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:

$$\int_S dS' K(\vec{r}, \vec{r}')\, r(\vec{r}') = f(\vec{r}), \quad \vec{r} \in S$$

$$K(\vec{r}, \vec{r}') = \frac{1}{|\vec{r} - \vec{r}'|}, \qquad \frac{e^{ik|\vec{r} - \vec{r}'|}}{|\vec{r} - \vec{r}'|}$$

Project the solution on a functional space:

$$r_n(\vec{r}') = \sum_{j=1}^{n} a_j b_j(\vec{r}'), \quad B_n = \text{span}\langle b_j(\vec{r}')\rangle$$

Integral Equation Method

Residual:

$$\mathrm{e}_{\mathrm{n}}(\vec{r}) = \int_S dS' K(\vec{r},\vec{r}')\, r_n(\vec{r}') - f(\vec{r})$$

Enforce the residual to be orthogonal to another functional space:

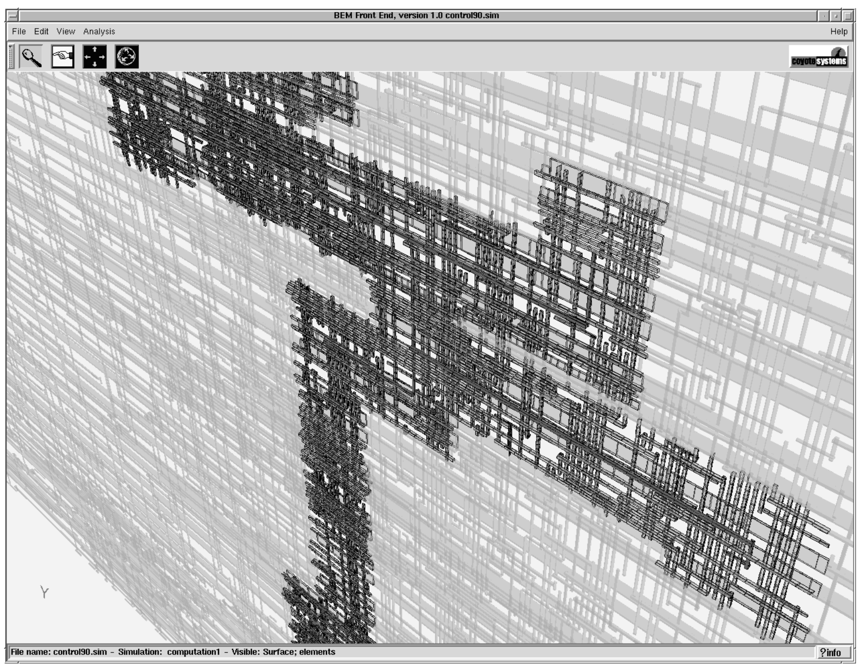
$$\left\langle t_i(\vec{r}), \mathrm{e}_{\mathrm{n}}(\vec{r}) \right\rangle = 0,\ T_n = \mathrm{span}\left\langle t_i(\vec{r}) \right\rangle$$

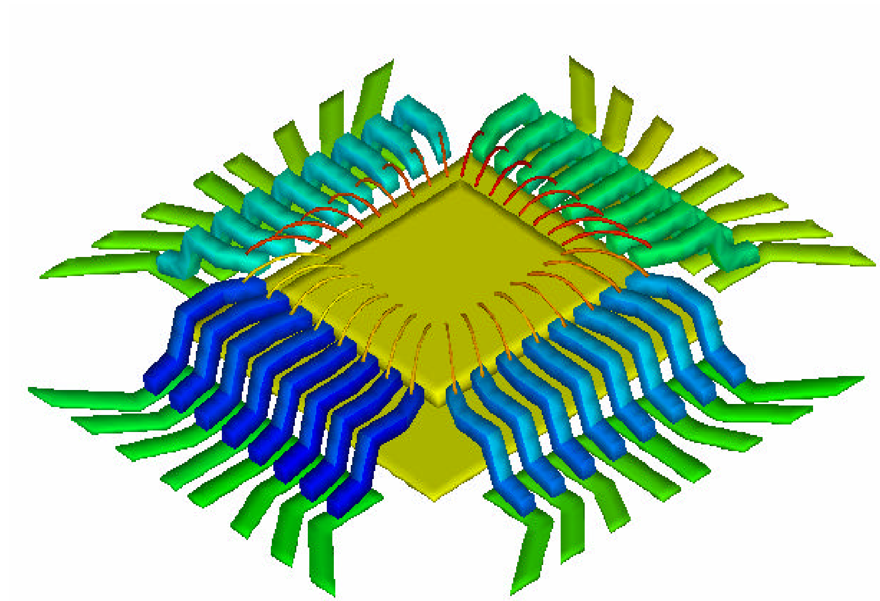
A dense linear system: $A\overline{a} = \bar{f}$

