并行计算在计算物理问题中的应用

1 计算物理问题并行求解的一般步骤

计算物理问题的并行求解一般由以下几步：

1 建立物理问题的数学模型

2 物理问题计算域的离散

3 控制方程离散

4 求解离散的控制方程

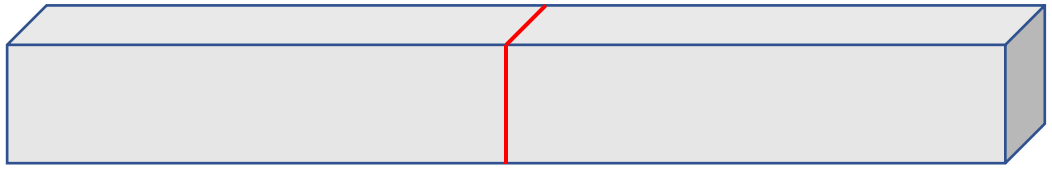
5 求解过程并行化

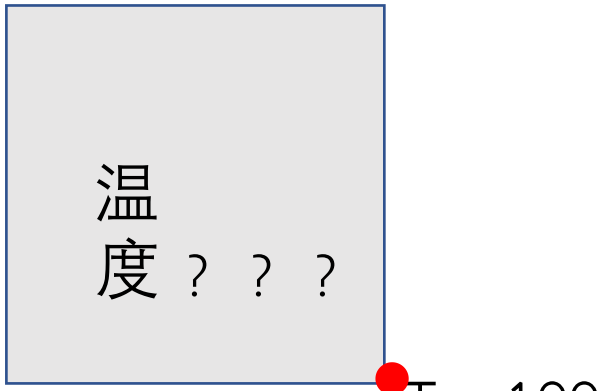
6 优化并行求解过程

2 物理问题描述

对无限长的长方体金属块进行加热，该长方体的截面为正方形，通过一定的加热方式能够保证长方体金属块的温度保持恒定。该三维问题可以看作一个计算域为二维正方形的热传导问题。

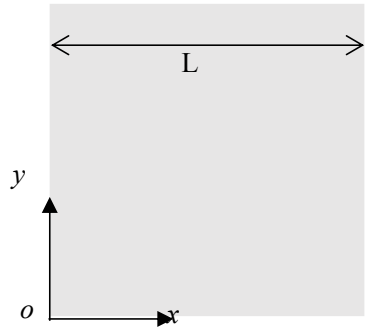
具体的加热方式描述为对正方形截面右下角进行加热，保持右下角温度T_0=100摄氏度(℃)，控制加热方式，使得在右下角相交的两个边的温度从100 ℃线性降低到0 ℃ 。保持其他两个边的温度为0 ℃。温度达到稳定状态后是如何在正方形截面上分布的呢？





3 建立数学模型——控制方程

建立数学模型，在如图所示笛卡尔坐标系统(xoy)下构建边长为L的方形计算域，未知变量温度T定义在方形计算域上，对于热传导问题，温度T在金属块的分布满足Laplace方程，即



$$\frac{\partial^2 \mathrm{T}}{\partial \mathrm{x}^2}+\frac{\partial^2 \mathrm{T}}{\partial \mathrm{y}^2}=0$$

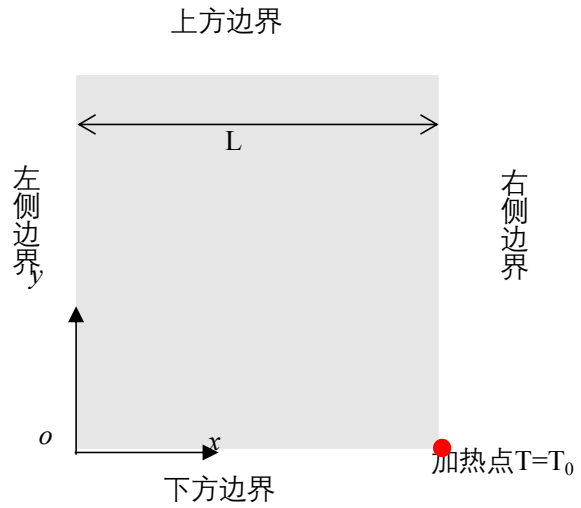
该物理问题的控制方程是一个二阶偏微分方程。在正方形计算域及其边界上的温度分布都应满足这个控制方程[1]。

3 建立数学模型——边界条件

从数学的角度看，偏微分方程只有具备边界条件才能求得计算域中的特定解。

从物理角度看，加热方式的描述实际上就是该物理问题的边界条件。相应的数学语言为：

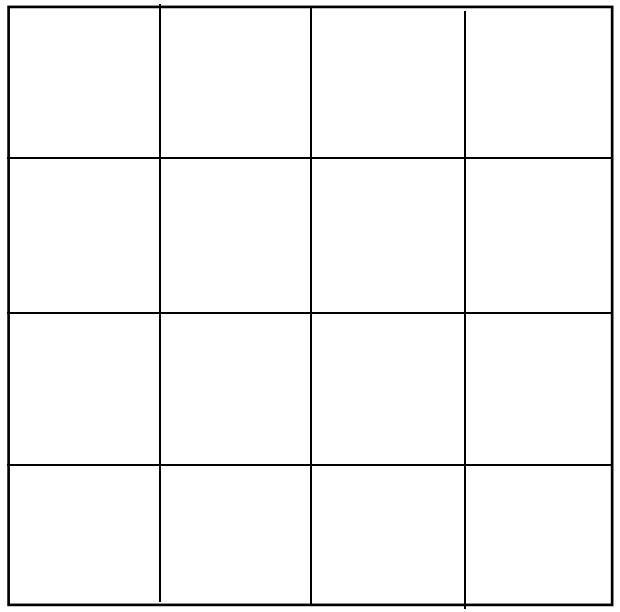
- 上方边界：T=0（y=L，0≤x≤L）
- 左侧边界：T=0（x=0，0≤y≤L）
- 下方边界：$T = \frac{T_0}{L}x$（0≤x≤L，y=0）
- 右侧边界：$T = -\frac{T_0}{L}y + T_0$（0≤y≤L，x=0）



4 计算域离散——计算网格

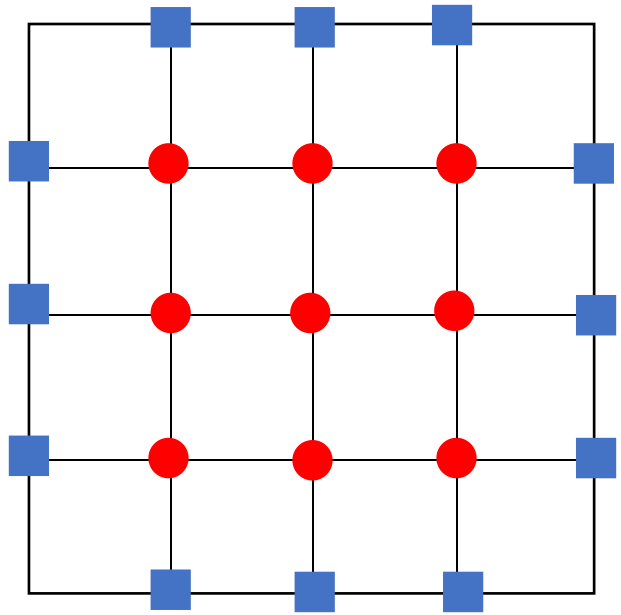
为了采用数值方法求解，将物理空间上连续的计算域离散，以适应计算机的求解方式。

本课程采用有限差分的计算方法，用计算网格填充计算域，如图所示采用4X4的方格离散计算域。



4 计算域离散——计算点边界点

本课程采用有限差分方法，将未知量T分布在计算域内部方格的交叉点上，如图红色圆点所示，这些点是计算点，需要通过数值计算求出。边界上的已知量用蓝色方点表示，在计算过程中由边界条件决定。



5 控制方程离散——差分

控制方程中连续的偏导数$\frac{\partial^2 \mathrm{T}}{\partial \mathrm{x}^2}$和$\frac{\partial^2 \mathrm{T}}{\partial \mathrm{y}^2}$无法在离散的计算域上使用，这些偏导数要用离散计算域中计算点和边界点的差分表示。差商是导数的近似值。使用差商会带来计算误差，差分和计算误差之间的关系可以由泰勒展开得到。

对于在x_0点附近连续可导的函数f(x)，通过泰勒展开可以得到f(x)在x_0点附近的值

$$f(x)=f(x_0)+f'(x_0)\cdot(x-x_0)+\frac{f''(x_0)\cdot(x-x_0)^2}{2!}+\ldots+\frac{f^{(n)}(x_0)\cdot(x-x_0)^n}{n!}+O[(x-x_0)^n\Big]$$

其中，$f^{(n)}(x_0)$是函数f(x)在x_0点的n阶导数。$O[(x-x_0)^n]$是$(x-x_0)^n$的高阶无穷小，即$\lim_{x\to x_0}\frac{O[(x-x_0)^n]}{(x-x_0)^n}=0$

