HPC Final Project Report

12132400 李桓彬

1. Problem statement

The problem is to solve the transient heat equation in a one-dimensional domain Ω := (0, 1). The boundary of the domain is Γ = {0, 1}. Let f be the heat supply per unit volume, u be the temperature, ρ be the density, c be the heat capacity, u0 be the initial temperature, κ be the conductivity, n_x be the Cartesian components of the unit outward normal vector . The boundary data involves the prescribed temperature g on Γ_g and heat flux h on Γ_h. The boundary Γ admits a non-overlapping decomposition: $\Gamma = \Gamma_g \cup \Gamma_h$ and $\varnothing = \Gamma_g \cap \Gamma_h$.

The transient heat equation may be stated as follows.

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

To make problems simple,the following options are applied

$$f = \sin(l\pi x), \quad u_0 = e^x, \quad u(0,t) = u(1,t) = 0, \quad \kappa = 1.0.$$

2. Code development and Compiling environment

For the explicit euler scheme,

$$\frac{\partial u_j^n}{\partial t} = \frac{u_j^{n+1} - u_j^n}{\Delta t}$$

$$\frac{\partial^2 u_j^n}{\partial x^2} = \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}$$

And a matrix multiplication could be constructed to do the iteration

$$A\vec{u}(n)+\vec{f}=\vec{u}(n+1)$$

$$A=\begin{bmatrix} 1-2a & a & ... & ... & 0 \\ a & 1-2a & a & ... & ... \\ ... & & ... & & ... \\ ... & & a & 1-2a & a \\ 0 & & & a & 1-2a \end{bmatrix}$$

$$a=\frac{kdt}{\rho c(dx)^2}$$

For the explicit euler scheme,

$$\frac{\partial u_j^n}{\partial t}=\frac{u_j^n-u_j^{n\text{-}1}}{\Delta t}$$

$$\frac{\partial^2 u_j^n}{\partial x^2}=\frac{u_{j+1}^n-2u_j^n+u_{j-1}^n}{\Delta x^2}$$

And a linear system should be solved to do the iteration.A KSP linear solver is used in the implicit code.

$$A\vec{u}(n+1)=\vec{u}(n)+f$$

$$A=\begin{bmatrix} 1+2a & -a & ... & ... & 0 \\ -a & 1+2a & -a & ... & ... \\ ... & & ... & & ... \\ ... & & -a & 1+2a & -a \\ 0 & & & -a & 1+2a \end{bmatrix}$$

$$a=\frac{kdt}{\rho c(dx)^2}$$

The code and compiling environment are uploaded to the github repository https://github.com/huanbin-li/hpc-project

3. Profiling analysis

Use Valgrind to do the profiling analysis.The profiling is performed in condition of dx =0.01 and dt = 0.00001.

(1) explicit code

```
==375305== Memcheck, a memory error detector
==375305== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==375305== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==375305== Command: /share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/mpi/intel6
4/bin/mpirun ./explicit_euler.out
==375305==
==375309==
==375309== HEAP SUMMARY:
==375309==     in use at exit: 57,094 bytes in 1,212 blocks
==375309==   total heap usage: 1,768 allocs, 556 frees, 94,972 bytes allocated
==375309==
==375309== LEAK SUMMARY:
==375309==    definitely lost: 0 bytes in 0 blocks
==375309==    indirectly lost: 0 bytes in 0 blocks
==375309==      possibly lost: 0 bytes in 0 blocks
==375309==    still reachable: 57,094 bytes in 1,212 blocks
==375309==         suppressed: 0 bytes in 0 blocks
==375309== Rerun with --leak-check=full to see details of leaked memory
==375309==
==375309== For counts of detected and suppressed errors, rerun with: -v
==375309== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

A total of 1768 blocks are allocated , 556 blocks are freed, 1212 blocks are in use for a run of n=100. 94972 bytes are allocated and 57094 bytes are in use at exit.There are no memory losts and all the memories are still reachable.There are no other errors in the code.

(2)implicit code

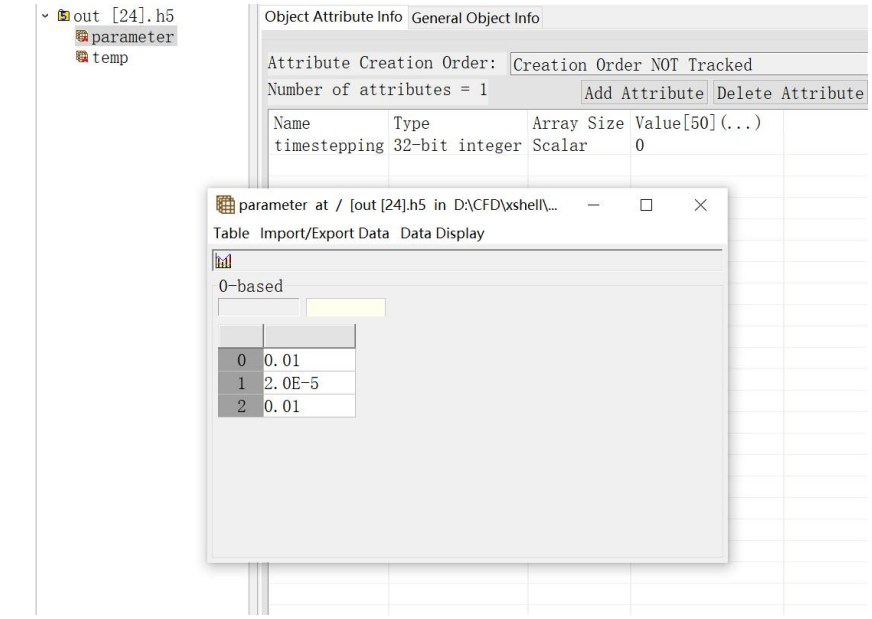
```
==69498== Memcheck, a memory error detector
==69498== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==69498== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==69498== Command: /share/intel/2018u4/compilers_and_libraries_2018.5.274/linux/mpi/intel64
/bin/mpirun ./implicit_euler.out
==69498==
==69500==
==69500== HEAP SUMMARY:
==69500==     in use at exit: 57,204 bytes in 1,212 blocks
==69500==   total heap usage: 1,768 allocs, 556 frees, 95,125 bytes allocated
==69500==
==69500== LEAK SUMMARY:
==69500==    definitely lost: 0 bytes in 0 blocks
==69500==    indirectly lost: 0 bytes in 0 blocks
==69500==      possibly lost: 0 bytes in 0 blocks
==69500==    still reachable: 57,204 bytes in 1,212 blocks
==69500==         suppressed: 0 bytes in 0 blocks
==69500== Rerun with --leak-check=full to see details of leaked memory
==69500==
==69500== For counts of detected and suppressed errors, rerun with: -v
==69500== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