Some very useful applications

Electrostatic analysis to compute the capacitance

Magneto-quasi-static analysis to compute impedance



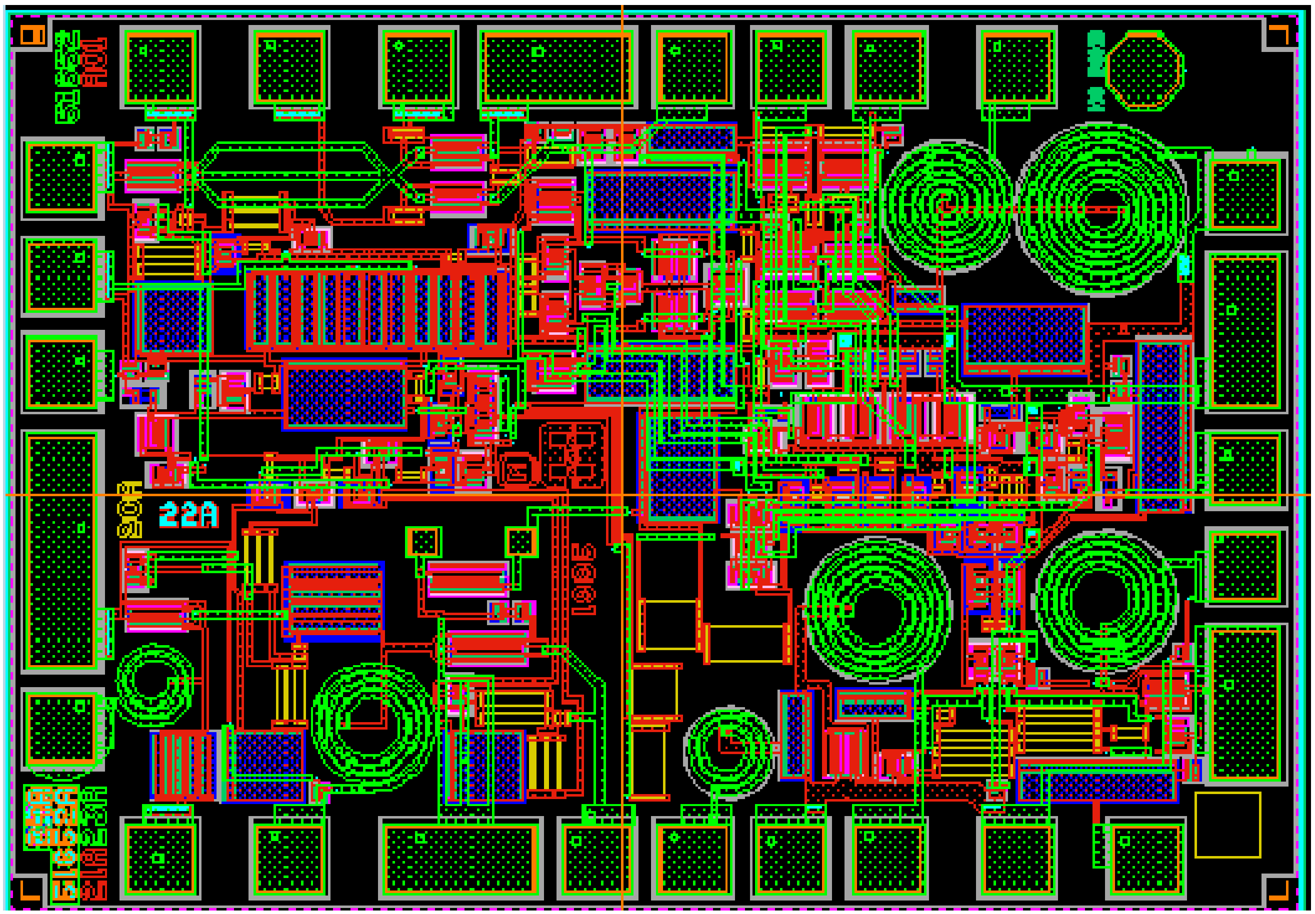


Figures thank to Coventor

Some very useful applications

EMQS analysis: coupling and resonance

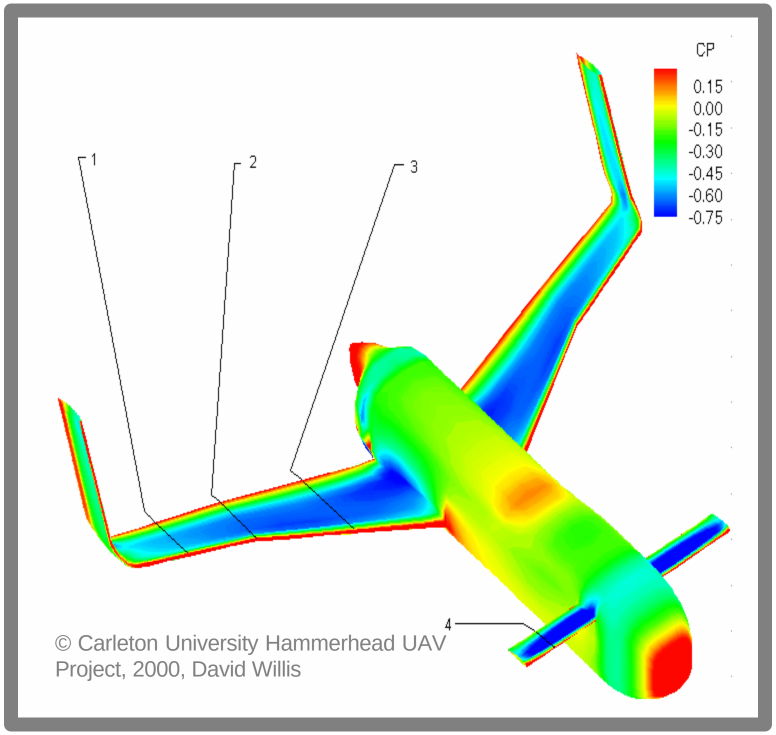
Fullwave analysis: radiation



Courtesy of Harris semiconductor

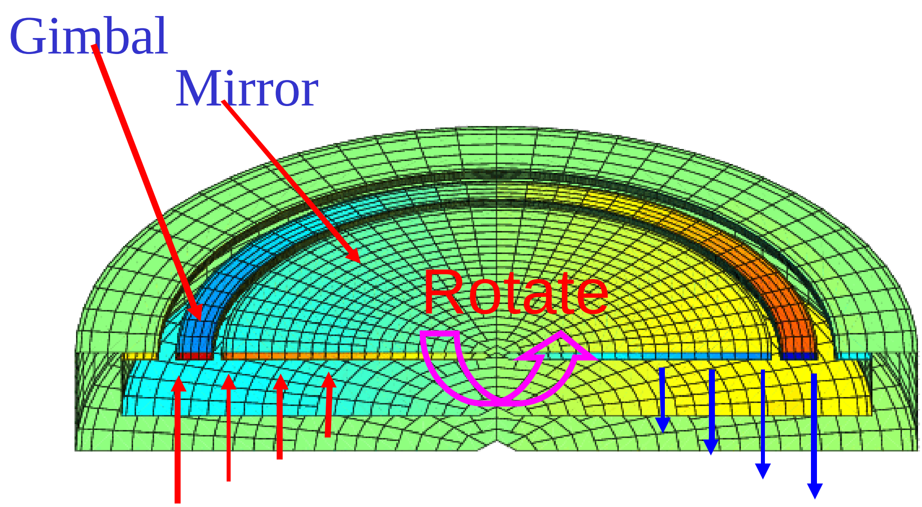
Some very useful applications

Computational Aerodynamics



Picture thanks to David Joe Willis

Stokes Flow Solver Viscous drag



Picture thanks to Xin Wang

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:

$$O(N^2) \quad \Rightarrow \quad O(N) \text{ or } O(N\log(N))$$

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: $G(\vec{r}+\triangle\vec{r},\vec{r}\,'+\triangle\vec{r}) = G(\vec{r},\vec{r}\,')$

Commonly used Green's function all satisfy these requirements

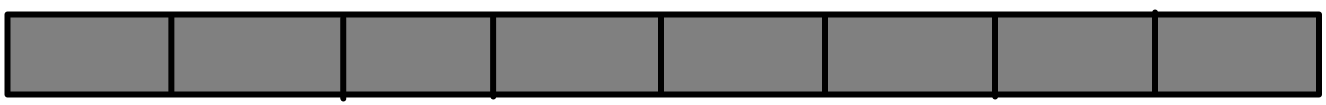
$$\frac{1}{|\vec{r}-\vec{r}\,'|},\quad \frac{e^{ik|\vec{r}-\vec{r}\,'|}}{|\vec{r}-\vec{r}\,'|},\quad \frac{\partial}{\partial \mathrm{n}}\left(\frac{1}{|\vec{r}-\vec{r}\,'|}\right),\quad \frac{\partial}{\partial \mathrm{n}}\left(\frac{e^{ik|\vec{r}-\vec{r}\,'|}}{|\vec{r}-\vec{r}\,'|}\right)$$

