

MAE 5032 HPC Final Project – the transient heat equation in a one-dimensional (1D) domain

Jiayi Huang

Abstract

本文利用有限差分法计算热方程的数值解，在一个一维热方程的具体问题中，使用显式欧拉法和隐式欧拉法分别求解，两种求解均基于PETsC库。根据逐步加细的网格和逐步减小的时间步长，分析并计算了截断误差如何依赖于网格间距和时间步长，并给出理论预测。

Keywords: Finite difference method, Explicit Euler and Implicit Euler methods

1. Introduction

首先列出要求解的一维传热问题

$$\begin{aligned}\rho c \frac{\partial u}{\partial t} - \kappa \frac{\partial^2 u}{\partial x^2} &= f && \text{on } \Omega \times (0, T) \\ u &= g && \text{on } \Gamma_g \times (0, T) \\ \kappa \frac{\partial u}{\partial x} n_x &= h && \text{on } \Gamma_h \times (0, T) \\ u|_{t=0} &= u_0 && \text{in } \Omega.\end{aligned}$$

其中 $u = u(x, t)$ 是要求解的 t 时刻， x 位置处的温度。可以画出下面的示意网格。

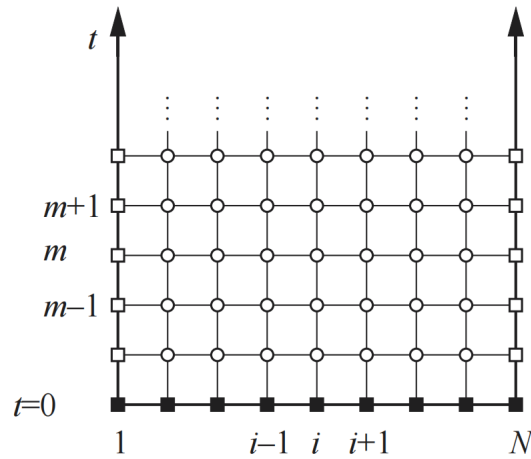


图 1: 解一维热方程的示意网格，实心方块代表已知位置，空心的则代表用有限差分近似的位置

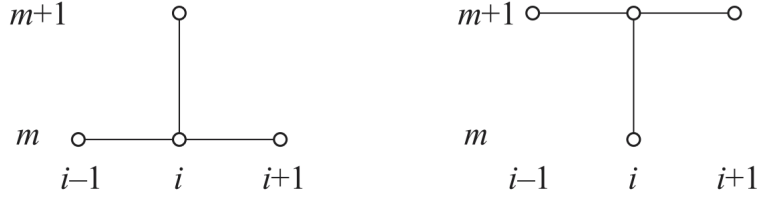


图 2: 关于有限差分的计算子, 左边是Explicit Euler, 右边是Implicit Euler

根据上面的差分算子示意图, 可以推导出下面的两个 Explicit Euler 和 Implicit Euler 迭代公式。

$$u_i^{m+1} = \alpha u_{i+1}^m + (1 - 2\alpha)u_i^m + \alpha u_{i-1}^m + \beta \cdot f \quad (1.1)$$

$$u_i^{m-1} = -\alpha u_{i+1}^m + (1 + 2\alpha)u_i^m - \alpha u_{i-1}^m - \beta \cdot f \quad (1.2)$$

其中 $\alpha = \kappa\beta/\Delta x^2$, $\beta = \Delta t/\rho c$,

代入边界条件 $f = \sin(l\pi x)$, $u_0 = e^x$, $u(0, t) = u(1, t) = 0$, $\kappa = 1.0$, 进而获得对应的三对角迭代矩阵, 再套用之前作业中完成的解三对角矩阵程序进行求解。

2. Code development

2.1. Codes for explicit Euler and implicit Euler methods

这里附上设置 α , β 等等常数参数, 以及设置系数矩阵Matrix A的代码

```
//Iteration coefficient
const double Beta = dt / (Rho_density * C_Heat_Capacity);
const double Alpha = (Beta * K_Conductivity) / (dx * dx);

const double ll = 1.0;

// Matrix_A values for x, y
double * Matrix_A_X = new double[Nodes_Num + 1]();
for(int ii=1; ii<Nodes_Num + 1; ii++) {
    Matrix_A_X[ii] = Matrix_A_X[ii-1] + dx;
}

// f = sin(l*pi*x)
double * Heat_Source = new double[Nodes_Num + 1]();
for(int ii=0; ii<Nodes_Num + 1; ii++) {
    Heat_Source[ii] = Beta * sin(ll * pi * Matrix_A_X[ii]);
}
```