5 控制方程离散——一阶差分格式

由泰勒展开可以得到$f'(x_0)=\frac{f(x)-f(x_0)}{x-x_0}+\frac{f''(x_0)\cdot(x-x_0)}{2!}+\ldots$。$x_0$点附近的点x可以写为$x=x_0+\Delta x$，f(x)在$x_0$点的一阶导数可以通过差商近似表示

$$f'(x_0)=\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}$$

这种差分格式也称为一阶差分格式，差商和导数之间的误差第一项为$\frac{f''(x_0)\cdot(x-x_0)}{2!}$，随着$x-x_0$的减小，差商和导数之间的误差也随之减小，这表明加密计算网格可以减少差商带来的误差。

5 控制方程离散——二阶差分格式

为了减小差分带来的误差，$f'(x_0)$还可以采用高阶差商表示，这里仍用泰勒展开说明二阶差分格式。将x＝ x_0＋Δx和x＝ x_0－Δx分别带入泰勒公式

$$\begin{cases} f(x_0-\Delta x)=f(x_0)-f'(x_0)\cdot\Delta x+\dfrac{f''(x_0)\cdot(\Delta x)^2}{2!}+\ldots+\dfrac{f^{(n)}(x_0)\cdot(-\Delta x)^n}{n!}+O[(-\Delta x)^n] \\ f(x_0+\Delta x)=f(x_0)+f'(x_0)\cdot\Delta x+\dfrac{f''(x_0)\cdot(\Delta x)^2}{2!}+\ldots+\dfrac{f^{(n)}(x_0)\cdot(\Delta x)^n}{n!}+O[(-\Delta x)^n] \end{cases}$$

两式相减可以得到 $f'(x_0)=\dfrac{f(x_0+\Delta x)-f(x_0-\Delta x)}{2\Delta x}+\dfrac{f''(x_0)}{3}(\Delta x)^2+\ldots$

f(x)在x0点的一阶导数可以通过二阶差分近似表示为

$$f'(x_0)=\frac{f(x_0+\Delta x)-f(x_0-\Delta x)}{2\Delta x}$$

相应的误差为 $\dfrac{f''(x_0)}{3}(\Delta x)^2$

5 控制方程离散——二阶偏导数差分格式

二阶偏导数的差分格式仍然可以通过泰勒展开得到

$$\begin{cases} f(x_0-\Delta x)=f(x_0)-f'(x_0)\cdot\Delta x+\dfrac{f''(x_0)\cdot(\Delta x)^2}{2!}+\ldots+\dfrac{f^{(n)}(x_0)\cdot(-\Delta x)^n}{n!}+O[(-\Delta x)^n] \\ f(x_0+\Delta x)=f(x_0)+f'(x_0)\cdot\Delta x+\dfrac{f''(x_0)\cdot(\Delta x)^2}{2!}+\ldots+\dfrac{f^{(n)}(x_0)\cdot(\Delta x)^n}{n!}+O[(-\Delta x)^n] \end{cases}$$

两式相加可以得到 $f''(x_0)=\dfrac{f(x_0+\Delta x)-2f(x_0)+f(x_0-\Delta x)}{(\Delta x)^2}-\dfrac{f^{(4)}(x_0)\cdot(\Delta x)^2}{12}+\ldots$

f(x)在x_0点的二阶导数可以通过差商近似表示为

$$f''(x_0)=\frac{f(x_0+\Delta x)-2f(x_0)+f(x_0-\Delta x)}{(\Delta x)^2}$$

误差的第一项为$\dfrac{f^{(4)}(x_0)\cdot(\Delta x)^2}{12}$ 具有二阶精度

5 控制方程离散——Laplace方程

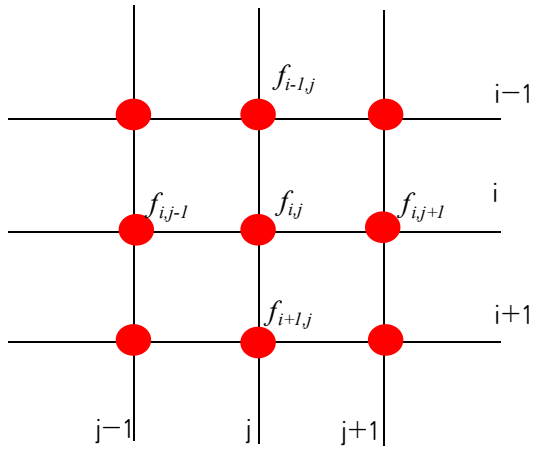
对于二维问题，在如图所示的离散域中，f(x，y)在第i行、j列所在点的x方向、y方向上的二阶偏导数可以借助具有二阶精度的差分格式分别表示为

$$\left(\frac{\partial^2 f}{\partial x^2}\right)_{i,j}=\frac{f_{i,j-1}-2f_{i,j}+f_{i,j+1}}{(\Delta L)^2} \qquad \left(\frac{\partial^2 f}{\partial y^2}\right)_{i,j}=\frac{f_{i-1,j}-2f_{i,j}+f_{i,j+1}}{(\Delta L)^2}$$

在第i行、j列所在点上，Laplace方程最终可以离散为

$$\frac{f_{i,j-1}-2f_{i,j}+f_{i,j+1}}{(\Delta L)^2}+\frac{f_{i-1,j}-2f_{i,j}+f_{i+!,j}}{(\Delta L)^2}=0$$

即 $f_{i,j-1}+f_{i,j+1}+f_{i-1,j}+f_{i+1,j}-4f_{i,j}=0$



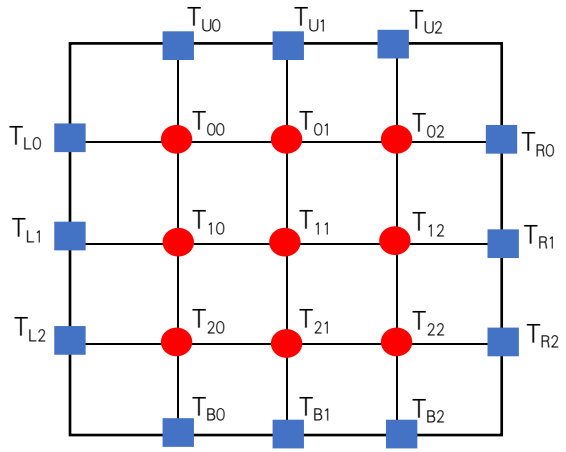
5 控制方程离散——线性方程组

计算域内第i行、第j列对应的关于未知量T的线性方程可以表示为

$$T_{i,j-1} + T_{i,j+1} + T_{i-1,j} + T_{i+1,j} - 4T_{i,j} = 0$$

将每个计算点对应的线性方程联立在一起，会得到NxN的线性方程组，N代表计算点上未知量的个数。

如右图所示9个未知量的计算网格，最终可以得到9x9的线性方程组。



5 控制方程离散——稀疏矩阵

$$\begin{cases} -4T_{00}+T_{01}+T_{10}=-T_{U0}-T_{L0} \\ T_{00}-4T_{01}+T_{02}+T_{11}=-T_{U1} \\ T_{01}-4T_{02}+T_{12}=-T_{U2}-T_{R0} \\ T_{00}-4T_{10}+T_{11}+T_{20}=-T_{L1} \\ T_{01}+T_{10}-4T_{11}+T_{12}+T_{21}=0 \\ T_{02}+T_{11}-4T_{12}+T_{22}=-T_{R1} \\ T_{10}-4T_{20}+T_{21}=-T_{L2}-T_{B0} \\ T_{11}+T_{20}-4T_{21}+T_{22}=-T_{B1} \\ T_{12}+T_{21}-4T_{22}=-T_{R2}-T_{B2} \end{cases}$$

$$\begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \cdot \begin{bmatrix} T_{00} \\ T_{01} \\ T_{02} \\ T_{10} \\ T_{11} \\ T_{12} \\ T_{20} \\ T_{21} \\ T_{22} \end{bmatrix} = \begin{bmatrix} -T_{U0}-T_{L0} \\ -T_{U1} \\ -T_{U2}-T_{R0} \\ -T_{L1} \\ 0 \\ -T_{R1} \\ -T_{L2}-T_{B0} \\ -T_{B1} \\ -T_{R2}-T_{B2} \end{bmatrix}$$

线性方程组经过整理写为矩阵乘法的形式，可以看到线性方程组左侧矩阵系数大部分都是0，非0项占很少部分。这种系数矩阵叫做稀疏矩阵

6 线性方程组求解——迭代方法

左侧系数矩阵大部分为0的稀疏线性方程组，在计算点较多的情况下通常采用迭代方法求解。

迭代方法的一般形式：

对于 $Ax=b$的线性方程组，由稀疏矩阵A和向量b得到变换矩阵G和转换向量c；结合第k次迭代的结果x_k可以得到第k+1次迭代结果： $x_{k+1}= Gx_k + c$；迭代是否终止一般通过临近两次比较结果比较决定。

