Parallel Scientific Computation

Parallelization of Finite Difference Method

J.-H. Parq
IPCST
Seoul National University

FDM for Elliptic PDEs

- Stencil + linear algebra
  - Stencil: 5-point, 7-point, ……
  - Linear algebra: direct or iterative
    - Direct: Gaussian elimination, LU factorization, ……
    - Iterative: steepest descent, conjugate gradient, Newton-Krylov, ……
- Iterative FDM
  - Jacobi, Gauss-Seidel, SOR, ……

FDM for 1-D Elliptic Model

- Poisson eq. $\triangle u = f(x) \quad \rightarrow \quad u_{xx} = f(x)$

  boundary condition: $u(a) = u(b) = 0$

  – A uniform grid by dividing the x-axis line

  $x_0 = a < x_1 < \ldots < x_{N-1} < x_N = b$

  $x_i = x_0 + ih \quad (h = \Delta x), \qquad 0 \leq i \leq N$

  $u(x_0) = u(x_N) = 0$

  – Applying the central 3-point second derivative finite difference to u_{xx},

  $u(x_{i-1}) - 2u(x_i) + u(x_{i+1}) = h^2 \cdot f(x_i)$

FDM for 1-D Elliptic Model

- Poisson eq. $\triangle u = f(x) \quad \rightarrow \quad u_{xx} = f(x)$
  - After applying the boundary condition, one can obtain a matrix for $u(x_1)$, $u(x_2)$, … , $u(x_{N\text{-}1})$

$$h^2 \triangle_h u = \begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{pmatrix}$$

$$\triangle u = f \quad \rightarrow \quad \boldsymbol{Au} = \boldsymbol{f} \quad (\boldsymbol{A} = \triangle_h u)$$

FDM for 1-D Elliptic Model

- Poisson eq. $\triangle u = f(x) \quad \rightarrow \quad u_{xx} = f(x)$

  $\triangle u = f \quad \rightarrow \quad \boldsymbol{Au} = \boldsymbol{f} \quad (\boldsymbol{A} = \triangle_h u)$

  – The same form is available for the boundary condition: $u(a) = \alpha$, $u(b) = \beta$

    by $f_1 \rightarrow f_1 - \alpha/h^2$ and $f_{N-1} \rightarrow f_{N-1} - \beta/h^2$

  – Truncation error at point x_i: $\tau_i \approx (1/12)h^2 u_{xxxx}$

  – Matrix analysis can prove convergence with order 2.

Five-point Stencil (2D)

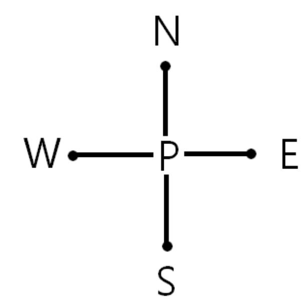
- 5-point discrete Laplacian (2D)

  $\triangle u = u_{xx} + u_{yy}$

  – Let $U_{i,j} = u(x_i, y_j)$ where $x_i = ih$, $y_j = jh$

  $$u_{xx,\,h} = (U_{i-1,j} - 2U_{i,j} + U_{i+1,j})/h^2$$

  $$u_{yy,\,h} = (U_{i,j-1} - 2U_{i,j} + U_{i,j+1})/h^2$$



$$\triangle_h u = u_{xx,\,h} + u_{yy,\,h}$$

$$= (U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j})/h^2$$

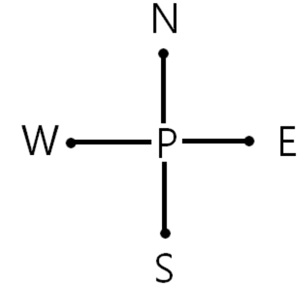
Five-point Stencil (2D)

- 5-point discrete Laplacian (2D)

$$\triangle_h u = (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j})/h^2$$

  - Local truncation error

    $$\tau_{i,j} = (u_{xxxx} + u_{yyyy})h^2/12 + O(h^4)$$



  - Convergent with order 2 for the equation

    $\triangle u = f(x)$

    $$(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j})/h^2 = f_{i,j}$$

Five-point Stencil (2D)

$$u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j} = h^2 f_{i,j}$$

- Applying boundary conditions
  - Dirichlet B.C.: replacement by values
    - Ex.) $u(x,0) = g(x) \rightarrow u_{i,0} = g_i$