图 3: α , β 以及热源初始条件的设置

如上图代码中所示, 使用了 *Bool* 变量 *Is_Explicit* 来区分显式欧拉和隐式欧拉

如 Figure 6 (*Implicit_Euler*) 图代码中, 区别于显式迭代, 隐式迭代在每个时间步需要解一个线性方程组, 可以设置迭代方法和预条件子。

```

for (PetscInt ii=rstart; ii<rend; ii++)
{
    PetscInt    index[3] = {ii-1, ii, ii+1};
    PetscScalar value[3];
    if (Is_Explicit) {
        value[0] = 1.0*Alpha;
        value[1] = 1.0-2.0*Alpha;
        value[2] = 1.0*Alpha;
    }
    else {
        value[0] = -1.0*Alpha;
        value[1] = 2.0*Alpha+1.0;
        value[2] = -1.0*Alpha;
    }

    if (ii == 0) {
        MatSetValues(A, 1, &ii, 2, &index[1], &value[1], INSERT_VALUES);
    }
    else if (ii == n-1) {
        MatSetValues(A, 1, &ii, 2, index, value, INSERT_VALUES);
    }
    else {
        MatSetValues(A, 1, &ii, 3, index, value, INSERT_VALUES);
    }
}

```

图 4: *Matrix_A* 中的各个元素的值的设置

```

for(int ii=0; ii<N; ii++) {
    time += dt;
    VecAXPY(b, 1.0, f);
    if (Boundary_Condition_x0_h) {
        VecSetValues(b, 1, &zero, &H_F_x0, INSERT_VALUES);
    }
    else {
        VecSetValues(b, 1, &zero, &Temp_x0, INSERT_VALUES);
    }
    if (Boundary_Condition_x0_t) {
        VecSetValues(b, 1, &Nodes_Num , &H_F_x1, INSERT_VALUES);
    }
    else {
        VecSetValues(b, 1, &Nodes_Num , &Temp_x0, INSERT_VALUES);
    }
    VecAssemblyBegin(b);
    MatMult(A, b, x);
    VecCopy(x, b);
    VecAssemblyEnd(b);
}
PetscPrintf(comm, "=====> Solving Process Done <=====\n");

```

图 5: *Explicit_Euler Codes*

```

// Initialize for KSP
KSPSetOperators(ksp, A, A);
KSPSetType(ksp, KSPQRRES);
KSPSetPC(ksp, spc);
PCSetType(pc, PCJACOBI);
KSPSetTolerances(ksp, 1.e-6, PETSC_DEFAULT, PETSC_DEFAULT, PETSC_DEFAULT);
KSPSetFromOptions(ksp);

// Solving Progress
PetscPrintf(comm, "=====> Index For Time: %d\t time: %g<=====\n", (int)zero, (double)time);

for(int ii=0; ii<N; ii++) {
    time += dt;
    VecAXPY(b, 1.0, f);
    if (Boundary_Condition_x0_h) {
        VecSetValues(b, 1, &zero, &H_F_x0, INSERT_VALUES);
    }
    else {
        VecSetValues(b, 1, &zero, &Temp_x0, INSERT_VALUES);
    }
    if (Boundary_Condition_x0_t) {
        VecSetValues(b, 1, &Nodes_Num , &H_F_x1, INSERT_VALUES);
    }
    else {
        VecSetValues(b, 1, &Nodes_Num , &Temp_x0, INSERT_VALUES);
    }
    VecAssemblyBegin(b);
    VecAssemblyEnd(b);
    KSPSolve(ksp, b, x);
    VecCopy(x, b);
    /* VecView(x, PETSC_VIEWER_STDOUT(comm));
    */
    KSPMonitor(ksp, its, rnorm);
    PetscPrintf(comm, "----- Step Solving Process Done. \n");
    PetscPrintf(comm, "----- Iteration OF KSP : %d\t r_norm: %g\n", its, (double)rnorm);
    PetscPrintf(comm, "=====> Index For Time: %d\t time: %g<=====\n", ii+1, (double)time);
}

```

图 6: *Implicit_Euler Codes*

2.2. The stability property and comparison with exact solution

2.2.1. Explicit stability

根据《微分方程数值解》(戴嘉尊)中的结论

$$\lambda_j = 1 - 2\alpha + 2\alpha \cos(j\pi\Delta x) = 1 - 4\alpha \sin^2 \frac{j\pi\Delta x}{2} \quad (j = 1, 2, \dots, M-1) \quad (2.1)$$