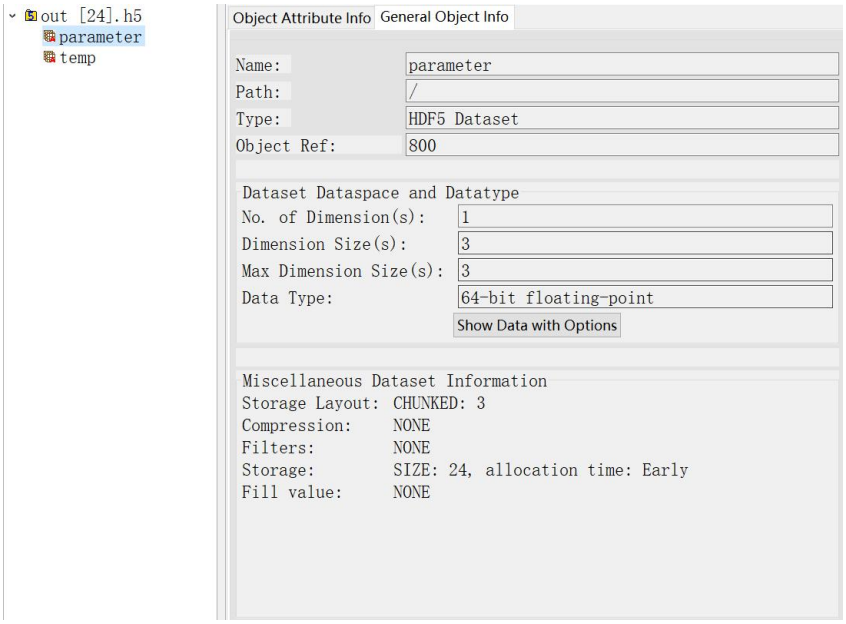
A total of 1768 blocks are allocated , 556 blocks are freed, 1212 blocks are in use for a run of n=100. 95125 bytes are allocated and 57204 bytes are in use at exit.There are no memory lost and all the memories are still reachable.There are no other errors in the code.

