# SSC 394 Class Project – 1D Heat Equation Solver

Ju Liu

February 13, 2014

## 1 Problem Setting

The mathematical problem of this project is to solve a 1D heat equation by numerical method. The heat equation is an important parabolic partial differential equation which describes the distribution of heat over a given region. Under the one dimensional setting, the heat equation reads as follows:

$$\frac{\partial T(x,t)}{\partial t} = \alpha \frac{\partial^2 T(x,t)}{\partial x^2} + q(x,t) \tag{1}$$

wherein, $t \geq 0$ is time; $x \in [0,1]$ is the spatial variable; $T = T(x,t)$ is the temperature; $\alpha$ is the thermal diffusivity constant; and $q = q(x,t)$ is the heat source function. To make the differential equation well setted, we need to give boundary condition and initial condition.

$$\begin{aligned} T(x,0) &= T_0(x), \qquad \text{for } x \in [0,1] \quad (2)\\ T(0,t) &= T_a(t), \qquad \text{for } t > 0 \quad (3)\\ T(1,t) &= T_b(t), \qquad \text{for } t > 0. \quad (4) \end{aligned}$$

## 2 Numerical Schemes

To solve this differential equation via numerical methods, we need to do the following discretization first. We discretize the spatial domain $[0,1]$ by a uniform mesh with $M$ elements. Then the spatial mesh size is $\Delta x = h = 1/M$. The time domain is discretized by $N$ elements, *i.e.* the mesh size in time direction is $\Delta t = k = 1/N$. Then the gird for the discretization is:

$$\begin{aligned} t_n &= nk \qquad n = 0, 1, \cdots, N \quad (5)\\ x_m &= mh \qquad m = 0, 1, \cdots, M. \quad (6) \end{aligned}$$

For convenience, we denote $T(x_m, t_n)$ as $T_m^n$. Then we state the numerical method.

**(1) Explicit Euler method**

We take central difference for the spatial derivative and take forward difference for the time derivative:

$$\begin{aligned} \frac{\partial^2 T}{\partial x^2}(x_m, t_n) &\approx \frac{T_{m+1}^n - 2T_m^n + T_{m-1}^n}{h^2} \quad (7)\\ \frac{\partial T}{\partial t}(x_m, t_n) &\approx \frac{T_m^{n+1} - T_m^n}{k} \quad (8) \end{aligned}$$

If we substitute the differential operator by the above finite difference, we may get the Explicit Euler Scheme:

$$\frac{T_m^{n+1} - T_m^n}{k} = \alpha \frac{T_{m+1}^n - 2T_m^n + T_{m-1}^n}{h^2} + q_m^n \tag{9}$$

If we denote $r = k/h^2$, we can get

$$T_m^{n+1} = \alpha r T_{m+1}^n + (1 - 2\alpha r) T_m^n + \alpha r T_{m-1}^n + k q_m^n \tag{10}$$

If we denote the coefficient matrix as A which is a tridiagonal matrix with main diagonal entries $1 - 2\alpha r$, and subdiagonal entries $\alpha r$. The explicit method is just

$$\mathbf{T}^{n+1} = A\mathbf{T}^n + k\mathbf{q}^n \tag{11}$$

This is simply a process of matrix vector multiplication.

**(2) Implicit Euler method**

We take central difference for the spatial derivative and take backward difference for the time derivative:

$$\frac{\partial^2 T}{\partial x^2}(x_m, t_{n+1}) \approx \frac{T_{m+1}^{n+1} - 2T_m^{n+1} + T_{m-1}^{n+1}}{h^2} \tag{12}$$

$$\frac{\partial T}{\partial t}(x_m, t_{n+1}) \approx \frac{T_m^{n+1} - T_m^n}{k} \tag{13}$$

If we substitute the differential operator by the above finite difference, we may get the Implicit Euler Scheme:

$$\frac{T_m^{n+1} - T_m^n}{k} = \alpha \frac{T_{m+1}^{n+1} - 2T_m^{n+1} + T_{m-1}^{n+1}}{h^2} + q_m^{n+1} \tag{14}$$

If we denote $r = k/h^2$, we can get

$$-\alpha r T_{m+1}^{n+1} + (1 + 2\alpha r) T_m^{n+1} - \alpha r T_{m-1}^{n+1} = T_m^n + k q_m^n \tag{15}$$

Then if we denote the coefficient matrix as B which is a tridiagonal matrix with main diagonal entries $1 + 2\alpha r$, and subdiagonal entries $-\alpha r$. The implicit method is just

$$B\mathbf{T}^{n+1} = \mathbf{T}^n + k\mathbf{q}^n \tag{16}$$

This is a process of solving linear system of equations at each time step.

**Initial and Boundary conditions:** For both the explicit and implicit method, we need to give discrete initial and boundary conditions. Here, the initial solution is $T_m^0 = T_0(x_m)$. And the boundary conditions are $T_0^n = T_a(t_n)$, $T_M^n = T_b(t_n)$.

Up to now, we have already setted up our numerical schemes for solving the 1D heat equation.