矩阵的特征值要小于 1，也即 $\alpha = \frac{\kappa\Delta t}{\rho c\Delta x^2} < \frac{1}{2}$ ，这是古典显式差分格式稳定的充分必要条件。在这里通过设置 Nodes_Num = 10，可知 $\Delta x^2 = 0.01$ ，取 $\kappa = \rho = c = 1$ ，推导出需要设置 $\Delta t \leq 0.005$ 才能收敛。

经过实验并做图，可以发现 ≤ 0.005 的黑线和橙线是稳定的，而红线、蓝线和黄线偏离的程度依次爆炸式增大。这符合之前的稳定性理论分析。可以看到实现的显式算法代码能够把解迭代到离精确解误差很小的范围内。

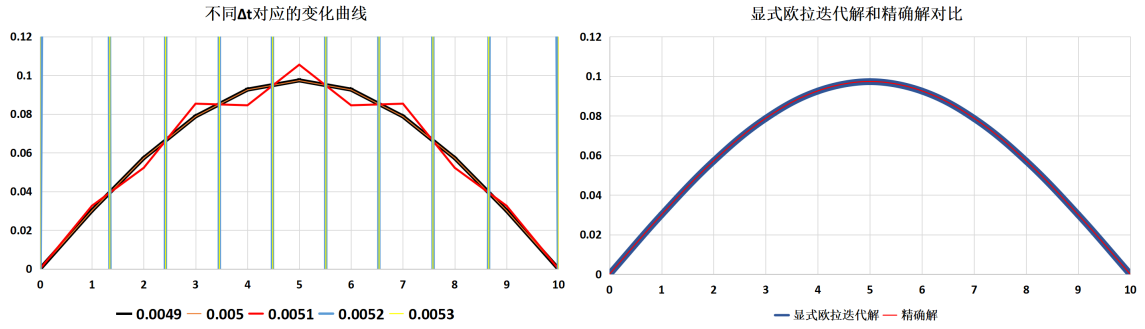


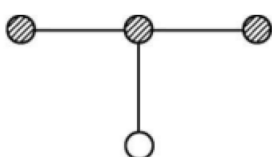
图 7: 显式欧拉迭代法的稳定性分析， ≤ 0.005 稳定

2.2.2. Implicit stability

根据《微分方程数值解》(戴嘉尊)中的结论和误差分析公式,可以知道隐式结构无条件稳定,从下图中同样能够看出,隐式欧拉迭代法能稳定迭代到离精确解很小的范围内。

$$|e^{\sigma \Delta t}| = \left| \frac{1}{1 + 2\beta[1 - \cos(\kappa \Delta x)]} \right| \leq 1 \quad (2.2)$$

3.



$$\frac{U_m^{n+1} - U_m^n}{k} = a \frac{(\delta^2 U)_m^{n+1}}{h^2}$$

Laasonen(1949)