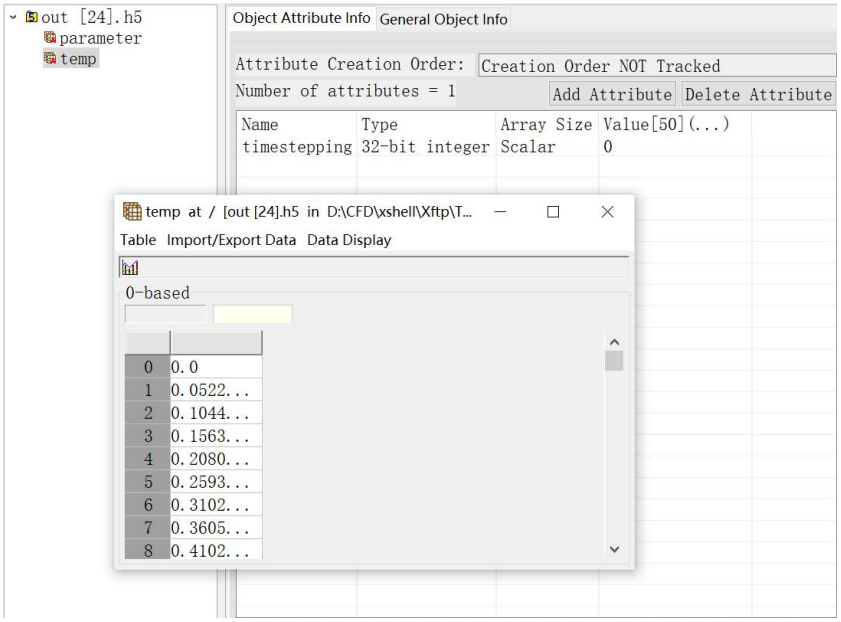
4.Use HDF5 for checkpoint restart

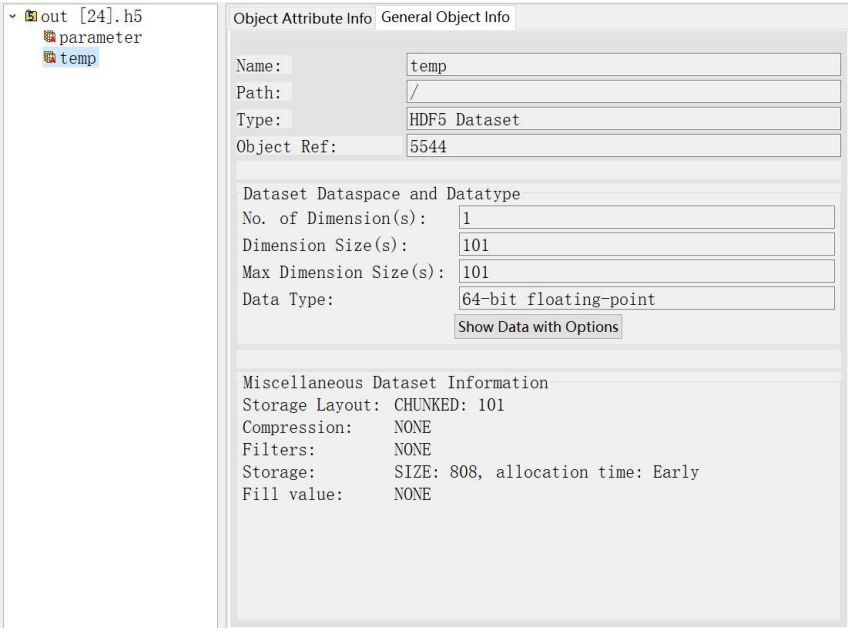
Each 10 timesteps the value temperature and iteration time are written into file out.h5 use VecView((variable name), h5). To restart from the checkpoint , use VecLoad((variable name), h5) to load the values from file out.h5 and continue calculation.

The output viewed in HDF5View is









Calculate the result at t=0.01 using following commands

```
mpirun -np 1 ./explicit_euler.out -n 100 -dt 0.00002 -t 0.01 -step 0 -ksp_type cg -pc_type
none -ksp_monitor -ksp_view -log_view \
```

Restart from t=0.01 and calculate to t=0.02,set step=1 to enable restart.

```
mpirun -np 1 ./explicit_euler.out -n 100 -dt 0.00002 -t 0.01 -step 1 -ksp_type cg -pc_type
none -ksp_monitor -ksp_view -log_view \
 > $LSB_JOBID.log 2>&1
```

t=0.02_restart [2].log

文件(F) 编辑(E) 格式(O)

0.
0.0522563
0.104418
0.15639
0.208081
0.259399
0.310257
0.360571
0.41026
0.45925
0.507469
0.554853
0.601343
0.646885
0.691435
0.734951
0.777401
0.818759
0.859005
0.898124
0.936111
0.972964
1.00869
1.04329
1.07679
1.1092
1.14056

t=0.02_no_restart [2].log -

文件(F) 编辑(E) 格式(O) 查看

0.
0.0522563
0.104418
0.15639
0.208081
0.259399
0.310257
0.360571
0.41026
0.45925
0.507469
0.554853
0.601343
0.646885
0.691435
0.734951
0.777401
0.818759
0.859005
0.898124
0.936111
0.972964
1.00869
1.04329
1.07679
1.1092
1.14056

The result is the same compared to directly calculate to t=0.02.

