IML

ACA

MLMDA

Fast Methods

MLFMA

Donald R. Wilton

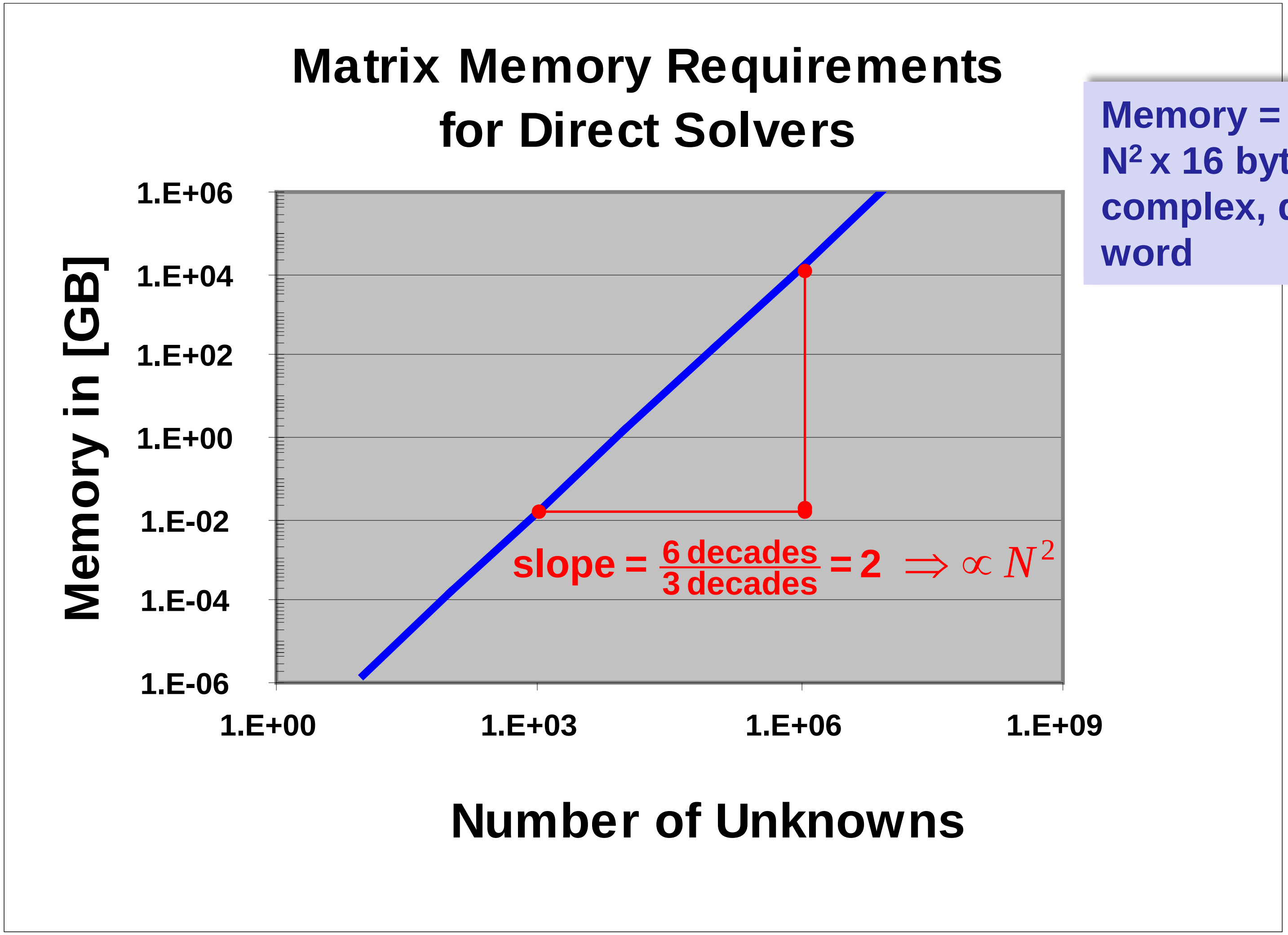
Vikram Jandhyala

SVD

