Parallel resolution of the heat equation

Exam !

Fabrice Deluzet[1] Chang Yang[2]

[1]Université de Toulouse; UPS, INSA, UT1, UTM,
Institut de Mathématiques de Toulouse,
CNRS, Institut de Mathématiques de Toulouse UMR 5219, France

[2]School of Mathematics, Harbin Institute of Technology,
No. 92 West Dazhi Street, Nangang District, 150001 Harbin, China.

June 2023

Implicit Euler Discretization I

The simplest implicit discretization consists in an implicit Euler discretization writing

$$\frac{1}{\Delta t}\left(u_h^{n+1} - u_h^n\right) - \Delta_h u_h^{n+1} = f_h^n \,,$$

which translates into, assuming $f \equiv 0$, and denoting $\mathbb{I}\mathrm{d}$ the identity matrix

$$\left(\mathbb{I}\mathrm{d} - \Delta t \Delta_h\right) u_h^{n+1} = u_h^n \,, \tag{1a}$$

with

$$\begin{aligned} \left(\left(\mathbb{I}\mathrm{d} - \Delta t \Delta_h\right) u_h^{n+1}\right)_{i,j} = u_{j,i}^{n+1}(1+4\lambda) - \lambda \left(u_{j-1,i}^{n+1} + u_{j,i-1}^{n+1}\right) \\ -\lambda \left(u_{j+1,i}^{n+1} + u_{j,i+1}^{n+1}\right) \,, \end{aligned} \tag{1b}$$

$$\lambda = \frac{\Delta t}{h^2} \,. \tag{1c}$$

Implicit Euler Discretization II

The computation of the solution u_h^{n+1} requires the resolution of the linear system

$$\begin{aligned} &\mathbb{A}x = b\,, \\ &\mathbb{A} = \mathbb{I}\mathrm{d} - \Delta t \Delta_h\,, \quad b = u_h^n\,. \end{aligned} \tag{2}$$

This linear system will be solved thanks to the Jacobi method, which amounts to construct a sequence $(x^{(k)})_{k>0}$ converging to x the solution of the system (2). The sequence is defined by

$$x^{(k+1)} = \mathbb{D}^{-1}\,(\mathbb{E} + \mathbb{F})\,x^{(k)} + \mathbb{D}^{-1} b \tag{3}$$

where $\mathbb{A} = \mathbb{D} - (\mathbb{E} + \mathbb{F})$ with

- $\mathbb{D}$ a diagonal matrix composed of the diagonal elements of $\mathbb{A}$;
- $\mathbb{E}$ (resp. $\mathbb{F}$) a lower (resp. upper) triangular diagonal with a zero diagonal.

Implicit Euler Discretization III

Owing to the definition of $\mathbb{A}$ (see Eqs. (1) and (2)), this yields the following recurrence

$$\begin{aligned} x_{j,i}^{(k+1)} &= \frac{\lambda}{1+4\lambda}\left(x_{j-1,i}^{(k)} + x_{j,i-1}^{(k)} + x_{j+1,i}^{(k)} + x_{j,i+1}^{(k)}\right) + \frac{1}{1+4\lambda} b_{j,i}\,, \\ \lambda &= \frac{\Delta t}{h^2}\,, \qquad b_{j,i} = u_{j,i}^{n}\,. \end{aligned} \tag{4}$$

The implicit Euler scheme together with the Jacobi iterations may be decomposed into the steps detailed in Alg. 3.

Jacobi iterations I

Algorithm 3: Jacobi sequence to solve $\mathbb{A}x = b$.

Provided: $\left(x^{(0)}\right)_{(j,i)\in[0,\mathrm{N}+1]^2}$ and $\left(b_{j,i}\right)_{(j,i)\in[1,\mathrm{N}]^2}$;
Tol and λ ;

Residu $\leftarrow$ 1;

$k \leftarrow 0$;

while Residu $>$ Tol **do**

Update $x_{j,i}^{(k+1)}$ for $(j,i) \in [1,\mathrm{N}]^2$ thanks to

$$x_{j,i}^{(k+1)} = \frac{\lambda}{1+4\lambda}\left(x_{j-1,i}^{(k)} + x_{j,i-1}^{(k)} + x_{j+1,i}^{(k)} + x_{j,i+1}^{(k)}\right) + \frac{1}{1+4\lambda} b_{j,i};$$

Residu $\leftarrow \max_{(j,i)\in[1,\mathrm{N}]^2} \left| x_{j,i}^{(k+1)} - x_{j,i}^{(k)} \right|$;

Update interface values of $x^{(k+1)}$;

Reduce Residu over the `MPI` processes;

$x^{(k)} \leftarrow x^{(k+1)}$;

$k \leftarrow k+1$;

end

Implicit Euler scheme I

Algorithm 4: Implicit Euler scheme with Jacobi iterations.