本课程分别采用Jacobi和Gauss—Seidel两种迭代方法在求解线性方程组。

6 线性方程组求解——Jacobi迭代法

二维方块热传导问题通过建立物理模型、离散计算域、离散控制方程后，计算域内第i行、第j列计算点对应线性方程

$$T_{i,j-1} + T_{i,j+1} + T_{i-1,j} + T_{i+1,j} - 4T_{i,j} = 0$$

可以得到

$$T_{i,j} = \frac{1}{4}\left(T_{i,j-1} + T_{i,j+1} + T_{i-1,j} + T_{i+1,j}\right)$$

利用上式，Jacobi迭代法的第k次迭代计算结果可由上一次（k－1）计算结果得到

$$T^{\mathrm{k}}{}_{i,j} = \frac{1}{4}\left(T^{\mathrm{k-1}}{}_{i,j-1} + T^{\mathrm{k-1}}{}_{i,j+1} + T^{\mathrm{k-1}}{}_{i-1,j} + T^{\mathrm{k-1}}{}_{i+1,j}\right)$$

迭代判据可以选取 $\max\left(\left|T^{\mathrm{k}}{}_{i,j} - T^{\mathrm{k-1}}{}_{i,j}\right|\right) < Tolerance$

7 Jacobi求解热传导问题——串行程序算法

Construct interior unknowns variables $T^{new}{}_{i,j}$ and $T^{old}{}_{i,j}$ with i=0 to n−2, j = 0 to n−2.
Construct boundary known variables TU_k, TB_k, TL_k, TR_k with k = 0 to n−2
while($\max\left(\left|T^{new}{}_{i,j} - T^{old}{}_{i,j}\right|\right) > Tolerance$)
Set $T^{old}{}_{i,j} \leftarrow T^{new}{}_{i,j}$
for i = 0 to n−2 do
for j =0 to n−2
if (j==0) Set $f_1 \leftarrow TL_i$ else Set $f_1 \leftarrow T^{old}{}_{i,j-1}$
if (j==n−2) Set $f_2 \leftarrow TB_j$ else Set $f_2 \leftarrow T^{old}{}_{i,j+1}$
if (i==0) Set $f_3 \leftarrow TU_j$ else Set $f_3 \leftarrow T^{old}{}_{i-1,j}$
if (i==n−2) Set $f_4 \leftarrow TB_j$ else Set $f_4 \leftarrow T^{old}{}_{i+1,j}$
Set $T^{new}{}_{i,j} \leftarrow \frac{1}{4}(f_1 + f_2 + f_3 + f_4)$
end for
end for
end while
Output of ($T^{new}{}_{i,j}$)

相应程序见jacobi2DSeries.c

7 Jacobi求解热传导问题——串行计算结果

未知数数量：NxN

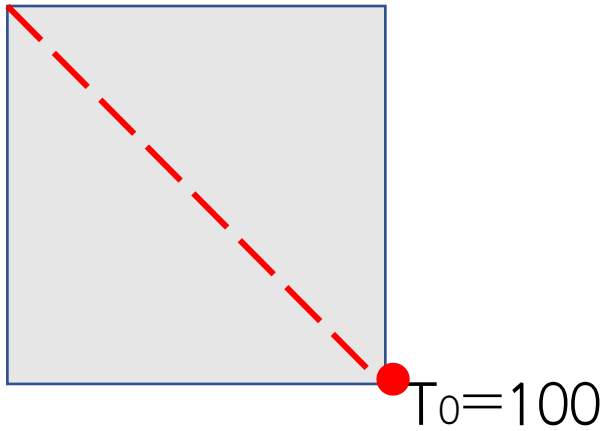
迭代判据： Tolerance

计算时间：
time

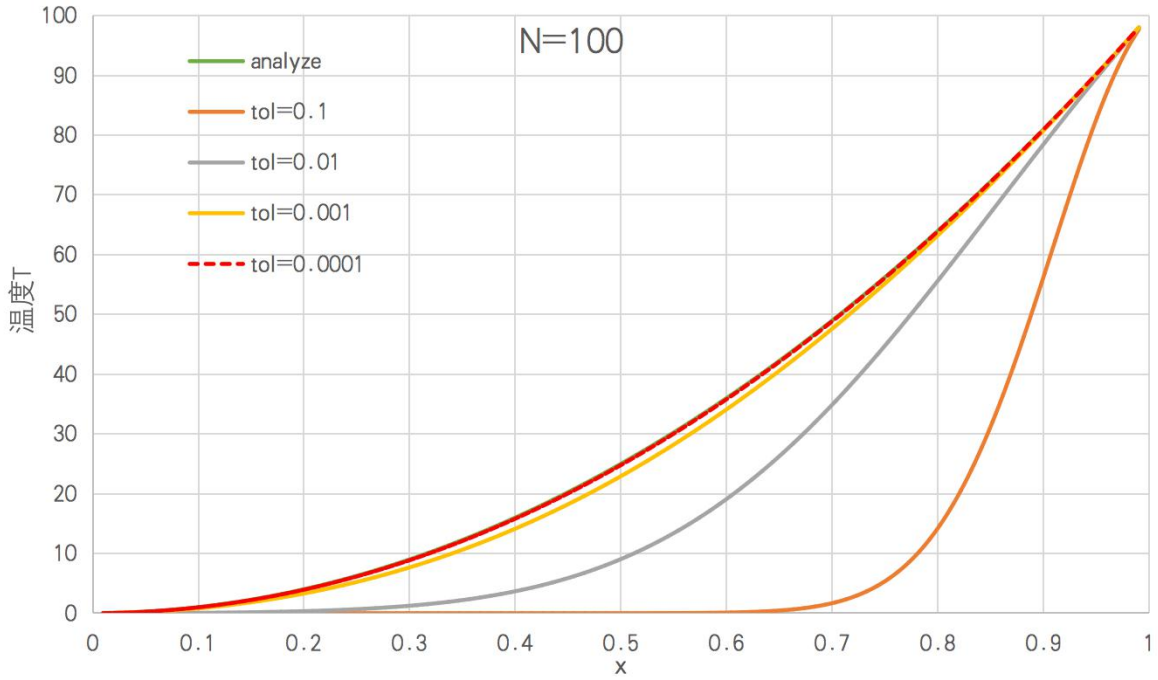
计算误差： error

startTime
while$\left(\max\left(\left|T^{new}{}_{i,j} - T^{old}{}_{i,j}\right|\right) > Tolerance\right)$
… …
… …
end while
endTime
time = startTime − endTime

$T(x,y) = T_0 \cdot x \cdot (1-y)$



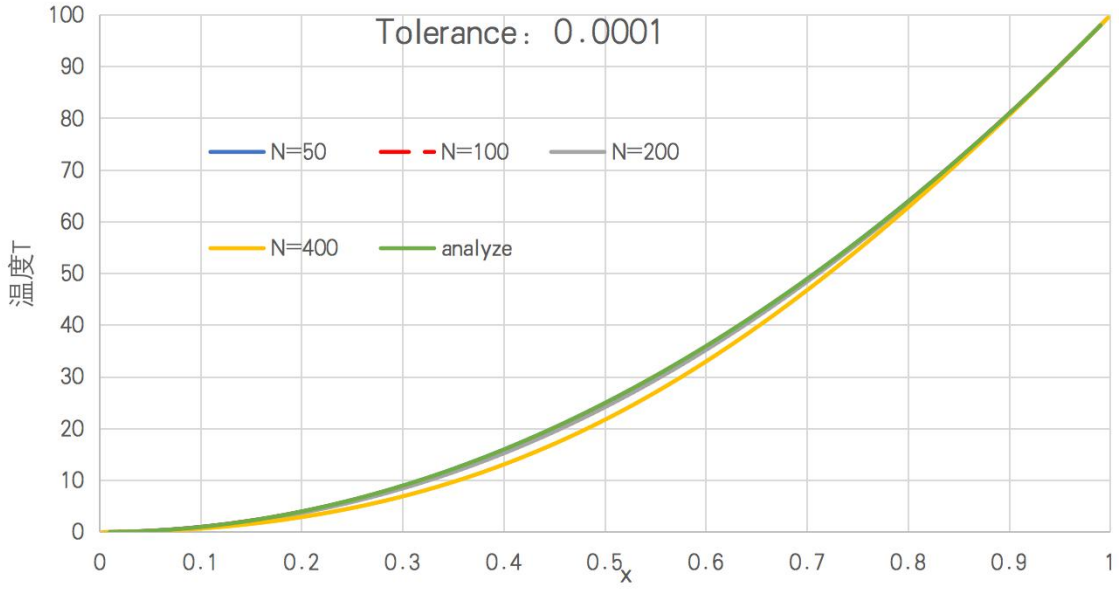
7 Jacobi求解热传导问题——串行计算结果



Resolution：N=100			
tolerance	Iteration steps	Average Error	Time/s
0.1	292	0.85	**0.03**
0.01	1893	0.55	0.15
0.001	6153	0.0907	0.22
0.0001	10911	0.0001	**0.38**

随着tolerance的**减小**，平均误差减小，迭代步数逐渐增加，计算时间**增长**

7 Jacobi求解热传导问题——串行计算结果



Tolerance：tol=0.0001			
N	Iteration steps	Average Error	Time/s
50	3501	0.0023	**0.03**
100	10911	0.00918	0.38
200	31946	0.0363	4.74
400	82238	0.141	**53.76**