CGFFT

Precorrected FFT

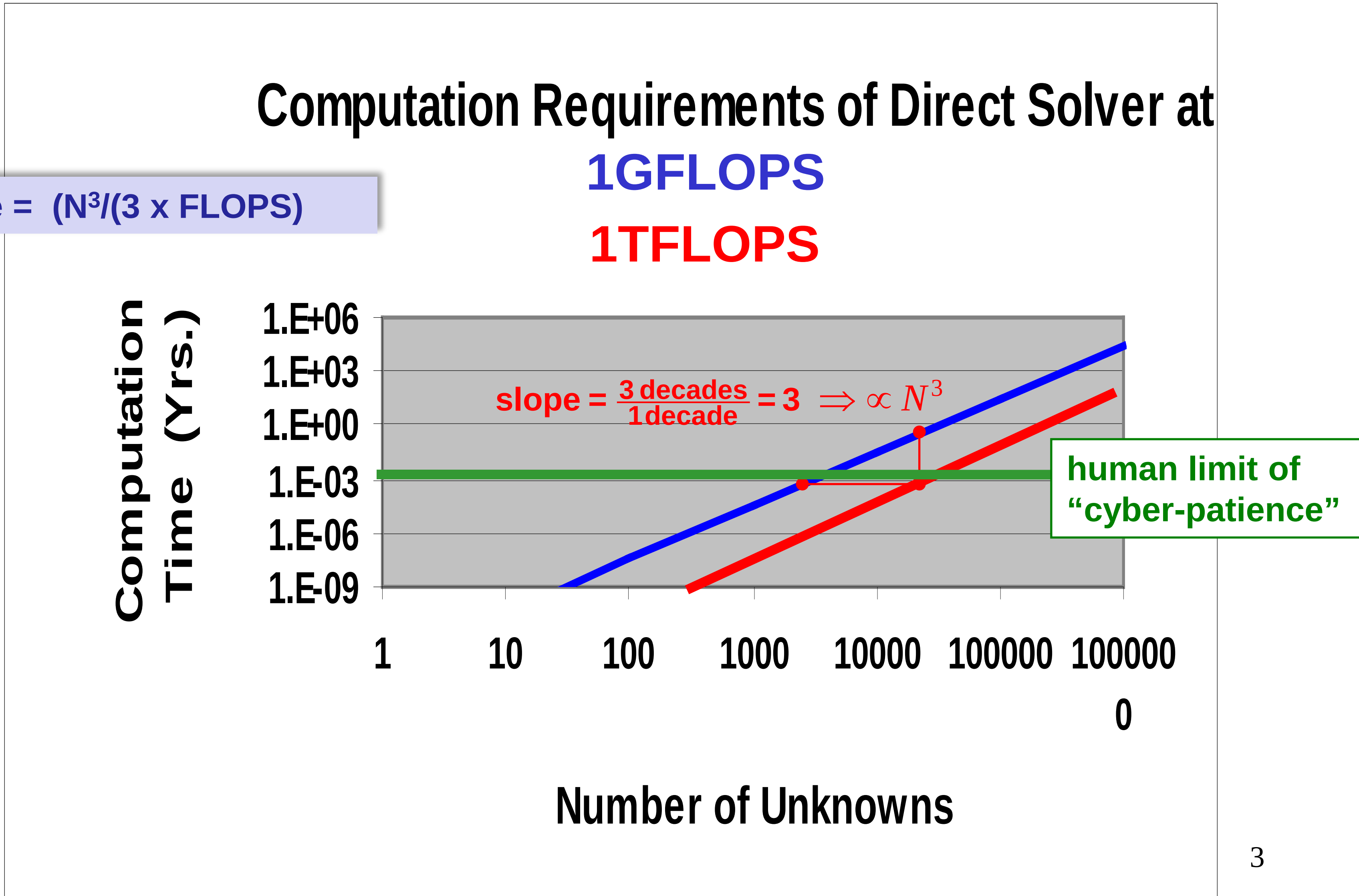
Why Are Fast Methods Needed for Large MoM Problems?