$n = 0$;

Initialize $u^0_{j,i}$ for $(j,i) \in [0, \mathrm{N}{+}1]^2$;

while $n < \mathrm{Nt}$ **do**

  $b \leftarrow u^n$;

  $x^{(0)} \leftarrow u^n$;

  Construct the Jacobi sequence $(x^{(k+1)})_{k \geq 0}$ to solve $\mathbb{A}x = b$ with precision Tol;

  $u^{n+1} \leftarrow x^{(k+1)}$;

  $n \leftarrow n + 1$;

end

Jacobi iterations I

Algorithm 5: Euler Implicit Scheme: Jacobi iterations (first part).

$n = 0$;
Initialize $u^0_{j,i}$ for $(j,i) \in [1,\mathrm{N}]^2$;
Update interface values of u^0;
while $n < \mathrm{Nt}$
  Residu $\leftarrow$ 1.;
  $b \leftarrow u^n$;
  $k \leftarrow 0$;
  $x^{(k)} \leftarrow u^n$;
  while Residu $>$ Tol
    Update $\left(x^{(k+1)}_{j,i}\right)_{(j,i)\in[1,\mathrm{N}]^2}$ thanks to Eq. (4) using $\left(x^{(k)}_{j,i}\right)_{(j,i)\in[0,\mathrm{N}+1]^2}$ and $(b_{j,i})_{(j,i)\in[1,\mathrm{N}]^2}$;
    Residu $\leftarrow \max_{(j,i)\in[1,\mathrm{N}]^2} \left| x^{(k+1)}_{j,i} - x^{(k)}_{j,i} \right|$;

Jacobi iterations II

Algorithm 3: Euler Implicit Scheme: Jacobi iterations (final part).

Update interface values of $x^{(k+1)}$;
Reduce Residu over the `MPI` processes;
$x^{(k)} \leftarrow x^{(k+1)}$;
$k \leftarrow k + 1$;

$u^{n+1} \leftarrow x^{(k+1)}$;
$n \leftarrow n + 1$;

Task 1: Parallel execution, no work-sharing I

In the file `Exam.cpp` implement

1. The launch and the termination of the MPI machinery.
2. The Broadcast of the numerical parameter `Nx`, `Ny`, `Nt`, `StabP` read by one of the `MPI` processes.
3. Check the environment consistency (number of subdomains equal to the number of `MPI` processes).

Task 2: Work-sharing I

In the file `Exam.cpp` implement

1. The definition of the Cartesian communicator `SBD_COMM`.
2. Store the Cartesian coordinates in the array `myCoord` and the rank (relative to `SBD_COMM`) of the neighbors in the `NeighbourRank`. Dump to the disk the files `Proc-X.txt` containing, for each `MPI` process, its rank, coordinates, the rank of its neighbors and their coordinates (see Practice1).
3. The computation of the Global indices of each subdomain nodes, and their physical coordinates. Write to the disk the file `SubDom-X.txt` containing the rank and the coordinates of an `MPI` process together with the range of local coordinates related to the subdomain (See practice 3).
4. Create the `colType` structure to exchange columns of 2D contiguous arrays.

Task 2: Work-sharing II

5. Replace the function `Jacobi` (`HeatUtils.cpp`) by `MPIJacobi` (`Interfaces.cpp`).
6. Free the memory related to `colType`.
7. In the file `Interfaces.cpp` the function

```
void MPIJacobi(double **x, double **x0, double **b, double &Residu, double Tol, int &k, int Nx, int Ny, double lambda, int NeighbourRank[], MPI_Comm SBD_COMM, MPI_Datatype colType, int myRank);
```

implements the Jacobi iterations as defined by Eq. (4) with a precision prescribed by `Tol`.
Using the function `Interfaces`, implement the update of the interface node values.

Task 2: Work-sharing III

8. Sanity checks: Run the implicit solver on 9 `MPI` processes with `N=50`, `Tol=1.e-6`, `StabP=100`, `Nt=5` and compare the outputs of your application with those below.

```
 Numerical Approximation: [Euler Implicit]
  ## Tol=1e-06 StabP=10
  ** N=50 N_tot=150
  ** h=0.00662252 Dt=0.000438577
  ** T=0.00219289 T/Dt=5
     n=0 Residu=9.85529e-07 k=250
     n=1 Residu=9.89344e-07 k=249
     n=2 Residu=9.93175e-07 k=248
     n=3 Residu=9.97022e-07 k=247
     n=4 Residu=9.75944e-07 k=247
  ** Error L2-norm: 0.00061681
  ** Error Li-norm: 0.00123349
```

Bonus questions

1. Task 3: Implement a convergence criteria with a global residual rather than the local residual;
2. Implement a non-blocking persistent communication protocol within the Jacobi iterations (see Practice 4).