Algorithms, Implementation and Applications of pFFT++: Overview

Zhenhai Zhu
RLE Computational prototyping group, MIT
www.mit.edu/people/zhzhu/pfft.html

Outline
- Brief introduction to fast IE solver
- FFT-based methods
- What pFFT++ does
- Project hierarchy of pFFT++
- Main classes of pFFT++
- User interface of pFFT++

Integral Equation Method

A simple integral equation:
\[ \int dS K(\vec{r}, \vec{r}') \rho(\vec{r}') = f(\vec{r}), \quad \vec{r} \in S \]
\[ K(\vec{r}, \vec{r}') = \frac{1}{|\vec{r} - \vec{r}'|} e^{-\frac{|\vec{r} - \vec{r}'|}{a}} \]

Project the solution on a functional space:
\[ \rho_{\chi}(\vec{r}') = \sum_j \alpha_j \beta_j(\vec{r}') \quad B_\chi = \text{span}\{\beta_j(\vec{r}')\} \]

 Integral Equation Method

Residual:
\[ e_{\chi}(\vec{r}) = \int dS K(\vec{r}, \vec{r}') \rho_{\chi}(\vec{r}') - f(\vec{r}) \]

Enforce the residual to be orthogonal to another functional space:
\[ \langle \hat{\iota}(\vec{r}), e_{\chi}(\vec{r}) \rangle = 0, \quad T_\chi = \text{span}\{\hat{\iota}(\vec{r})\} \]

A dense linear system:
\[ A\alpha = \tilde{f} \]

Some very useful applications
- Electrostatic analysis to compute the capacitance
- Magneto-quasi-static analysis to compute impedance
- EMQS analysis: coupling and resonance
- Fullwave analysis: radiation

Figures thanks to Coventor
Some very useful applications

Computational Aerodynamics
Stokes Flow Solver
Viscous drag

Recipe for A Fast Integral Equation Solver

- An iterative solver
  - no dense LU factorization \(O(N^3)\)
- A pre-conditioner (A sparse matrix solver)
  - Minimize number of iterations
- A matrix vector product accelerator
  - avoid filling the whole matrix which needs \(O(N^2)\) memory and \(O(N^2)\) CPU time

Fast Matrix-Vector Product

The most expensive step:

\[ Ax \]

Goal:

\[ O(N^2) \Rightarrow O(N) \text{ or } O(N \log(N)) \]

Well-known Fast Algorithms

- Fast Multiple Method
- Hierarchical SVD
- Panel Clustering Method

Key idea:
interaction matrix is low rank

Kernel “Independent” Technique

Basic requirements:
- Reciprocity: \(G(\vec{r}, \vec{r}') = G(\vec{r}', \vec{r})\)
- Shift invariance: \(G(\vec{r} + \vec{a}, \vec{r}' + \vec{a}) = G(\vec{r}, \vec{r}')\)

Commonly used Green’s function all satisfy these requirements

\[
\frac{1}{|\vec{r} - \vec{r}'|} \frac{e^{j|\vec{r} - \vec{r}'|}}{|\vec{r} - \vec{r}'|} \frac{\partial}{\partial n} \left( \frac{1}{|\vec{r} - \vec{r}'|} \right), \quad \frac{\partial}{\partial n} \left( \frac{e^{j|\vec{r} - \vec{r}'|}}{|\vec{r} - \vec{r}'|} \right)
\]

FFT-based Method

Key idea: kernel is shift-invariant

\[ G(\vec{r}, \vec{r}') = G(\vec{r} - \vec{r}', 0) = G(\vec{r} - \vec{r}') \]

A simple example:

\[
\int d S \, G(\vec{r}, \vec{r}') p(\vec{r}') = f(\vec{r}), \quad \vec{r} \in S
\]

\(H\vec{x} = \vec{f}\)
FFT-based Method

If collocation method with constant basis is used and all panels are identical

\[ H_{i,j} = \int_{\text{panel}_i} dS G(\vec{r}_i - \vec{r}_j) \]

Only \( H_{i,j} \) \((j = 1, 2, ..., N)\) are unique. \( H \) is a Toeplitz matrix. Matrix vector product could be computed using FFT in \( O(N \log(N)) \) time.