    → $u_{i-1,1} + u_{i+1,1} + u_{i,2} - 4u_{i,1} = h^2 f_{i,1} - g_i$
  - Neumann B.C.: replacement by equations
    - Forward FD ex.) $u_x(0,y) = g(y) \rightarrow$ $-3u_{0,j} + 4u_{1,j} - u_{2,j} = 2hg_j$

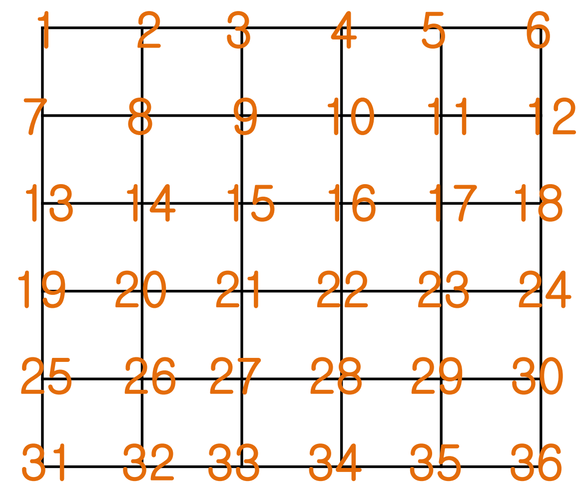
    → $(2/3)u_{2,j} + u_{1,j-1} + u_{1,j+1} - (8/3)u_{1,j} = h^2 f_{i,j} + (2/3)hg_j$
    - Ghost boundary ex.) $u_x(0,y) = g(y) \rightarrow u_{-1,j} - u_{1,j} = 2hg_j$

    → $2u_{1,j} + u_{0,j-1} + u_{0,j+1} - 4u_{0,j} = h^2 f_{i,j} + 2hg_j$

    ✓ Matrix elements for $u_{0,j}$ are needed.

Five-point Stencil (2D)

- Matrix representation
  - Usual ordering →
    - $\boldsymbol{A}$ (= $\triangle_h u$): $m^2 \times m^2$ matrix



m = 6 case

$$\frac{1}{h^2}\left[\begin{array}{c|c|c|c} T & I & 0 & 0 \\ \hline I & T & \ddots & 0 \\ \hline 0 & \ddots & \ddots & I \\ \hline 0 & 0 & I & T \end{array}\right]$$

$$I = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix} \quad T = \begin{bmatrix} -4 & 1 & & 0 \\ 1 & -4 & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -4 \end{bmatrix}$$

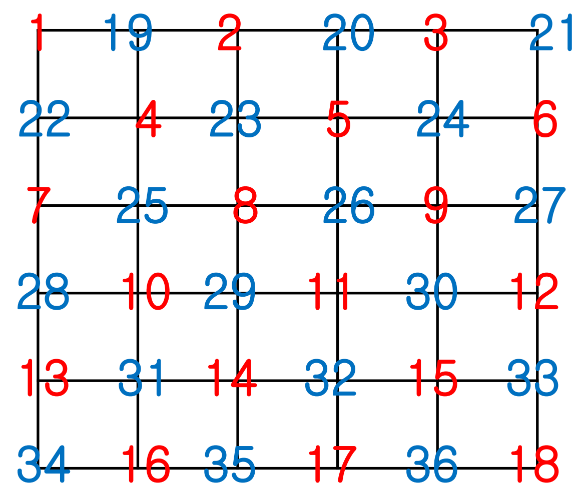
    - $\boldsymbol{T,I}$: $m \times m$ matrices

❖ Ill-conditioned for iterative linear algebra methods

Five-point Stencil (2D)

- Matrix representation
  - Alternative ordering →
    - **$\boldsymbol{A}$** (= $\triangle_h u$): $m^2 \times m^2$ matrix

$$\frac{1}{h^2}\left[\begin{array}{c|c} D & H \\ \hline H^T & D \end{array}\right]$$



$$H = \begin{bmatrix} 1 & \cdot & 0 & \cdot & 1 & & 0 \\ 1 & \ddots & & & & \ddots & \\ \vdots & & & & & & 1 \\ 0 & \ddots & & \ddots & & & \vdots \\ \vdots & & & & & & 0 \\ 1 & & & \ddots & & \ddots & \vdots \\ & \ddots & & & & & \\ 0 & & 1 & 0 & & 1 & 1 \end{bmatrix}$$

    - **$\boldsymbol{D}$** = $-4\boldsymbol{I}$
    - **$\boldsymbol{D}$,$\boldsymbol{H}$** : $(m^2/2)\times(m^2/2)$ matrices