随着计算规模的**增大**，平均误差增大，迭代步数逐渐增加，计算时间**显著增长**

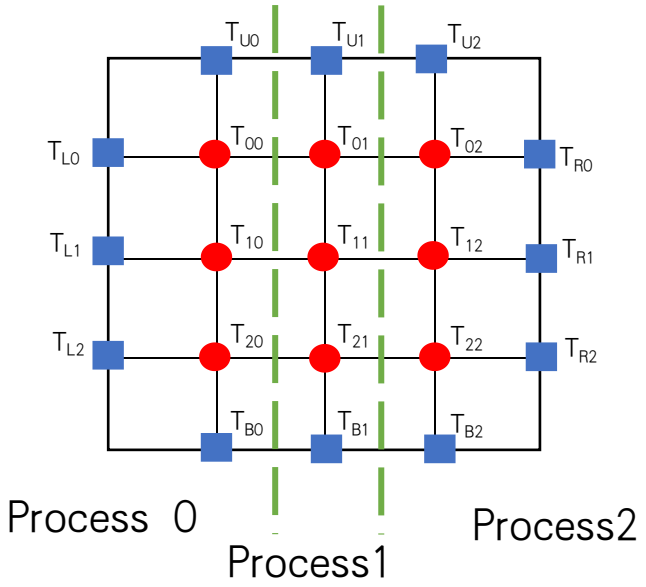
8 Jacobi法求解热传导问题计算并行化——并行策略和并行模型

热传导问题和实际物理区域（计算域）有关，一般采用数据并行策略，即将计算域划分为若干部分，利用多个进程针对不同计算域数据分别求解。这种并行策略也称为SPMD（Single Program Multiple Data）。 Jacobi法是求解热传导问题的关键。因此，并行计算的关键是Jacobi迭代求解线性方程组的并行化。

SPMD策略的具体实现可以使用共享内存模型或分布式内存模型，这两种并行模型又各有多种实现方式。本课程采用基于消息传递（MPI）的分布式内存模型。

8 Jacobi法求解热传导问题计算并行化——计算域划分

首先，按照进程数量将计算域划分，每个进程只计算分区中的计算点。区域划分要尽量保证每个进程中的计算点数量相等，以实现负载均衡。如下图所示为包含3x3计算点的计算域。只考虑沿x方向对计算域进行划分，比较容易在3个进程之间实现负载均衡。如果只有2个进程参与计算，其中1个进程计算3x1个计算点，另1个进程计算3x2个计算点，负载不再平衡。



8 Jacobi法求解热传导问题计算并行化——计算域划分

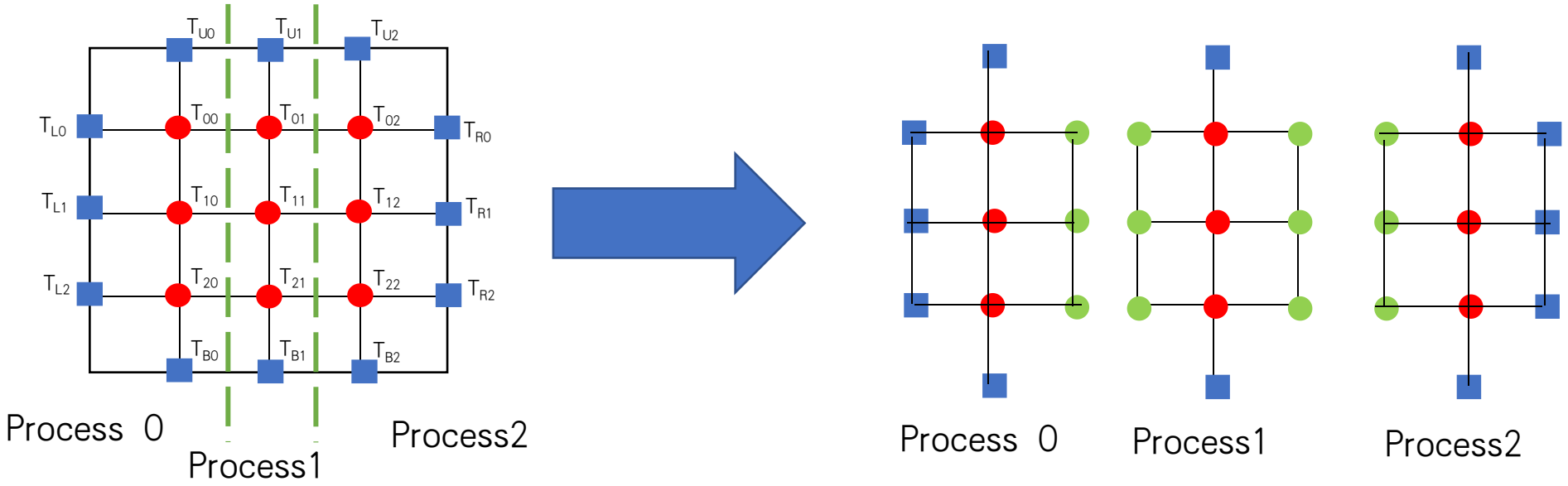
按照进程数量将计算域划分，每个进程只计算分区中的计算点。区域划分要尽量保证每个进程中的计算点数量相等，以实现负载均衡。本课程所附程序只考虑沿x方向对计算区域进行划分。假设包含48x48计算点的计算域，9个进程参与计算：3个进程分配48x6计算点，6个进程分配48x5计算点的负载分方式(算法1)要优于8个进程分配48x5计算点，1个进程分配48x8计算点的分配方式(算法2)。

```
Get num_process and process_rank
Set num_nodeX ← (n−1)/num_zone
If (process_rank < (n−1)%num_zone)
  num_nodeX ← num_nodeX+1
Endif
```

```
Get num_process and process_rank
If (process_rank ≠ num_zone − 1)
  Set num_nodeX ← (n−1)/num_zone
Else
  Set num_nodeX ← (n−1) − (n−1)/num_zone * (num_zone − 1 )
Endif
```

8 Jacobi法求解热传导问题计算并行化——交换点（ghost点、halo点）

并行计算需要解决近邻计算域的数据传递接收问题。常见的方法是建立交换点的方法来接收其他进程的数据。如下图所示，将3x3计算点沿x方向分给三个进程。绿色代表交换点。交换点由相邻进程的计算点提供数据。



8 Jacobi法求解热传导问题计算并行化——数据传递

沿x方向划分计算域，进程编号可以表征进程之间的依赖关系。例如0号进程可以代表最左侧的计算域，依次类推，m号进程代表的计算域位于(m−1)号进程计算域的右侧，最后一个进程号可以代表最右侧的计算域。这样一来，0号进程只在右侧有交换点，最后的进程只在左侧有交换点。其他进程则在两侧均有交换点。数据传递要模板计算开始前完成，相应算法描述为

```
If process_rank != 0
   Send the left compute nodes value to (process_rank − 1)'s right ghost nodes
  Receive the left ghost nodes value from (process_rank − 1)'s right compute nodes value
Endif
If process_rank != process_size - 1
  Send the right compute nodes value to (process_rank + 1)'s left ghost nodes
  Receive the right compute nodes value from (process_rank + 1)'s left interior nodes value
Endif
```

9消息传递模型的具体实现——常用接口

本课程采用消息传递模型（MPI）实现了并行计算热传导问题（Jacobi迭代法）。在优化的第一阶段，主要涉及到6种常用的、基本的MPI接口。

MPI接口名称	作用
MPIInit	启动消息传递模型
MPI_COMM_SIZE	返回参与并行计算的进程数量
MPI_COMM_RANK	返回进程编号
MPI_Send	传递消息
MPI_Receive	接收消息
MPI_Finalize	结束消息传递模型

9消息传递模型的具体实现——阻塞式通信

采用MPI_Send和MPI_Recv这两个函数完成进程间的通信，具体实现如下（jacobi2DParallelMPIReconstr.c）

```
if (process_rank != 0) {
    //copy left side compute nodes value into TOldLeft, which owns continuous addresses.
    for (i = 0; i < num_nodesI; i++) {
        TOldLeft[i] = TOld[i*num_nodesJ]; //Left boundary T=0, x=0, 0<=y<=L
    }
    MPI_Send(TOldLeft, num_nodesI, MPI_DOUBLE, process_rank-1, 0, MPI_COMM_WORLD);
    MPI_Recv(TL, num_nodesI, MPI_DOUBLE, process_rank-1, 0, MPI_COMM_WORLD, &status);
}
if (process_rank != process_size-1) {
    for (i = 0; i < num_nodesI; i++) {
        TOldRight[i] = TOld[i*num_nodesJ+num_nodesJ-1]; //Left boundary T=0, x=0,
0<=y<=L
    }
    MPI_Send(TOldRight, num_nodesI, MPI_DOUBLE, process_rank+1, 0, MPI_COMM_WORLD);
    MPI_Recv(TR, num_nodesI, MPI_DOUBLE, process_rank+1, 0, MPI_COMM_WORLD, &status);
}
```