5.Stability analysis

At n=100 (dx=0.01), for the explicit code the solution divergence at 0.00005<t, which corresponds to CFL>0.5 ,so the maximum time step size that can deliver stable calculations is 0.00005.The calculation of CFL is $dt/(dx)^2$.For the implicit scheme, the code is always stable.This result correspond to the theoretical result.

dt=0.00005 [3].log

文件(F) 编辑(E) 格式(O

0.
0.00318284
0.00636253
0.00953595
0.0127
0.0158514
0.0189873
0.0221044
0.0251996
0.02827
0.0313125
0.0343242
0.0373019
0.0402428
0.043144
0.0460026
0.0488159
0.0515809
0.0542951
0.0569556
0.05956
0.0621056
0.0645899
0.0670104
0.0693648
0.0716508
0.073866

dt=0.000051.log

文件(F) 编辑(E) 格式(

0.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.
-inf.
inf.

Theoretical result of explicit method using Von-Neumann stability analysis (α =CFL):

$$\delta u_j^{n+1} = \delta u_j^n + \alpha\left[\delta u_{j-1}^n - 2\delta u_j^n + \delta u_{j+1}^n\right] = (1-2\alpha)\delta u_j^n + \alpha\left[\delta u_{j-1}^n + \delta u_{j+1}^n\right]$$

$$\delta u_j^n \sim e^{\sigma t^n} \cdot e^{i(k\cdot x_j)} \sim e^{\sigma\cdot n\Delta t} \cdot e^{i(k\cdot j\Delta x)}$$

$$e^{\sigma\cdot(n+1)\Delta t} \cdot e^{i(k\cdot j\Delta x)} = (1-2\alpha)e^{\sigma\cdot n\Delta t} \cdot e^{i(k\cdot j\Delta x)} + \alpha\left[e^{\sigma\cdot n\Delta t} \cdot e^{i(k\cdot(j-1)\Delta x)} + e^{\sigma\cdot n\Delta t} \cdot e^{i(k\cdot(j+1)\Delta x)}\right]$$

$$e^{\sigma\cdot\Delta t} = (1-2\alpha) + \alpha\left[e^{i(-k\cdot\Delta x)} + e^{i(k\cdot\Delta x)}\right] = (1-2\alpha) + 2\alpha\cos(k\Delta x)$$

$$\begin{gathered} |(1-2\alpha) + 2\alpha\cos(k\Delta x)| \le 1 \\ -1 \le 1-4\alpha \le 1 \\ \alpha \le \frac{1}{2} \end{gathered}$$

Theoretical result of implicit method using Von-Neumann stability analysis :

$$\delta u_j^{n+1} = \delta u_j^n + \alpha\left[\delta u_{j-1}^{n+1} - 2\delta u_j^{n+1} + \delta u_{j+1}^{n+1}\right]$$

$$\delta u_j^n \sim e^{\sigma t^n} \cdot e^{i(k\cdot x_j)} \sim e^{\sigma\cdot n\Delta t} \cdot e^{i(k\cdot j\Delta x)}$$

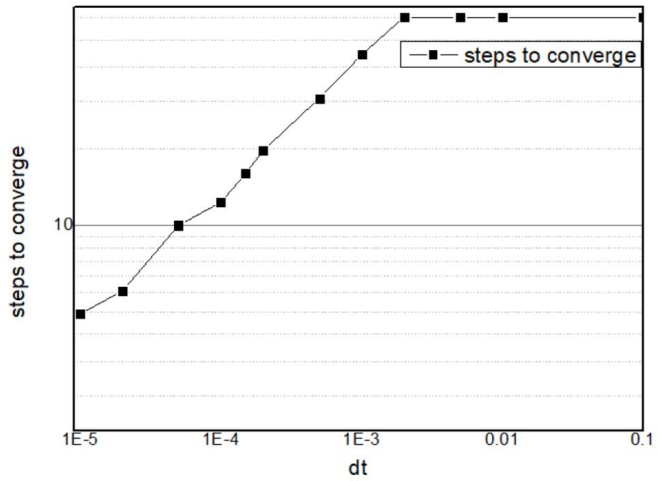
$$e^{\sigma\cdot(n+1)\Delta t} \cdot e^{i(k\cdot j\Delta x)} = e^{\sigma\cdot n\Delta t} \cdot e^{i(k\cdot j\Delta x)} + \alpha \cdot e^{\sigma\cdot(n+1)\Delta t}\left[e^{i(k\cdot(j-1)\Delta x)} - 2e^{i(k\cdot j\Delta x)} + e^{i(k\cdot(j+1)\Delta x)}\right]$$

$$e^{\sigma\cdot\Delta t} = 1 + 2\alpha\left[-1+\cos(k\Delta x)\right] e^{\sigma\Delta t}$$

$$\begin{gathered} e^{\sigma\cdot\Delta t} = \frac{1}{1+2\alpha\left[1-\cos(k\Delta x)\right]} \\ \left|\frac{1}{1+4\alpha}\right| \le \left|\frac{1}{1+2\alpha\left[1-\cos(k\Delta x)\right]}\right| \le 1 \end{gathered}$$

For the implicit method, test different time step size and the convergence speed of KSP solver,the relative tolerance is set at 10^{-7}.

dt	Steps to converge
0.00001	5
0.00002	6
0.00005	10
0.0001	12
0.00015	15
0.0002	18
0.0005	27
0.001	38
0.002	51
0.005	51
0.01	51
0.1	51



It can be seen that the convergence speed decrease as dt increase,and the convergence speed becomes stable at dt>0.002.The plot shows a relationship clos to exponential between steps to converge and dt .One can choose 0.0001<dt<0.001 for relatively large time step and convergence speed.

