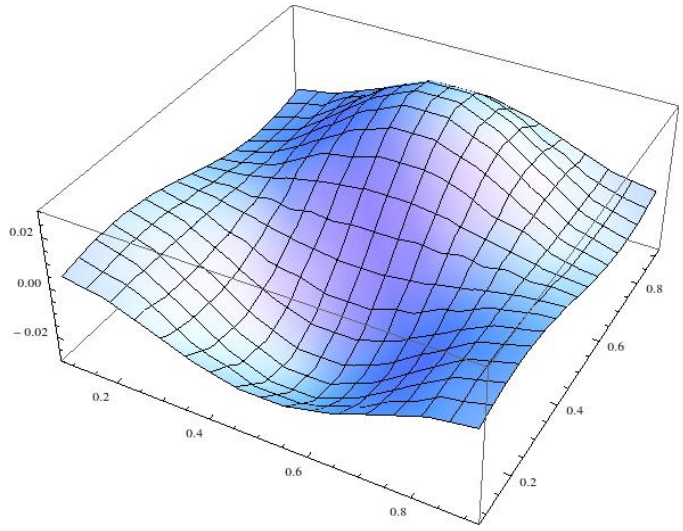

图 10 $k=10$ 的误差

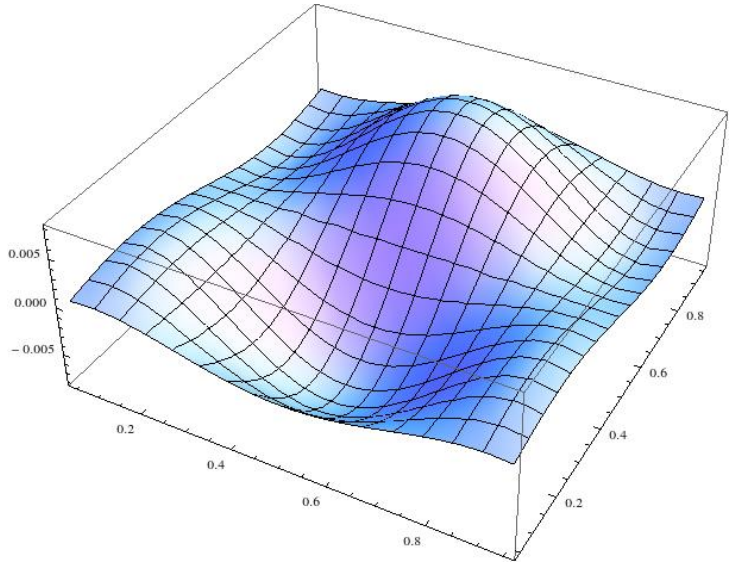

图 11 $k=20$ 的误差

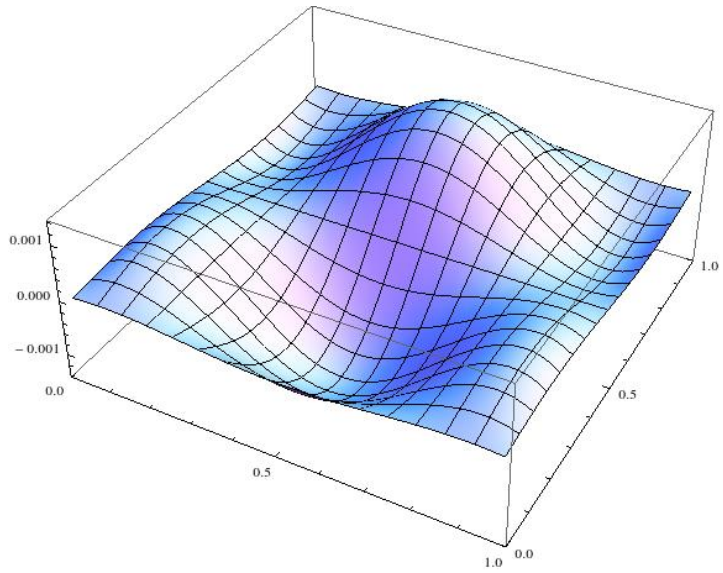

图 12 $k=50$ 的误差

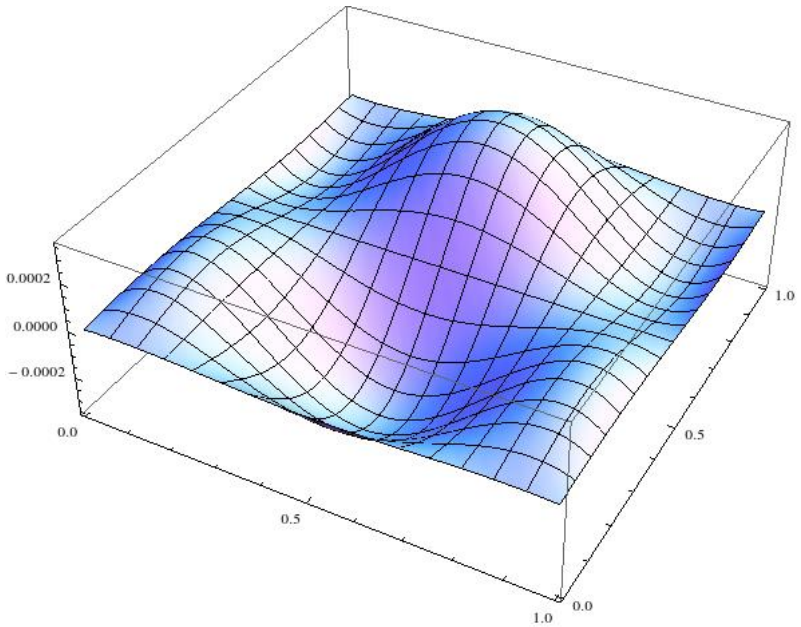

图 13 $k=100$ 时的误差

可以证明：如果二维泊松方程的第一边值问题

$$\begin{cases}\Delta u = f(x,y),(x,y)\in D \\ u(x,y)=\alpha(x,y),(x,y)\in \partial D\end{cases} \tag{8}$$

的解在 $D\cup\partial D$ 上有四阶连续的偏导数，则中心差分格式收敛且有估计式

$$\max_{D_A}\left|u_{ij}-u(x_i,y_i)\right|\le K\left(h^2+k^2\right) \tag{9}$$

对比上面各图，结论是一致的；而对于一维泊松方程，也有类似的结果。

四、并行效果分析：

对于不同的k、不同的处理器数，每种情况运行 5 次取运行（不含输出）时间的平均值，结果如下：

处理器数	1	2	3	4
运行时间/s	0.0010588	0.0003766	0.0002996	0.0004098
加速比	1	2.811471057	3.534045394	2.583699366
效率	1	1.405735528	1.178015131	0.645924841

表 1 一维泊松方程的并行运行结果($k=100$)

处理器数	1	2	3	4
运行时间/s	0.0360308	0.0238146	0.0209852	0.0181868
加速比	1	1.512971035	1.716962431	1.981151165
效率	1	0.756485517	0.57232081	0.495287791

表 2 一维泊松方程的并行运行结果($k=1000$)

处理器数	1	2	3	4
运行时间/s	0.01814	0.0116492	0.0139766	0.0102382
加速比	1	1.557188476	1.297883605	1.771795823
效率	1	0.778594238	0.432627868	0.442948956

