Algorithms, Implementation and Applications of pFFT++:

Projection and Interpolation

Zhenhai Zhu

RLE Computational prototyping group, MIT

www.mit.edu/people/zhzhu/pfft.html

Outline

- Grid-based interpolation
- Interpolation matrix
- Projection matrix
- Implementation details

Grid-based interpolation

Suppose charge is at origin, then potential is

$$f = \frac{1}{r}$$

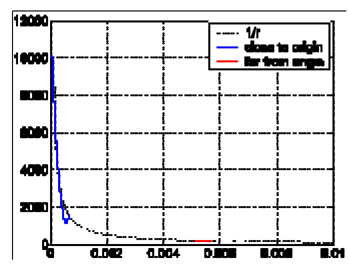
Using a second-order interpolation polynomial, the error is

$$e \approx \frac{1}{r}\left(\frac{h}{r}\right)^3$$

where *h* is the uniform grid spacing.
Far field can be accurately approximated with low-order polynomials.

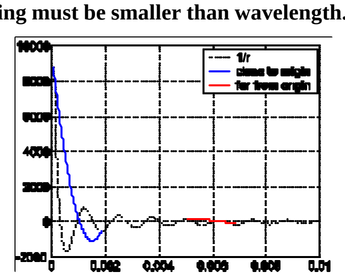
Grid-based interpolation

Interpolation error decreases with the increase of *r*.

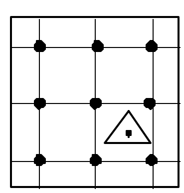


Grid-based interpolation

Interpolation error does not necessarily decrease with the increase of *r*. Grid Spacing must be smaller than wavelength.



pFFT Algorithm: Interpolation Matrix



Given $\overline{f}_g$

Compute $f(x, y)$

pFFT Algorithm: Interpolation Matrix

$$f(x,y)=\sum_k c_k f_k(x,y)=\bar{f}^{\,t}(x,y)\bar{c}$$

An example of $f_k(x,y)$:

$$1, x, x^2, y, xy, x^2y, y^2, xy^2, x^2y^2$$

pFFT Algorithm: Interpolation Matrix

$$f(x,y)=\left[f_1(x,y)\; f_2(x,y)\; \cdots\; f_9(x,y)\right]\begin{bmatrix} c_1 \\ \vdots \\ c_9 \end{bmatrix}=\bar{f}^{\,t}(x,y)\bar{c}$$

$$\bar{f}_g=\begin{bmatrix} f_{g,1} \\ f_{g,2} \\ \vdots \\ f_{g,9} \end{bmatrix}=\begin{bmatrix} f_1(x_1,y_1) & f_2(x_1,y_1) & \cdots & f_9(x_1,y_1) \\ f_1(x_2,y_2) & f_2(x_2,y_2) & \cdots & f_9(x_2,y_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_9,y_9) & f_2(x_9,y_9) & \cdots & f_9(x_9,y_9) \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_9 \end{bmatrix}=[F]\bar{c}$$

$$f(x,y)=\bar{f}^{\,t}(x,y)[F]^{-1}\bar{f}_g$$

pFFT Algorithm: Interpolation Matrix

$$\Psi_i=\langle t_i(\vec{r}), f(\vec{r})\rangle=\int_{\Delta_i^t} dSt_i(\vec{r})\bar{f}^{\,t}(\vec{r})[F]^{-1}\bar{f}_g=\bar{W}^t\bar{f}_g$$

$$\bar{\Psi}=\begin{bmatrix} \vdots \\ \Psi_i \\ \vdots \end{bmatrix}=\begin{bmatrix} \cdots & W_1 & \cdots & W_9 & \cdots \end{bmatrix}\begin{bmatrix} \vdots \\ f_{g,1} \\ \vdots \\ f_{g,9} \\ \vdots \end{bmatrix}=[I]\bar{f}_g$$

Operations: 9N_b Memory: 9N_b

pFFT Algorithm: Outer Differential Operator

If the kernel has a differential operator outside:

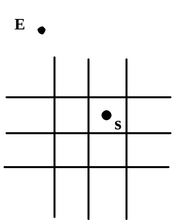
$$\frac{\partial}{\partial n(\vec{r})}\int_S dS' G(\vec{r},\vec{r}')r(\vec{r}')$$

The operator works on the interpolation

$$\frac{\partial}{\partial n(\vec{r})}f(\vec{r})=\frac{\partial}{\partial n(\vec{r})}\bar{f}^{\,t}(\vec{r})F^{-1}\bar{f}_g$$

$$\bar{W}_n^t=\int_{\Delta_i^t} dSt_i(\vec{r})\frac{\partial}{\partial n(\vec{r})}\bar{f}^{\,t}(\vec{r})[F]^{-1}=\int_{\Delta_i^t} dSt_i(\vec{r})\hat{n}(\vec{r})\bullet\nabla\bar{f}^{\,t}(\vec{r})[F]^{-1}$$