6.Result analysis

6.1 Code verification

At large t $\frac{\partial u}{\partial t} \approx 0$,and the analytical solution can be approximately calculated:

$$\frac{\partial^2 u}{\partial x^2} = -\frac{1}{\kappa} f = -\frac{1}{\kappa}\sin(l\pi x)$$

$$u = \frac{\sin(l\pi x)}{\kappa l^2 \pi^2} + C_1 x + C_2$$

$$u(0,t) = 0; u(1,t) = 0$$

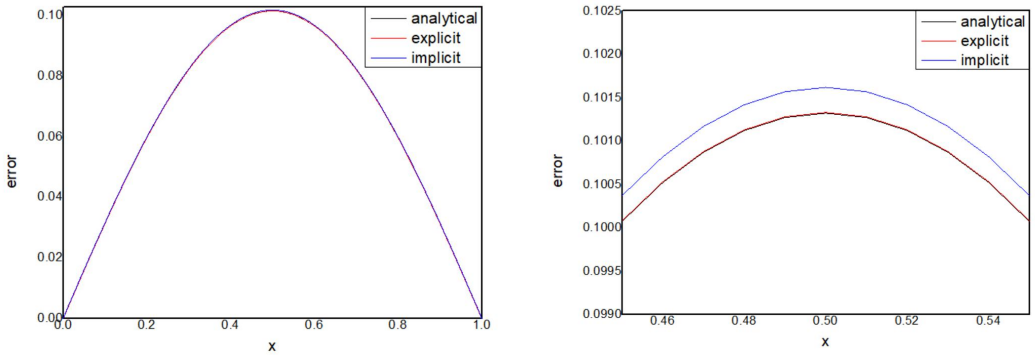
$$C_2 = 0; \frac{\sin(l\pi)}{\kappa l^2 \pi^2} + C_1 = 0$$

$$C_1 = -\frac{\sin(l\pi)}{\kappa l^2 \pi^2}$$

$$u = \frac{\sin(l\pi x)}{\kappa l^2 \pi^2} - \frac{\sin(l\pi)}{\kappa l^2 \pi^2} x$$

If all the constants equals 1,the analytical result can be simplified to

$$u = \frac{\sin(\pi x)}{\pi^2} - \frac{\sin(\pi)}{\pi^2} x$$
$$= \frac{\sin(\pi x)}{\pi^2}$$



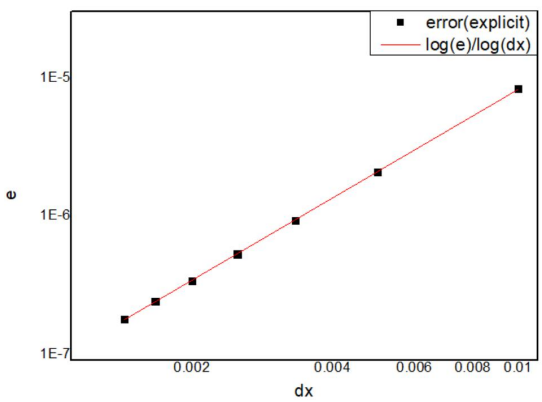
Plot the analytical result and explicit ,implicit result at dx=0.01,dt=0.00001.The results are very close to the analytical solution, which verifies that correctness of the code.It can be seen from the plot that the implicit scheme is less accurate than the explicit scheme.

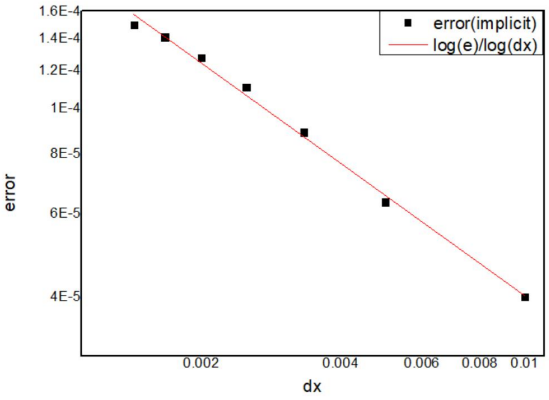
6.2 Error analysis

The error is related to the mesh resolution as $e \approx C_1\Delta x^{\alpha} + C_2\Delta t^{\beta}$.