# 3 Numerical Tests

## 3.1 Method Stability

### 3.1.1 Explicit Method

Here, we test our code with the following setting: initial solution is $T_0 = e^x$; boundary condition are homogeneous $T_a(t) = T_b(t) = 0$; thermal diffusivity $\alpha = 1$; and the heat source function is $q(x,t) = sin(l\pi x)$. The space discretization is fixed at $h = 10^{-2}$ and we tried different time step.

$l = 1$ We tried time step $k = 1.0 \times 10^{-1}, 1.0 \times 10^{-2}, 1.0 \times 10^{-3}, 1.0 \times 10^{-4}, 5.0 \times 10^{-5}, 1.0 \times 10^{-5}$. Explicit method diverges until we took time step $k = 5.0 \times 10^{-5}$, and $k = 5.0 \times 10^{-5}$ is the biggest time step to achieve stability. Actually, in numerical analysis, the time step size $k$ and space step size $h$ need to satisfy the requirement

$$\frac{\alpha k}{h^2} \leq \frac{1}{2}$$

to ensure stable explicit scheme. In our case, this means $k \leq 0.5 \times (0.01)^2 = 5.0 \times 10^{-5}$. Below we plotted the visualization of the solution with $k = 1.0 \times 10^{-2}$ and $k = 5.0 \times 10^{-5}$.

```
fig1.png
```

Figure 1: Solution of explicit method with $l = 1$ $k = 1.0e^{-2}$ (left) and $k = 5.0e^{-5}$ (right)

$l = 5$ Here we give a higher frequency in the source term. Time step are tried with $k = 1.0 \times 10^{-1}, 1.0 \times 10^{-2}, 1.0 \times 10^{-3}, 1.0 \times 10^{-4}, 5.0 \times 10^{-5}, 1.0 \times 10^{-5}$. The explicit converges with biggest time step $k = 5.0 \times 10^{-5}$. Visualization of the solution with $k = 1.0 \times 10^{-2}$ and $k = 5.0 \times 10^{-5}$ are given below.

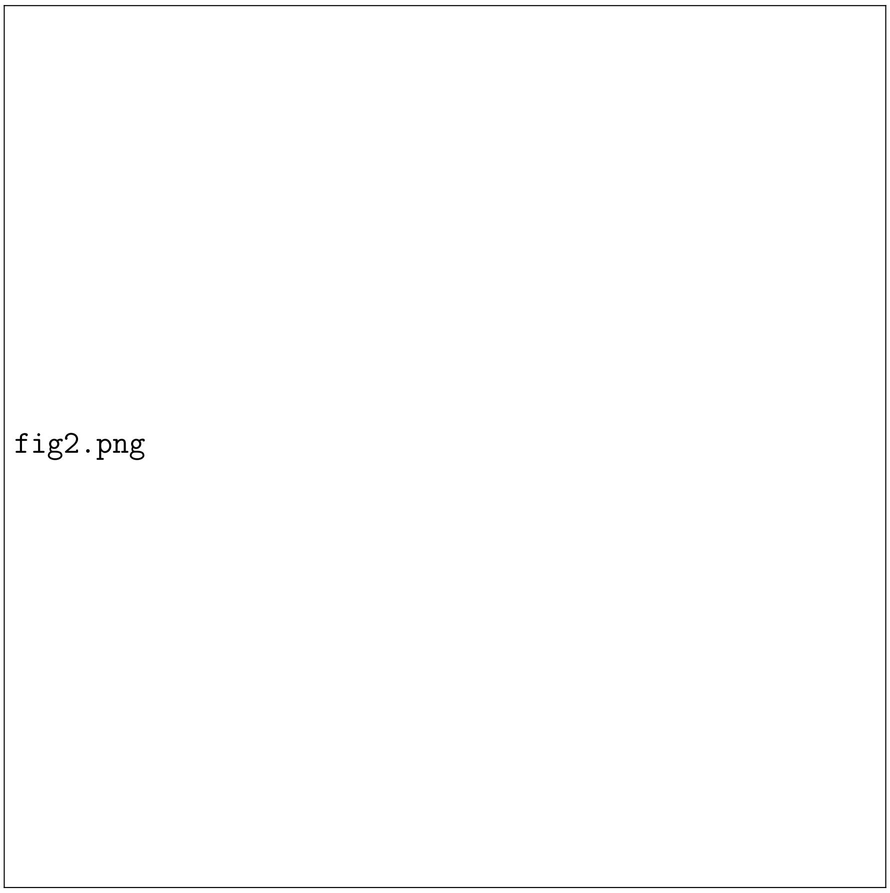

Figure 2: Solution of explicit method with $l = 5$ $k = 1.0e^{-2}$ (left) and $k = 5.0e^{-5}$ (right)

$l = 7$ Here we give a more higher frequency in the source term. Time step are tried with $k = 1.0 \times 10^{-1}, 1.0 \times 10^{-2}, 1.0 \times 10^{-3}, 1.0 \times 10^{-4}, 5.0 \times 10^{-5}, 1.0 \times 10^{-5}$. The explicit converges with biggest time step $k = 5.0 \times 10^{-5}$. Visualization of the solution with $k = 1.0 \times 10^{-2}$ and $k = 5.0 \times 10^{-5}$ are given below.