$$E = O(k) + O(h^2)$$

隐式,无条件稳定

图 8: 隐式欧拉迭代法的稳定性分析, 无条件稳定

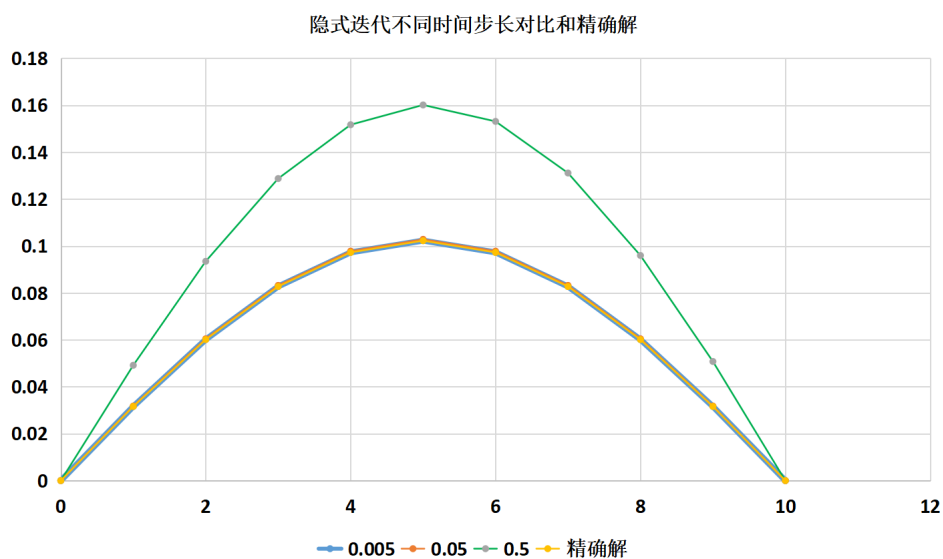


图 9: 隐式欧拉迭代法的不同时间步长对比, 可以看出0.05时即可达到较好精度

2.3. The restart facility using HDF5

```
static char help[] = "Demonstrate HDF5 parallel load-save-reload cycle\n\n";

#include <petscdmplex.h>
#include <petscviewerhdf5.h>
#define EX "restart.c"

typedef struct {
    char    infile[PETSC_MAX_PATH_LEN]; /* Input mesh filename */
    char    outfile[PETSC_MAX_PATH_LEN]; /* Dump/reload mesh filename */
    PetscViewerFormat informat;          /* Input mesh format */
    PetscViewerFormat outformat;         /* Dump/reload mesh format */
    PetscBool heterogeneous;            /* Test save on N / load on M */
    PetscInt ntimes;                    /* How many times do the cycle */
} AppCtx;

static PetscErrorCode ProcessOptions(MPI_Comm comm, AppCtx *options)
{
    PetscBool    flg;
    PetscErrorCode ierr;

    PetscFunctionBeginUser;
    options->infile[0] = '\0';
    options->outfile[0] = '\0';
    options->informat = PETSC_VIEWER_HDF5_XDMF;
    options->outformat = PETSC_VIEWER_HDF5_XDMF;
    options->heterogeneous = PETSC_FALSE;
    options->ntimes = 2;
    ierr = PetscOptionsBegin(comm, "", "Meshing Problem Options", "DMPLEX"); PetscCall(ierr);
    PetscCall(PetscOptionsString("-infile", "The input mesh file", EX, options->infile, sizeof(options->infile), &flg));
    PetscCheck(flg, comm, PETSC_ERR_USER_INPUT, "-infile needs to be specified");
    PetscCall(PetscOptionsString("-outfile", "The output mesh file (by default it's the same as infile)", EX, options->outfile, sizeof(options->outfile), &flg));
    PetscCheck(flg, comm, PETSC_ERR_USER_INPUT, "-outfile needs to be specified");
    PetscCall(PetscOptionsEnum("-informat", "Input mesh format", EX, PetscViewerFormats, (PetscEnum)options->informat, (PetscEnum*)options->informat));
    PetscCall(PetscOptionsEnum("-outformat", "Dump/reload mesh format", EX, PetscViewerFormats, (PetscEnum)options->outformat, (PetscEnum*)options->outformat));
    PetscCall(PetscOptionsBool("-heterogeneous", "Test save on N / load on M", EX, options->heterogeneous, &options->heterogeneous, NULL));
    PetscCall(PetscOptionsInt("-ntimes", "How many times do the cycle", EX, options->ntimes, &options->ntimes, NULL));
    ierr = PetscOptionsEnd(); PetscCall(ierr);
    PetscFunctionReturn(0);
};
```

图 10: PETsC提供的HDF5并行重启示例程序

```
for (i=0; i<user.ntimes; i++) {
    if (i==0) {
        /* Use infile/informat for the initial load */
        infilename = user.infile;
        informat = user.informat;
    } else {
        /* Use outfile/outformat for all I/O except the very initial load */
        infilename = user.outfile;
        informat = user.outformat;
    }

    if (user.heterogeneous) {
        Is_restart = (PetscMPIInt) (grank > user.ntimes-i);
    } else {
        Is_restart = (PetscMPIInt) 0;
    }
    PetscCall(MPI_Comm_split(PETSC_COMM_WORLD, Is_restart, grank, &comm));

    if (Is_restart == 0) {
        /* Load/Save only on processes with Is_restart == 0 */
        DM dm;
        PetscViewer v;

        PetscCall(PetscPrintf(comm, "Begin cycle %D\n", i));

        /* Load data from XDMF into dm in parallel */
        /* We could also use
        PetscCall(DMplexCreateFromFile(PETSC_COMM_WORLD, user.filename, "ex5_plex", PETSC_TRUE, &dm));
        This currently support a few more formats than DMLoad().
        */
        PetscCall(PetscViewerHDF5Open(comm, infilename, FILE_MODE_READ, &v));
        PetscCall(PetscViewerPushFormat(v, informat));
        PetscCall(DMCreate(comm, &dm));
        PetscCall(DMSetType(dm, DM_PLEX));
        PetscCall(PetscObjectSetName((PetscObject) dm, exampleDMplexName));
        PetscCall(DMSetOptionsPrefix(dm, "loaded_"));
        PetscCall(DMLoad(dm, v));
        PetscCall(DMplexDistributeSetDefault(dm, PETSC_FALSE));
        PetscCall(DMSetFromOptions(dm));
        PetscCall(DMplexFromOptions(dm, NULL, "-dm.viewer"));
    }
}
```