First do the calculation of α .To calculate α ,choose a fixed $dt=10^{-6}$ and total time t=2.0,and calculate errors from n = 100 to n = 700.The error calculation is $e= \max_{1\le i\le n}|u_{exact,i} - u_{num,i}|$, where $u_{exact,i}$ and $u_{num,i}$ are the i-th component of the solution vectors with the vector length being n.

n	dx	Error(explicit)	Error(implicit)
100	0.01	0.00000834	0.00004002
200	0.005	0.00000209	0.00006337
300	0.003333	0.00000093	0.00008895
400	0.0025	0.00000053	0.00011050
500	0.002	0.00000034	0.00012780
600	0.001667	0.00000024	0.00014105
700	0.001429	0.00000018	0.00014998

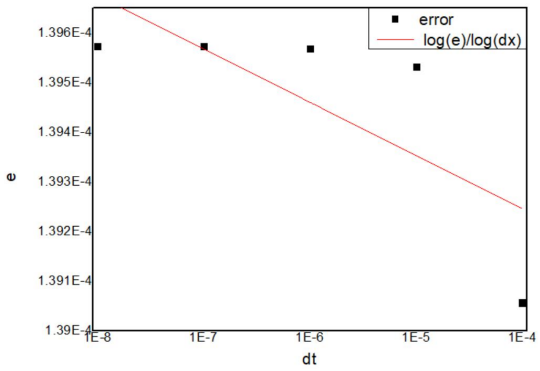


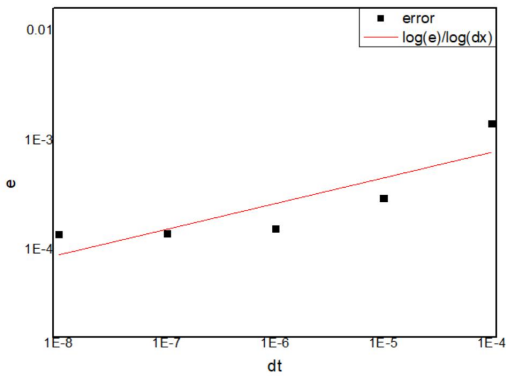


Plot the error and dx in logarithmic coordinate,and do linear fit to the points.The slope of the fitting straight line is the value of α .For explicit scheme, α =1.9755,for implicit scheme, α =-0.7016.The relation between log(e) and log(Δ x) is $\log(e) = \alpha \log(\Delta x) + \log(C_1)$ .The intercept of the line on y axis is log(C_1).For explicit scheme, C_1=0.3224, For implicit scheme, C_1=0.003034.

Choose a relatively low grid resolution to calculate β ,choose a fixed dx=0.02 or n=50 ,and calculate errors from dt=2×10^{-7} to 1×10^{-6}.For the explicit

dt	Error(explicit)	Error(implicit)
10^{-4}	0.0001390570	0.00143085
10^{-5}	0.0001395319	0.00029698
10^{-6}	0.0001395680	0.00015568
10^{-7}	0.0001395726	0.00014161
10^{-8}	0.0001395730	0.00013978





Plot the error and dx in logarithmic coordinate,and do linear fit to the points.The slope of the fitting straight line is the value of β .For explicit scheme, β =0.00034378,for implicit scheme, β =0.2342.The relation between log(e) and log(Δ t) is $\log(e) = \beta \log(\Delta t) + \log(C_2)$ .The intercept of the line on y axis is log(C_2).For explicit scheme, C_2=0.02111, For implicit scheme, C_2=0.1140.

For explicit scheme, $e = 0.3224\,(\Delta x)^{1.9755} + 0.02111\,(\Delta t)^{0.00034378}$ .For implicit

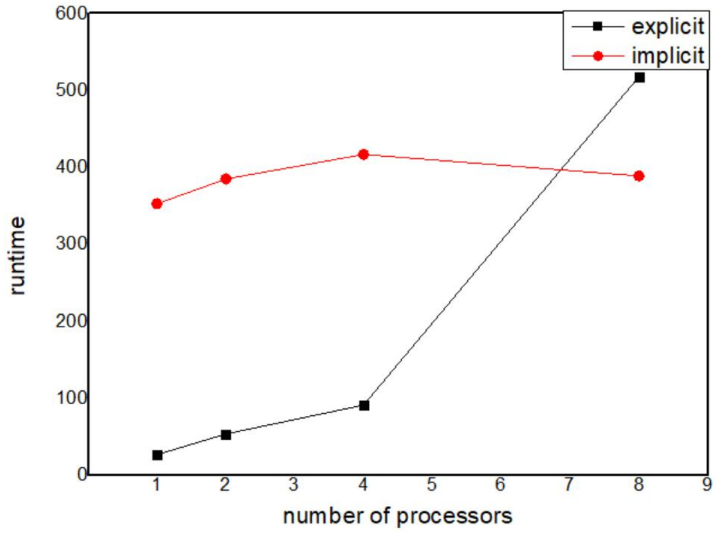
scheme, $e = 0.003034\,(\Delta x)^{-0.7016} + 0.1140\,(\Delta t)^{0.2342}$.