FFT-based Method

Key idea: kernel is shift-invariant

$$G(\vec{r},\vec{r}') = G(\vec{r}-\vec{r}',0) = \tilde{G}(\vec{r}-\vec{r}')$$

A simple example:



$$\int_S dS' G(\vec{r},\vec{r}')\, r(\vec{r}') = f(\vec{r}), \quad \vec{r} \in S$$

$$H\bar{a} = \bar{f}$$

FFT-based Method

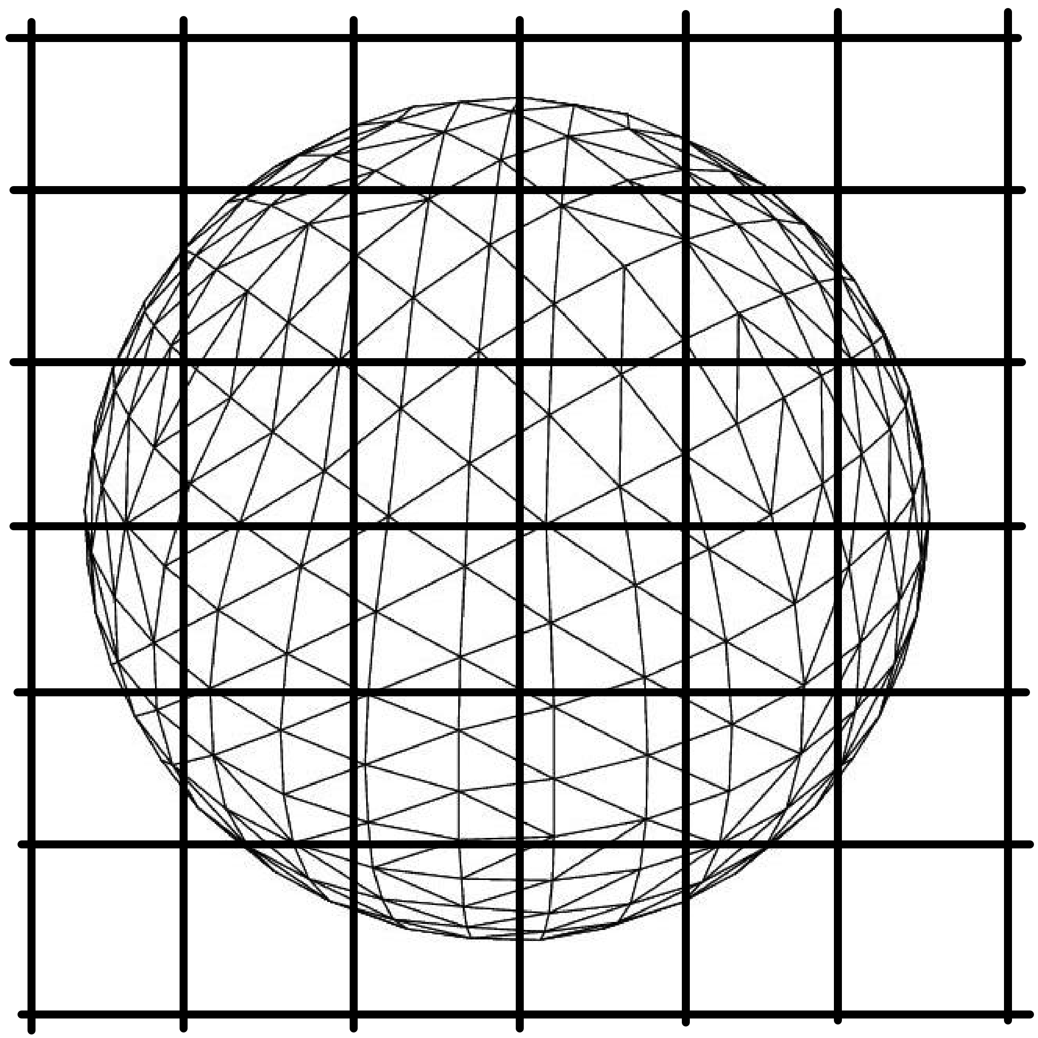
If collocation method with constant basis is used and all panels are identical