图 11: 具体代码细节, 根据输入的PetscBool来判别是否进行断点重启运算

经过测试, 可以完成断点恢复, 不用重新从头计算。

2.4. Defensive manner

如下面的PETSc手册和代码图片所示，采用ierr和PetscCall来进行防御性编程；

采用-Wall来找到提示警告的地方来进行防御性编程；

用assert来找到变量异常的地方来进行防御性编程，前面稳定性分析要求显式欧拉需要 $\alpha \leq 0.5$ 。

Most PETSc programs begin with a call to

```
ierr = PetscInitialize(int *argc, char ***argv, char *file, char *help); if (ierr) return,
-ierr;
```

```
PetscCall(VecCreate(PETSC_COMM_WORLD, &x));
PetscCall(PetscObjectSetName((PetscObject) x, "Solution"));
PetscCall(VecSetSizes(x, PETSC_DECIDE, n));
PetscCall(VecSetFromOptions(x));
PetscCall(VecDuplicate(x, &b));
PetscCall(VecDuplicate(x, &u));
```

图 12: 使用ierr来侦测错误，使用PetscCall来找到出错的地方

```
hjj@ubuntu:~/Desktop/HPC/final_pro/HPC-Final-Project-main/BeatEquation1D$ make e
x1
/home/hjj/Soft/petsc-3.17.1-opt/bin/mpicxx -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -O3 -march=native -mtune=native -Wall -Wwrite-strings -Wno-strict-aliasing -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -O3 -march=native -mtune=native -std=gnu++17 -I/home/hjj/Soft/petsc-3.17.1-opt/include ex1.cpp -Wl,-rpath,/home/hjj/Soft/petsc-3.17.1-opt/lib -L/home/hjj/Soft/petsc-3.17.1-opt/lib -Wl,-rpath,/home/hjj/Soft/petsc-3.17.1-opt/lib -L/home/hjj/Soft/petsc-3.17.1-opt/lib -Wl,-rpath,/home/hjj/lib/hdf5-1.13.1/lib -L/home/hjj/lib/hdf5-1.13.1/lib -Wl,-rpath,/usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/gcc/x86_64-linux-gnu/7 -Wl,-rpath,/home/hjj/Soft/petsc-3.17.1 -L/home/hjj/Soft/petsc-3.17.1 -lpetsc -lhypre -lcumumps -ldmumps -lsmumps -lzmumps -lmumps_common -lpord -lpthread -lscalapack -lflapack -lfbblas -lpthread -lhdf5_hl -lhdf5 -lmetis -lm -lstdc++ -ldl -lmpifort -lmpi -lgfortran -lm -lgfortran -lm -lgcc_s -lquadmath -lstdc++ -ldl -o ex1
ex1.cpp: In function 'void Implicit_Euler(MPI_Comm, Mat, Vec, Vec, Vec, KSP, PC, PetscReal, PetscReal, PetscInt, PetscInt, PetscInt, PetscReal, PetscReal, PetscReal, PetscReal)':
ex1.cpp:263:13: warning: 'its' may be used uninitialized in this function [-Wmaybe-uninitialized]
    KSPMonitor(ksp, its, rnorm);
                ^
ex1.cpp:263:13: warning: 'rnorm' may be used uninitialized in this function [-Wmaybe-uninitialized]
```

```
// write current solution
/* Assert that the alpha <= 0.5 */
PetscReal alpha = kappa * dt / ( rho * c * delta_x * delta_x );
assert ( alpha <= 0.5 );
```

图 13: 使用-Wall来找到提示警告的地方，使用assert来找到变量异常的地方

2.5. Compiling

```
hhy@ubuntu:~/Desktop/HPC/final_pro/HPC-Final-Project-main/HeatEquation1D$ make ex_and_implicit_restart
/home/hhy/Soft/petsc-3.17.1-opt/bin/mpicc -Wall -Wwrite-strings -Wno-unknown-pragmas -fstack-protector -fvisibility=hidden -O3 -march=native -mtune=native -Wall -Wwrite-strings -Wno-unknown-pragmas -fstack-protect
or -fvisibility=hidden -O3 -march=native -mtune=native -I/home/hhy/Soft/petsc-3.17.1-opt/include ex_
and_implicit_restart.c -Wl,-rpath,/home/hhy/Soft/petsc-3.17.1-opt/lib -L/home/hhy/Soft/petsc-3.17.1-opt/li
b -Wl,-rpath,/home/hhy/Soft/petsc-3.17.1-opt/lib -L/home/hhy/Soft/petsc-3.17.1-opt/lib -Wl,-rpath,/home/hhy
/lib/hdf5-1.13.1/lib -L/home/hhy/lib/hdf5-1.13.1/lib -Wl,-rpath,/usr/lib/gcc/x86_64-linux-gnu/7 -L/usr/lib/
gcc/x86_64-linux-gnu/7 -Wl,-rpath,/home/hhy/Soft/petsc-3.17.1 -L/home/hhy/Soft/petsc-3.17.1 -lpetsc -lHYPRE
-lcmumps -ldmumps -lsmumps -lzmumps -lmumps_common -lpord -lpthread -lscalapack -lflapack -lfbblas -lpthrea
d -lhdf5_hl -lhdf5 -lmetis -lm -lstdc++ -ldl -lmport -lmpi -lgfortran -lm -lgfortran -lm -lgcc_s -lquadma
th -lstdc++ -ldl -o ex_and_implicit_restart
```