7.Parallelism

7.1 Fixed size scalability

Plot the run time and the number of processors for both schemes.Other parameters are dx=0.001 or n=1000,dt=10^{-6},t=2s.

Number of processors	Run time(explicit)	Run time(implicit)
1	27.37s	353.0s
2	54.06s	385.1s
4	91.70s	417.1s
8	518.0s	389.1s
16	1035s	11410s



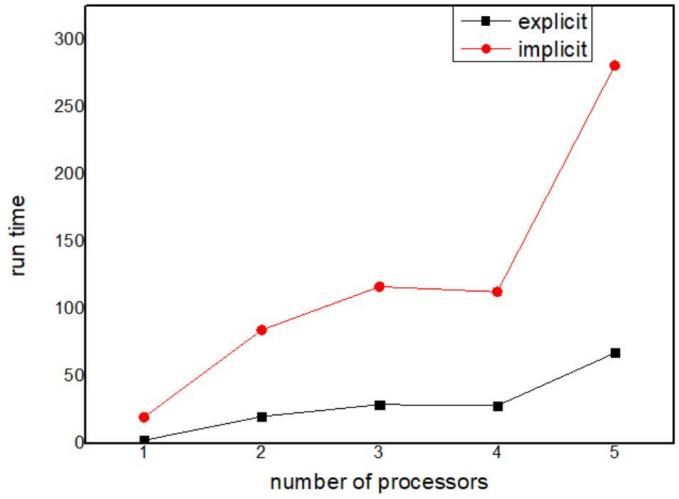
It can be seen that the run time generally increases as the number of processors increase.This is probably because of the grid resolution is not large enough and the time reduced by paralleling is smaller than the time increased by the communication between processors and part of the code which are not affected by parallel computing.For implicit code the run time decrease as the number of processors is increased from 8 to 16,but drastically increase at 32 processors.This is probably because of the communication between processors and part of the code which are not affected by parallel computing takes more time in the implicit code.

7.1 Isogranular scalability

Plot the run time and the number of processors for both schemes.Each processor do the calculation of 100 grids.Other parameters are the same as the fixed-size scalability analysis.

Number of	Number of grids	Run time(explicit)	Run time(implicit)

processors			
1	100	2.649s	19.81s
2	200	20.33s	84.56s
3	300	29.20s	116.7s
4	400	28.38s	112.9s
5	500	67.58s	281.1s



Similarly the run time generally increases as the number of processors increase.

For implicit code the run time decrease as the number of processors is increased from 3 to 4,but drastically increase at 5 processors.For both scalability,the run time is the smallest when the number of processors is 1.

7.3 Comparison of different linear solvers

```
mpirun -np 2 ./implicit_euler.out -ksp_type gmres \
  -ksp_gmres_restart 30 -ksp_rtol 1.0e-10 \
  -ksp_atol 1.0e-50 -ksp_max_it 1500 \
  -ksp_gmres_modifiedgramschmidt \
  -pc_type jacobi \
  -sub_ksp_type richardson \
  -sub_pc_type icc -ksp_monitor_short \
  -ksp_converged_reason \
  -n 100 -dt 0.00004 -t 1.0 -step 0 \
  -ksp_view \
  -log_view > $LSB_JOBID.log 2>&1
```

```
mpirun -np 2 ./implicit_euler.out -ksp_type gmres \
  -ksp_gmres_restart 30 -ksp_rtol 1.0e-10 \
  -ksp_atol 1.0e-50 -ksp_max_it 1500 \
  -ksp_gmres_modifiedgramschmidt \
  -pc_type asm \
  -sub_ksp_type richardson \
  -sub_pc_type icc -ksp_monitor_short \
  -ksp_converged_reason \
  -n 100 -dt 0.00004 -t 1.0 -step 0 \
  -ksp_view \
  -log_view > $LSB_JOBID.log 2>&1
```

```
mpirun -np 2 ./implicit_euler.out -ksp_type preonly \
  -ksp_gmres_restart 30 -ksp_rtol 1.0e-10 \
  -ksp_atol 1.0e-50 -ksp_max_it 1500 \
  -ksp_gmres_modifiedgramschmidt \
  -pc_type lu \
  -sub_ksp_type richardson \
  -sub_pc_type icc -ksp_monitor_short \
  -ksp_converged_reason \
  -ksp_view \
  -pc_factor_mat_solver_type mumps \
  -n 100 -dt 0.00004 -t 1.0 -step 0 \
  -ksp_view \
  -log_view > $LSB_JOBID.log 2>&1
```