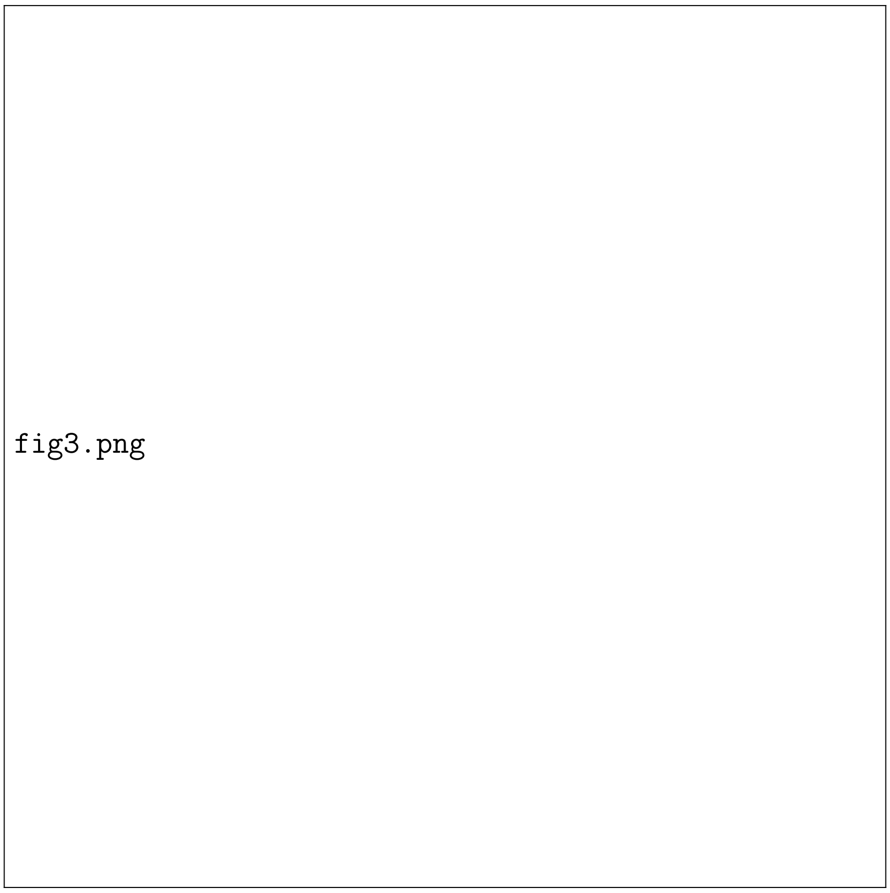

Figure 3: Solution of explicit method with $l = 7$ $k = 1.0e^{-2}$ (left) and $k = 5.0e^{-5}$ (right)

### 3.1.2 Implicit Method

First we solve the Implicit Scheme via direct method. We use options *-ksp_type preonly -pc_type lu* to make it direct solver. We tested it with time step size $k = 0.2, 0.1, 0.01, 0.001, 0.0001$ and space step size is fixed at $h = 0.01$. We tried with $l = 1$ and $l = 5$ The results are stable for all the time steps. We give the visualization of the solution in below.

`fig4.png`

Figure 4: Solution of implicit method with direct solver, $l = 1$, k=0.2, 0.1, and 0.01 respetively

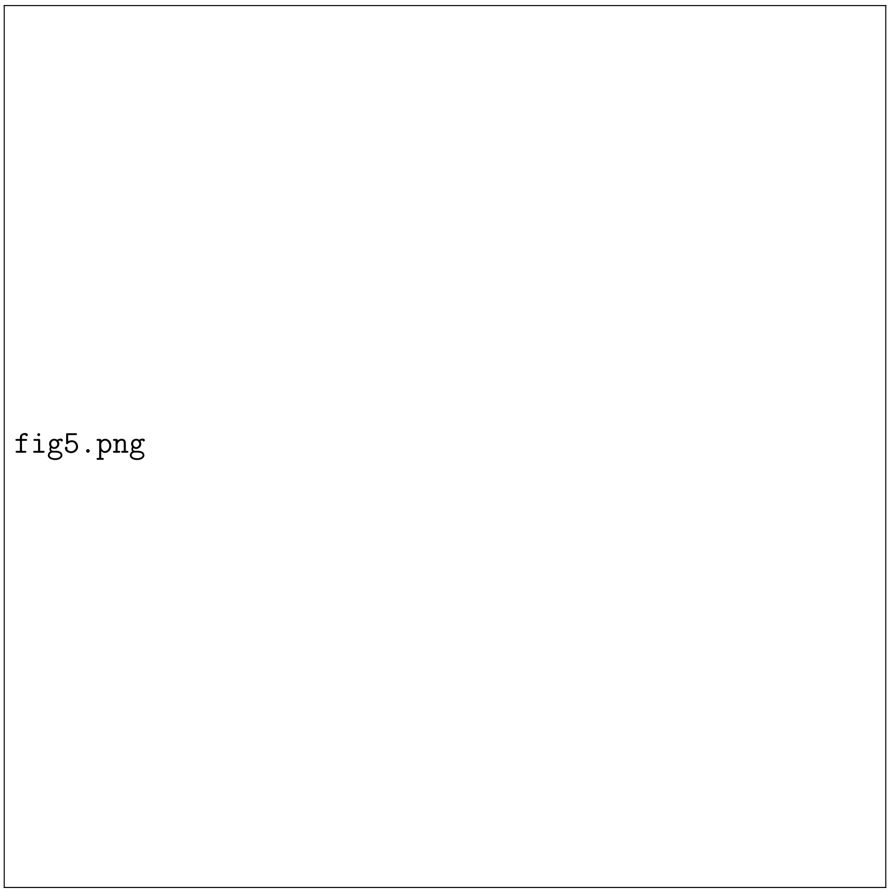

Figure 5: Solution of implicit method with direct solver, $l = 5$, k=0.2, 0.1, 0.01 and 0.001 respetively

