FMMTL: FMM Template Library

Generalized Framework for Kernel Matrices

Cris Cecka, Simon Layton

Institute for Applied Computational Science
School of Engineering and Applied Sciences
Harvard University

November 15, 2013

Dense Matrices

$$\mathbf{K} = \begin{bmatrix} K_{00} & K_{01} & \cdots & K_{0N} \\ K_{10} & K_{11} & \cdots & K_{1N} \\ \vdots & \vdots & \ddots & \vdots \\ K_{N0} & K_{N1} & \cdots & K_{NN} \end{bmatrix} \in \mathbb{K}^{N,N}$$

- Storage: $\mathcal{O}(N^2)$
- Matrix-vector product: $\mathcal{O}(N^2)$ ops.
- Matrix-matrix product: $\mathcal{O}(N^3)$ ops.
- Matrix factorizations (LU, QR, SVD): $\mathcal{O}(N^3)$ ops.

Structured Dense Matrices

- Low-rank Matrices
- Heirarchical Matrices

Kernel Matrices

$$\mathbf{K} = \begin{bmatrix} K(t_0, s_0) & K(t_0, s_1) & \cdots & K(t_0, s_N) \\ K(t_1, s_0) & K(t_1, s_1) & \cdots & K(t_1, s_N) \\ \vdots & \vdots & \ddots & \vdots \\ K(t_N, s_0) & K(t_N, s_1) & \cdots & K(t_N, s_N) \end{bmatrix} = \left[K_{ij}\right]$$

and the matrix equation

$$r_i = K(t_i, s_j)\, c_j$$

where

- $K(\cdot, \cdot)$: The *kernel* generating matrix elements,
- s_j: The *sources* of the kernel,
- c_j: The *charges* of the sources,
- t_i: The *targets* of the kernel (could be $= s_j$),
- r_j: The *results* (*potentials*) of the targets.

Kernel Matrices

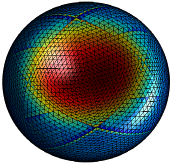
$$\mathbf{K} = \begin{bmatrix} K(t_0, s_0) & K(t_0, s_1) & \cdots & K(t_0, s_N) \\ K(t_1, s_0) & K(t_1, s_1) & \cdots & K(t_1, s_N) \\ \vdots & \vdots & \ddots & \vdots \\ K(t_N, s_0) & K(t_N, s_1) & \cdots & K(t_N, s_N) \end{bmatrix} = [K_{ij}]$$

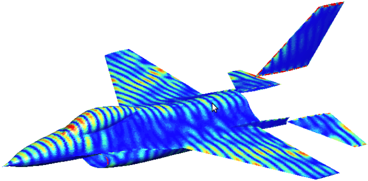
Appear in

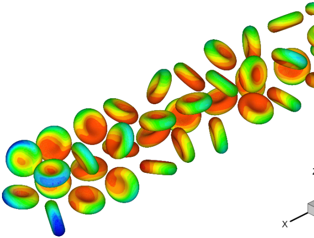
Name/Equation	$K(x,y)$	**Field**
Laplace, Poisson	$1/\lvert x-y\rvert$	$\mathbb{R}^3 \to \mathbb{R}$
Yukawa, Helmholtz	$e^{k\lvert x-y\rvert}/\lvert x-y\rvert$	$\mathbb{R}^3 \to \mathbb{C}$
Stokes	$\frac{1}{\lvert x-y\rvert}\left(\mathbf{I} + \frac{(x-y)(x-y)^T}{\lvert x-y\rvert^2}\right)$	$\mathbb{R}^3 \to \mathbb{R}^{3,3}$
Gaussian	$e^{-\varepsilon\lvert x-y\rvert^2}$	$\mathbb{R}^n \to \mathbb{R}$

- Other physics:
  - Elastics, Maxwell, Biot-Savart, any Green's function.
- Other machine learning, interpolation
  - (Multi)quadric, Inv (Multi)quadric, splines, any RBF.