图 14: 使用PETsC里面自带的tutorial包含的makefile, 方便快捷

2.6. Profiling analysis

```
hhy@ubuntu:~/Desktop/HPC/final_pro/HPC-Final-Project-main/HeatEquation1D$ valgrind ./explicit
==25087== Memcheck, a memory error detector
==25087== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==25087== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==25087== Command: ./explicit
==25087==
==25087== HEAP SUMMARY:
==25087==   in use at exit: 5,932 bytes in 19 blocks
==25087==   total heap usage: 24 allocs, 5 frees, 86,776 bytes allocated
==25087==
==25087== LEAK SUMMARY:
==25087==   definitely lost: 0 bytes in 0 blocks
==25087==   indirectly lost: 0 bytes in 0 blocks
==25087==   possibly lost: 0 bytes in 0 blocks
==25087==   still reachable: 5,932 bytes in 19 blocks
==25087==   suppressed: 0 bytes in 0 blocks
==25087== Rerun with --leak-check=full to see details of leaked memory
==25087==
==25087== For counts of detected and suppressed errors, rerun with: -v
==25087== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
Illegal instruction (core dumped)
```

图 15: 使用valgrind进行Profiling, 寻找性能瓶颈, 提高运行效率

从上面valgrind分析截图中可以看出没有内存泄漏, 但是堆没有及时释放。

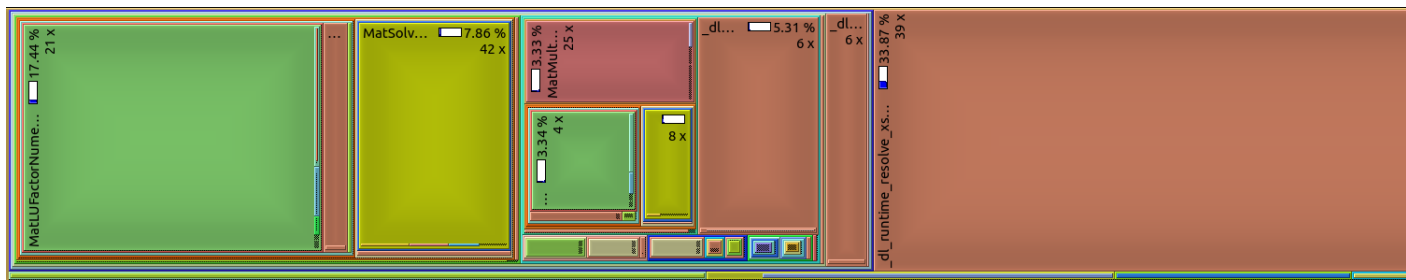


图 16: 使用kcachegrind画出分析可视图：隐式

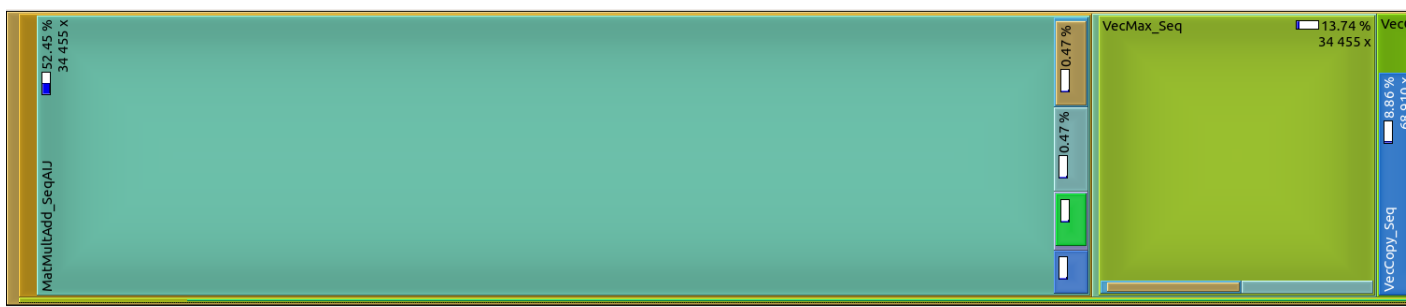


图 17: 使用kcachegrind画出分析可视图：显式

2.7. Visualization utility

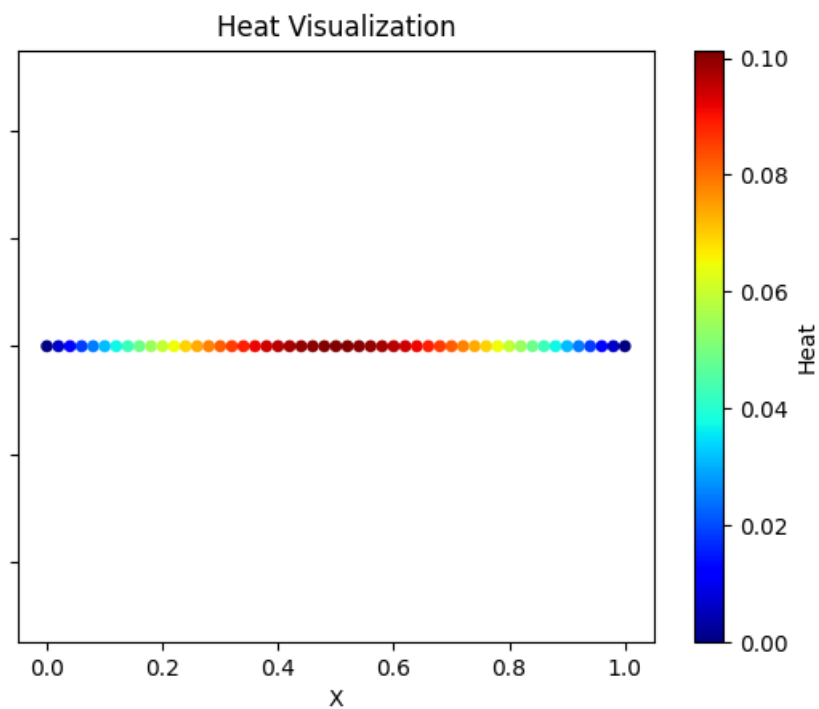


图 18: 使用包含vtk库的vedo工具包，可视化热量分布

如上图所示，选取 50 个点，时间步取 0.0001，用显式欧拉迭代出温度值，并用 vedo 包含的 `vtk_python` 画出可视化温度分布。

3. Code testing

3.1. Manufactured solution method

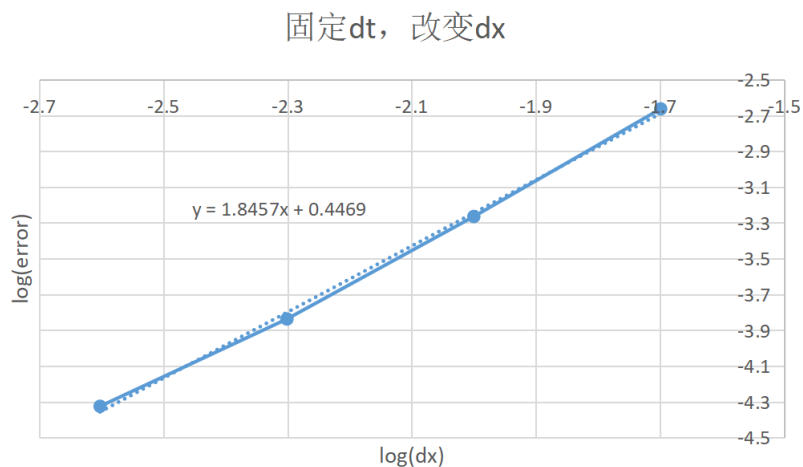


图 19: 确定 Δx 的幂次, 这里取1.8457

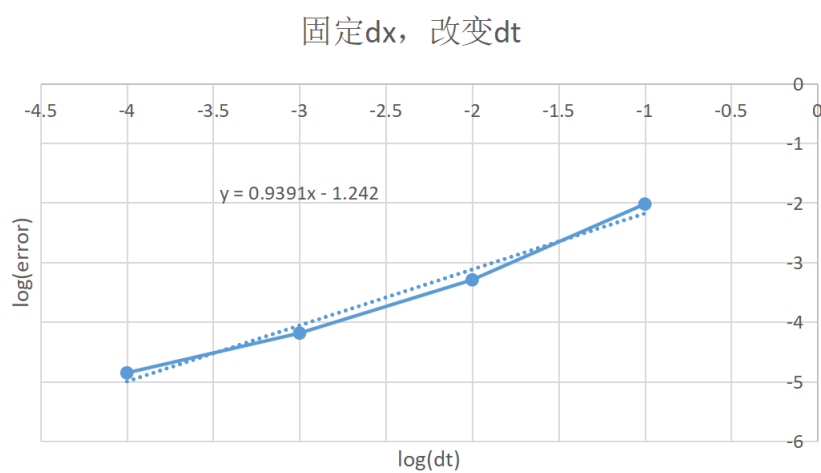


图 20: 确定 Δt 的幂次, 这里取0.9391

根据精确解和隐式迭代算法所得解之间的误差, 计算获得了如上的两个曲线图, 拟合后取 $\alpha = 1.8457, \beta = 0.9391$, 用来估计误差 $e \approx C_1 \Delta x^\alpha + C_2 \Delta t^\beta$ 。