Secondly, we solve the implicit scheme via iterative method. We acheive this via the options *-ksp_type cg -pc_type jacobi*. We pick conjugate gradient method because the coefficient matrix is symmetric. The method is stable with the iterative method. We test the iterative with small tolerance and compare it with the solution of large tolerance. The two tolerance we chose are: small tolerance $1.0 \times 10^{-16}$, and large tolerance $1.0 \times 10^{-1}$. We also compare differnt time steps $k = 0.2$ and 0.01. The solution of the four different simulation are plotted below by gnuplot. We can see that even with tolerace 0.1, we still can get stable solution with the conjugate gradient method. Time step here seems to play a big role in accuracy. For bigger time steps, the solution is worse than the solution with small time steps. We conclude here that the time step here play a bigger role in influencing the solution.

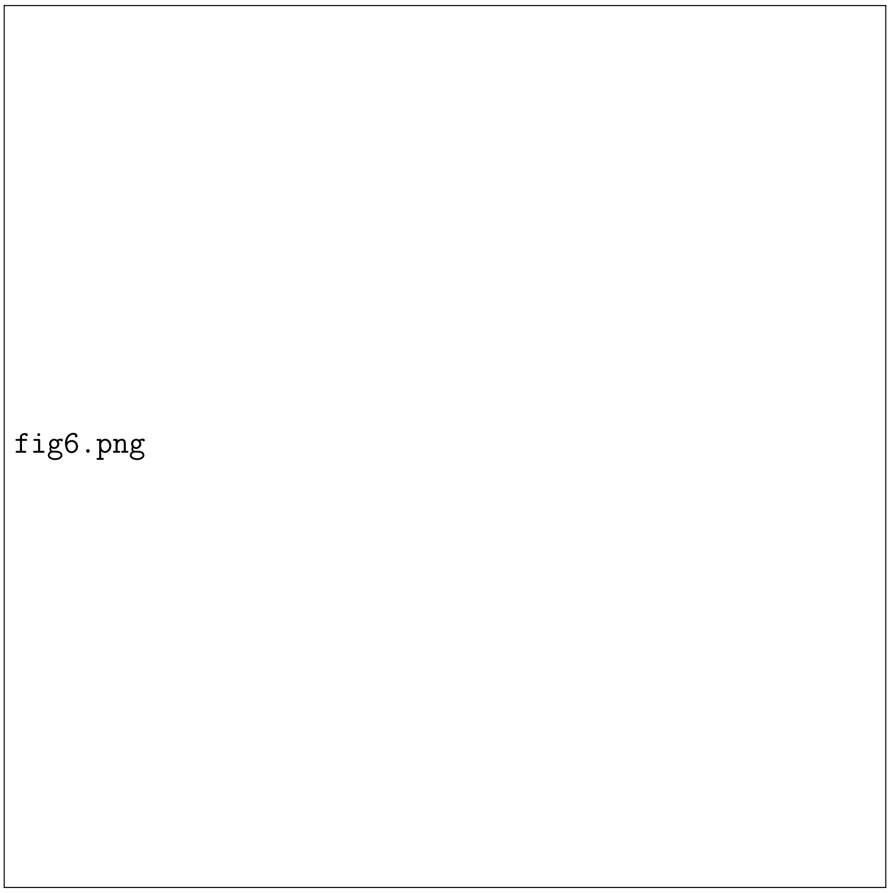

Figure 6: Solution of implicit scheme with iterative method, $l = 1$, $k = 0.2$ $tol = 0.1$, $k = 0.01$ $tol = 0.1$, $k = 0.2$ $tol = 1.0 \times 10^{-16}$ and $k = 0.01$ $tol = 1.0 \times 10^{-16}$

### 3.1.3 Steady State Solution

If we take the final time big enough, we should be able to get the steady state solution $T_\infty$. The reason is that as $t \to \infty$, $\frac{\partial T}{\partial t} \to 0$. Then in our case,

$$
\begin{aligned}
0 &= \alpha \frac{\partial^2 T_\infty}{\partial x^2} + q(x,t) && (17)\\
\Rightarrow \frac{\partial^2 T_\infty}{\partial x^2} &= -\sin(l\pi x), \qquad \text{with } T_\infty(0) = T_\infty(1) = 0 && (18)\\
\Rightarrow T_\infty(x) &= \frac{1}{(l\pi)^2} \sin l\pi x && (19)
\end{aligned}
$$

$l = 1$ In this case, the steady state solution should be

$$
T_\infty(x) = \frac{1}{(l\pi)^2} \sin l\pi x = \frac{1}{\pi^2} \sin \pi x \tag{20}
$$