Parallel Linear Algebra Methods

- Parallel libraries
  - OpenMP: Some BLAS or LAPACK libraries provide runtime routines or environment variables for OpenMP (ex. MKL)
  - MPI: PBLAS+ScaLAPACK, PLAPACK, Elemental (C or Python. MPI-2)
  - CUDA: cuBLAS, cuSPARSE, cuSolver, ……
  - OpenCL: clBLAS, clBLAST
  - MPI+GPU: LAMA (C++)
  - Many-core CPU: PLASMA, MAGMA

Iterative Methods for Elliptic PDEs

- For the 2-D equation $\triangle u = f(x)$
  - Jacobi

$$u_{i,j}^{k+1} = (u_{i-1,j}^{k} + u_{i+1,j}^{k} + u_{i,j-1}^{k} + u_{i,j+1}^{k} - h^2 f_{i,j})/4$$

    - This can be derived from the 5-point stencil.
  - Gauss-Seidel

$$u_{i,j}^{k+1} = (u_{i-1,j}^{k+1} + u_{i+1,j}^{k} + u_{i,j-1}^{k+1} + u_{i,j+1}^{k} - h^2 f_{i,j})/4$$

    - Twice faster than Jacobi for serial computing

❖ Computational time ~ $O(m^4 \log m)$ for serial comput.

➢ $O(m^2)$ per each iteration × $O(m^2 \log m)$ iterations

Iterative Methods for Elliptic PDEs

$$u_{i,j}^{k+1} = (u_{i-1,j}^{k} + u_{i+1,j}^{k} + u_{i,j-1}^{k} + u_{i,j+1}^{k} - h^2 f_{i,j})/4$$

- Applying boundary conditions (in Jacobi)
  - Dirichlet B.C.: replacement by values
    - Ex.) $u(x,0) = g(x) \rightarrow u_{i,0} = g_i$

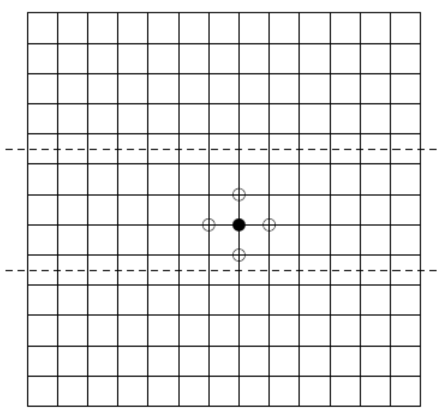
    → $u_{i,j}^{k+1} = (u_{i-1,1}^{k} + u_{i+1,1}^{k} + u_{i,2}^{k} - h^2 f_{i,1} + g_i)/4$
  - Neumann B.C.: replacement by equations
    - Forward FD ex.) $u_x(0,y) = g(y) \rightarrow -3u_{0,j} + 4u_{1,j} - u_{2,j} = 2hg_j$

    → $u_{1,j}^{k+1} = (2u_{2,j}^{k} + 3u_{1,j-1}^{k} + 3u_{1,j+1}^{k} - 3h^2 f_{i,j} - 2hg_j)/8$
    - Ghost boundary ex.) $u_x(0,y) = g(y) \rightarrow u_{-1,j} - u_{1,j} = 2hg_j$

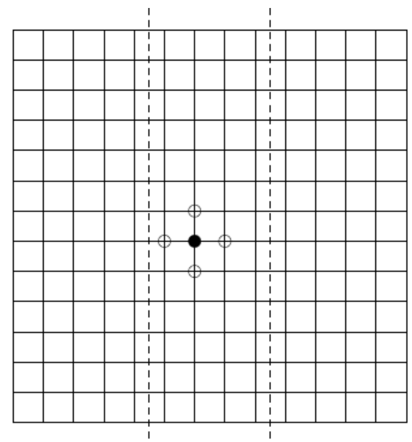
    → $u_{0,j}^{k+1} = (2u_{1,j}^{k} + u_{0,j-1}^{k} + u_{0,j+1}^{k} - h^2 f_{i,j} - 2hg_j)/4$