pFFT Algorithm: Projection Matrix



Assume a unit charge at point S

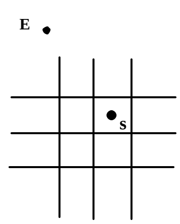
$$f_E^{(1)}=G(\vec{r}_s,\vec{r}_E)$$

find grid charge $\bar{r}_g$

$$f_E^{(2)}=\sum_i r_{g,i}G(\vec{r}_i,\vec{r}_E)=(\bar{r}_g)^t\bar{f}_g$$

such that $f_E^{(1)}=f_E^{(2)}$

pFFT Algorithm: Projection Matrix



Expand the Green's function

$$G(\vec{r},\vec{r}_E)=\sum_k f_k(\vec{r})c_k$$

match both sides at grid point $\vec{r}_i$

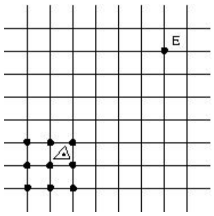
$$\bar{c}=F^{-1}\bar{f}_g$$

$$f_E^{(1)}=G(\vec{r}_s,\vec{r}_E)=\bar{f}^{\,t}(\vec{r}_s)F^{-1}\bar{f}_g$$

$$f_E^{(2)}=\sum_i r_{g,i}G(\vec{r}_i,\vec{r}_E)=(\bar{r}_g)^t\bar{f}_g$$

$$(\bar{r}_g)^t=\bar{f}^{\,t}(\vec{r}_s)[F]^{-1}$$

pFFT Algorithm: Projection Matrix



For unit point charge

$$(\bar{r}_g)^t = \bar{f}^t(\bar{r}_s)[F]^{-1}$$

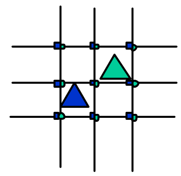
If the charge is a distributi on $b_j(\bar{r})$

$$(\bar{r}_g^{(j)})^t = \int_{\Delta_j^b} dSb_j(\bar{r})\bar{f}^t(\bar{r})[F]^{-1}$$

pFFT Algorithm: Projection Matrix

For multiple panels:

$$r(\bar{r}) = \sum_j a_j b_j(\bar{r})$$



$$(\bar{r}_g^{(j)})^t = \int_{\Delta_j^b} dSb_j(\bar{r})\bar{f}^t(\bar{r})[F]^{-1}$$

$$\bar{Q}_g = \sum_{j=1}^{N_b} a_j (\bar{r}_g^{(j)})^t$$

pFFT Algorithm: Projection Matrix

$$\bar{Q}_g = \sum_{j=1}^{N_b} a_j (\bar{r}_g^{(j)})^t$$

$$\bar{Q}_g = \begin{bmatrix} \vdots \\ Q_{g,1} \\ \vdots \\ Q_{g,9} \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & 0 & \vdots & 0 & \vdots \\ \vdots & r_{g,1}^{(j)} & \vdots & r_{g,1}^{(k)} & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & r_{g,9}^{(j)} & \vdots & r_{g,9}^{(k)} & \vdots \\ \vdots & 0 & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ a_j \\ \vdots \\ a_k \\ \vdots \end{bmatrix} = [P]\bar{a}$$

Operations: $9N_b$ Memory: $9N_b$

pFFT Algorithm: Inner Differential Operator

If the kernel has a differential operator inside:

$$\int_S dS' \frac{d}{dn(\bar{r}')} G(\bar{r}, \bar{r}') r(\bar{r}')$$

The operator works on the projection

$$\frac{d}{dn(\bar{r})} f(\bar{r}_s) = \frac{d}{dn(\bar{r})} \bar{f}^t(\bar{r}_s) F^{-1} \bar{f}_g$$

$$\bar{r}_g^t = \int_{\Delta_j^b} dSb_j(\bar{r}) \frac{d}{dn(\bar{r})} \bar{f}^t(\bar{r})[F]^{-1}$$

pFFT Algorithm: Duality of [*I*] and [*P*]

***i*th row of [*I*]:** $\int_{\Delta_i^t} dSt_i(\bar{r})\bar{f}^t(\bar{r})[F]^{-1}$

***j*th column of [*P*]:** $\int_{\Delta_j^b} dSb_j(\bar{r})\bar{f}^t(\bar{r})[F]^{-1}$

If $t_i(\bar{r}) = b_j(\bar{r})$, or $T_n = B_n$, then

$$P = I^t$$

pFFT Algorithm: Summary of *P* and *I*

operator	none	d/dn
[*I*]	$\int_{\Delta_i^t} dSt_i(\bar{r})\bar{f}^t(\bar{r})[F]^{-1}$	$\int_{\Delta_i^t} dSt_i(\bar{r})\frac{\partial}{\partial n(\bar{r})}\bar{f}^t(\bar{r})[F]^{-1}$
[*P*]	$\int_{\Delta_j^b} dSb_j(\bar{r})\bar{f}^t(\bar{r})[F]^{-1}$	$\int_{\Delta_j^b} dSb_j(\bar{r})\frac{d}{dn(\bar{r})}\bar{f}^t(\bar{r})[F]^{-1}$

pFFT Algorithm: Summary of *P* and *I*

For a general kernel
$$\frac{\partial}{\partial n(\vec{r})}\int_s \frac{\partial}{\partial n(\vec{r}\,')} G(\vec{r},\vec{r}\,')d\vec{r}\,'$$