3.2. Parallelism

3.2.1. Strong scaling analysis

对于强可扩展，设置节点数量50，时间步长0.00001，从下图可以看出，问题规模较小的时候，单核效率最高，因为这时候增加核数，运行时间反而随着CPU之间通信消耗显著增加。

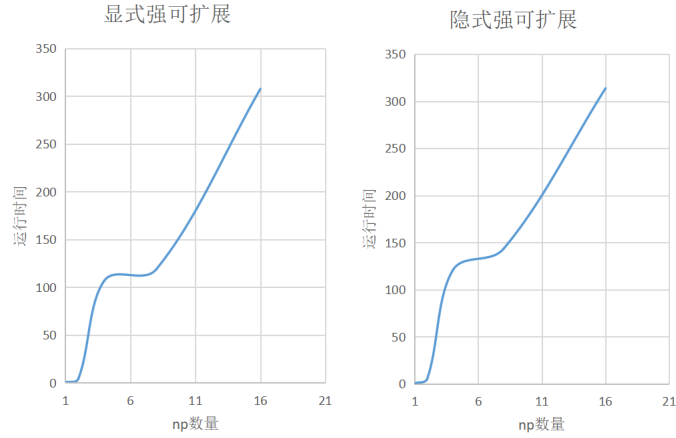


图 21: 强可扩展运行时间不减反增

3.2.2. Weak scaling analysis

对于弱可扩展，设置节点数量随着核数一同增加，如下图所示，时间步长仍然设为0.00001。和强可扩展结论一样，问题规模较小的时候，单核效率最高，多核的运行时间因为CPU之间通信耗散显著增加。

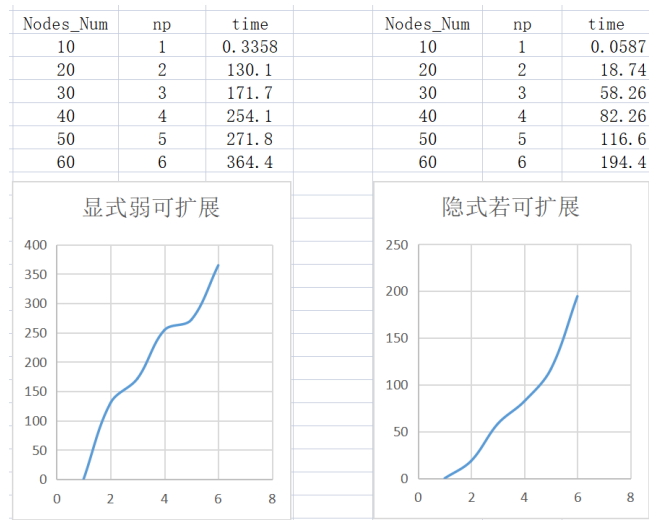


图 22: 弱可扩展运行时间不减反增，而且由于问题规模比较小，Nodes_Num节点数的增多并未对时间有显著影响

3.2.3. PC options

对于预条件子等，如下表所示，可以看出在隐式差分下，都能正确求解到要求精度前提下，lu 分解最快，ksp_type = cg pc_type = jacobi 次之，它们迭代速度快，求解效率高，适合并行加速运算。

ksp_type	pc_type	time
gmres	jacobi	8.367
cg	jacobi	0.8267
gmres	additive	8.438
cg	additive	8.368
preonly	lu\mumps	0.1748

图 23: 达到同样误差精度，lu分解速度最快

References

- [1] Recktenwald, Gerald. “Finite-Difference Approximations to the Heat Equation.” (2004).
- [2] 微分方程数值解法/戴嘉尊,邱建贤编著.-南京:东南大学出版社,2002.02.-233页