拉普拉斯方程解题报告

1. 问题描述

给定一个正方形均质的薄板，板的顶部和底部绝缘，除板的边缘外，板上任意点的温度完全由它周围的点的温度决定。板的边缘温度固定，三边被蒸汽所环绕(100℃)，第四边接触着冰块(0℃)。求出盘上各点的稳态温度分布[1]。

2. 解题思路

2.1 分析

属于稳态热传导(steady-state heat transfer)问题[2]，可用拉普拉斯方程(Laplace's equation[3])描述，加上边界条件，在数学上属于狄利克雷问题(Dirichlet problem[4])，存在唯一解。该问题的数学形式如下：

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=0 \qquad (1)Laplace's\ equation$$

$$\text{s.t.}$$

$$\begin{cases}T(x,0)=C_0\\T(x,b)=C_1\end{cases},0\le x\le b$$

$$\begin{cases}T(y,0)=C_2\\T(y,b)=C_3\end{cases},0\le y\le b$$

其中 x、y 是正方形(边长为 b)内点的坐标，函数 T(x, y)是在稳态时点(x, y)处的温度，C_0-C_3对应着各个边的温度，是常数。

2.2 求解

可用有限差分法(Finite difference method[5])得到该方程的数值解。该方法的基本思路是用差分(difference)近似微分(differential)，将微分方程转换成一系列差分方程，求解差分方程组即可得到数值解。

2.3 公式推导

如下图，将二维平面离散化

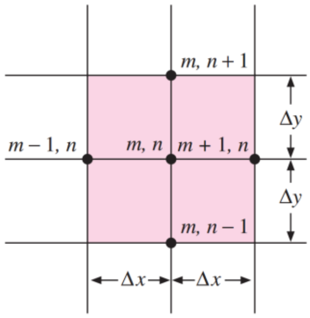

Figure 1 **二维平面离散化**

差分近似微分

$$\left.\frac{\partial \mathrm{T}}{\partial \mathrm{x}}\right|_{m+\frac{1}{2},n} \approx \frac{T_{m+1,n}-T_{m,n}}{\Delta x} \tag{2}$$

$$\left.\frac{\partial \mathrm{T}}{\partial \mathrm{x}}\right|_{m-\frac{1}{2},n} \approx \frac{T_{m,n}-T_{m-1,n}}{\Delta x} \tag{3}$$

$$\left.\frac{\partial \mathrm{T}}{\partial \mathrm{x}}\right|_{m,n+\frac{1}{2}} \approx \frac{T_{m,n+1}-T_{m,n}}{\Delta y} \tag{4}$$

$$\left.\frac{\partial \mathrm{T}}{\partial \mathrm{x}}\right|_{m,n-\frac{1}{2}} \approx \frac{T_{m,n}-T_{m,n-1}}{\Delta y} \tag{5}$$

则有

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{m,n} \approx \frac{\left.\frac{\partial T}{\partial x}\right|_{m+\frac{1}{2},n}-\left.\frac{\partial T}{\partial x}\right|_{m-\frac{1}{2},n}}{\Delta x} = \frac{T_{m+1,n}+T_{m-1,n}-2T_{m,n}}{(\Delta x)^2} \tag{7}$$

$$\left.\frac{\partial^2 T}{\partial y^2}\right|_{m,n} \approx \frac{\left.\frac{\partial T}{\partial y}\right|_{m,n+\frac{1}{2}}-\left.\frac{\partial T}{\partial y}\right|_{m,n-\frac{1}{2}}}{\Delta y} = \frac{T_{m,n+1}+T_{m,n-1}-2T_{m,n}}{(\Delta y)^2} \tag{8}$$

将上面的两个式子带入拉普拉斯方程(公式 1)得到

$$\frac{T_{m+1,n}+T_{m-1,n}-2T_{m,n}}{(\Delta x)^2}+\frac{T_{m,n+1}+T_{m,n-1}-2T_{m,n}}{(\Delta y)^2}=0 \tag{9}$$