采用MPI_Send和MPI_Recv这两个函数的特点是当消息发送和接收未完成的时候，程序无法向下执行，这种通信模式称为阻塞式通信

9消息传递模型的具体实现——正确性分析

N=100，tol = 0.0001				
num_process	iteration	error (series)	error (Analyze)	time/s
1	10911	x	0.00918	1.482
2	10911	0	0.00918	0.748
4	10911	0	0.00918	0.414
8	10911	0	0.00918	0.312
16	10911	0	0.00918	0.301
32	10911	0	0.00918	0.539

N=50，tol=0.0001				
num	iteration	error (series)	error (Analyze)	time/s
1	3501	x	0.0023	0.224
2	3501	0	0.0023	0.140
4	3501	0	0.0023	0.098
8	3501	0	0.0023	0.084

9消息传递模型的具体实现——加速比、并行效率、可扩展性

- 加速比：串行程序运行时间Ts和并行程序运行时间Tp的比值，反映了并行程序的加速能力。

$$S = \frac{T_S}{T_p}$$

- 并行效率：加速比S与并行计算进程数p的比值，反映了并行程序利用计算资源的能力。

$$E = \frac{S}{p} = \frac{T_S}{p \cdot T_p}$$

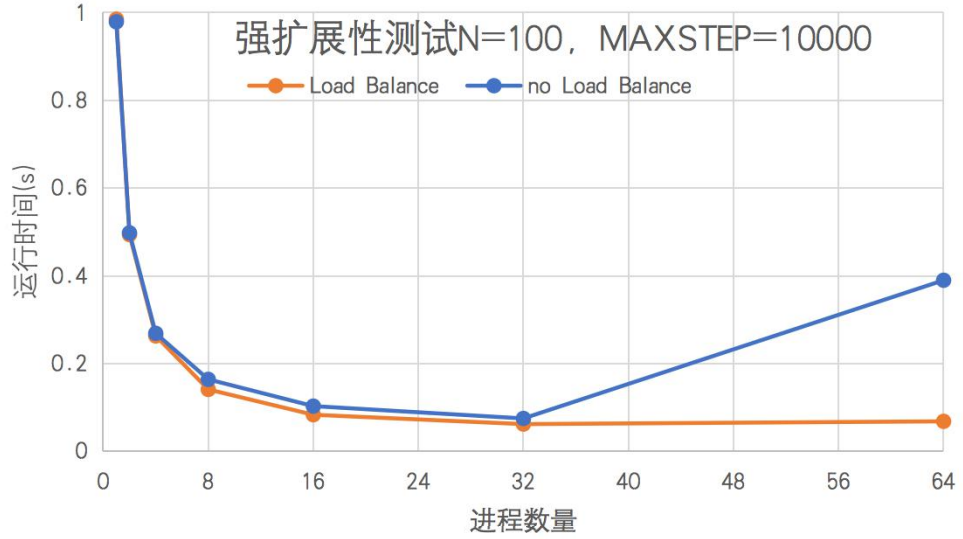
- 可扩展性：如果在计算规模保持不变，增加进程数量，并行效率仍然保持不变，则称并行程序具有强可扩展性。如果只能以相同倍率增大问题规模才能保证进程数目增大情况下并行效率不变，则称并行程序为弱可扩展性的。反映了不同计算规模下的并行效率

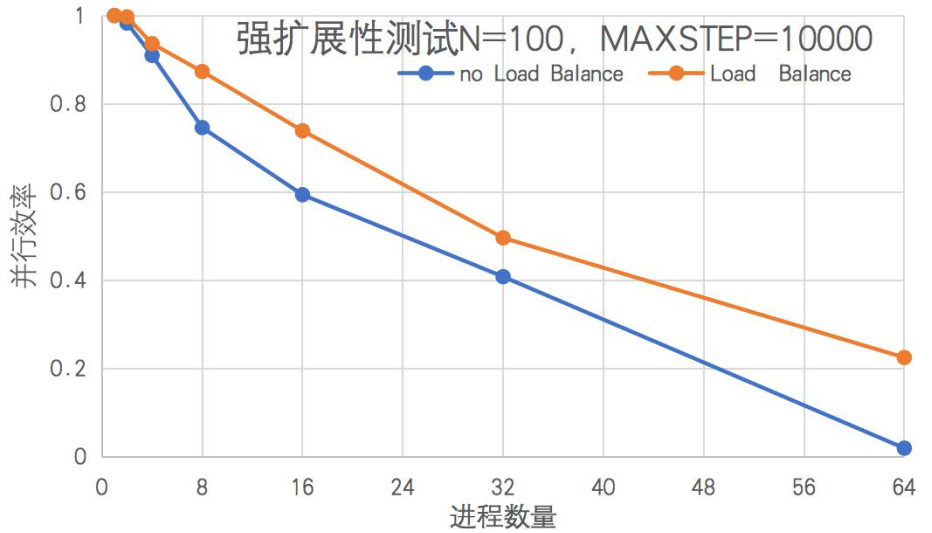
9消息传递模型的具体实现——加速比、并行效率、可扩展性

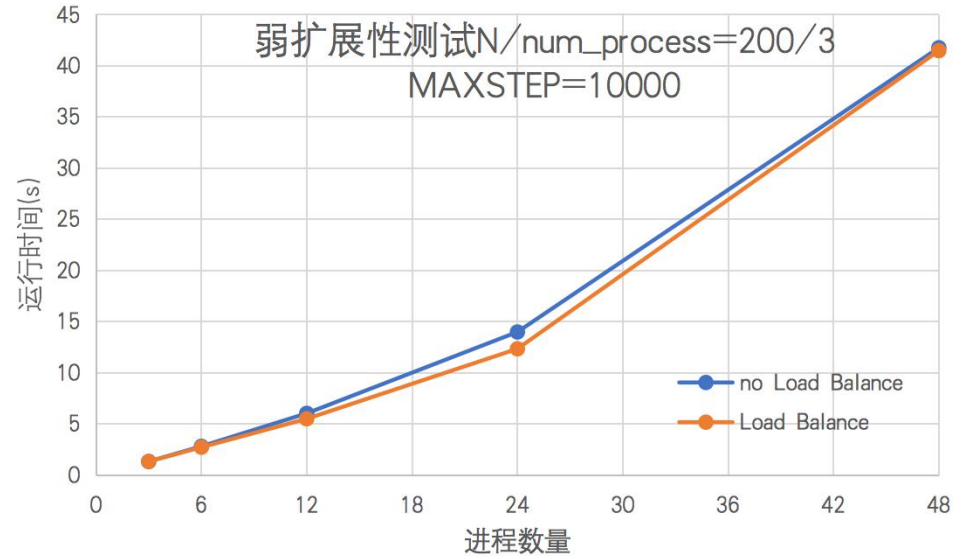
N=100，tol = 0.0001				
num_process	iteration	time/s	S（加速比）	E（并行效率）
1	10911	1.482	x	x
2	10911	0.748	1.981417	99.07%
4	10911	0.414	3.575754	89.39%
8	10911	0.312	4.755045	59.44%
16	10911	0.301	4.923201	30.77%
32	10911	0.539	2.749722	8.59%

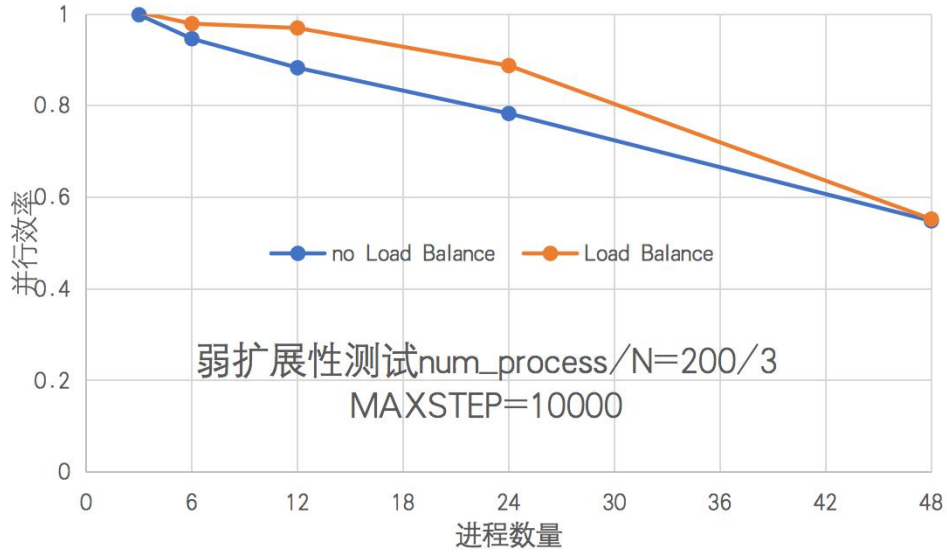
N=50，tol=0.0001				
num	iteration	time/s	S（加速比）	E（并行效率）
1	3501	0.224	x	x
2	3501	0.140	1.597718	79.89%
4	3501	0.098	2.276423	56.91%
8	3501	0.084	2.669845	33.37%