We run the code with implicit method, final time is set to be $T_{final} = 1000.0$, and the time step size is $k = 0.1, 0.01$, space step size is taken to be 0.01. The code gives solution pretty close to the analytical steady state solution. The PETSc restult is visualized as below:

fig7.png

Figure 7: Solution of steady state solution with $l = 1$

fig8.png

Figure 8: Solution of steady state solution with $l = 2$

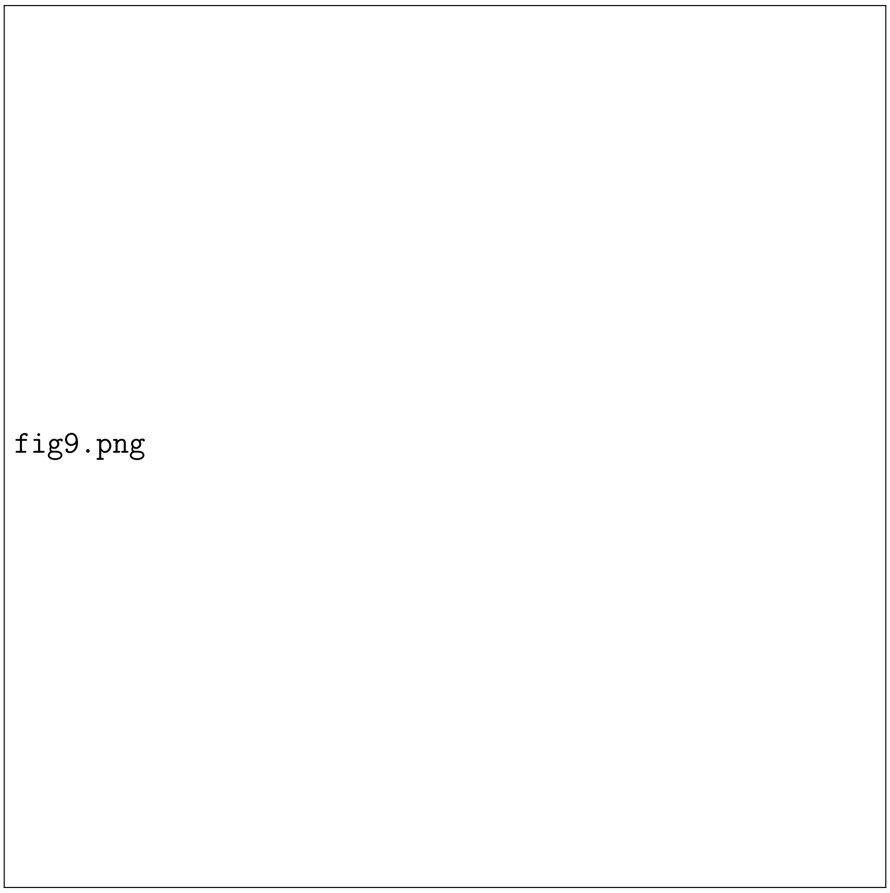

Figure 9: Solution of steady state solution with $l = 3$

**$l = 2$ and $l = 3$** Here we test the problem with frequency $l = 2$ and $l = 3$. the steady state solution should be

$$T_\infty(x) = \frac{1}{(l\pi)^2}\sin l\pi x = \frac{1}{(2\pi)^2}\sin 2\pi x \tag{21}$$

$$T_\infty(x) = \frac{1}{(l\pi)^2}\sin l\pi x = \frac{1}{(3\pi)^2}\sin 3\pi x \tag{22}$$

Same scheme is applied. The code gives us the numerical solution pretty close to the analytical steady state solution. The results are visualized in above.

As for the time step issue, we observed that the implicit method will always gives a satisfactory steady state solution. For small time step, the solution is more accurate but it takes a long computation time to obtain the solution. For large time step, the error is relatively big but the speed is high. Since for implicit method, the *priori* error is of order $O(h^2) + O(k)$, for a fixed space mesh, the time step should not be over $h^2$. We can not get a better convergence rate with time step size bigger than $h^2$.

## 3.2 Timing

### 3.2.1 Dense Matrix Performance Test

**Explicit Method:** For the test, we take spatial step size $h = 0.01$ and time step size $k = 5.0\times10^{-5}$, the frequency of heat source $l = 1$, and the test result is:

| Total memory | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| $2.268\times10^{5}$ bytes | $5.416\times10^{1}$ sec | $2.708\times10^{-3}$ sec | $6.648\times10^{9}$ | $1.228\times10^{8}$ |

**Implicit Method:** In the implicit method, we take the same mesh as we did in explicit method for comparisom, the solver we use is direct method:

| Total memory | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| $1.255\times10^{5}$ bytes | $7.966\times10^{0}$ sec | $3.983\times10^{-4}$ sec | $4.101\times10^{8}$ | $5.148\times10^{7}$ |

From the above results, we can see that the implicit method uses less memory and floating point operation than the explicit method. The reason for this phenomena is that for dense matrix, the explicit method needs to do multiplication involving all the entries in the coefficient matrix. However, as for the implicit method, the coefficient matrix is a tridiagonal matrix. The time spent on solving the tridiagonal system is $O(n)$. Therefore, the floating point operation needed for implicit method is much less than the explicit method. This advantage directly leads to the efficiency of implicit method in solving this problem.