$$H_{i,j} = \int_{panel_j} dS' \tilde{G}(\vec{r}_i - \vec{r}_j')$$

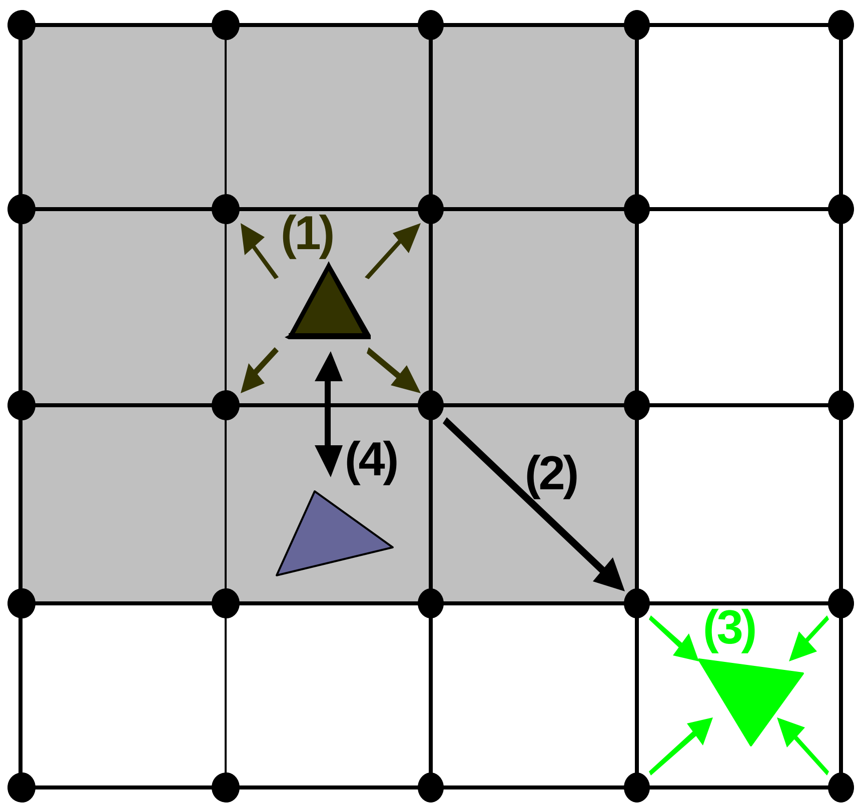
Only $H_{1,j}$ $(j = 1, 2, ..., N)$ are unique. H is a Toeplitz matrix. Matrix vector product could be computed using FFT in $O(N\log(N))$ time.

Operations: O(Nlog(N))   Memory: O(N)

Separation of Regular Grid From Discretization Panels



pFFT Algorithm: Basic steps



$(1)\ \text{Project}: \ \overline{Q_g} = [P]\overline{a}$

$(2)\ \text{Convolve}: \ \overline{f_g} = [H]\overline{Q_g}$

$(3)\ \text{Interpolate}: \ \overline{\Psi_g} = [I]\overline{f_g}$

$(4)\ \text{Direct}: \ \overline{\Psi_d} = [D]\overline{a}$

$$\Psi = \Psi_g + \Psi_d = ([D] + [I][H][P])\overline{a}$$

pFFT Algorithm: Basic Idea

A sparse representation of the system matrix

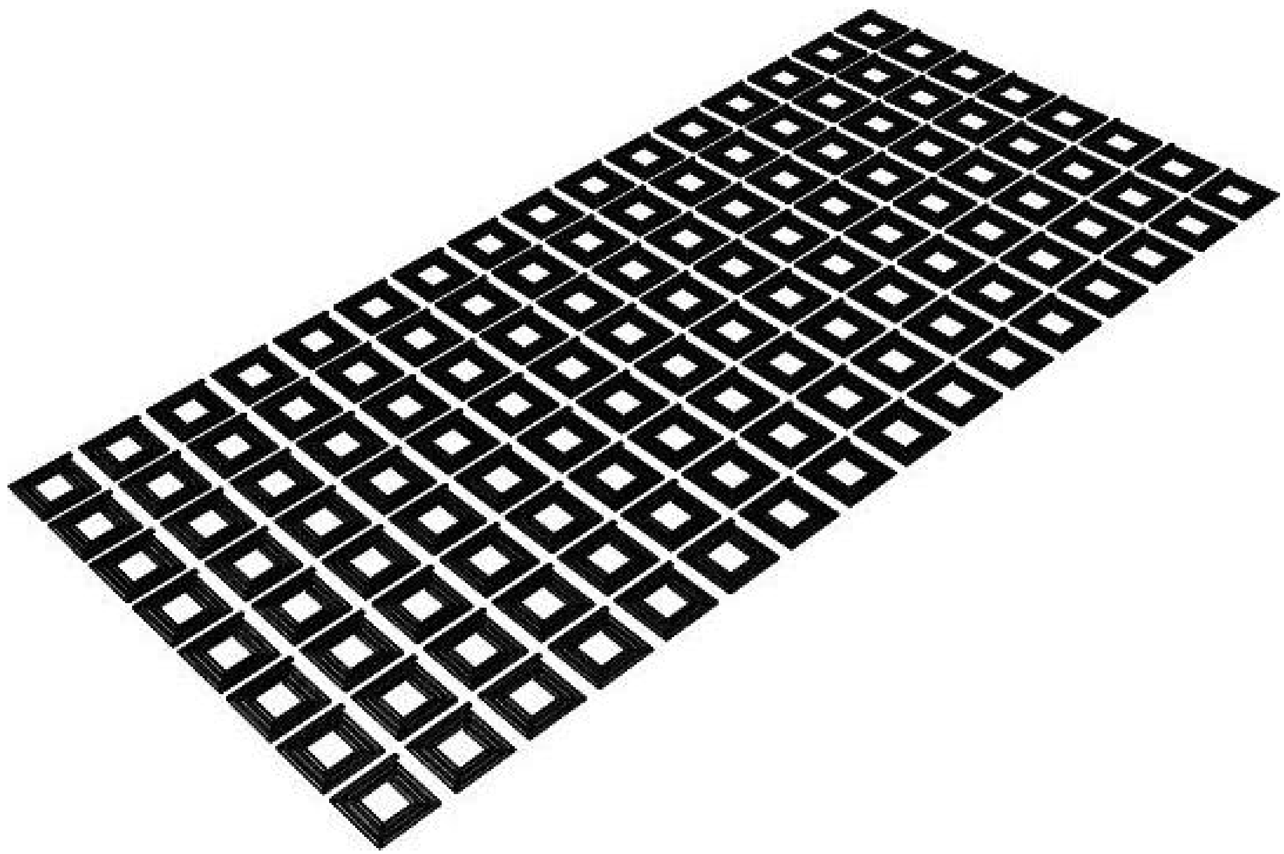
$$[A]_{N_b \times N_b} = [D]_{N_b \times N_b} + [I]_{N_b \times N_g} [H]_{N_g \times N_g} [P]_{N_g \times N_b}$$