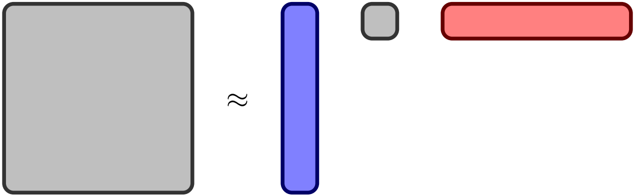
Flashy Applications







Low Rank Matrices



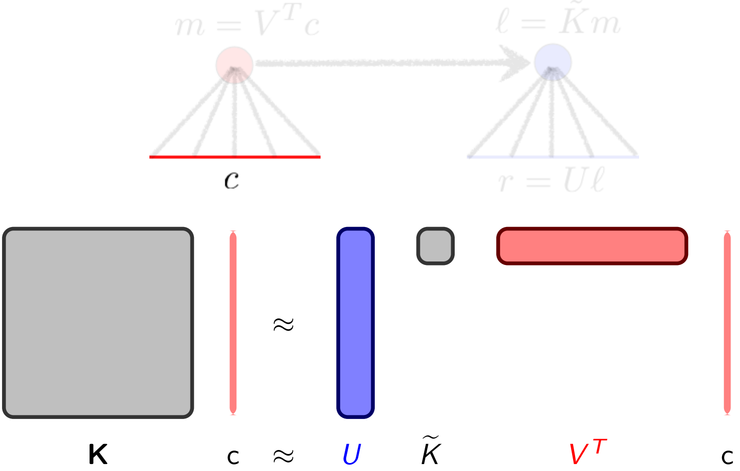
$$\mathbf{K} \approx U \widetilde{K} V^T$$

- Storage: $\mathcal{O}(rN)$ where $\widetilde{K} \in \mathbb{K}^{r,r'}$
- Matrix-vector product: $\mathcal{O}(rN)$
- Constructed with:

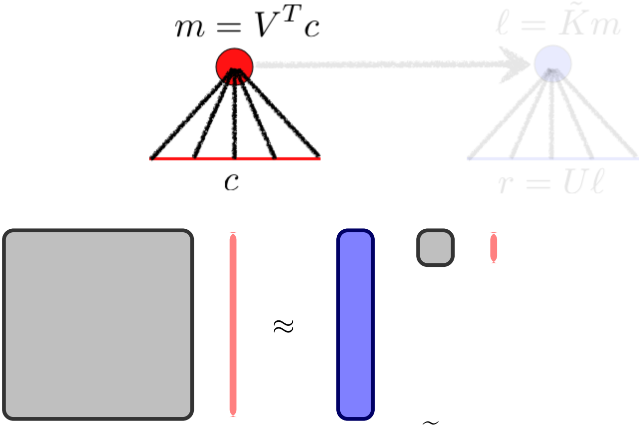
Algebraic techniques	Cost
SVD	$\mathcal{O}(N^3)$
RRLU, RRQR	$\mathcal{O}(rN^2)$
ACA	$\mathcal{O}(r^2 N)$
Psuedo-skeletal,CUR	$\mathcal{O}(rN)$

Analytic techniques	Cost
Series expansion	$\mathcal{O}(rN)$
Interpolation	$\mathcal{O}(rN)$

Low Rank Interpretation

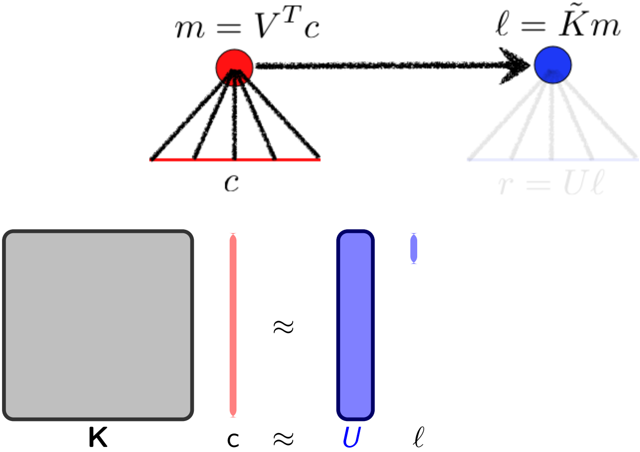


Low Rank Interpretation

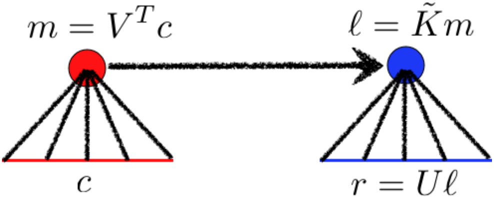


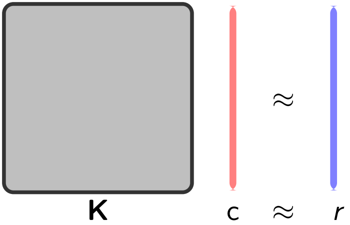
$$\mathbf{K} \quad c \quad \approx \quad U \quad \widetilde{K} \quad m$$