The interaction is
$$A_{i,j} = \int_{\Delta_i^t} d\vec{r}\, t_i(\vec{r}) \frac{\partial}{\partial n(\vec{r})} \int_{\Delta_j^b} d\vec{r}\,' b_j(\vec{r}\,') \frac{\partial}{\partial n(\vec{r}\,')} G(\vec{r},\vec{r}\,')$$

$\Downarrow$ [*I*] $\qquad$ $\Downarrow$ [*P*]

Both matrices are independent of the Green's function

Implementation: How to Fill the Interpolation Matrix

- Row index = panel index
- Column index = index of interpolation grid for the panel

$$\begin{bmatrix} \vdots \\ \Psi_i \\ \vdots \end{bmatrix} = \begin{bmatrix} \cdots & W_1 & \cdots & W_9 & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ f_{g,1} \\ \vdots \\ f_{g,9} \\ \vdots \end{bmatrix}$$

Implementation: How to Fill the Projection Matrix

- Column index = panel index
- Row index = index of interpolation grid for the panel

$$\begin{bmatrix} \vdots \\ Q_{g,1} \\ \vdots \\ Q_{g,9} \\ \vdots \end{bmatrix} = \begin{bmatrix} \ddots & 0 & \vdots \\ \vdots & r_{s,1}^{(j)} & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & r_{s,9}^{(j)} & \vdots \\ \vdots & 0 & \vdots \end{bmatrix} \begin{bmatrix} \vdots \\ a_j \\ \vdots \end{bmatrix}$$

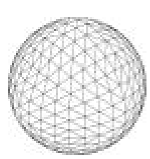
Implementation: Source codes

- See interpMat.cc
- See projectMat.cc

Numerical Experiments

On the surface of a sphere with radius *R*
$$\int_S dS' K(\vec{r},\vec{r}\,')\, r(\vec{r}\,') \Rightarrow Ax$$

Let *x* be a random vector



$$y_1 = Ax$$
$$y_2 = pfft(x)$$
$$error = \frac{\|y_1 - y_2\|_2}{\|y_1\|_2}$$

Numerical Experiments: Error vs. polynomial order

	P = 3	*P* = 5	*P* = 7
$1/r$	8.4e-5	1.3e-6	4.3e-9
$\frac{\partial}{\partial n} 1/r$	8.5e-3	1.1e-4	8.4e-7
e^{ikr}/r *kR* = 1.1e-9	8.3e-5	1.3e-6	1.7e-9
$\frac{\partial}{\partial n} e^{ikr}/r$ *kR* = 1.1e-9	6.0e-3	7.5e-5	5.9e-7
e^{ikr}/r *kR* = 11.1	4.9e-4	1.1e-5	4.0e-7
$\frac{\partial}{\partial n} e^{ikr}/r$ *kR* = 11.1	1.4e-2	2.8e-4	6.5e-6

setup time vs. polynomial order (seconds)

	$P = 3$	$P = 5$	$P = 7$
$1/r$	3.76	39.48	305.61
$\frac{\partial}{\partial n} 1/r$	4.28	45.96	326.47
e^{ikr}/r kR = 1.1e-9	55.66	249.01	1022.05
$\frac{\partial}{\partial n} e^{ikr}/r$ kR = 1.1e-9	47.80	229.02	971.32
e^{ikr}/r kR = 11.1	53.06	242.65	1082.36
$\frac{\partial}{\partial n} e^{ikr}/r$ kR = 11.1	47.99	226.89	967.58

Memory usage vs. polynomial order (Mb)

	$P = 3$	$P = 5$	$P = 7$
$1/r$	10.75	35.18	87.94
$\frac{\partial}{\partial n} 1/r$	10.75	35.18	87.94
e^{ikr}/r kR = 1.1e-9	16.04	47.3	114.5
$\frac{\partial}{\partial n} e^{ikr}/r$ kR = 1.1e-9	16.04	47.3	114.5
e^{ikr}/r kR = 11.1	16.04	47.3	114.5
$\frac{\partial}{\partial n} e^{ikr}/r$ kR = 11.1	16.04	47.3	114.5

Matrix vector product time vs. polynomial order

	$P = 3$	$P = 5$	$P = 7$
$1/r$	0.07	0.11	0.17
$\frac{\partial}{\partial n} 1/r$	0.07	0.11	0.17
e^{ikr}/r kR = 1.1e-9	0.20	0.33	0.64
$\frac{\partial}{\partial n} e^{ikr}/r$ kR = 1.1e-9	0.20	0.33	0.64
e^{ikr}/r kR = 11.1	0.19	0.32	0.63
$\frac{\partial}{\partial n} e^{ikr}/r$ kR = 11.1	0.19	0.32	0.63

Moral of the story

- pFFT++ is excellent for 4-5 digit accuracy
- Use with precaution for higher accuracy level

Next Lecture

- Loss of Accuracy in double-layer potential
- Compact projection and interpolation stencil