MAE5032_FinalProject

1.Introduction

Two-dimensional Heat Transfer Simulator (HTS2D) 是由 陈鹏翰和黄灏开发的二维传热模拟器软件，能够在给定必要物理条件的情况下，使用偏微分方程数值方法对二维平面传热问题进行模拟。软件依赖项包括PETSc、HDF5、MPI等，支持在装有Linux操作系统的超算上并行计算。

2.Method

2.1 Problem

We consider the transient heat equation in a two-dimensional (2D) unit square $\Omega := (0,1)\times(0,1)$. The boundary of the domain Γ can be decomposed into

the bottom face $\Gamma_b := \{(x,y) : x \in (0,1), y = 0\}$

the top face $\Gamma_t := \{(x,y) : x \in (0,1), y = 1\}$

the left face $\Gamma_l := \{(x,y) : y \in (0,1), x = 0\}$

the right face $\Gamma_r := \{(x,y) : y \in (0,1), x = 1\}$

Let f be the heat supply per unit volume, u be the temperature, ρ be the density, c be the heat capacity, u_0 be the initial temperature, κ be the conductivity, n_x and n_y be the Cartesian components of the unit outward normal vector. The boundary data involves the prescrived temperature g on Γ_g and heat flux h on Γ_h. The boundray Γ admits a non-overlapping decomposition: $\Gamma = \Gamma_g \cup \Gamma_t$ and $\varnothing = \Gamma_g \cap \Gamma_t$. The transient heat equation may be stated as follows.

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

2.2 显式

2.2.1内部点

对于内部控制点P，二维显式瞬态数值传热问题可以用以下差分形式表示，

$$
\rho c\Delta x\Delta y(T_P^{t+1} - T_P^t) = \kappa(\frac{T_E^t - T_P^t}{\delta x_e} - \frac{T_P^t - T_W^t}{\delta x_w})\Delta y\Delta t +
$$

$$
\kappa(\frac{T_N^t - T_P^t}{\delta y_n} - \frac{T_P^t - T_S^t}{\delta y_s})\Delta x\Delta t + f\Delta x\Delta y\Delta t
$$

其中，ρ 为密度，κ 为导热率，Δx，Δy，Δt 分别为 x 轴和 y 轴空间步长以及时间步长。δx_e，δx_w，δy_n，δy_s 分别为当前控制点到东、西、北、南四个方向的控制点的距离。T_P^t，T_E^t，T_W^t，T_N^t，T_S^t 分别为上一时刻该控制点及其周围4个控制点的温度值，T_P^{t+1} 为需要计算的当前时刻该控制点的温度值，该方程可以化简为以下形式。

$$a_P^{t+1}T_P^{t+1} = a_E^tT_E^t + a_W^tT_W^t + a_N^tT_N^t + a_S^tT_S^t + a_P^tT_P^t + s$$
$$s = f\Delta x\Delta y$$

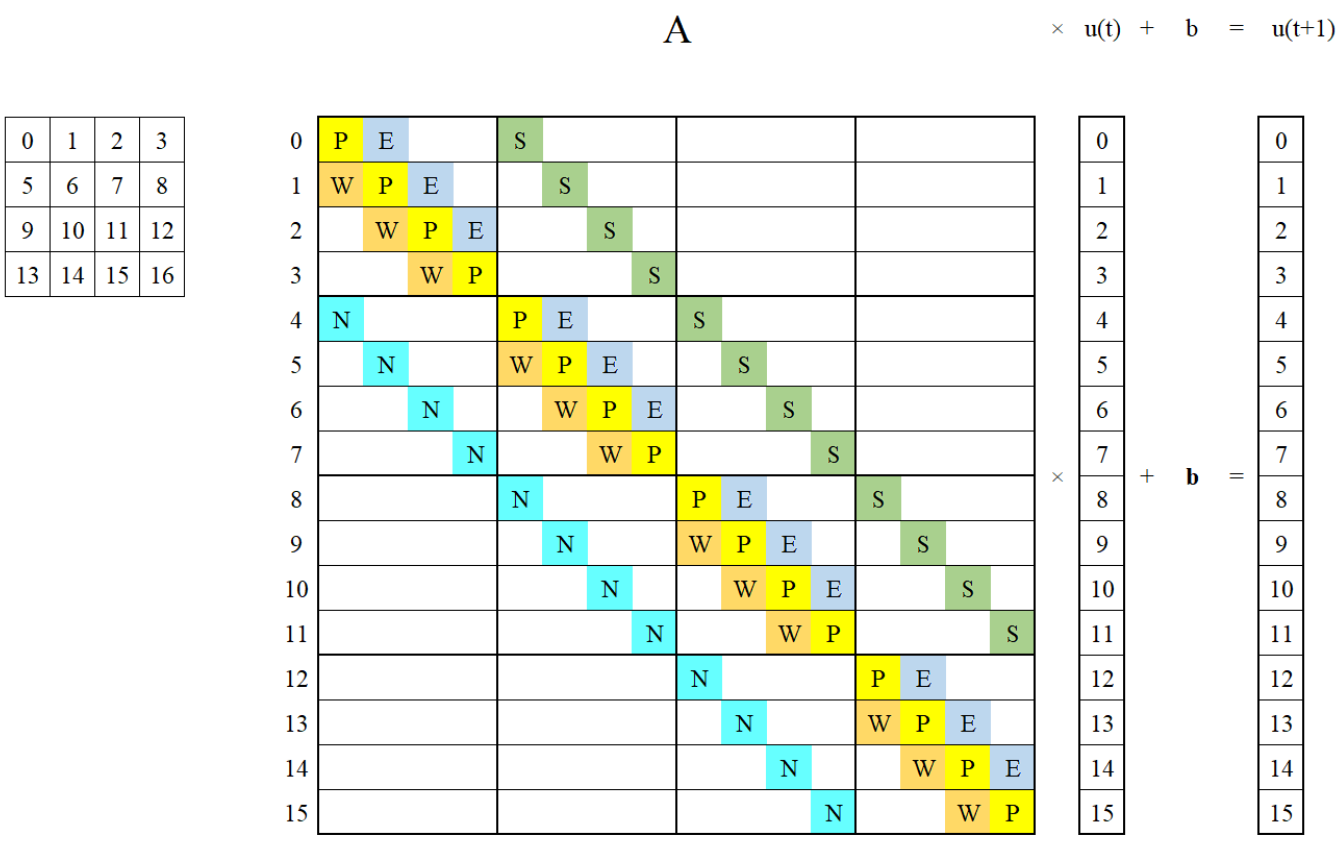
在计算中，我们一般认为各方向的空间步长一致，均为 Δl， 对于内部点，我们认为 δx_e，δx_w，δy_n，δy_s 均与 Δl 相等，于是，进一步，我们可以得到

$$a_P^{t+1} = \frac{\rho c\Delta^2 l}{\Delta t} \qquad a_E^t = a_W^t = a_N^t = a_S^t = \kappa$$

进一步，化简为矩阵乘法 $u^{T+1} = Au^T + b$ 的形式，其中A是系数矩阵，我们可以得到

$$u_P^{T+1} = Wu_W^T + Eu_E^T + Nu_N^T + Su_S^T + Pu_p^T + b$$
$$P = 1 - 4\times\frac{\kappa\Delta t}{\rho c\Delta^2 l} \qquad W = E = N = S = \frac{\kappa\Delta t}{\rho c\Delta^2 l} \qquad b = \frac{f\Delta t}{\rho c}$$