9消息传递模型的具体实现——强扩展性测试和弱扩展性测试









9消息传递模型的具体实现——非阻塞式通信

前面提到了MPI_Send和MPI_Recv。如果调换程序中二者的位置会出现什么现象呢？（jacobi2DParaIIeIMPIReconstrDeadLock.c ）

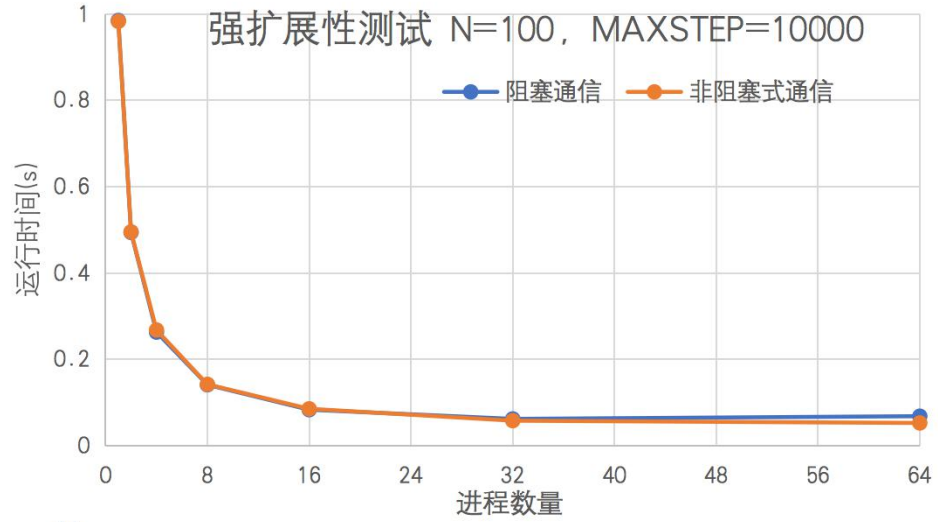
```
if (process_rank != 0) {
            //copy left side compute nodes value into TOIdLeft, which owns continuous addresses.
            for (i = 0; i < num_nodesI; i++) {
                TOIdLeft[i] = TOId[i*num_nodesJ]; //Left boundary T=0, x=0, 0<=y<=L
            }
            MPI_Recv(TL, num_nodesI, MPI_DOUBLE, process_rank-1, 0, MPI_COMM_WORLD, &status);
            MPI_Send(TOIdLeft, num_nodesI, MPI_DOUBLE, process_rank-1, 0, MPI_COMM_WORLD);
}
        if (process_rank != process_size-1) {
            for (i = 0; i < num_nodesI; i++) {
                TOIdRight[i] = TOId[i*num_nodesJ+num_nodesJ-1]; //Left boundary T=0, x=0,
0<=y<=L
            }
            MPI_Recv(TR, num_nodesI, MPI_DOUBLE, process_rank+1, 0, MPI_COMM_WORLD, &status);
            MPI_Send(TOIdRight, num_nodesI, MPI_DOUBLE, process_rank+1, 0, MPI_COMM_WORLD);
}
```

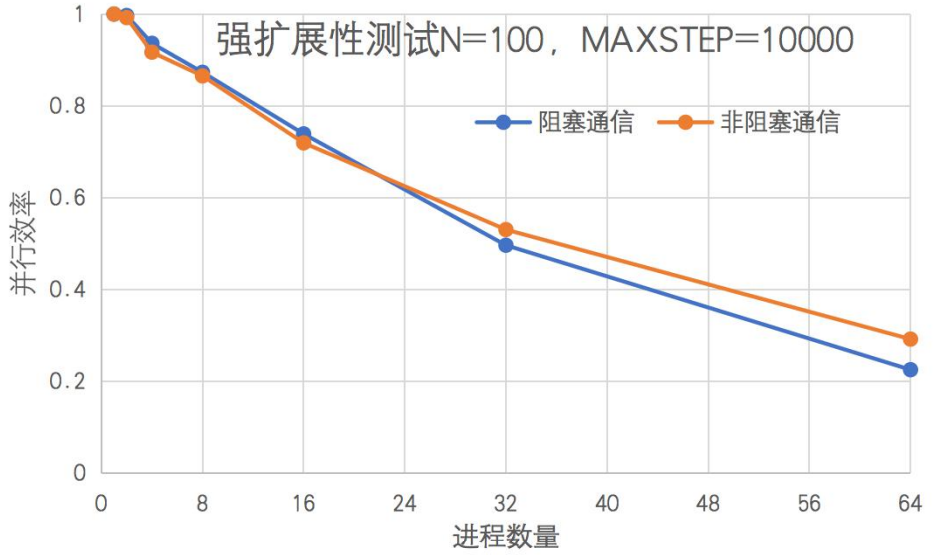
9消息传递模型的具体实现——非阻塞式通信

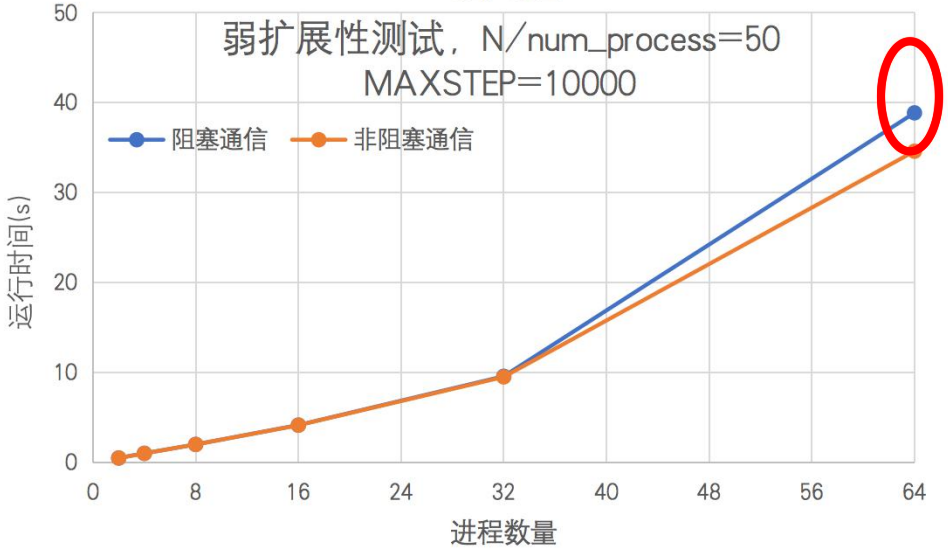
为了避免死锁状态，加快程序速度，采用MPI_Isend和MPI_Irecv实现进程间通信。（jacobi2DParallelMPIReconstrIsendIrecv.c ）

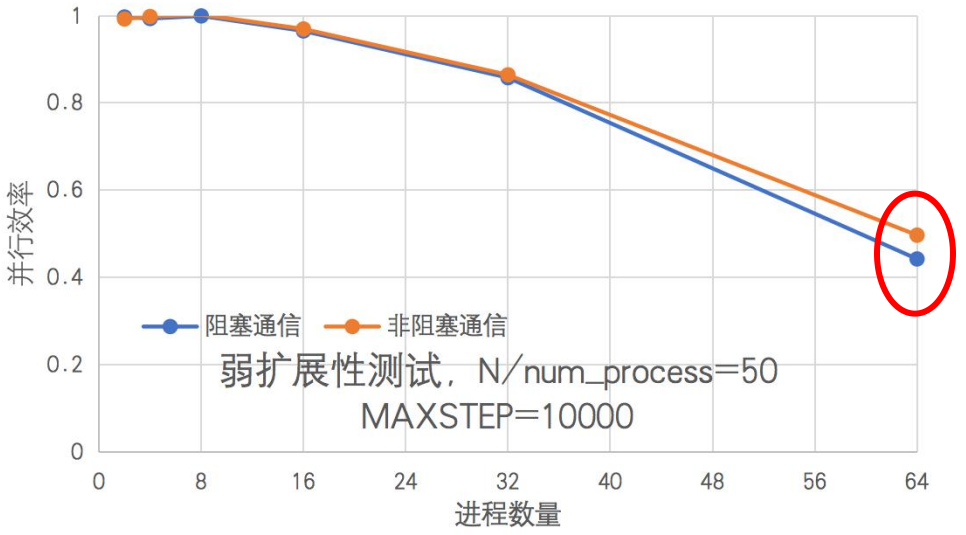
```
if (process_rank != 0) {
    //copy left side interior nodes into TOldLeft, which owns continuous addresses.
    for (i = 0; i < num_nodesI; i++) {
        TOldLeft[i] = TOld[i*num_nodesJ]; //Left boundary T=0, x=0, 0<=y<=L
    }
    MPI_Irecv(TL, num_nodesI, MPI_DOUBLE, process_rank-1, 0, MPI_COMM_WORLD, &req[0]);
    MPI_Isend(TOldLeft, num_nodesI, MPI_DOUBLE, process_rank-1, 0, MPI_COMM_WORLD, &req[1]);
}
if (process_rank != process_size-1) {
    //copy right interior nodes into TOldRight, which owns continuous addresses.
    for (i = 0; i < num_nodesI; i++) {
        TOldRight[i] = TOld[i*num_nodesJ+num_nodesJ-1]; //Left boundary T=0, x=0, 0<=y<=L
    }
    MPI_Irecv(TR, num_nodesI, MPI_DOUBLE, process_rank+1, 0, MPI_COMM_WORLD, &req[2]);
    MPI_Isend(TOldRight, num_nodesI, MPI_DOUBLE, process_rank+1, 0, MPI_COMM_WORLD, &req[3]);
}
MPI_Waitall(4, req, st);
```