令$\Delta \mathrm{x}=\Delta \mathrm{y}$，得到

$$\mathrm{T_{m+1,n}}+\mathrm{T_{m-1,n}}+\mathrm{T_{m,n+1}}+\mathrm{T_{m,n-1}}-4\mathrm{T_{m,n}}=0 \tag{10}$$

针对每个内部节点，均可由公式 10 得到一个方程，组合所有内部节点的方程得到一个线性方程组，求解这个线性方程组即可得到每个内部节点的温度。

$$\begin{gathered}\mathrm{a}_{1,1}T_1+a_{1,2}T_2+\cdots+a_{1,n}T_n=0\\ \mathrm{a}_{2,1}T_1+a_{2,2}T_2+\cdots+a_{2,n}T_n=0\\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\\ \mathrm{a}_{n,1}T_1+a_{n,2}T_2+\cdots+a_{n,n}T_n=0\end{gathered}$$

即

$$\boldsymbol{A}\times\boldsymbol{T}=\mathbf{0} \tag{11}$$

其中

$$\boldsymbol{A}=\begin{bmatrix}a_{1,1} & \cdots & a_{1,n}\\ \vdots & \ddots & \vdots\\ a_{n,1} & \cdots & a_{n,n}\end{bmatrix}$$

$$\boldsymbol{T} = \begin{bmatrix} T_1 \\ \vdots \\ T_n \end{bmatrix}$$

$\boldsymbol{T}$代表每个内部节点的温度，$\boldsymbol{A}$是将公式 10 扩展之后变量的系数，注意，每个内部节点的温度只与 4 个邻居节点的温度有关，所以$\boldsymbol{A}$是个稀疏矩阵。

2.4 求解线性方程组

求解稀疏的线性方程组一般使用雅可比算法(Jacobi method[6])，该算法的一个显著优点是容易并行化。算法的伪代码如下：

```
Input:  初始值t^(0)和A
Output:  t的数值解
Comments:  t是T中的一个变量

k = 0
while convergence not reached do
        for i := 1 step until n do
                σ = 0
                for j := 1 step until n do
                        if  j ≠ i  then
                                σ = σ + a_{i,j} x_j^(k)
                        end
                  end
                  x_i^(k+1) = (1/a_ii)(b_i − σ)
        end
         k = k + 1
end
```

3. 实现

因为拉普拉斯方程的系数矩阵 A 是稀疏的，所以在实现雅可比算法时并不需要遍历元素所在的列，更新等式可简化成

$$\mathrm{x}_{\mathrm{i}}^{(\mathrm{k}+1)} = \frac{\mathrm{x}_{\mathrm{i,up}}^{(\mathrm{k})} + \mathrm{x}_{\mathrm{i,down}}^{(\mathrm{k})} + \mathrm{x}_{\mathrm{i,left}}^{(\mathrm{k})} + \mathrm{x}_{\mathrm{x,right}}^{(\mathrm{k})}}{4} \tag{12}$$

3.1 串行化实现

```
void solve_equations(float* T, int length, float tolerance) {
    float *Tn = new float[length*length];
    for(int i = 0; i < length; i++) {
        for (int j = 0; j < length; j++) {
            Tn[i*length+j] = T[i*length+j];
        }
    }
    float *T0 = T, *Tk = Tn;
    int N = length - 1;
    while(true) {
        float max_dif = -INFINITY;
        for(int i = 1; i < N; i++) {
            for(int j = 1; j < N; j++) {
                float t = (T0[length*(i-1)+j]+
                        T0[length*(i+1)+j]+
                        T0[length*i+j-1]+
                        T0[length*i+j+1])/4;
                Tk[i*length+j] = t;
                max_dif = max(max_dif, abs(t-T0[i*length+j]));
            }
        }
        if(max_dif <= tolerance) {
            break;
        }
        else {
            swap(T0, Tk);
        }
    }
    delete [] Tn;
}
```