其中，P, W, E, N, S 该控制点的系数矩阵 A 对应行5个值，所以我们的系数矩阵为一个五对角矩阵。



对于内部点，我们可以根据此矩阵向量乘法进行迭代计算出每一时刻的各网格温度值。

2.2.2 边界点

假设边界存在虚控制点，则δx_e，δx_w，δy_n，δy_s 变为 Δl 的一半，故对该控制点的边界温度权重W, E, N, S 中的边界方向的值变为原来的两倍，对该控制点上一时刻的权重对应减小。

对于狄利克雷边界，我们将该方向上的系数乘以温度值作为 b 项中的部分，对于诺依曼边界条件，我们将热流密度除以导热率κ再乘以系数作为b项中的部分，由此可以得到下式。其中，N_1, N_2 分别是该点边界条件的数量，U_i，H_i 为边界控制点的边界温度值或者热流密度。

$$P = 1 - (4 + N_1 - N_2)\frac{k\Delta t}{\rho c \Delta^2 l}$$

$$b = \frac{f\Delta t}{\rho c} + 2\frac{k\Delta t}{\rho c \Delta^2 l}\sum_{i=1,2} U_i + \frac{\Delta t}{\rho c \Delta^2 l}\sum_{i=1,2} H_i$$

最终，我们要迭代的矩阵向量乘法即为，其中 A 为五对角系数矩阵，b 为热源部分，$\mathbf{u}^t$，$\mathbf{u}^{t+1}$ 分别为上一时刻一致到温度和该时刻需要求解的温度。

$$A\mathbf{u}^t + \mathbf{b} = \mathbf{u}^{t+1}$$

2.3 隐式

隐式的与显式的区别在于，是使用当前时刻还是上一时刻的周围温度控制点温度作来计算该控制点温度。

$$Pu_P^{T+1} - Wu_W^{T+1} - Eu_E^{T+1} - Nu_N^{T+1} - Su_S^{T+1} = u_P^T + b$$

在计算P 时，区别于显式

$$P = 1 + (4 + N_1 - N_2)\frac{k\Delta t}{\rho c \Delta^2 l}$$

最终，我们要迭代的线性方程形式为

$$A\mathbf{u}^{t+1} = \mathbf{u}^t + \mathbf{b}$$

3. Installation and Running

HTS2D使用git进行版本控制，并上传到github仓库，使用以下命令克隆至本地

```
git clone https://github.com/PerhapsChen/MAE5032_FinalProject
```

克隆完成后，进入build文件夹

```
cd build
```

加载相关的依赖后使用 `make` 进行构建得到 `HST2D` 可执行文件。

```
module load purge
module load mpi/intel/2018.4

make HTS2D
```

在build文件夹下，`HST2D`的基本使用格式

```
mpirun -np X ./HST2D [FUNCTION] [PARAMETERS] [OTHERS]
```

其中 X 表示使用的核心数

- `FUNCTION`为要使用的功能，目前包括
  - generator: 生成输入数据，包括边界条件，初始温度，其他常量等。
  - explicit: 显式迭代计算，可以指定最大迭代次数，可以重启
  - implicit: 隐式迭代计算，可以指定最大迭代次数，可以重启
- `PARAMETERS`功能要用到的输入参数，具体见下面部分的介绍。
- `OTHERS`表示其他flag，比如 `-log_view`等

推荐使用太乙脚本运行

3.1 generator

generator可以生成计算所需要的HDF5文件，文件中记载了包括边界条件，各物理常量值，每次迭代得到的结果以及迭代次数等信息。

使用generator需要按照以下格式

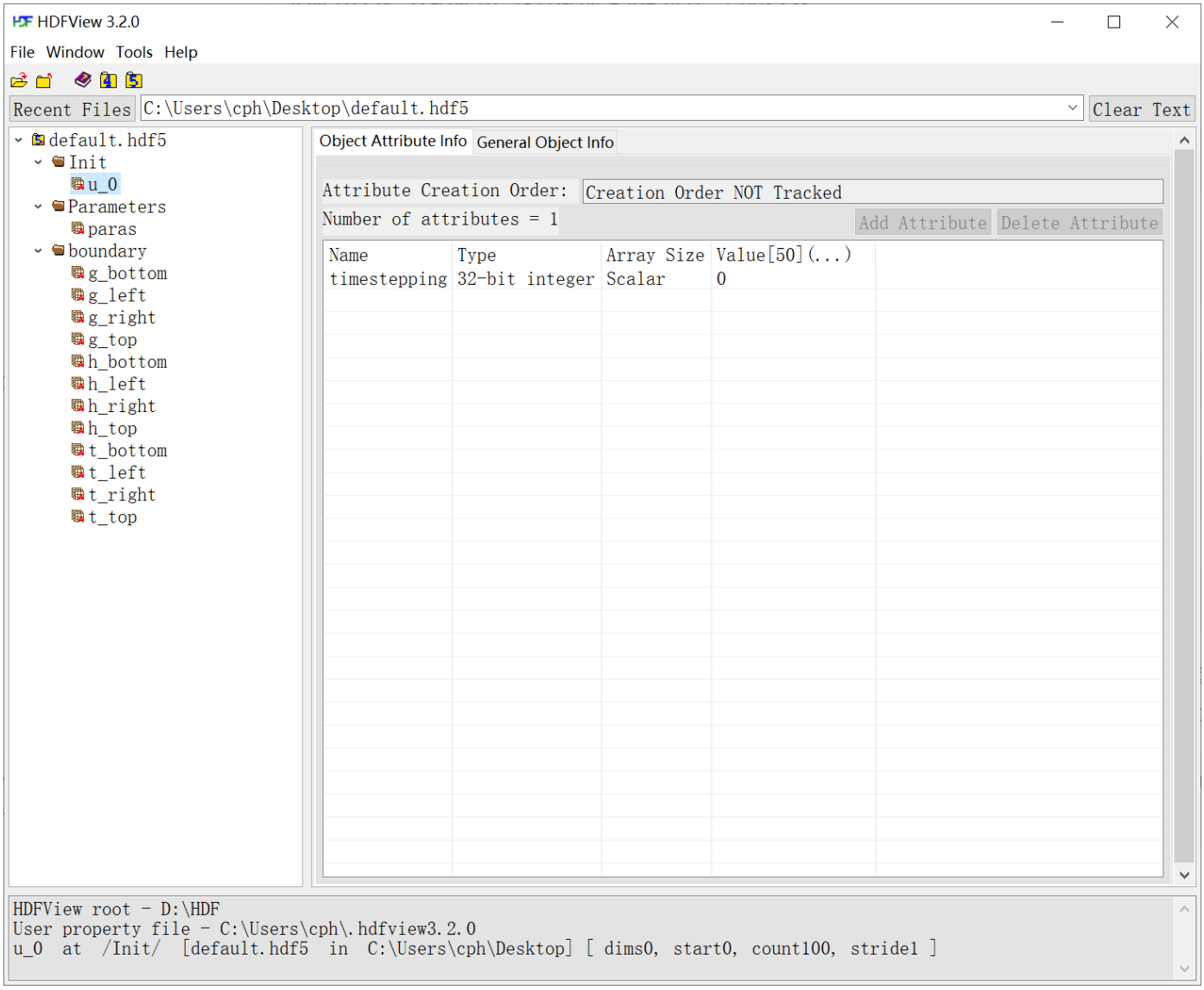
```
mpirun -np X ./HTS2D generator [PARAMETERS] [OTHERS]
```