$$O(N_b^{\ 2}) \quad O(N_b) \quad O(N_b) \quad O(N_g \log(N_g)) \quad O(N_b)$$

$$O(N_b^{\ 2}) \quad O(N_b) \quad O(N_b) \quad O(N_g) \quad O(N_b)$$

Application: FastImp
16 x 8 3-turn spiral array

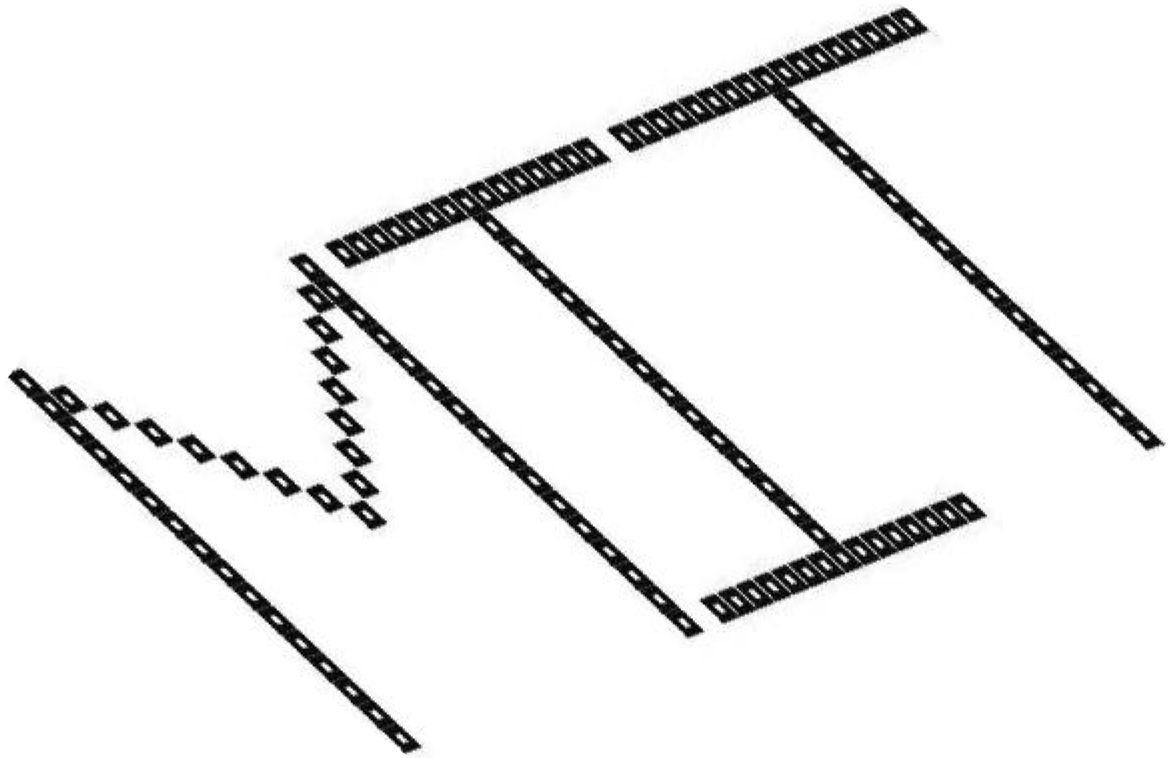
180k panels, 1.44 million unknowns, grid 256 x 128 x 8