Change the flags in the command line to enable different solvers.From left to right are the flags and for (a) Jacobi; (b) Additive Schwarz; (c) LU with MUMPS.Run the calculation on dt=0.00004,t=1,n=100 or dx=0.01.

```
./implicit_euler.out on a  named r01n08 with 2 processors, by mae-lihb Thu Jun  9 21:43:54 2022
Using Petsc Release Version 3.16.6, Mar 30, 2022

                         Max       Max/Min     Avg       Total
Time (sec):           1.750e+01     1.000   1.750e+01
error = 0.10132118
Objects:              4.600e+01     1.000   4.600e+01
Flop:                 6.555e+08     1.020   6.490e+08  1.298e+09
Flop/sec:             3.745e+07     1.020   3.708e+07  7.416e+07
MPI Messages:         4.005e+05     1.000   4.005e+05  8.010e+05
MPI Message Lengths:  3.205e+06     1.000   8.003e+00  6.411e+06
MPI Reductions:       2.683e+06     1.000

Flop counting convention: 1 flop = 1 real number operation of type (multiply/divide/add/subtract)
                            e.g., VecAXPY() for real vectors of length N --> 2N flop
                            and VecAXPY() for complex vectors of length N --> 8N flop

Summary of Stages:   ----- Time ------  ----- Flop ------  --- Messages ---  -- Message Lengths --  -- Reductions --
                        Avg     %Total     Avg     %Total    Count   %Total     Avg         %Total    Count   %Total
 0:      Main Stage: 1.7504e+01 100.0%  1.2981e+09 100.0%  8.010e+05 100.0%  8.003e+00      100.0%  2.683e+06 100.0%

------------------------------------------------------------------------------------------------------------------------
```

For Jacobi ,the run time is 17.5s and it took 13 steps for KSP solver to converge.

```
./implicit_euler.out on a  named r01n15 with 2 processors, by mae-lihb Thu Jun  9 21:43:59 2022
Using Petsc Release Version 3.16.6, Mar 30, 2022

                         Max       Max/Min     Avg       Total
Time (sec):           9.615e+00     1.000   9.615e+00
error = 0.10132118
Objects:              2.506e+04     1.000   2.506e+04
Flop:                 1.905e+08     1.020   1.886e+08  3.772e+08
Flop/sec:             1.981e+07     1.020   1.962e+07  3.923e+07
MPI Messages:         2.750e+05     1.000   2.750e+05  5.500e+05
MPI Message Lengths:  2.101e+06     1.000   7.641e+00  4.202e+06
MPI Reductions:       3.000e+05     1.000

Flop counting convention: 1 flop = 1 real number operation of type (multiply/divide/add/subtract)
                            e.g., VecAXPY() for real vectors of length N --> 2N flop
                            and VecAXPY() for complex vectors of length N --> 8N flop

Summary of Stages:   ----- Time ------  ----- Flop ------  --- Messages ---  -- Message Lengths --  -- Reductions --
                        Avg     %Total     Avg     %Total    Count   %Total     Avg         %Total    Count   %Total
 0:      Main Stage: 9.6150e+00 100.0%  3.7721e+08 100.0%  5.500e+05 100.0%  7.641e+00      100.0%  3.000e+05 100.0%

------------------------------------------------------------------------------------------------------------------------
```

For Additive Schwarz ,the run time is 9.615s and it took 3 steps for KSP solver to converge.

```
./implicit_euler.out on a  named r01n08 with 2 processors, by mae-lihb Thu Jun  9 21:47:02 2022
Using Petsc Release Version 3.16.6, Mar 30, 2022

                         Max       Max/Min     Avg       Total
Time (sec):           5.954e+01     1.000   5.954e+01
error = 0.10132118
Objects:              2.502e+04     1.000   2.502e+04
Flop:                 2.920e+07     1.825   2.260e+07  4.520e+07
Flop/sec:             4.904e+05     1.825   3.796e+05  7.591e+05
MPI Messages:         4.125e+05     1.000   4.125e+05  8.250e+05
MPI Message Lengths:  6.257e+08     1.000   1.517e+03  1.251e+09
MPI Reductions:       1.250e+05     1.000

Flop counting convention: 1 flop = 1 real number operation of type (multiply/divide/add/subtract)
                            e.g., VecAXPY() for real vectors of length N --> 2N flop
                            and VecAXPY() for complex vectors of length N --> 8N flop

Summary of Stages:   ----- Time ------  ----- Flop ------  --- Messages ---  -- Message Lengths --  -- Reductions --
                        Avg     %Total     Avg     %Total    Count   %Total     Avg         %Total    Count   %Total
 0:      Main Stage: 5.9541e+01 100.0%  4.5198e+07 100.0%  8.250e+05 100.0%  1.517e+03      100.0%  1.250e+05 100.0%

------------------------------------------------------------------------------------------------------------------------
```

For LU with MUMPS ,the run time is 59.54s and it took 1 step for KSP solver to converge.

It can be concluded that the LU with MUMPS uses the most time and Additive Schwarz uses the least time.The Jacobi solver converges slowest and the LU with MUMPS converges the fastest.