参数说明

PARAMETERS	说明
-fname	指定文件名
-n	指定矩阵规模，为空间步长dl的倒数
-g	指定所有边界给定g，使用给定的g的值，不能为负
-h	指定所有边界给定h，使用给定的h的值h
-gl	指定左边界给定g，其他边同理(gr, gt, gb)
-hl	指定左边界给定h，其他边同理(hr, ht, hb)
	不同的边可以指定h或者g，例如 -gl 10 -hr 20 -ht 30 -gb 35
-dt	指定时间步长(s)
-dl	指定空间步长
-rho	指定密度
-c	指定heat capacity
-k	指定conductivity
-f	指定heat supply
-u0	指定初始平面温度，需要大于0

例如，使用单核处理器，生成时间步长为1，空间步长为0.1，规模为10，上边界、左边界温度值为200，有边界、下边界热流密度为100，密度为300，head capacity 为1000，conductivity 为10 ，heat supply 为0，初始温度平面为50的文件名为default.hdf5的命令为

```
mpirun -np 4 ./HTS2D generator -fname ../data/test01_ex.hdf5 -n 100 -dl 0.01 -dt 100 -g 100 -rho 5000 -c 1000 -k 1 -f 0 -u0 50
```



生成的文件如上图所示，其中u_0中记录了每次时间步长迭代得到的温度值， Parameters记录了相关常量物理参数，boundary记录了边界条件，g表示温度，h表示热流密度，t指出该点的边界类型（1代表狄利克雷边界，2代表诺依曼边界条件）

如果希望使用更复杂的边界条件，可以修改HDF5文件中对应变量的值。

3.2 Explicit

根据generator生成的数据，我们可以用来选择显式还是隐式计算每次时间步长迭代得到的温度值

使用以下命令选择使用显式计算

```
mpirun -np X ./HTS2D explicit [PARAMETERS] [OTHERS]
```

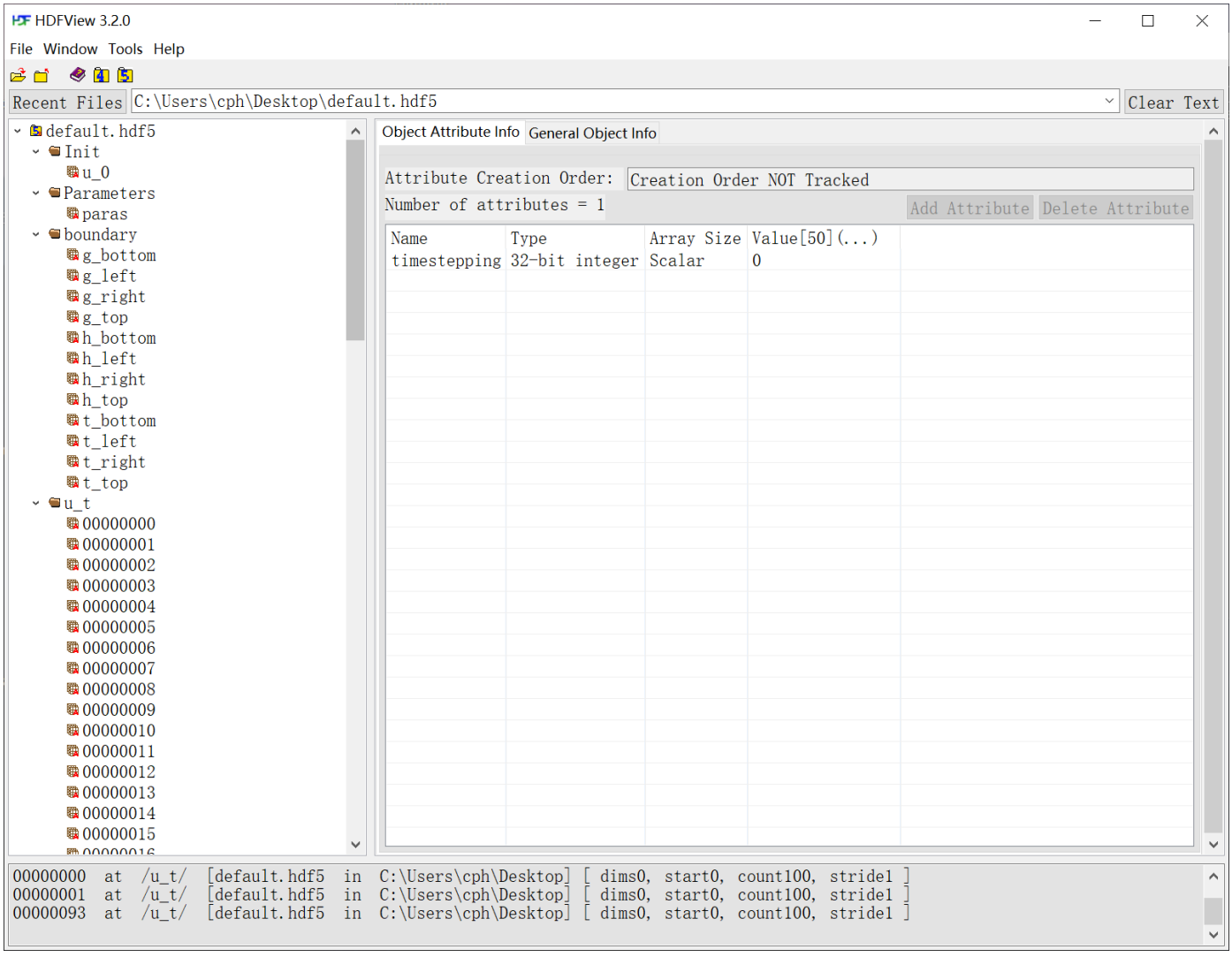
参数说明

PARAMETERS	说明
-fname	指定存储信息的HDF5文件路径及文件名
-maxItsW	多次重启任务中，最大迭代次数
-maxIts	该次任务中，累计最大迭代次数
-restart	0代表非重启型任务，1代表重启任务

例如，我们希望使用4个处理器内核对3.1中生成的数据进行迭代计算，设置该任务总迭代次数最多为100000次，该次迭代次数为100次（不希望一次性计算完毕，或者中途异常退出），首次运行时候，太乙脚本中参考以下命令