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)

	MIT logo	16x8 array
P and *I* matrices	890	746
D and *H* matrices	13638	14353
Form the preconditioner P_r	54	53
LU factorization of P_r	1512	1927
GMRES (tol=1e-3)	32424 (77 iter)	25168 (80 iter)
total	51244	42247

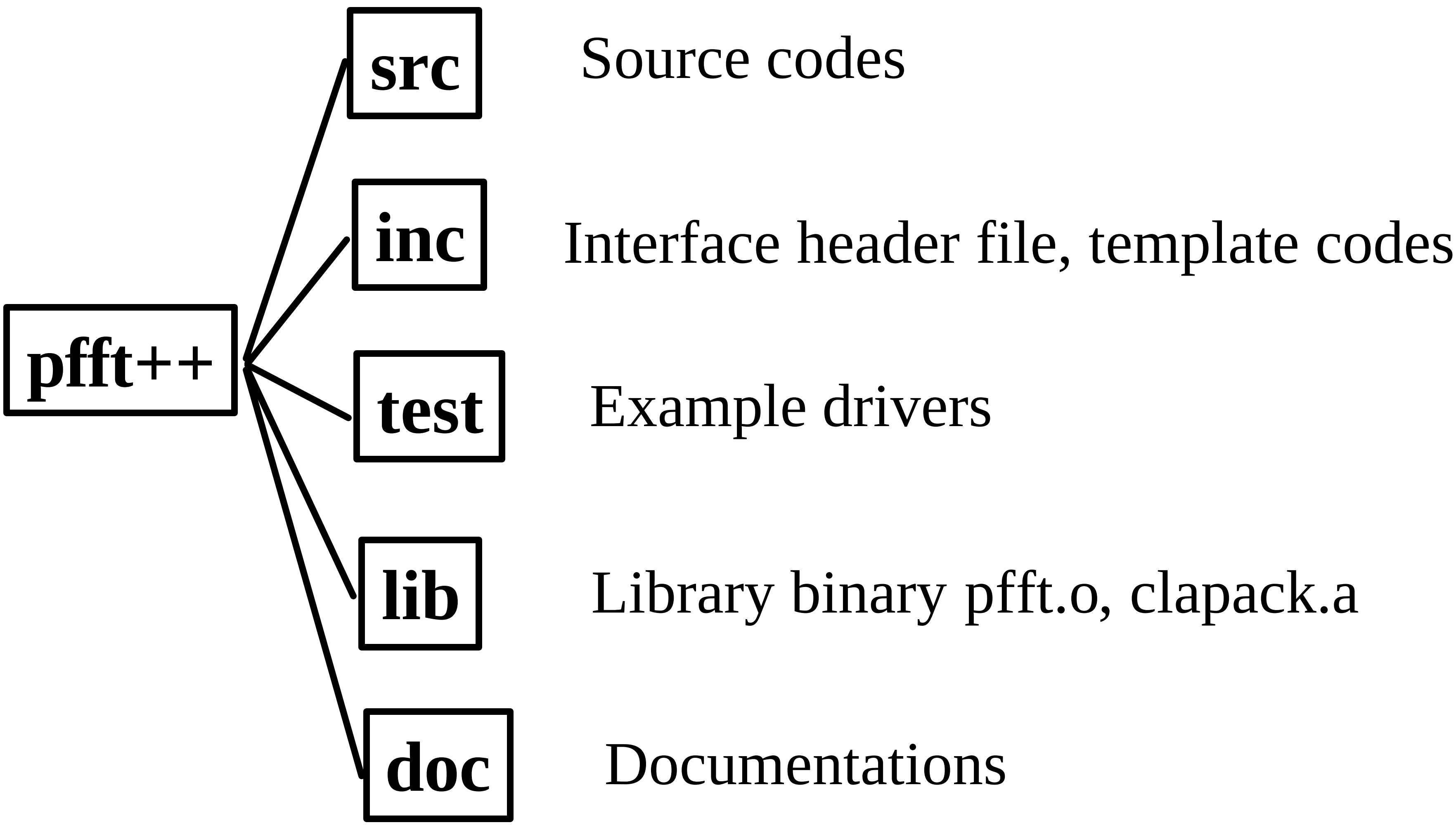
Notice the difference in GMRES is only about 25%, small considering the grid size is different by a factor of 8.