Memory = N^2 x 16 bytes/ complex, double word

Approximate Computation Times for Large Problems

Time = $(N^3/(3 \times \text{FLOPS})$



Main Features of Fast Methods

- **We assume solution uses an *iterative*, not a *direct* method**
- **Use *redundant information* in Mom matrix and/or Green's function to reduce storage requirements ("compress" the matrix) and speed up the solution process**

Iterative Methods

- Instead of directly solving

$$\mathbf{Ax} = \mathbf{b}$$

by, e.g. Gaussian elimination, we iterate on an equation of the form

$$\mathbf{x}_n = \mathbf{B}_n \mathbf{x}_{n-1} + \mathbf{c}_n, \quad n = 1, 2, \cdots, \qquad \boxed{\mathbf{B}_n = \mathbf{B}_n(\mathbf{A}, \mathbf{x}_{n-1}, \mathbf{x}_{n-2}, \cdots, \mathbf{r}_{n-1})}$$

where $\mathbf{x}_0$ is an initial guess, until we achieve

convergence, say $\|\mathbf{x}_n - \mathbf{x}_{n-1}\| < \varepsilon_1$, and / or $\|\overbrace{\mathbf{Ax}_n - \mathbf{b}}^{\mathbf{r}_n}\| < \varepsilon_2$.

- The process must usually be sped up by *preconditioning* the system, i.e., premultiplying by a matrix $\mathbf{P}$ and solving the modified system

$$\mathbf{PAx} = \mathbf{Pb}$$

Iterating the Preconditioned System

- The preconditioner should in some sense approximate the inverse of the system matrix, $\mathbf{P} \approx \mathbf{A}^{-1}$, or equivalently $\mathbf{PA} \sim \mathbf{I}$
- When this is the case, we may view the term $\left(\mathbf{I}-\mathbf{PA}\right)$ in the *identity*

$$\mathbf{x} = \underbrace{\left(\mathbf{I}-\mathbf{PA}\right)}_{\approx \mathbf{0}}\mathbf{x} + \underbrace{\mathbf{Pb}}_{\approx \mathbf{A}^{-1}\mathbf{b}} \quad \Longleftarrow$$

add:

$$\mathbf{0} = -\mathbf{PAx} + \mathbf{Pb}$$

$$\mathbf{x} = \mathbf{Ix}$$

as a "small" correction to the RHS, leading to the simple iterative procedure

$$\mathbf{x}_n = \left(\mathbf{I}-\mathbf{PA}\right)\mathbf{x}_{n-1} + \mathbf{Pb} \quad \Leftrightarrow \quad \mathbf{x}_n = \mathbf{B}_n\mathbf{x}_{n-1} + \mathbf{c}_n, \;\; n = 1, 2, \cdots$$

Iterative Convergence of the Preconditioned System

- Beginning with $\mathbf{x}_0 \equiv \mathbf{0}$, successive iterations of the simple iterative procedure yield

$$\boxed{\mathbf{x}_n = (\mathbf{I}-\mathbf{PA})\mathbf{x}_{n-1} + \mathbf{Pb}}$$

$$\begin{aligned}
\mathbf{x}_0 &= \mathbf{0} \\
\mathbf{x}_1 &= \mathbf{Pb} \\
\mathbf{x}_2 &= (\mathbf{I}-\mathbf{PA})\mathbf{Pb} + \mathbf{Pb} \\
\mathbf{x}_3 &= (\mathbf{I}-\mathbf{PA})^2\mathbf{Pb} + (\mathbf{I}-\mathbf{PA})\mathbf{Pb} + \mathbf{Pb} \\
&\vdots \\
\mathbf{x}_{n+1} &= \left[\sum_{i=0}^{n} (\mathbf{I}-\mathbf{PA})^i\right]\mathbf{Pb}
\end{aligned}$$

- Identifying $(\mathbf{I}-\mathbf{PA}) \equiv \mathbf{R}$ in the identity, and noting

$$\sum_{i=0}^{\infty}\mathbf{R}^i = (\mathbf{I}-\mathbf{R})^{-1},\ \|\mathbf{R}\| < 1 \qquad \left(\Rightarrow \sum_{i=0}^{\infty}(\mathbf{I}-\mathbf{PA})^i = (\mathbf{PA})^{-1}\right)$$

we see that the solution converges to

$$\mathbf{x} = (\mathbf{PA})^{-1}\mathbf{Pb} = \mathbf{A}^{-1}\mathbf{P}^{-1}\mathbf{Pb} = \mathbf{A}^{-1}\mathbf{b}$$

if $\|\mathbf{I}-\mathbf{PA}\| < 1$.