Iterative Methods for Elliptic PDEs

- Jacobi with MPI
  - Cartesian communicators are useful.
  - Space decomposition



or

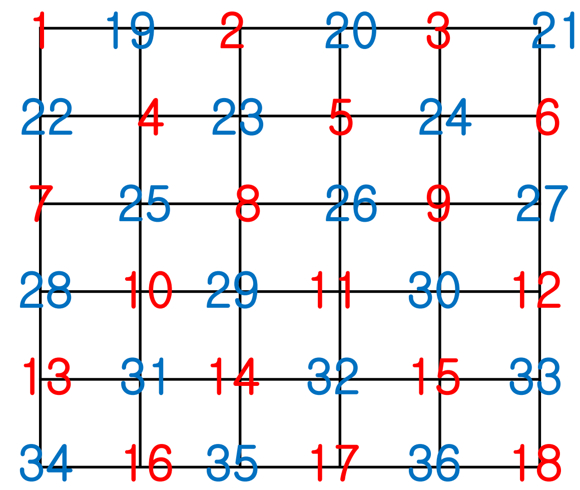


  - Exchange borders: sendrecv or non-blocking
  - Error estimation: collective operation

Figure from Baolai

Iterative Methods for Elliptic PDEs

- Parallelized Gauss-Seidel
  - Change the order of updates
  - Then do Jacobi



    - So-called 'red-black' Gauss-Seidel

  - Reduction should be necessary to estimate errors.

FDM for Parabolic PDE

- Method of lines
- Forward Euler

Parallelization: You can apply ways similar to those for Jacobi (space decomposition)

- Backward Euler → Parallelization: Linear algebra or transformation to Jacobi

- Alternate Direction Implicit method

  → Parallelization: Shared-memory parallelism (OpenMP or CUDA)

- Crank-Nicolson method

  → Parallelization: Linear algebra or transformation to Jacobi

Crank-Nicolson method

  - Based on the trapezoidal rule (implicit method)
- If a PDE has the form of

  $\partial_t u = f(u, x, y, t, \partial_x u, \partial_y u, \partial^2{}_x u, \partial^2{}_y u)$

  - By discretization, Crank-Nicolson method gives

  $(U_i^{n+1} - U_i^n)/\delta = (F_i^n + F_i^{n+1})/2$

  where $U_i^n = u(x_i, t_n)$ and F_i^n : value of f at t_n and x_i $\delta = \Delta t$

  - Time: 2-point. Space: any finite difference
- Unconditional stability and 2nd order accuracy
- Used for
  - Parabolic PDEs and advection equations

Crank-Nicolson method

- Ex.) Heat equation $u_t = \triangle u$

$$\frac{u(\vec{x},t+\delta)-u(\vec{x},t)}{\delta} = \frac{\triangle_h u(\vec{x},t+\delta)+\triangle_h u(\vec{x},t)}{2}$$

  – For the 2-D case,

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \frac{1}{2}\frac{\delta}{h^2}\left[\left(u_{i+1,j}^{n+1}+u_{i-1,j}^{n+1}+u_{i,j+1}^{n+1}+u_{i,j-1}^{n+1}-4u_{i,j}^{n+1}\right)\right.$$
$$\left.+\left(u_{i+1,j}^{n}+u_{i-1,j}^{n}+u_{i,j+1}^{n}+u_{i,j-1}^{n}-4u_{i,j}^{n}\right)\right]$$

$$\rightarrow \quad (1+2\mu)u_{i,j}^{n+1}-\frac{\mu}{2}\left(u_{i+1,j}^{n+1}+u_{i-1,j}^{n+1}+u_{i,j+1}^{n+1}+u_{i,j-1}^{n+1}\right)$$
$$=(1-2\mu)u_{i,j}^{n}+\frac{\mu}{2}\left(u_{i+1,j}^{n}+u_{i-1,j}^{n}+u_{i,j+1}^{n}+u_{i,j-1}^{n}\right).$$

where $\mu = \delta/h^2$

    - matrix form: $(\boldsymbol{I}+\boldsymbol{C})\,\boldsymbol{u}^{n+1} = (\boldsymbol{I}-\boldsymbol{C})\,\boldsymbol{u}^{n}$

Crank-Nicolson method

- Parallelization
  - It is possible to apply Jacobi
    - Ex.)

$$(1+2\mu)\, u_{i,j}^{n+1}[k+1] = (1-2\mu)\, u_{i,j}^{n} + \{F^n - F^{n+1}[k]\}\, \mu/2$$

where $F^n = u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}$

(k : Jacobi iteration number)

    - Tips for parallelization for Jacobi are applicable.