Breakdown of Memory (Mb)

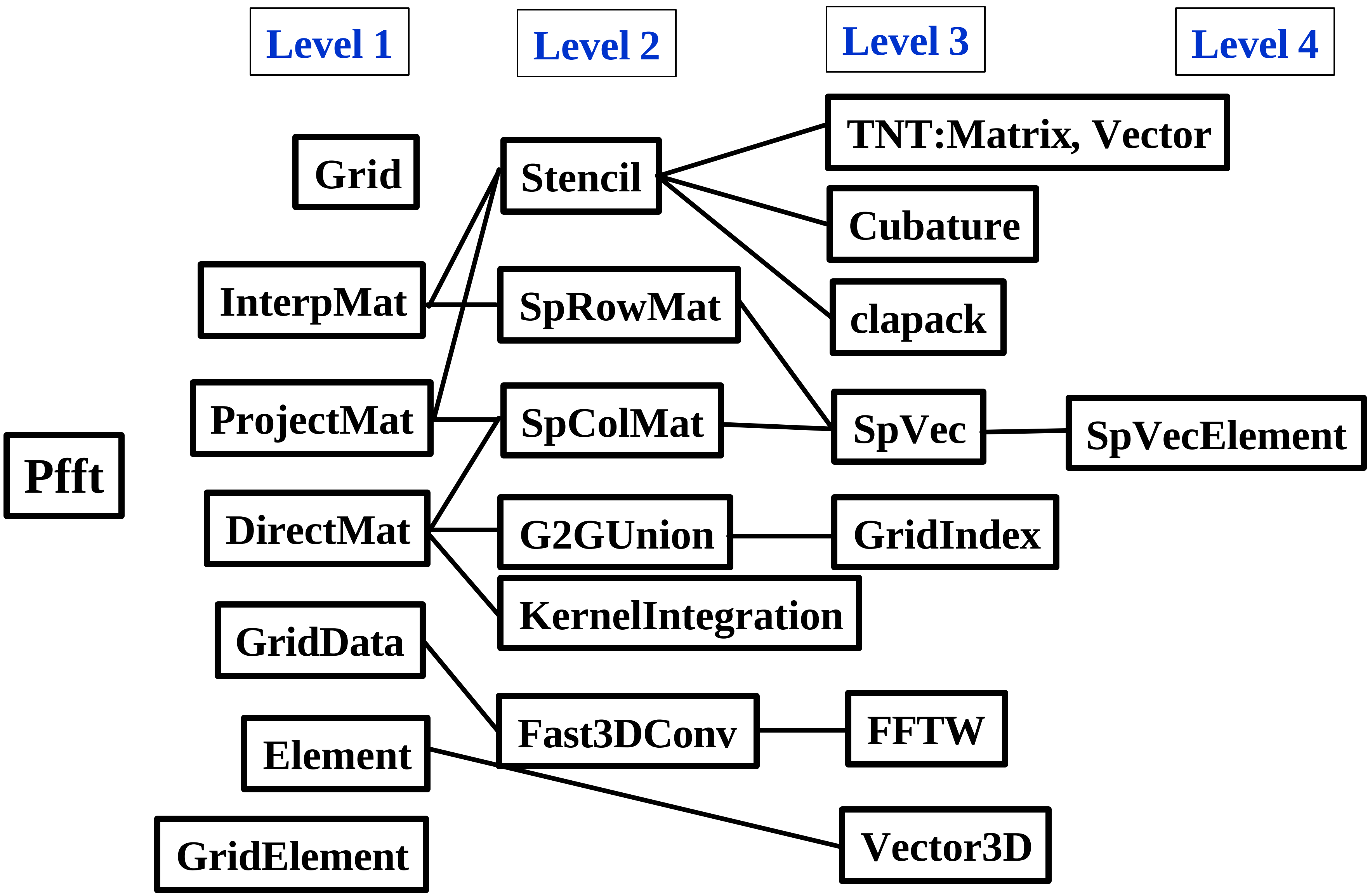
	MIT logo	16x8 array
Direct matrix	5.17	5.54
Projection matrix	0.38	0.39
Interpolation matrix	0.22	0.23
Convolution matrix	0.68	0.13
Maps between grids and panels	0.65	0.70
Pre-conditioner	2.72	2.76
GMRES	2.03	2.21
total	11.85	11.96

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

Project hierarchy of pFFT++



Main classes of pFFT++



Accessory classes of pFFT++

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

Spare matrix classese

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**

$$\int_S dS' \left[G(\vec{r},\vec{r}')\, r(\vec{r}') + \frac{dG(\vec{r},\vec{r}')}{dn(\vec{r}')}\, s(\vec{r}') \right]$$

Next

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