### 3.2.2 Sparse Matrix Performance Test

**Explicit Method:** Here we list the performace of the sparse matrix with explicit method. Here we solve a problem with space step size $h = 0.01$ and time step size $k = 5.0 \times 10^{-5}$. The frequency of heat source function $l = 1$. The problem is tested on a single CPU.

| Total memory | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| $5.206\times10^{4}$ bytes | $2.079\times10^{0}$ sec | $1.040\times10^{-4}$ sec | $1.406\times10^{7}$ | $6.764\times10^{6}$ |

**Implicit Method:** For the implicit method, we use the same parameter as we used in explicit method for comparisom. The solver we use is conjugate method with preconditioner of Jacobi type

| Total memory | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| $6.493\times10^{4}$ bytes | $5.291\times10^{0}$ sec | $2.646\times10^{-4}$ sec | $7.773\times10^{7}$ | $1.469\times10^{7}$ |

If we use direct method to solve this implicit method, the performace is:

| Total memory | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| $7.517\times10^{4}$ bytes | $3.129\times10^{0}$ sec | $1.565\times10^{-4}$ sec | $2.110\times10^{7}$ | $6.744\times10^{6}$ |

Here the implicit method behaves poorly for sparse matrix. The reason for this phenomena is that solving the linear system in sparse matrix takes more time than matrix vector multiplication in sparse matrix form. In our case, the sparse matrix vector multiplication involves three entries at most in each row. However, solving the linear system has the Flops of order $O(n)$. For sparse matrix, explicit method is more efficiency in solving the discrete problem with the same size.

## 3.3 Restart

In our code, we give one more option *-save SaveFlag*. If *SaveFlag* is 0, we will run the simulation without saving any iteration results; however, if the *SaveFlag* is nonzero, we will save the iteration results, time step, space step, and $l$ into a HDF5 file. The HDF5 file is named "fileN", where N means the file saves the iteration results at step $10 \times N$.

Another option we added is *-restart RestartFlag*. If *RestartFlag* is 0, the program will run normally; otherwise (*i.e.* *RestartFlag* is nonzero), the program will run the simulation first, save iteration results every ten steps and then load the HDF5 file "fileN", wherein N = RestartFlag.

Our test is: running the program for 25 iterations, and restart the code at iteration 20. We compare the values at iteration 20, 21, 22, 23, 24, and 25. To make the problem convenient to show in our paper, we did the test first for $h = 1/5$, $k = 1/50$, $l = 1$. Both explicit and implicit method are tested.

| | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.0772914 | 0.0772914 |
| 3 | 0.12809 | 0.12809 |
| 4 | 0.12506 | 0.12506 |
| 5 | 0.0791643 | 0.0791643 |
| 6 | 0 | 0 |

Table 1: Explicit method at time step 20

| | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.0758009 | 0.0758009 |
| 3 | 0.120197 | 0.120197 |
| 4 | 0.122649 | 0.122649 |
| 5 | 0.0742858 | 0.0742858 |
| 6 | 0 | 0 |

Table 2: Explicit method at time step 21

|  | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.0718541 | 0.0718541 |
| 3 | 0.118246 | 0.118246 |
| 4 | 0.116262 | 0.116262 |
| 5 | 0.07308 | 0.07308 |
| 6 | 0 | 0 |

Table 3: Explicit method at time step 22

|  | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 |  |
| 2 | 0.0708786 | 0.0708786 |
| 3 | 0.113079 | 0.113079 |
| 4 | 0.114684 | 0.114684 |
| 5 | 0.0698869 | 0.0698869 |
| 6 | 0 | 0 |

Table 4: Explicit method at time step 23

|  | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.0682954 | 0.0682954 |
| 3 | 0.111802 | 0.111802 |
| 4 | 0.110504 | 0.110504 |
| 5 | 0.0690977 | 0.0690977 |
| 6 | 0 | 0 |

Table 5: Explicit method at time step 24

|  | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.0676569 | 0.0676569 |
| 3 | 0.108421 | 0.108421 |
| 4 | 0.109471 | 0.109471 |
| 5 | 0.0670079 | 0.0670079 |
| 6 | 0 | 0 |

Table 6: Explicit method at time step 25

| | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.0965991 | 0.0965991 |
| 3 | 0.156313 | 0.156313 |
| 4 | 0.156327 | 0.156327 |
| 5 | 0.0966231 | 0.0966231 |
| 6 | 0 | 0 |

Table 7: Implicit method at time step 20

| | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.0909823 | 0.0909823 |
| 3 | 0.14722 | 0.14722 |
| 4 | 0.147228 | 0.147228 |
| 5 | 0.0909965 | 0.0909965 |
| 6 | 0 | 0 |

Table 8: Implicit method at time step 21

| | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.086265 | 0.086265 |
| 3 | 0.139584 | 0.139584 |
| 4 | 0.139589 | 0.139589 |
| 5 | 0.0826733 | 0.0826733 |
| 6 | 0 | 0 |