Advection equation

$$\frac{\partial \psi}{\partial t} + \nabla \cdot (\psi \mathbf{u}) = 0$$

- For incompressible flows ($\nabla \cdot \mathrm{u} = 0$),

$$\frac{\partial \psi}{\partial t} + \mathbf{u} \cdot \nabla \psi = 0.$$

- The simplest 1-D case

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0$$

a: constant

$$\left(\frac{\partial}{\partial t} - a\frac{\partial}{\partial x}\right)\left(\frac{\partial}{\partial t} + a\frac{\partial}{\partial x}\right)u = \left(\frac{\partial^2}{\partial t^2} - a^2\frac{\partial^2}{\partial x^2}\right)u = 0$$

FDM for advection equations

- Implicit methods
  - Backward central
  - Crank-Nicolson

Parallelization: Linear algebra or transformation to Jacobi

- Explicit methods
  - Upwind methods
  - Lax-Friedrichs
  - Leapfrog
  - Lax-Wendroff

Parallelization: You can apply ways similar to those for Jacobi (space decomposition)

Convection-diffusion equation

- Also known as *advection-diffusion equation*

- $u_t + au_x = Du_{xx}$
  - Various methods are applicable.
  - Even forward time central space (FTCS) FDM is possible. (The same stable condition with forward Euler for diffusion equation)
  - You can also apply the method of lines.
  - Parallelization is similar to that of advection equation or diffusion equation.

FDM for wave equations

- A PDE like $\partial^2{}_t u = a^2 \triangle u$ can become a system of 1st order PDEs with auxiliary variables.

  → Apply FDM for advection equations
- System of equations
  - For example, $q = a u_x$, $r = a u_y$ & $s = u_t$

  $\partial^2{}_t u = a^2 \triangle u$ →
  $$q_t = a s_x$$
  $$r_t = a s_y$$
  $$s_t = a(q_x + r_y)$$

FDM for wave equations

- Alternative way
  - Centered second order time difference

$$\frac{\partial^2 u}{\partial t^2} \rightarrow \frac{u(\vec{x}, t+\delta) - 2u(\vec{x}, t) + u(\vec{x}, t-\delta)}{\delta^2}$$

Ex.) $\partial^2{}_t u = a^2 \triangle u$

$\rightarrow u(\vec{x}, t+\delta) = 2u(\vec{x}, t) - u(\vec{x}, t-\delta) + a^2 \delta^2 \triangle_h u(\vec{x}, t)$

  - Parallelization is similar to that of advection equation.

In CUDA

- For iterative FDM or time-dependent FDM,
  - At least 2 arrays are necessary: the current state and the next state
    - SIMT techniques: similar to those of OpenMP
  - If you have two or more GPUs,
    - Space decomposition: overlapping regions at borders → exchange data
    - Techniques similar to those of MPI

Domain Decomposition

- BVP on domain → BVPs on subdomains
  - In addition to boundary conditions, we need conditions at interfaces or in overlapping regions
- Usefulness
  1. Efficient parallel computing
  2. It is often useful to use different time steps or grids on different subdomains.

Domain Decomposition

- Overlap conditions
  1. Subdomains overlap
  2. Subdomains do not overlap, but they are appended with buffer regions
  3. Without buffer regions, subdomains intersect only along an interface

Domain Decomposition

- Simple example: FDM of 1-D heat eq.
    - $\partial_t u = \partial^2{}_x u$
    - $u(x\,, 0) = f(x)$; $u(0, t) = u(1, t) = 0$
  - Let $U_i^n \equiv u(x_i\,, t_n)$ where $x_i = ih$, $t_n = n\delta$
  - Assume each subdomain ranges from one interface point to the next interface point. Then,
    - $U_i^n = 0$ at boundary points
    - $\partial_{t,\,\delta} U_i^n = \partial^2{}_{x,\,h}\, U_i^{n\text{-}1}$ at interface points
    - $\partial_{t,\,\delta} U_i^n = \partial^2{}_{x,\,h}\, U_i^n$ at interior points

Domain Decomposition

- Simple example: FDM of 1-D heat eq.
    - $U_i^n = 0$ at boundary points
    - $\partial_{t,\delta} U_i^n = \partial^2_{x,h} U_i^{n-1}$ at interface points
    - $\partial_{t,\delta} U_i^n = \partial^2_{x,h} U_i^n$ at interior points

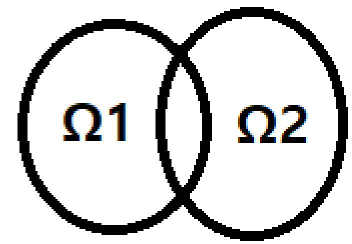
  where $\partial_{t,\delta} U_i^n = (U_i^n - U_i^{n-1})/\delta$,

  $$\partial^2_{x,h} U_i^n = (U_{i-1}^n - 2U_i^n + U_{i+1}^n)/h^2$$

  – Explicit for interface and implicit for interior
    - After computing the interface values, the interior values in each subdomain are computed.