**Operations:** \( O(N \log(N)) \) **Memory:** \( O(N) \)

---

pFFT Algorithm: Basic steps

1. Project: \( \bar{Q}_j = [P] \bar{\mathbf{x}} \)
2. Convolve: \( \bar{\phi}_k = [H] \bar{Q}_j \)
3. Interpolate: \( \Psi_p = [I] \bar{\phi}_k \)
4. Direct: \( \Psi_d = [D] \bar{\mathbf{x}} \)

\[ \Psi = \Psi_p + \Psi_d = ([D] + [I] [H] [P]) \bar{\mathbf{x}} \]

---

Application: FastImp

16 x 8 3-turn spiral array

180k panels, 1.44 million unknowns, grid 256 x 128 x 8

FastImp: 11.7 hours, 11.9 Gb

---

Application: FastImp

MIT logo with 123 3-turn spirals

173k panels, 1.38 million unknowns, grid 1024 x 256 x 8

FastImp: 14.2 hours, 11.8 Gb
Breakdown of CPU time (seconds)

<table>
<thead>
<tr>
<th>Component</th>
<th>MIT logo</th>
<th>16x8 array</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$ and $I$ matrices</td>
<td>890</td>
<td>746</td>
</tr>
<tr>
<td>$D$ and $H$ matrices</td>
<td>13638</td>
<td>14353</td>
</tr>
<tr>
<td>Form the preconditioner $P_c$</td>
<td>54</td>
<td>53</td>
</tr>
<tr>
<td>LU factorization of $P_c$</td>
<td>1512</td>
<td>1927</td>
</tr>
<tr>
<td>GMRES ($tol=1e^{-3}$)</td>
<td>32424 (77 iter)</td>
<td>25168 (80 iter)</td>
</tr>
<tr>
<td>total</td>
<td>51244</td>
<td>42247</td>
</tr>
</tbody>
</table>

Notice the difference in GMRES is only about 25%, small considering the grid size is different by a factor of 8.

Breakdown of Memory (Mb)

<table>
<thead>
<tr>
<th>Component</th>
<th>MIT logo</th>
<th>16x8 array</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct matrix</td>
<td>5.17</td>
<td>5.54</td>
</tr>
<tr>
<td>Projection matrix</td>
<td>0.38</td>
<td>0.39</td>
</tr>
<tr>
<td>Interpolation matrix</td>
<td>0.22</td>
<td>0.23</td>
</tr>
<tr>
<td>Convolution matrix</td>
<td>0.68</td>
<td>0.13</td>
</tr>
<tr>
<td>Maps between grids and panels</td>
<td>0.65</td>
<td>0.70</td>
</tr>
<tr>
<td>Pre-conditioner</td>
<td>2.72</td>
<td>2.76</td>
</tr>
<tr>
<td>GMRES</td>
<td>2.03</td>
<td>2.21</td>
</tr>
<tr>
<td>total</td>
<td>11.85</td>
<td>11.96</td>
</tr>
</tbody>
</table>

Notice the difference in convolution matrix is consistent with the difference in grid size.

---

Project hierarchy of pFFT++

- **pfft++**
  - **src**
    - Source codes
  - **inc**
    - Interface header file, template codes
  - **test**
    - Example drivers
  - **lib**
    - Library binary pfft.o, clapack.a
  - **doc**
    - Documentations

---

Main classes of pFFT++

- **Pfft**
  - **Level 1**
    - Grid
  - **Level 2**
    - Stencil
    - InterpMat
    - SpRowMat
  - **Level 3**
    - SpColMat
    - ProjectMat
    - DirectMat
    - GridData
    - Element
    - Fast3DCase
    - FFTW
    - GridElement
  - **Level 4**
    - TNT::Matrix, Vector
    - Cubature
    - clapack
    - SpVec
    - SpVecElement

---

Accessory classes of pFFT++

- Discretization
  - element
- Kernels
  - EikOverR
  - OneOverR
  - EkrOverR
- Panel integration
  - StaticCollocation
  - FullwaveCollocation
- Iterative solver
  - gmres
- Pre-conditioner
  - SuperLU

---

Spare matrix classes

Sparse matrix is extensively used in pFFT++. It is one of the building blocks.

- Compressed Column format
  - see spColMat.h
- Compressed row format
  - see spRowMat.h
User interface of pFFT++

- See pfft++/test/driver1.cc
- See pfft++/test/driver2.cc
- See pfft++/test/driver3.cc
- Demo of three drivers

Today’s Goals

- Download
- Compile
- Run three drivers
- Write a simple driver of your own for a two kernel case

\[
\int dS \left[ G(\vec{r}, \vec{r}') p(\vec{r}') + \frac{dG(\vec{r}, \vec{r}')}{dn(\vec{r}') - \sigma(\vec{r}')} \right]
\]

Next

- Algorithms: Projection and Interpolation
- Implementation: projectMat.h and interpMat.h