9消息传递模型的具体实现——性能分析





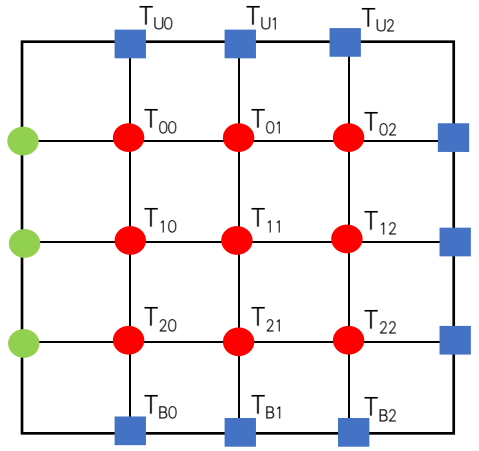




9消息传递模型的具体实现——计算通信重叠

通过观察如图所示某进程内的计算点（红点）、交换点（绿点）可以发现，有的计算点并不需要交换点参与计算，如T_{12}、T_{11}计算点。对于这样的计算点，可以先行计算，结合非阻塞式通信，实现计算和通信的重叠。

在具体实现过程中需要将计算模板分为两部分，与交换点有关的计算和无关的计算。同时需要保证通信能够在与交换点有关的计算开始前完成。

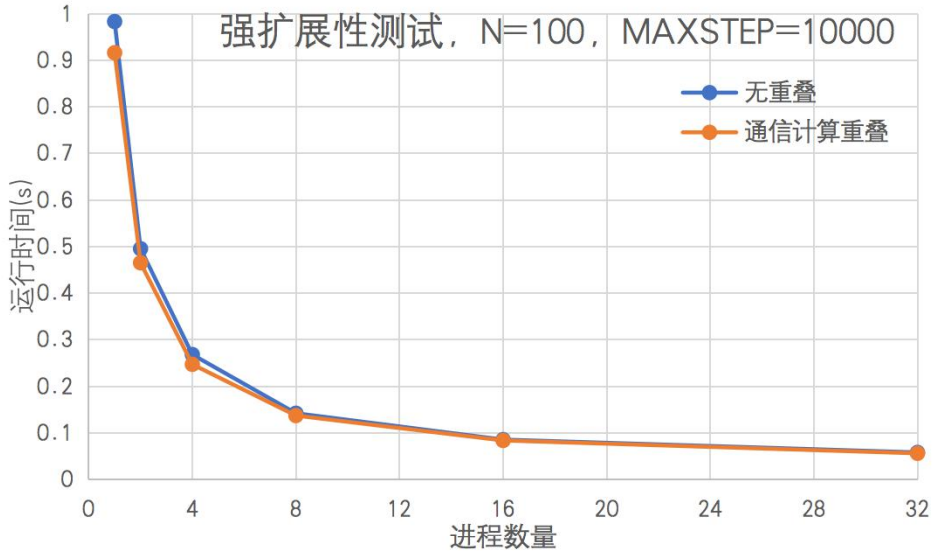


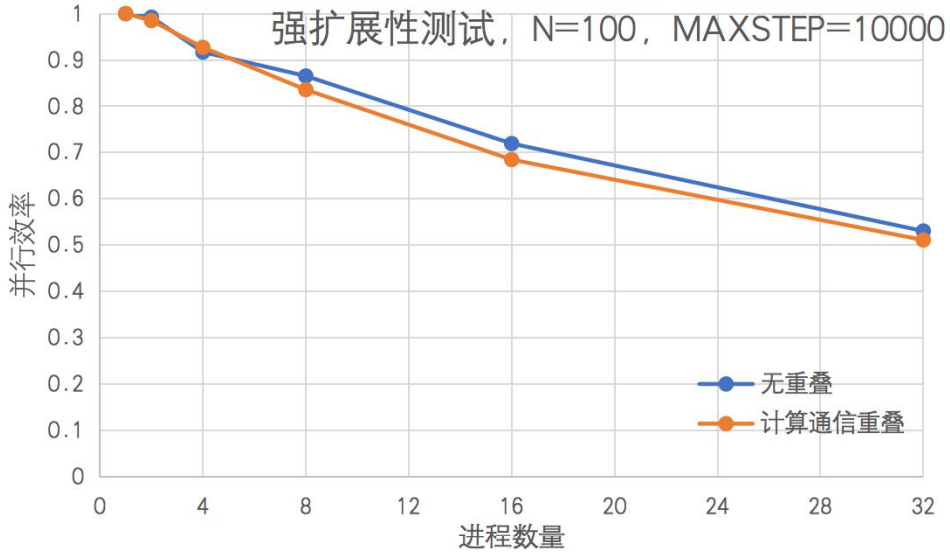
9消息传递模型的具体实现——计算通信重叠

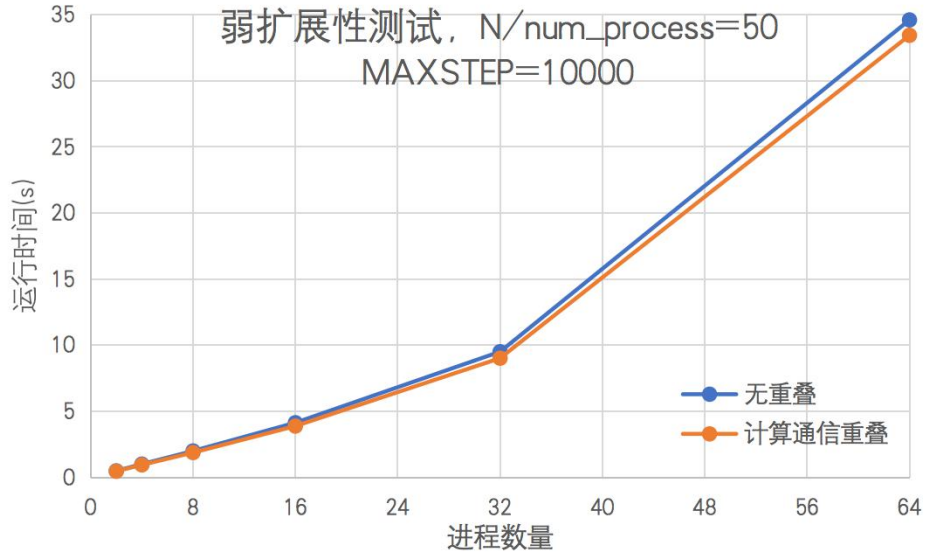
计算通信重叠的具体实现如下所示，要特别注意阻塞点的放置位置，以保证进程间通信在交换点相关计算开始前完成。

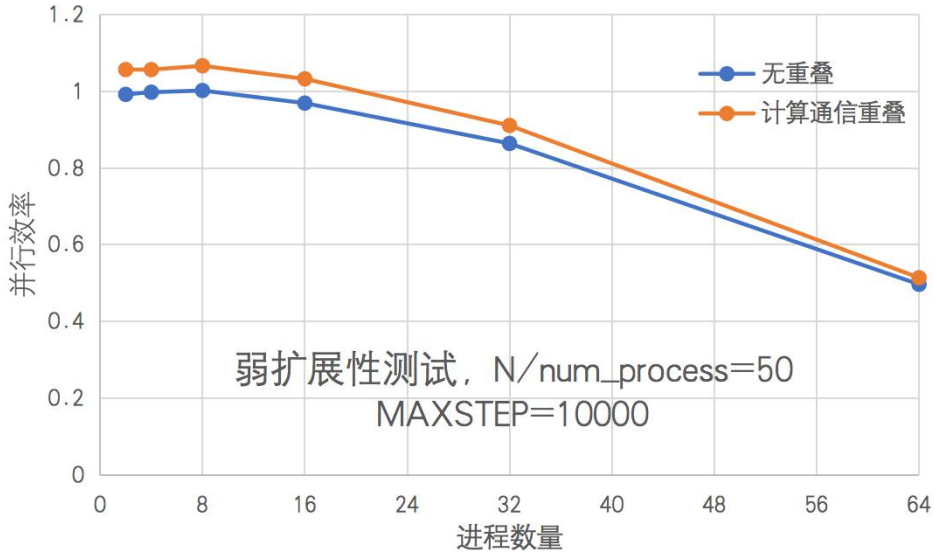
```
if (process_rank != 0) {
      … …
    MPI_Irecv(TL, num_nodesI, MPI_DOUBLE, process_rank-1, 0, MPI_COMM_WORLD, &req[0]);
    MPI_Isend(TOIdLeft, num_nodesI, MPI_DOUBLE, process_rank-1, 0, MPI_COMM_WORLD, &req[1]);
}
if (process_rank != process_size-1) {
    … …
    MPI_Irecv(TR, num_nodesI, MPI_DOUBLE, process_rank+1, 0, MPI_COMM_WORLD, &req[2]);
    MPI_Isend(TOIdRight, num_nodesI, MPI_DOUBLE, process_rank+1, 0, MPI_COMM_WORLD, &req[3]);
}
CalculateComputeNodesWithoutGhostNodes();
MPI_Waitall(4, req, st);
CalculateComputeNodesWithGhostNodes();
```