Domain Decomposition

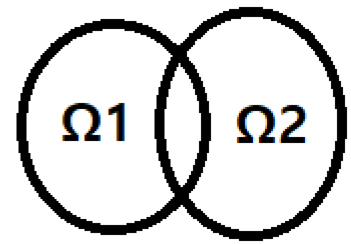
- Overlapping domain cases
    - $\Omega = \Omega_1 \cup \Omega_2$



  – Original Schwarz iteration
    - PDE → $\mathbf{Au} = \mathbf{b}$ form
    - Supposing $\mathbf{Au}_1{}^k = \mathbf{b}$ & $\mathbf{Au}_2{}^k = \mathbf{b}$,
    - Solve $\mathbf{Au}_1{}^{k+1} = \mathbf{b}$ & $\mathbf{Au}_2{}^{k+1} = \mathbf{b}$ under the B. C. $\mathbf{u}_1{}^{k+1} = \mathbf{u}_2{}^k$ on $\partial\Omega_1 \cap \Omega_2$ and $\mathbf{u}_2{}^{k+1} = \mathbf{u}_1{}^{k+1}$ on $\partial\Omega_2 \cap \Omega_1$
    - Convergence depends on boundary conditions and size of the overlapping region

Domain Decomposition

- Overlapping domain cases
    - $\Omega = \Omega_1 \cup \Omega_2$
  - Jacobi Schwarz method
    - PDE → $\mathbf{Au} = \mathbf{b}$ form
    - Supposing $\mathbf{Au}_i^{\,k} = \mathbf{b}$,
    - Solve $\mathbf{Au}_i^{\,k+1} = \mathbf{b}$ under the B. C. $\mathbf{u}_i^{\,k+1} = \mathbf{u}_{3-i}^{\,k}$ on $\partial\Omega_i \cap \Omega_{3-i}$



❖ See also additive Schwarz method

Spectral Method

- Fourier transform of spatial derivatives

  $$\mathrm{FFT}(\partial^n u/\partial x^n) = (ik)^n\,\mathrm{FFT}(u)$$

- This makes some finite difference equations easier – spectral equations

- Parallelization
  - Collective operations are sufficient for error estimation.
  - FFT can be parallelized by parallel FFT libraries or slab decomposition or domain decomposition.

References

- W. Gropp, E. Lusk, and A. Skjellum, “Using MPI”
- R. J. LeVeque, “Finite Difference Methods for Ordinary and Partial Differential Equations”
  - Steady-state and Time-dependent Problems
- G. Baolai, “Parallel Numerical Solution of PDEs with Message Passing”

References

- C. Douglas, G. Haase, & U. Langer, "A Tutorial on Elliptic PDE Solvers and their Parallelization"
- Z. Wei *et al.*, "Parallelizing Alternating Direction Implicit Solver on GPUs", Procedia Comput. Sci. 18, 389 (2013).
- J. Furumura, "Large-scale parallel simulation of seismic wave propagation and strong ground motions for the past and future earthquakes in Japan", J. Earth Simulator 3, 29 (2005).

References

- T. F. Chan & T. P. Mathew, “Domain decomposition algorithms”, Acta Numerica 3, 61 (1994).

- A. Quarteroni & A. Valli, “Domain Decomposition Methods for Partial Differential Equations”

- B. F. Smith, “Domain decomposition methods for partial differential equations”, Parallel Numerical Algorithms, pp. 225-243 (1997)

References

- B. F. Smith, P. E. Bjørstad, & W. D. Gropp, "Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations"

- A. Toselli & O. Widlund, "Domain Decomposition Methods: Algorithms and Theory"

- V. Dolean, P. Jolivet, F. Nataf, "An Introduction to Domain Decomposition Methods: algorithms, theory and parallel implementation"

References

- M. Kaiho *et al.*, "Parallel overlapping scheme for viscous incompressible flows", Int. J. Numer. Methods Eng. 24, 1341 (1997).

- G. Chen *et al.*, "Parallel Spectral Numerical Methods"

- J. Cheng, M. Grossman, and T. McKercher, "Professional CUDA C Programming"

- C. Moler, "Numerical Computing with MATLAB"