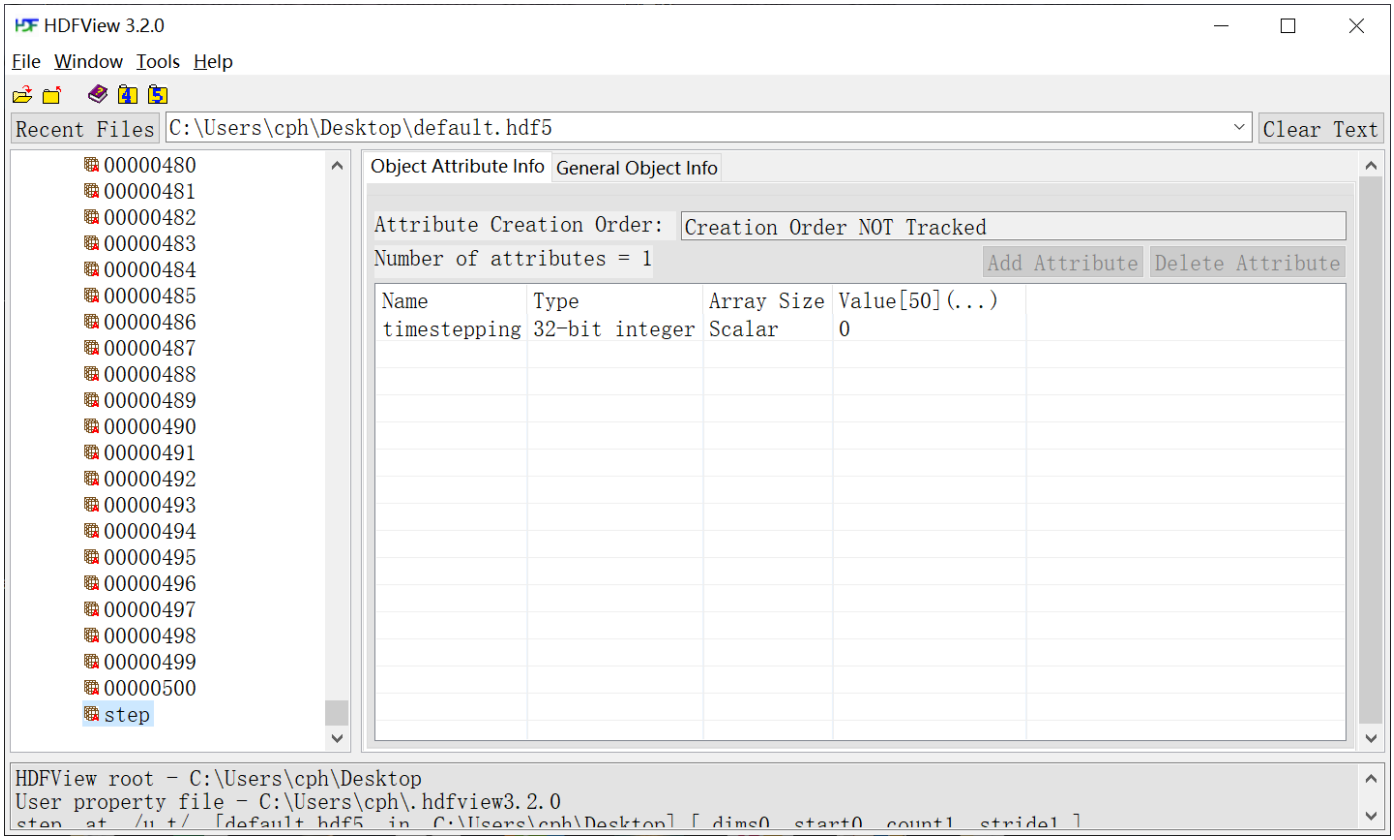
```
mpirun -np 4 ./HTS2D explicit -fname ../data/test01_ex.hdf5 -tol 1e-7 -ps 1000 -maxItsW
100000 -maxIts 100 -restart 0
```

即可在该HDF5文件中记录每次迭代的结果及迭代次数，存放在u_t的group中



如果我们希望重启任务，从上次迭代结束的次数重新计算至500次，则可以考虑以下命令

```
mpirun -np 4 ./HTS2D explicit -fname ../data/test01_ex.hdf5 -tol 1e-7 -ps 1000 -maxItsW
100000 -maxIts 500 -restart 1
```



3.3 Implicit

隐式计算与显式计算相同，只需要将`[FUNCTION]`项修改为 `implicit`

```
# 生成数据
mpirun -np 4 ./HTS2D generator -fname ../data/test01_im.hdf5 -n 100 -dl 0.01 -dt 100 -g
100 -rho 5000 -c 1000 -k 1 -f 0 -u0 50

# 首次迭代100次
mpirun -np 4 ./HTS2D implicit -fname ../data/test01_im.hdf5 -tol 1e-7 -ps 1000 -maxItsW
100000 -maxIts 100 -restart 0

# 重启迭代至500次
mpirun -np 4 ./HTS2D implicit -fname ../data/test01_im.hdf5 -tol 1e-7 -ps 1000 -maxItsW
100000 -maxIts 500 -restart 1
```

4 Profiling and Stability

使用以下太乙脚本进行测试后分析测试结果

```
#!/bin/bash
#BSUB -J HTS2D
#BSUB -q debug
#BSUB -n 10
#BSUB -W 00:10
#BSUB -e %J-petsc.err
#BSUB -o %J-petsc.out
```

```
module purge
module load mpi/intel/2018.4

HTS2D_DIR="/work/ese-chenph/github/MAE5032_FinalProject"
cd $HTS2D_DIR
cd build
make HTS2D
rm ../data/*
ITS=100000

mpirun -np 10 ./HTS2D generator -fname ../data/test02_ex.hdf5 -n 100 -dl 0.01 -dt 100 -g 100 -rho 5000 -c 1000 -k 1 -f 0 -u0 50
mpirun -np 10 ./HTS2D explicit -fname ../data/test02_ex.hdf5 -ps 1000 -tol 1e-7 -maxIts $ITS -maxItsW $ITS -restart 0 -log_view > test01_ex.log 2>&1

mpirun -np 10 ./HTS2D generator -fname ../data/test02_im.hdf5 -n 100 -dl 0.01 -dt 100 -g 100 -rho 5000 -c 1000 -k 1 -f 0 -u0 50
mpirun -np 10 ./HTS2D implicit -fname ../data/test02_im.hdf5 -ps 1000 -tol 1e-7 -maxIts $ITS -maxItsW $ITS -restart 0 -log_view > test01_im.log 2>&1
```

迭代结果

```
# 显式
Explicit iteration done.
Iteration times:68105, break error:7.10543e-14 .
Norm difference to solution: 8.78215e-09 .

# 隐式
Implicit iteration done.
Iteration times:65205, break tolerance:1.56319e-13 .
Difference to solution: 0.172211 .
```

4.1 Explicit

对于显式迭代计算进行稳定性分析，使用解析解 $u = 200$ 进行测试，进行有限步迭代，设置tolerance为1e-20，发现对于显式迭代，进行到65105步的时候迭代退出，此时的与精确解的误差为8.78215e-09（NORM_2）。