Table 9: Implicit method at time step 22

| | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | |
| 2 | 0.0823034 | 0.0823034 |
| 3 | 0.133172 | 0.133172 |
| 4 | 0.133175 | 0.133175 |
| 5 | 0.0823083 | 0.0823083 |
| 6 | 0 | 0 |

Table 10: Implicit method at time step 23

| | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.0789766 | 0.0789766 |
| 3 | 0.127788 | 0.127788 |
| 4 | 0.12779 | 0.12779 |
| 5 | 0.0789795 | 0.0789795 |
| 6 | 0 | 0 |

Table 11: Implicit method at time step 24

| | Solution without restart | Solution with restart |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0.0761831 | 0.0761831 |
| 3 | 0.123268 | 0.123268 |
| 4 | 0.123269 | 0.123269 |
| 5 | 0.0761848 | 0.0761848 |
| 6 | 0 | 0 |

Table 12: Implicit method at time step 25

We did test on bigger problems, and the results all show that the restart function of the program gives exactly the same solution as the result of the non-restart program. We would suggest to use HDF5 to record the interation results every 10 or more time steps, especially for big problems. The reason is that for big problems, the simulation time could be pretty long. It is possible that the computer may break down or strop working for some reason. Then we may resume the computation from the last records we recorded. This mechanism may save a lot of time and money. Otherwise, if we are running a relatively small problem, we can run the code directly without saving the middle iteration results, since I/O may consume some time.

## 3.4 Parallellism

### 3.4.1 Explicit Method

Now we run our code in parallel with 16 cores on Ranger. We did the test on explicit method first. We run the problem over a various problem size, and the performace of the code is reported in the table below. Meanwhile, we compare the performance of the parallel/sequential code by looking at the speedup number $S_{16} = T_1/T_{16}$. This number is 16 in theoretical condition. However, in realistic, every code has sequential part which limit the potential speed up.

Here we run our problem with the same heat source function with frequency $l = 1$. The mesh size is $h = 2.0 \times 10^{-6}$ and $k = 5.0 \times 10^{-6}$. The results show that $S_{16} = 8.338$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $3.929 \times 10^2$ sec | $1.965 \times 10^{-3}$ sec | $3.750 \times 10^{10}$ | $9.544 \times 10^7$ |
| 1 core | $3.276 \times 10^3$ sec | $1.64 \times 10^{-2}$ sec | $6.000 \times 10^{11}$ | $1.832 \times 10^8$ |

Second test, mesh size is $h = 1.25 \times 10^{-6}$ and $k = 2.0 \times 10^{-3}$. The results show that $S_{16} = 3.981$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $3.489 \times 10^{0}$ sec | $6.978\times10^{-3}$ sec | $1.504\times10^{8}$ | $4.309 \times10^{7}$ |
| 1 core | $1.389 \times 10^{1}$ sec | $2.778\times10^{-2}$ sec | $2.406\times10^{9}$ | $1.732 \times10^{8}$ |

Third test, mesh size is $h = 1.0 \times 10^{-6}$ and $k = 2.0 \times 10^{-3}$. The results show that $S_{16} = 5.485$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $3.302 \times 10^{0}$ sec | $6.604\times10^{-3}$ sec | $1.879\times10^{8}$ | $5.693 \times10^{7}$ |
| 1 core | $1.811 \times 10^{1}$ sec | $3.622\times10^{-2}$ sec | $3.007\times10^{9}$ | $1.660 \times10^{8}$ |

Forth test, mesh size is $h = 5.56 \times 10^{-7}$ and $k = 5.0 \times 10^{-4}$. The results show that $S_{16} = 6.942$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $1.743 \times 10^{1}$ sec | $8.715\times10^{-3}$ sec | $1.351\times10^{9}$ | $7.773 \times10^{7}$ |
| 1 core | $1.210 \times 10^{2}$ sec | $6.050\times10^{-2}$ sec | $2.161\times10^{10}$ | $1.787 \times10^{8}$ |

Fifth test, mesh size is $h = 5.0 \times 10^{-7}$ and $k = 1.0 \times 10^{-3}$. The results show that $S_{16} = 5.616$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $1.205 \times 10^{1}$ sec | $1.205\times10^{-2}$ sec | $7.509\times10^{8}$ | $7.355 \times10^{7}$ |
| 1 core | $6.767 \times 10^{1}$ sec | $6.767\times10^{-2}$ sec | $1.201\times10^{10}$ | $1.775 \times10^{8}$ |

In our sixth test, we did a small problem: $h = 1.0 \times 10^{-2}$ and $k = 2.0 \times 10^{-2}$. The results show that $S_{16} = 0.323$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $8.139 \times 10^{-2}$ sec | $1.628\times10^{-3}$ sec | $2.149\times10^{3}$ | $2.660 \times10^{4}$ |
| 1 core | $2.628 \times 10^{-2}$ sec | $5.256\times10^{-4}$ sec | $3.080\times10^{4}$ | $1.172 \times10^{6}$ |

### 3.4.2 Implicit Method

Here we test the implicit method on ranger. Parallel and serial code are tested with conjugate gradient iterative solver and jacobi preconditioner. We choose conjugate gradient method because the coefficient matrix is symmetric, which is perfectly suitable for the conjugate gradient method.