Observations on the Iterative Procedure

- Our *simple* procedure *may* not converge at all (i.e., if $\|\mathbf{I}\text{-}\mathbf{PA}\| > 1$) though in principle, that is not the case with more sophisticated iterative algorithms. Commonly used algorithms include BiCGSTAB, GMRES, QMR, etc.
- Convergence speeds up the closer $\mathbf{P}$ is to $\mathbf{A}^{-1}$.
- The main computational bottleneck is then the repeated calculation of the matrix / vector ("*matvec*") products $(\mathbf{I}\text{-}\mathbf{PA})\mathbf{x}_{n-1}$. All so-called "fast methods" attempt to speed up the matrix / vector product ("matvec") computation.
- Modern iterative solvers require that the *user* implement the matvec computations like $\mathbf{PAx}_{n-1}$ to allow use of the most appropriate speedup method.

Matrix/Vector Products

- The *inner product* between two vectors generates a *scalar* given by

$$\langle \mathbf{u},\mathbf{v}\rangle = \mathbf{u}^t\mathbf{v} = [u_1, u_2, \cdots, u_N]\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix} = u_1v_1 + u_2v_2 + \cdots + u_Nv_N$$

The product requires approximately N operations.

- The *outer product* between two vectors generates a *matrix* given by

$$\mathbf{u}\mathbf{v}^t = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_M \end{bmatrix}[v_1, v_2, \cdots, v_N] = \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_N \\ u_2v_1 & u_2v_2 & \cdots & u_2v_N \\ \vdots & & \ddots & \vdots \\ u_Mv_1 & u_Mv_2 & \cdots & u_Mv_N \end{bmatrix}$$

Matrix rank = number of rows (columns) of largest submatrix with non-vanishing determinant

Since all rows and columns are proportional, the matrix is only rank 1.

Forming the product on the RHS requires approximately MN operations.

- But storage of **u, v** requires only $M+N$ memory locations and $\mathbf{u}\mathbf{v}^t\mathbf{x}$ can be evaluated in only $M+N$ multiplications!

Matrix/Vector Products with Low Rank Matrices

- **The sum of r *outer products* of independent vectors,**

$$\sum_{p=1}^{r} \mathbf{u}_p \mathbf{v}_p^t \equiv \mathbf{U}\mathbf{V}^t \text{ where } \mathbf{U} \equiv \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_r \end{bmatrix}, \mathbf{V}^t \equiv \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_r \end{bmatrix}^t = \begin{bmatrix} \mathbf{v}_1^t \\ \mathbf{v}_2^t \\ \vdots \\ \mathbf{v}_r^t \end{bmatrix}$$

is a matrix of rank r; conversely, such matrices can be so factorized.

- **The matrix / vector product $\mathbf{Ab} \left(= \mathbf{A}_{M\times N} \mathbf{b}_{N\times 1}\right)$ generally requires MN multiplies, but the product**

$$\left(\sum_{p=1}^{r} \mathbf{u}_p \mathbf{v}_p^t \right) \mathbf{b} = \sum_{p=1}^{r} \mathbf{u}_p \left(\mathbf{v}_p^t \mathbf{b} \right) = \mathbf{U}\left(\mathbf{V}^t \mathbf{b} \right)$$

> **Storage of $\mathbf{A}$: MN**
>
> **Storage of $\mathbf{u}_p, \mathbf{v}_p$: $r\left(M+N\right)$**

requires only about $r\left(N+M\right)$ multiplies when performed using the RHS grouping. If $r << \min(M, N)$, $\Rightarrow$ significant speedup!