使用 `-log_view` 查看程序表现（以下省略了部分信息）

```
------------------------------------------------------------------------
Event                Count   --- Global ---  --- Stage ----  Total
                   Max Ratio %T %F %M %L %R  %T %F %M %L %R Mflop/s
------------------------------------------------------------------------

--- Event Stage 0: Main Stage

BuildTwoSided          3 1.0  5  0  0  0  0   5  0  0  0  0     0
```

```
BuildTwoSidedF         2 1.0  5   0   0   0   0    5   0   0   0   0      0
VecView               43 1.0 21   0   0   0   0   21   0   0   0   0      0
VecMax                 1 1.0  0   0   0   0   0    0   0   0   0   0      0
VecNorm            41484 1.0  9 17   0   0 100   9 17   0   0100   4184
VecCopy                1 1.0  0   0   0   0   0    0   0   0   0   0      0
VecSet                 2 1.0  0   0   0   0   0    0   0   0   0   0      0
VecAXPBYCZ             1 1.0  0   0   0   0   0    0   0   0   0   0   1029
VecAssemblyBegin       1 1.0  0   0   0   0   0    0   0   0   0   0      0
VecAssemblyEnd         1 1.0  0   0   0   0   0    0   0   0   0   0      0
VecLoad                1 1.0  0   0   0   0   0    0   0   0   0   0      0
VecScatterBegin    41483 1.0  4   0 100 100 0    4   0100100   0      0
VecScatterEnd      41483 1.0  2   0   0   0   0    2   0   0   0   0      0
MatMultAdd         41483 1.0 21 83100100   0   21 83100100   0   8698
MatAssemblyBegin       1 1.0  5   0   0   0   0    5   0   0   0   0      0
MatAssemblyEnd         1 1.0  0   0   0   0   0    0   0   0   0   0      0
SFSetGraph             1 1.0  0   0   0   0   0    0   0   0   0   0      0
SFSetUp                1 1.0  0   0   0   0   0    0   0   0   0   0      0
SFPack             41483 1.0  0   0   0   0   0    0   0   0   0   0      0
SFUnpack           41483 1.0  0   0   0   0   0    0   0   0   0   0      0
------------------------------------------------------------------------
```

发现程序耗时最多的部分是 `VecView` 和 `MatMultAdd`，即对向量的写操作和矩阵向量乘法，发现IO在任务中占据了部分时间，可以考虑减少迭代后的向量写的次数，**可以考虑多次不必记录每次迭代的结果，例如只记录每1000次迭代的结果，减少IO次数。**程序计算最密集的部分为 `MatMultAdd` 占据83%的计算次数，其次是 `VecNorm`，占据了17%的计算次数 。

分析内存分配释放情况

```
Memory usage is given in bytes:

Object Type          Creations   Destructions     Memory  Descendants' Mem.
Reports information only for process 0.

--- Event Stage 0: Main Stage

              Viewer     3              2         1616     0.
              Vector 100010          100010    975274200     0.
              Matrix     3              3       170040     0.
           Index Set     2              2         2184     0.
   Star Forest Graph     1              1         1200     0.
```

分析表明，仅有一个Viewer未被释放，内存使用情况良好。

4.2 Implicit

对于隐式迭代计算进行稳定性分析，使用解析解 $u = 200$ 进行测试，进行有限步迭代，设置tolerance为1e-20，发现对于隐式迭代，进行到65205步的时候迭代退出，此时的与精确解的误差为0.172211（NORM_2）。

使用 `-log_view` 查看程序表现（以下省略了部分信息）

```
------------------------------------------------------------------------
Event                Count   --- Global ---  --- Stage ----  Total
                   Max Ratio %T %F %M %L %R  %T %F %M %L %R Mflop/s
------------------------------------------------------------------------

--- Event Stage 0: Main Stage

BuildTwoSided          3 1.0   1  0  0  0  0   1  0  0  0  0     0
BuildTwoSidedF         2 1.0   1  0  0  0  0   1  0  0  0  0     0
VecView               43 1.0  5  0  0  0  0   5  0  0  0  0     0
VecMax                 1 1.0  0  0  0  0  0   0  0  0  0  0     0
VecMDot           414990 1.0 20 29  0  0 45  20 29  0  0 45 18434
VecNorm           497989 1.0  9  6  0  0 55   9  6  0  0 55  9328
VecScale          456489 1.0  1  3  0  0  0   1  3  0  0  0 39644
VecCopy            41500 1.0  0  0  0  0  0   0  0  0  0  0     0
VecSet             83000 1.0  2  0  0  0  0   2  0  0  0  0     0
VecAXPY            41499 1.0  0  1  0  0  0   0  1  0  0  0 52349
VecAXPBYCZ         41500 1.0  0  1  0  0  0   0  1  0  0  0 51117
VecMAXPY          456489 1.0  7 34  0  0  0   7 34  0  0  0 69412
VecAssemblyBegin       1 1.0  0  0  0  0  0   0  0  0  0  0     0
VecAssemblyEnd         1 1.0  0  0  0  0  0   0  0  0  0  0     0
VecPointwiseMult  456489 1.0  3  3  0  0  0   3  3  0  0  0 15382
VecLoad                1 1.0  0  0  0  0  0   0  0  0  0  0     0
VecScatterBegin   414990 1.0  8  0100100  0   8  0100100  0     0
VecScatterEnd     414990 1.0  4  0  0  0  0   4  0  0  0  0     0
VecNormalize      456489 1.0 10  9  0  0 50  10  9  0  0 50 11579
MatMult           414990 1.0 37 23100100  0  37 23100100  0  7675
MatAssemblyBegin       1 1.0   1  0  0  0  0   1  0  0  0  0     0
MatAssemblyEnd         1 1.0  0  0  0  0  0   0  0  0  0  0     0
SFSetGraph             1 1.0  0  0  0  0  0   0  0  0  0  0     0
SFSetUp                1 1.0  0  0  0  0  0   0  0  0  0  0     0
SFPack            414990 1.0  1  0  0  0  0   1  0  0  0  0     0
SFUnpack          414990 1.0  0  0  0  0  0   0  0  0  0  0     0
KSPSetUp               1 1.0  0  0  0  0  0   0  0  0  0  0     0
KSPSolve           41499 1.0 84 98100100 95  84 98100100 95 16435
KSPGMRESOrthog    414990 1.0 27 58  0  0 45  27 58  0  0 45 27545
PCSetUp                1 1.0  0  0  0  0  0   0  0  0  0  0     0
PCApply           456489 1.0  3  3  0  0  0   3  3  0  0  0 12098
------------------------------------------------------------------------
```

发现程序耗时最多的部分是 `KSPSolve` ，其次是 `VECMDot` 即矩阵向量方程求解和向量的内积，程序计算最密集的部分为 `KSPSolve` 。

分析内存分配释放情况

```
Memory usage is given in bytes:

Object Type          Creations   Destructions     Memory  Descendants' Mem.
Reports information only for process 0.

--- Event Stage 0: Main Stage

              Viewer     3              1          808     0.
              Vector 100026          100007    975252272     0.
              Matrix     3              0            0     0.
           Index Set     2              2         2184     0.
   Star Forest Graph     3              0            0     0.
       Krylov Solver     1              0            0     0.
      Preconditioner     1              0            0     0.
    Distributed Mesh     1              0            0     0.
     Discrete System     1              0            0     0.
           Weak Form     1              0            0     0.
```

发现大部分分配的空间都进行了回收，部分未回收对象可能是函数值传递的过程中产生的，此外内存使用情况良好。

5. Parallelism and Scalability

用于考察并行可扩展性的脚本位于 `/work/ese-chenph/github/MAE5032_FinalProject/TaiYi_Material/ty_parallelism`