表 3 二维泊松方程的并行运行结果($k=50$)

处理器数	1	2	3	4
运行时间/s	0.049604	0.034005	0.0367816	0.0363376
加速比	1	1.458720776	1.34860365	1.365081899
效率	1	0.729360388	0.44953455	0.341270475

表 4 二维泊松方程的并行运行结果($k=100$)

对于数据规模较小的情况，运行的时间会出现一些波动；当处理器增加时，加速比没有明显的上升。进行共轭梯度算法中的每一步基本上都实现了并行，但是效率并不高，说明数据的传输量比较大，一定程度上抑制了并行的效果。

程序刚开始时构造向量的过程没有并行化，但是构造向量的运算量相对来解方程说很少，另外实际计时得到的结果表明构造向量的时间基本可以忽略，因此在讨论并行效果的时候可以不考虑串行构造向量带来的影响。

五、进一步讨论

在一维泊松问题的求解中，为了方便起见，构造了一个与系数矩阵等效的$k\times 3$矩阵，后面的计算中是直接进行一般的矩阵-向量乘法。这种方法在解泊松问题的时候比较浪费，因为系数矩阵是稀疏的，进行了大量无意义的乘法运算，而且系数是一致的，计算的时候不需要将系数存储起来；不过这种构造等效的$k\times 3$矩阵的方法适用于解一般的三对角方程组，且有较好的效果。

此外，在进行矩阵-向量乘法的过程中，为了方便起见采用了复制向量的方法，而实际上也可以采用解二维情况中的方法，将乘法过程所需的额外的向量的数据和自己对应的一部分分配给每个进程，这样就不用复制整个向量。在二维泊松问题的求解中，在矩阵-向量乘法之中分配给各个进程的向量是先进行了一次 MPI_Gatherv 后进行一次 MPI_Scatterv 得到的，而实际上并不需要这样，只要各进程之间相互交换数据即可，但这样做在程序编写上带来了更多的工作，处理起来会很麻烦。如果实现的话对减少通信量有很大帮助。

在二维泊松问题的求解中没有存储系数矩阵，而是直接根据边界情况进行计算，这在这个问题中减少了很多计算量，但是只适用于二维泊松方程这种特殊的椭圆方程，如果系数矩阵更复杂一些就不适用了。

二维泊松方程求解的 matrix_vector_product 函数中的代码重复的情况可以减少。

另外正如前面提过的，如果将程序开头构造向量的部分也并行化，可以减少一些运行的时间，不过由于构造向量部分的计算并不多，对总运行时间的影响不大。