9消息传递模型的具体实现——性能分析









10 线性方程组求解——Gauss-Sidel迭代法

前面介绍的Jacobi迭代方法，迭代次数过多，收敛速度较慢，Gauss—Sidel迭代法可以克服这些缺点。另外，Gauss—Sidel方法无需存储上次迭代结果，节省了存储空间。

针对二维方块热传导问题，由第i行、j列的方程$T_{i,j-1}+T_{i,j+1}+T_{i-1,j}+T_{i+1,j}-4T_{i,j}=0$，可以得到Gauss—Sidel的格式为

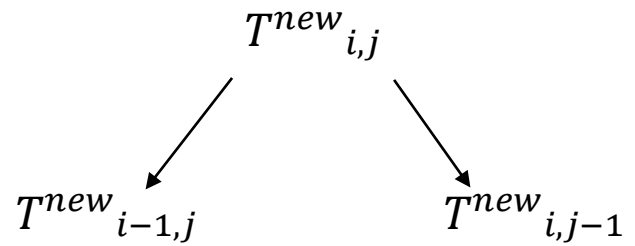
$$T^{\mathrm{k}}{}_{i,j}=\frac{1}{4}\left(T^{k}{}_{i,j-1}+T^{\mathrm{k}-1}{}_{i,j+1}+T^{k}{}_{i-1,j}+T^{\mathrm{k}-1}{}_{i+1,j}\right)$$

在计算过程中，新的迭代更新不仅和上一次迭代计算结果有关，还与已经更新后的计算点有关，更新速度更快。

10 线性方程组求解——Gauss-Sidel迭代法

二维热传导求解(Gauss—Sidel)的算法描述如下(gaussSidel2DSeries.c)

Construct interior unknowns variables $T^{new}{}_{i,j}$ with i=0 to n−2, j = 0 to n−2.
Construct boundary known variables TU_k, TB_k, TL_k, TR_k with k = 0 to n−2
while($\max\left(\left|T^{new}{}_{i,j} - T^{old}\right|\right) > Tolerance$)
Set $T^{old} \leftarrow T^{new}{}_{i,j}$
for i = 0 to n−2 do
for j =0 to n−2
if (j==0) Set $f_1 \leftarrow TL_i$ else Set $f_1 \leftarrow T^{new}{}_{i,j-1}$
if (j==n−2) Set $f_2 \leftarrow TB_j$ else Set $f_2 \leftarrow T^{new}{}_{i,j+1}$
if (i==0) Set $f_3 \leftarrow TU_j$ else Set $f_3 \leftarrow T^{new}{}_{i-1,j}$
if (i==n−2) Set $f_4 \leftarrow TB_j$ else Set $f_4 \leftarrow T^{new}{}_{i+1,j}$
Set $T^{new}{}_{i,j} \leftarrow \frac{1}{4}(f_1 + f_2 + f_3 + f_4)$
end for
end for
end while
Output of ($T^{new}{}_{i,j}$)



11 Gauss-Sidel求解热传导问题——串行计算结果

tolerance = 0.0001				
N	迭代步(Jacobi)	迭代步(GaussSidel)	平均误差(Jacobi)	平均误差(GaussSidel)
50	3501	1949	0.0023	0.0012
100	10911	6202	0.00918	0.00469
200	31946	18870	0.0363	0.0184
400	82238	52486	0.141	0.072

N = 100				
tol	迭代步(Jacobi)	迭代步(GaussSidel)	平均误差(Jacobi)	平均误差(GaussSidel)
0.1	292	279	0.85	0.799
0.01	1893	1505	0.55	0.38
0.001	6153	3822	0.0907	0.04675
0.0001	10911	6202	0.00918	0.00469

12 Gauss-Sidel法求解热传导问题计算并行化——数据依赖性

Jacobi $$T^{\mathrm{k}}{}_{i,j} = \frac{1}{4}\left(T^{\mathrm{k-1}}{}_{i,j-1} + T^{\mathrm{k-1}}{}_{i,j+1} + T^{\mathrm{k-1}}{}_{i-1,j} + T^{\mathrm{k-1}}{}_{i+1,j}\right)$$

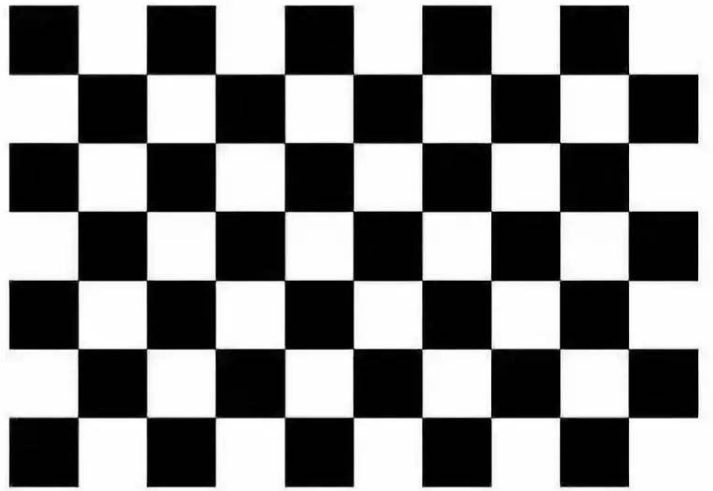
Gauss−Sidel $$T^{\mathrm{k}}{}_{i,j} = \frac{1}{4}\left(T^{k}{}_{i,j-1} + T^{\mathrm{k-1}}{}_{i,j+1} + T^{k}{}_{i-1,j} + T^{\mathrm{k-1}}{}_{i+1,j}\right)$$

对比Jacobi方法，可见Gauss−Sidel的方法具有一定的数据依赖性。每个计算点的更新依赖于已经更新的计算点，从这个角度看，计算具有先后顺序，不能并行 。如何破除这种数据依赖性是实现Gauss−Sidel法求解热传导问题并行计算的关键。

12 Gauss-Sidel法求解热传导问题计算并行化——红黑着色法

着色法是在并行计算中解决数据依赖的常用方法。通过将计算点着色、分组，具有数据依赖性的计算点着不同的颜色，分在不同组中，破除数据依赖性。

对于Gauss—Sidel方法，已经更新的计算点和未更新的计算点存在数据依赖关系。因此，只需要将计算点分成两组，每组计算点的更新只用到未更新的计算点。这种方法又叫做红黑Gauss—Sidel方法，染色后的计算点类似棋盘，如图所示。



12 Gauss-Sidel法求解热传导问题计算并行化——红黑着色法

采用红黑着色方法的计算模板可以表示为

$$T^{\mathrm{k},black}{}_{i,j} = \frac{1}{4}\left(T^{\mathrm{k}-1,red}{}_{i,j-1} + T^{\mathrm{k}-1,red}{}_{i,j+1} + T^{\mathrm{k}-1,red}{}_{i-1,j} + T^{\mathrm{k}-1,red}{}_{i+1,j}\right)$$

$$T^{\mathrm{k},red}{}_{i,j} = \frac{1}{4}\left(T^{\mathrm{k}-1,black}{}_{i,j-1} + T^{\mathrm{k}-1,black}{}_{i,j+1} + T^{\mathrm{k}-1,black}{}_{i-1,j} + T^{\mathrm{k}-1,black}{}_{i+1,j}\right)$$

在迭代完一次后，交换点需要进行数据更新。算法描述如下：

Update ghost nodes's value by sending and receiving data.
Red Compute nodes' Stencil Calculation
Update ghost nodes's value by sending and receiving data.
Black Compute nodes' Stencil Calculation

12 Gauss-Sidel法求解热传导问题计算并行化——正确性分析

- 红黑Gauss Sidel并行算法同串行Gauss Sidel算法不完全等价，反映在单进程的数值计算结果中。红黑单进程的迭代步更少，平均误差也更小。

N = 100				
tol	迭代步(串行)	迭代步(红黑单进程)	平均误差(串行)	平均误差(红黑单进程)
0.1	279	269	0.799	0.799
0.01	1505	1483	0.38	0.378
0.001	3822	3793	0.04675	0.04574
0.0001	6202	6172	0.00469	0.004597

思考题

1 课程提供的并行计算代码只在x方向上实现了区域划分。请同学们在x方向和y方向上同时实现区域划分。

2 比较jacobi并行程序和红黑Gauss—Sidel并行程序的性能（强扩展性测试、弱扩展性测试比较）。