生成的日志文件位于 `/work/ese-chenph/github/MAE5032_FinalProject/TaiYi_Material/scalability/` 文件夹。

针对1~9个处理器核心和10000~90000次迭代，考察并行可扩展性

太乙脚本如下

```
#!/bin/bash
#BSUB -J HTS2D
#BSUB -q debug
#BSUB -n 10
#BSUB -W 00:10
#BSUB -e %J-petsc.err
#BSUB -o %J-petsc.out

HTS2D_DIR="/work/ese-chenph/github/MAE5032_FinalProject"
cd $HTS2D_DIR
```

```
module purge
module load mpi/intel/2018.4

cd build
make HTS2D

rm -rf ../data/*

# 针对不同任务规模和不同核心进行计算
for ITS in {10000,20000,30000,40000,50000,60000,70000,80000,90000}
do
    for K in {1,2,3,4,5,6,7,8,9}
    do
    mpirun -np $K ./HTS2D generator -fname ../data/test01_ex.hdf5 -n 100 -dl 0.01 -dt
10 -g 200 -rho 5000 -c 1000 -k 1 -f 0 -u0 50
    mpirun -np $K ./HTS2D explicit -fname ../data/test01_ex.hdf5 -maxItsW $ITS -maxIts
$ITS -restart 0 -log_view > test01_ex.log 2>&1


    mpirun -np $K ./HTS2D generator -fname ../data/test01_im.hdf5 -n 100 -dl 0.01 -dt
10 -g 200 -rho 5000 -c 1000 -k 1 -f 0 -u0 50
    mpirun -np $K ./HTS2D implicit -fname ../data/test01_im.hdf5 -tol 1e-20 -ps 1000 -
maxItsW $ITS -maxIts $ITS -restart 0 -log_view > test01_im.log 2>&1

    mv test01_ex.log ../TaiYi_Material/scalability/EX_ITS${ITS}K${K}.log
    mv test01_im.log ../TaiYi_Material/scalability/IM_ITS${ITS}K${K}.log
    rm -rf ../data/*
    done
done
```

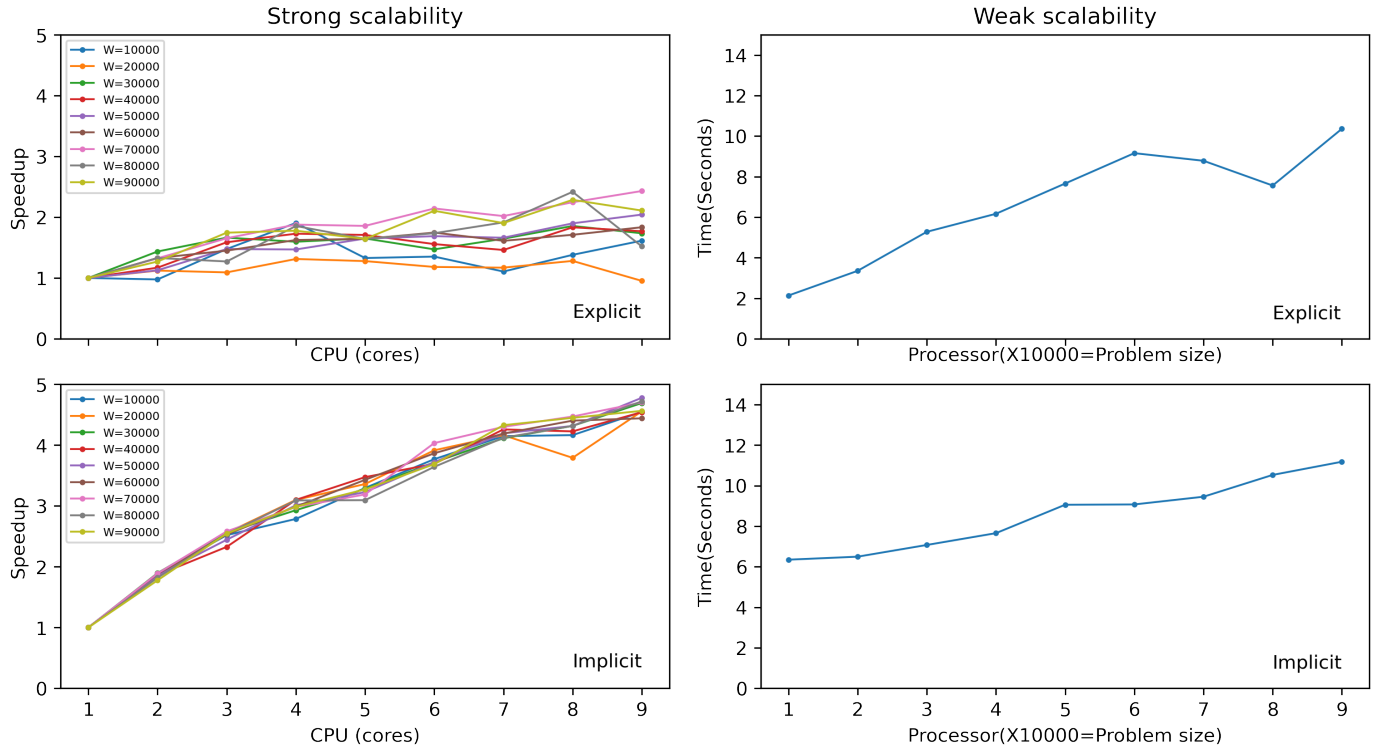
显式计算可扩展性如下表所示

单位: 秒(s)	1核	2核	3核	4核	5核	6核	7核	8核	9核
10000次	2.133	2.186	1.445	1.120	1.606	1.578	1.929	1.545	1.322
20000次	3.962	3.532	3.629	3.019	3.101	3.354	3.387	3.094	4.159
30000次	8.781	6.117	5.278	5.502	5.320	5.971	5.336	4.732	5.061
40000次	10.66	9.113	6.714	6.168	6.241	6.844	7.299	5.815	6.026
50000次	12.63	11.21	8.557	8.598	7.662	7.483	7.598	6.652	6.184
60000次	16.00	12.04	11.04	9.847	9.736	9.160	9.931	9.355	8.721
70000次	17.73	13.40	10.72	9.441	9.549	8.272	8.784	7.896	7.295
80000次	18.28	13.76	14.38	9.855	11.12	10.53	9.551	7.562	12.03
90000次	21.85	17.20	12.52	12.29	13.30	10.38	11.48	9.561	10.36

隐式计算可扩展性

单位: 秒(s)	1核	2核	3核	4核	5核	6核	7核	8核	9核
10000次	6.347	3.375	2.519	2.278	1.926	1.684	1.530	1.524	1.396
20000次	12.09	6.495	4.731	3.901	3.598	3.088	2.904	3.189	2.654
30000次	18.12	9.564	7.073	6.184	5.526	4.873	4.406	4.197	3.862
40000次	23.73	12.74	10.20	7.656	6.831	6.427	5.572	5.615	5.224
50000次	29.23	15.72	11.96	9.707	9.058	7.859	6.970	6.773	6.117
60000次	35.08	18.67	13.87	11.74	10.23	9.073	8.372	7.967	7.899
70000次	40.66	21.53	15.75	13.63	12.75	10.08	9.454	9.095	8.647
80000次	45.46	24.99	17.89	14.71	14.69	12.48	11.03	10.53	9.630
90000次	51.03	28.73	20.04	17.08	15.60	13.84	11.79	11.47	11.18