3.2 并行化实现

使用 MPI 和 OpenMP 做并行化，主节点基于 MPI 将矩阵按行分发给每个子节点，子节点基于 OpenMP 计算分配的任务。雅可比算法的每次迭代，主节点都会和子节点交换数据，直到算法收敛。

主节点基于 MPI 的任务分发代码

```
        while(true) {
            float max_dif = -INFINITY;

            for(int i = 1; i < world_size; i++) {
                MPI_Send(data0+senddispls[i], sendcounts[i],
                    MPI_FLOAT, i, 0, MPI_COMM_WORLD);
            }
            memcpy(part0, data0+senddispls[world_rank], sizeof(float)*sendcounts[world_rank]);
            max_dif = max(max_dif, solve_part_equations(
                part0, partk, sendcounts[world_rank]/length, length));
            memcpy(datak+senddispls[world_rank], partk, sizeof(float)*sendcounts[world_rank]);
            for(int i = 1; i < world_size; i++) {
                MPI_Recv(datak+recvdispls[i], recvcounts[i],
                    MPI_FLOAT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                float dif = INFINITY;
                MPI_Recv(&dif, 1, MPI_FLOAT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                max_dif = max(max_dif, dif);
            }
            if(max_dif <= tolerance) {
                break;
            }
            else {
                swap(data0, datak);
            }
        }
```

子节点基于 OpenMP 的代码

```
float solve_part_equations(float *p0, float *pk, int rows, int cols) {
    float max_dif = -INFINITY;
    #pragma omp parallel for collapse(2) reduction (max:max_dif)
    for(int i = 1; i < rows-1; i++) {
        for(int j = 1; j < cols-1; j++) {
            float t = (p0[i*cols+j-1]+p0[i*cols+j+1]+p0[(i-1)*cols+j]+p0[(i+1)*cols+j])/4;
            pk[i*cols+j] = t;
            max_dif = max(max_dif, abs(t-p0[i*cols+j]));
        }
    }
    return max_dif;
}
```