- **But $r < M, N \Rightarrow \left(\sum_{p=1}^{r} \mathbf{u}_p \mathbf{v}_p^t \right)$ is singular; hence it must be that only *subblocks* of $\mathbf{A}$, *not* the entire system matrix, can be represented in this form. Such matrices are said to be *rank deficient.***

Obtaining Low Rank Matrices

- **Fast methods approximate *off-diagonal blocks* of the system matrix as low rank matrices that can be represented in product form, UV^t. Such blocks typically represent *far interactions* between closely grouped observation and source element *clusters*.**

- **There are two approaches to obtaining reduced rank blocks:**

  1) Represent the Green's function in *separable* or *degenerate* form over the block's observer and source domains.

  2) Use matrix algebraic methods to directly find reduced-rank block representations

Matrix-Vector Product for Sums of Separable Matrices

- **Separable kernels lead to separable matrix blocks:**

  E.g., for simple integral eq. $\int_{\mathcal{D}} G(\mathbf{r},\mathbf{r}')\,x(\mathbf{r}')\,d\mathcal{D} = f(\mathbf{r}),\ \mathbf{r} \in \mathcal{D}$

  with kernel expansion $G(\mathbf{r},\mathbf{r}') \approx \sum_{p=1}^{r} u_p(\mathbf{r}) v_p(\mathbf{r}'),\quad \mathbf{r},\mathbf{r}' \in$ ***subregion* of** $\mathcal{D}$

  and basis representation $x(\mathbf{r}) \approx = \sum_{n} x_n b_n(\mathbf{r}) = \left[b_n(\mathbf{r})\right]^t \left[x_n\right]$ **,**

  contributions to a block of the Galerkin system matrix are

$$\mathbf{U}\mathbf{V}^t\mathbf{x},$$

  where

$$\mathbf{U} = \left[< b_m, u_p >\right]_{M\times r} \qquad \mathbf{x} = \left[x_n\right]_{N\times 1}$$

$$\mathbf{V} = \left[< b_n, v_p >\right]_{N\times r}$$

Matrix-Vector Product for Separable Matrix:

$$\mathbf{U}\mathbf{V}^t\mathbf{x} \equiv \left[\sum_{p=1}^{r} \mathbf{u}_p \mathbf{v}_p^{\,t}\right]\mathbf{x} = \left[\mathbf{u}_1\, \mathbf{u}_2 \cdots \mathbf{u}_r\right]\left[\mathbf{v}_1\, \mathbf{v}_2 \cdots \mathbf{v}_r\right]^t \mathbf{x}$$

- $r(M+N)$ **operations if performed right-to-left**
- $MN(r+1)$ **operations if performed left-to-right**

Approach Generalizes to More Complex Operators

- **A block of an EFIE matrix becomes**

$$
\begin{aligned}
\mathbf{Z}_{M\times N} \quad &= \quad j\omega\mu \sum_{p=1}^{r} [<\Lambda_m, u_p>]_{M\times r} \cdot [<\Lambda_n, v_p>]_{N\times r}^{t} \\
&\quad + \frac{1}{j\omega\varepsilon} \sum_{p=1}^{r} [<\nabla\cdot\Lambda_m, u_p>]_{M\times r} [<\nabla\cdot\Lambda_n, v_p>]_{N\times r}^{t} \\
&= \quad j\omega\mu \mathbf{U}\cdot\mathbf{V}^{t} + \frac{1}{j\omega\varepsilon} \mathbf{U}'\mathbf{V}'^{t}
\end{aligned}
$$

where

$$
\mathbf{U} = [<\Lambda_m, u_p>]_{M\times r}, \qquad \mathbf{V} = [<\Lambda_n, v_p>]_{N\times r}
$$

$$
\mathbf{U}' = [<\nabla\cdot\Lambda_m, u_p>]_{M\times r} \quad \mathbf{V}' = [<\nabla\cdot\Lambda_n, v_p>]_{N\times r}
$$

Fast Methods Often Combine Separable Matrix Approximation with Hierarchical Methods

- **Separable matrices reduce both storage and matrix vector multiplication counts from** MN **to** $r(M+N)$

- **Unfortunately it is not possible to approximate the *entire* system matrix by a separable matrix---it will be *rank deficient* and hence have no inverse (i.e., no solution)**

- **Nevertheless, nearly all fast methods in computational electromagnetics are based on approximating *blocks* of the system matrix by separable matrices**

- **For additional speed, some hierarchical scheme generally must be used to transfer information at one discretization level to another**

The End