显式及隐式程序的强可扩展性及弱可扩展性如图所示，可以发现隐式计算的强可扩展性和弱可扩展性较好，显式计算的扩展性一般，当问题规模提升时扩展性有所提升（图中W为问题规模，使用开源软件python绘制）。



6. Comparison of KSP&PC

用于比较的太乙脚本位于 `/work/ese-chenph/github/MAE5032_FinalProject/TaiYi_Material/ty_ksppc`

对于隐式方法，使用不同的KSP和PC类别，限定使用20个处理器核心，迭代上线设置为50000，考察不同类别的用时

	pc: jacobi	pc:sor	pc:none	pc: asm	pc:hypre	pc:lu
ksp: richardson	9.175	8.311	/	/	/	/
ksp: gmres	6.263	/	7.189	7.768	11.89	/
ksp: cg	6.272	/	7.230	8.365	11.39	/
ksp: preonly	/	/	/	/	/	时间过长

分析发现选择GMRES/CG的KSP类别和Jacobi的PC类别有较高的计算效率，使用PREONLY和LU的组合计算时间过长。

7. Visualization

使用VTK和Paraview进行可视化，可视化模块位于 `visualization` 文件夹。

进入 `visualization` 文件夹，提供有两种可视化方法：`StructuredGrid` 为二维平面结构化网格，ElevationFilter为三维高程可视化（讲温度变化假设为高程）。

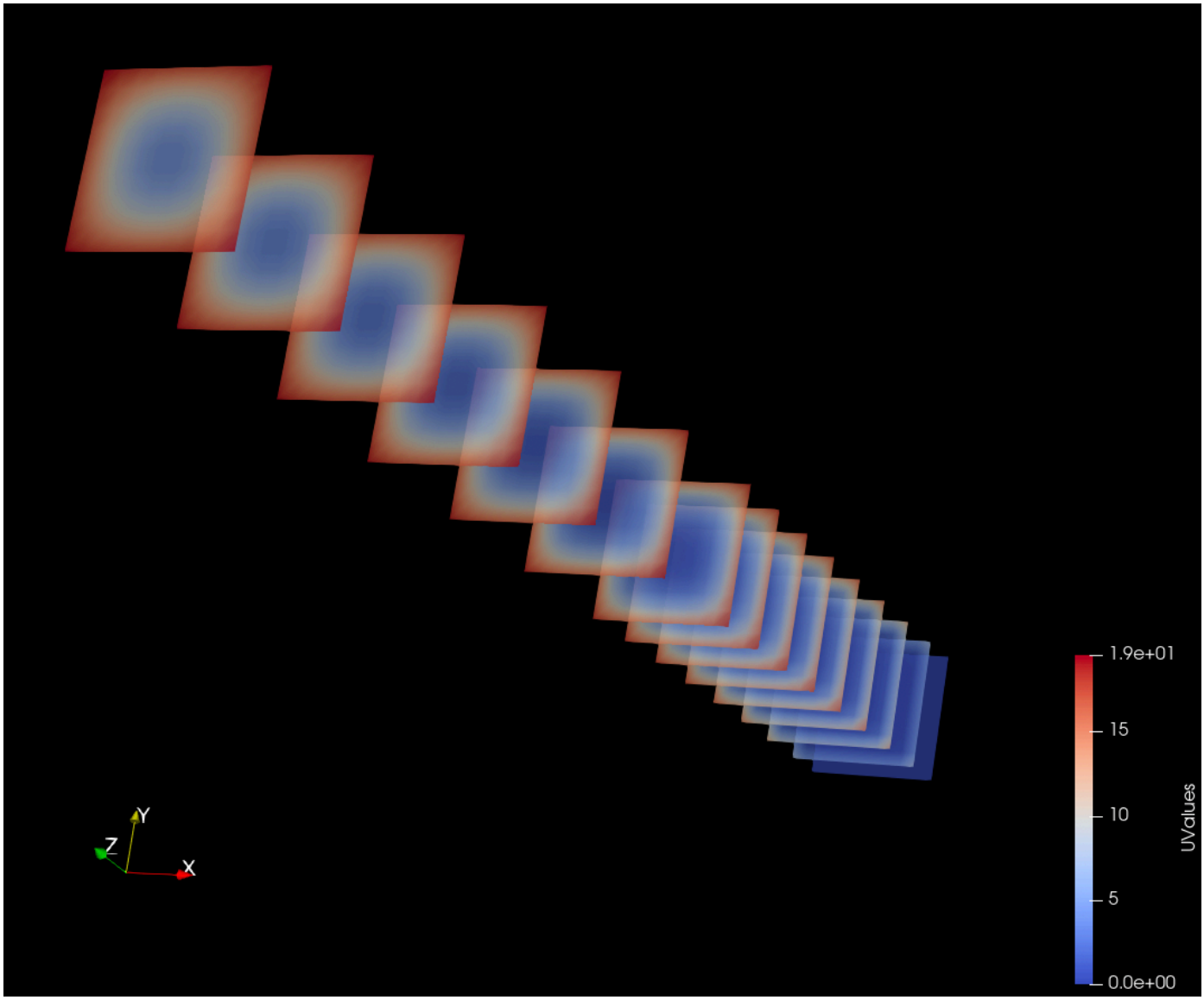
文件内提供CMakeLists.txt，根据是在太乙服务器上还是本地机上修改VTK路径（都测试了，不要的注释掉就行）。**请确保VTK的版本为8.2.0**。以StructuredGrid为例：

```
cd visualization
cd StructuredGrid
mkdir build && cd build
cmake .. && make
```

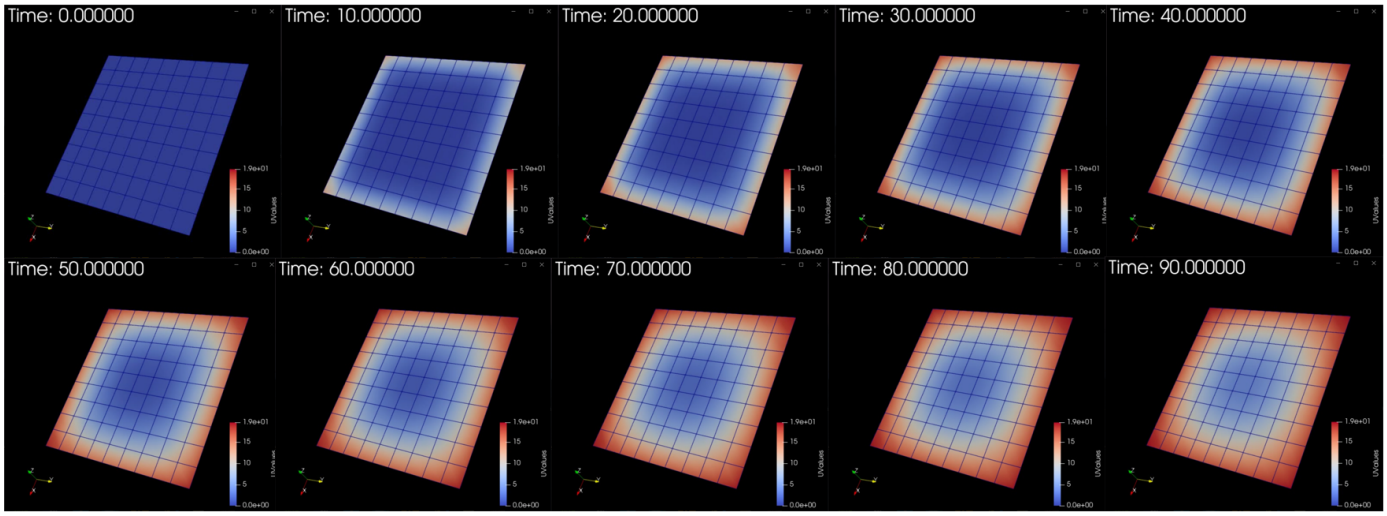
build文件内产生可执行文件，可在本地机直接 `./vtk_StruGrid`，也可以在太乙上提交 `visualization/StructuredGrid` 文件夹中的 `ty_script`