4. 实验截图

Figure 2 一万内部节点并行化

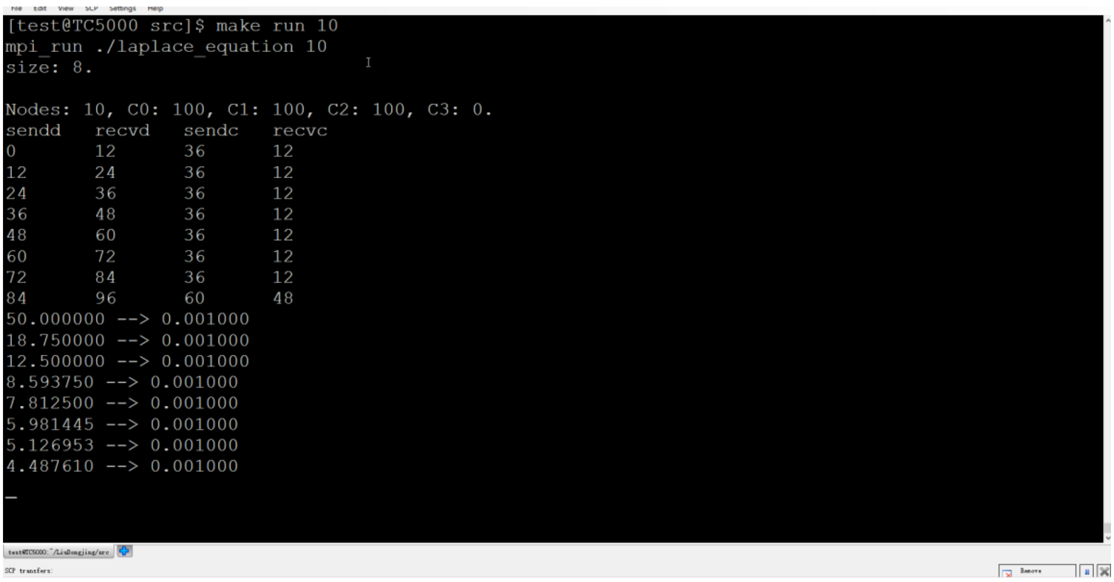

Figure 3 一百内部节点，开始

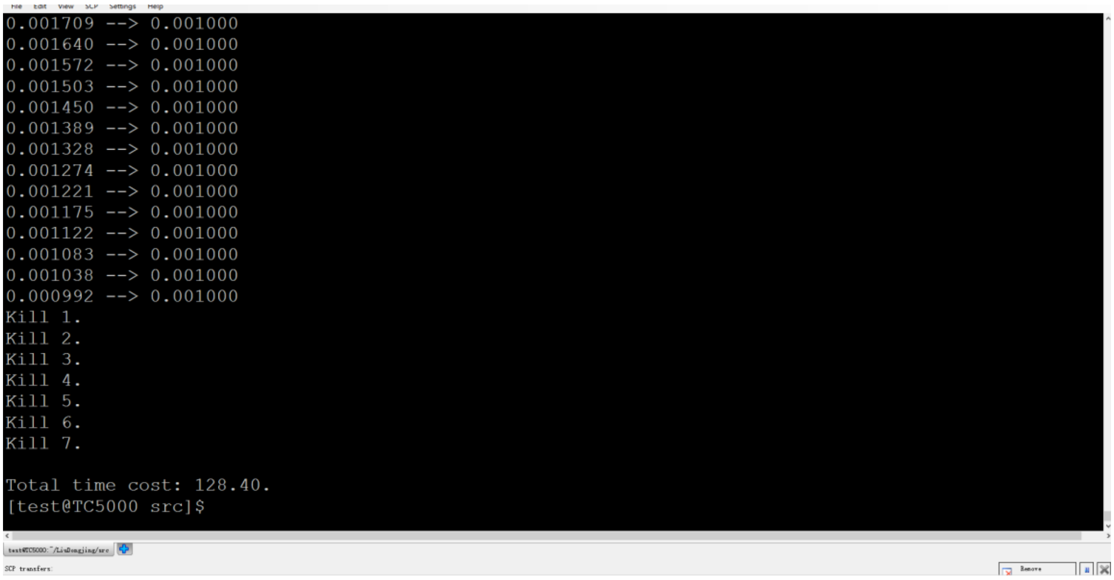

Figure 4 **一百内部节点，结束**

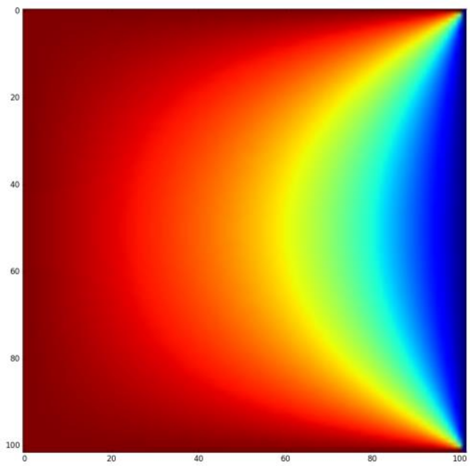

Figure 5 **一万内部节点，可视化**

5. 存在的问题

并行化代码中，主节点和子节点每次迭代都会交换数据，通信开销比较大。对解的精度要求不高(内部节点少)时，串行的算法更高效。所以并行化算法还有较大的优化空间，比如减少传输的数据量、主节点与子节点的通信并行化等等。

参考文献

[1] 赵旭升。MPI+OpenMP 混合求解偏微分方程
[2] http://jingweizhu.weebly.com/uploads/1/3/5/4/13548262/steady-state_conduction_multiple_dimensions.pdf

[3] https://en.wikipedia.org/wiki/Laplace%27s_equation
[4] https://en.wikipedia.org/wiki/Dirichlet_problem
[5] https://en.wikipedia.org/wiki/Finite_difference_method
[6] https://en.wikipedia.org/wiki/Jacobi_method