we run our problem with the same heat source function with frequency $l = 1$. The mesh size is $h = 1.0 \times 10^{-4}$ and $k = 5.0 \times 10^{-3}$. The results show that $S_{16} = 1.915$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $2.951 \times 10^{2}$ sec | $1.476\times10^{0}$ sec | $1.816\times10^{10}$ | $6.153 \times10^{7}$ |
| 1 core | $5.650 \times 10^{2}$ sec | $2.825\times10^{0}$ sec | $2.900\times10^{11}$ | $5.726 \times10^{8}$ |

For our second test with mesh size is $h = 5.0 \times 10^{-5}$ and $k = 5.0 \times 10^{-3}$. The results show that $S_{16} = 3.310$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $3.931 \times 10^{2}$ sec | $1.966 \times 10^{0}$ sec | $4.525 \times 10^{10}$ | $1.151 \times 10^{8}$ |
| 1 core | $1.301 \times 10^{3}$ sec | $6.505 \times 10^{0}$ sec | $7.236 \times 10^{11}$ | $5.564 \times 10^{8}$ |

For our third test with mesh size is $h = 2.0 \times 10^{-5}$ and $k = 5.0 \times 10^{-3}$. The results show that $S_{16} = 10.741$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $4.753 \times 10^{2}$ sec | $2.377 \times 10^{0}$ sec | $1.131 \times 10^{11}$ | $2.380 \times 10^{8}$ |
| 1 core | $5.105 \times 10^{3}$ sec | $2.552 \times 10^{1}$ sec | $1.809 \times 10^{12}$ | $3.543 \times 10^{8}$ |

For our forth test with mesh size is $h = 1.25 \times 10^{-5}$ and $k = 2.0 \times 10^{-2}$. The results show that $S_{16} = 15.862$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $1.445 \times 10^{2}$ sec | $2.890 \times 10^{0}$ sec | $4.591 \times 10^{10}$ | $3.716 \times 10^{8}$ |
| 1 core | $2.292 \times 10^{3}$ sec | $4.584 \times 10^{1}$ sec | $7.344 \times 10^{11}$ | $3.204 \times 10^{8}$ |

For our fifth test with mesh size is $h = 1.0 \times 10^{-5}$ and $k = 2.0 \times 10^{-2}$. The results show that $S_{16} = 15.947$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $1.883 \times 10^{2}$ sec | $3.766 \times 10^{0}$ sec | $5.738 \times 10^{10}$ | $3.537 \times 10^{8}$ |
| 1 core | $3.003 \times 10^{3}$ sec | $6.006 \times 10^{1}$ sec | $9.180 \times 10^{11}$ | $3.057 \times 10^{8}$ |

In our sixth test, we did a small problem: $h = 1.0 \times 10^{-2}$ and $k = 2.0 \times 10^{-2}$. The results show that $S_{16} = 1.032$.

| | Runtime | Runtime per time step | Flops | Flops/sec |
|---|---|---|---|---|
| 16 cores | $7.335 \times 10^{-2}$ sec | $1.467 \times 10^{-3}$ sec | $4.367 \times 10^{5}$ | $5.955 \times 10^{5}$ |
| 1 core | $7.567 \times 10^{-2}$ sec | $1.513 \times 10^{-3}$ sec | $6.370 \times 10^{6}$ | $8.418 \times 10^{7}$ |

## 4 Appendix

### 4.1 Note

Our simulation is based on the code "heatsolver" written by Ju Liu. This code is written in `C++` with PETSc [?] and HDF5 [?]. Postprocessing of the simulation is done with GNUplot [?]. The version control of the development of the code is done by subversion control system, and the repository is provided by Google Code.

The code is compiled and tested on Ranger at Texas Advanced Computing Center (TACC).

## 4.2 User Manual

We give the manual for the code. Here, to run the code, the user should input, for example,

$ibrun./heatsolver-t2000-x1000-final1.0-l1.0-alpha1.0-exim0.0-matrix0-log\_summary-ksp\_typecg - pc\_typejacobi - save0 - restart0$

Here ibrun is the command in ranger to run MPI executable program. heatsolve is the name of the program.

-t 2000 means there are 2000 elements in t direction;

-x 1000 means there are 1000 elements in x direction;

-final 1.0 means the time domain is [0, 1.0];

-l 1.0 means that the frequency number $l$ in our program is 1.0;

-alpha 1.0 means thermal diffusivity is 1.0;

-exim is the flag for time step scheme. 1.0 means we use explicit time stepping, while other number gives us implicit time step method;

-matrix 0 is the flag for coefficient matrix type. 1.0 means construct it as dense matrix, otherwise sparse matrix;

$-log\_summary$ means we need the basic profile information about the program;

$-ksp\_type$ cg means the iterative solver type. cg means conjugate gradient method;

$-pc\_type$ jacobi means the preconditioner type of the iterative solver. Here in our example, we use jacobi preconditioner;

-save 0 is the flag about if we should save the solution every 10 steps or not. 0 means do not save the iteration every 10 steps, otherwise, we save the iteration results every 10 time steps;

-restart 0 is the flag about if we should restart the code or not. O means we do not restart the program. If we add -restart n, we will restart the program from the time step 10*n. Note, if you want to run the program with restart function, -save should be nonzero, so that there are files saved for restarting;