7.1 StructuredGrid

在 `visualization/StructuredGrid` 文件夹中， 在太乙脚本使用以下命令对demo文件 `default.hdf5` 进行可视化，边缘温度为20K，初始表面温度为0K，温度由边缘传递到中间。

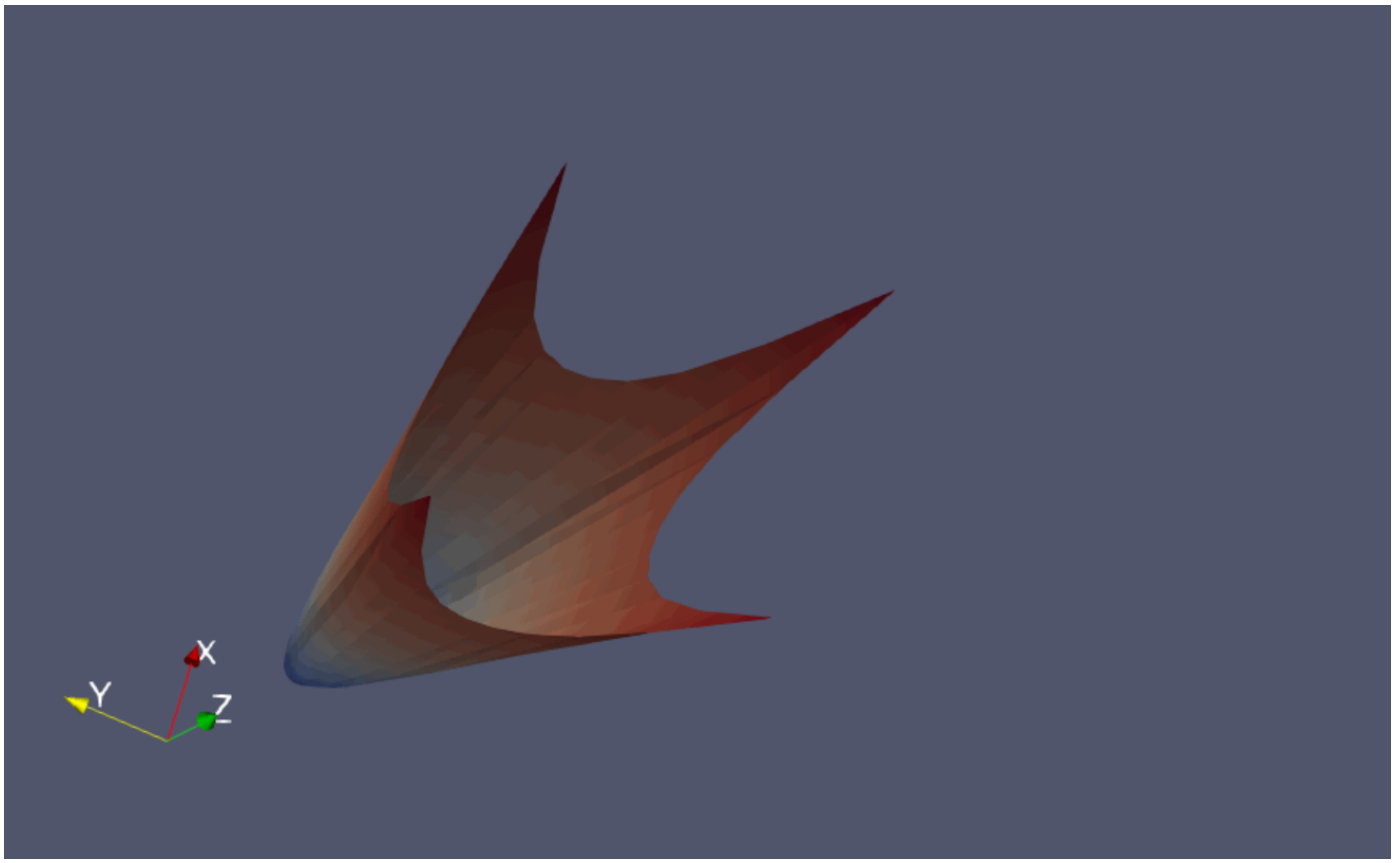


在`output`目录可以生成一系列文件供可视化，显式及隐式计算得到的HDF5文件是每个时刻的温度一个dataset，所以暂时只能实现每个时刻生成一个文件，再用Paraview后处理成动画，查看动画。



7.2 ElevationFilter

在`visualization/ElevationFilter`文件夹中，可以对某一时刻的二维温度进行三维数字高程绘制。



8. 不足和展望

- **代码复用性一般** 生成输入数据部分正常实现，能够生成用户给定的数据组合，并存储到指定HDF5文件中，且能够较好的识别例如温度值（K）为负的输入方式并退出报错，但包括读取该数据部分的代码较为冗长，可以尝试更好的输入及读取方式。
- **BUG** 显式及隐式计算部分，如果输入特定的物理参数组合，可能会导致系数矩阵，即五对角矩阵的主对角线值为负，从而造成迭代异常，该问题发现较晚，可能由于在数学和物理上的理解有些偏差，该问题由于时间原因暂未修复。
- **内存使用方面** 显示及隐式计算输出到HDF5文件的部分，可以实现每个时间步长都输入到文件中，也可以用户指定每迭代多少次输出到HDF5文件中。但无论那种方式，都需要提前分配足够的向量数组空间，分配个数达到百万级别时，似乎会导致内存不足的情况导致程序异常退出。后续改进方向可以考虑探索动态分配向量数组的内存，或者只存储每100次或者1000次迭代的结果，从而节省内存的使用。
- **可视化方面** 提供的cpp代码默认读取HDF5文件的名字为u_t的group的Dataset，时间步长时按照(0), 1, 2, 3, ...读取，如果HDF5的时间步长改变，需要更改变量。可视化未实现并行，如果剖分的网格很密或者时间步长很多，也许会需要耗时多一点点；尝试了动画渲染，未成功，仅实现到了本地记的离屏渲染单张照片（而且VTK-8.2.0中不知道为什么没有找到AVIWriter），最终用ParaView实现动画。