Low Rank Interpretation



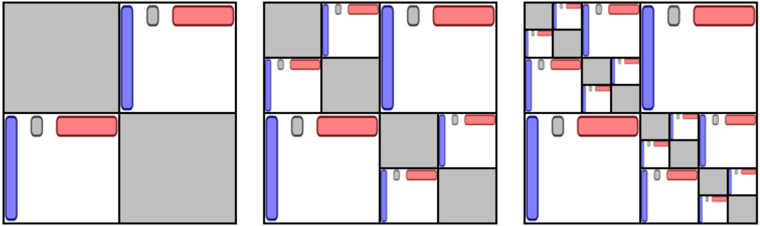
Low Rank Interpretation





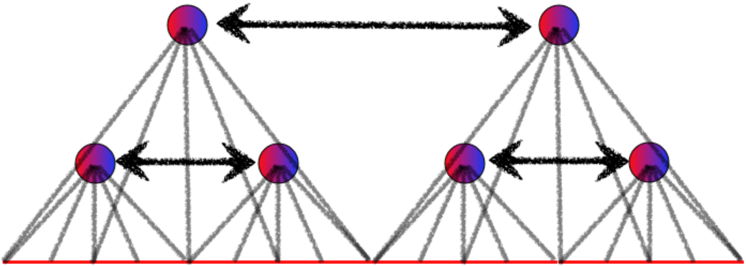
Hierarchically Low Rank

- Fully low rank matrices are scarce.
- Instead, hierarchically off-diagonal low-rank (HODLR):



HODLR Interpretation

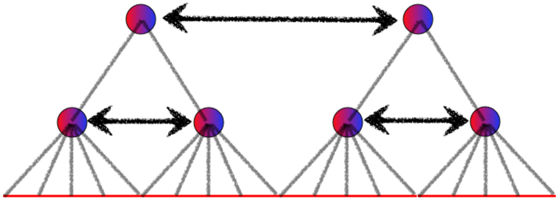
Disjoint “clusters” of sources and targets have low-rank interaction.



Additional Low Rank Operations

- Nested low-rank property.
  - Multipole/Local expansions may be passed up/down the tree.

$$\mathbf{K}_{IJ} = U_{C(I)} \widetilde{U}_I \widetilde{K}_{IJ} \widetilde{V}_J^T V_{C(J)}^T$$



  - Called heiarchically semiseparable (HSS) matrices.
- Well-separated condition
  - Stricter than simply disjoint clusters: Need well-separated.
  - Forces off-diagonal dense blocks.
  - Called $\mathcal{H}$ and $\mathcal{H}^2$ matrices.

Another View

Using a low rank approximation,

$$K(\mathbf{r}_i + \mathbf{r}_0 + \mathbf{r}_j) = \sum_{p=1}^{r} \sum_{q=1}^{r'} U_p(\mathbf{r}_i)\, \widetilde{K}_{pq}(\mathbf{r}_0)\, V_q^T(\mathbf{r}_j) + \varepsilon$$

Another View

Using a low rank approximation,

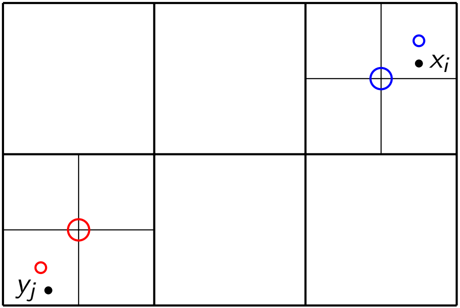
$$K(\mathbf{r}_i + \mathbf{r}_0 + \mathbf{r}_j) = \sum_{p=1}^{r} \sum_{q=1}^{r'} U_p(\mathbf{r}_i)\, \widetilde{K}_{pq}(\mathbf{r}_0)\, V_q^T(\mathbf{r}_j) + \varepsilon$$

Another View

Using a low rank approximation,

$$K(\mathbf{r}_i + \mathbf{r}_0 + \mathbf{r}_j) = \sum_{p=1}^{r} \sum_{q=1}^{r'} U_p(\mathbf{r}_i)\, \widetilde{K}_{pq}(\mathbf{r}_0)\, V_q^T(\mathbf{r}_j) + \varepsilon$$

We define "multipole" and "local" fields at the nodes of a tree:

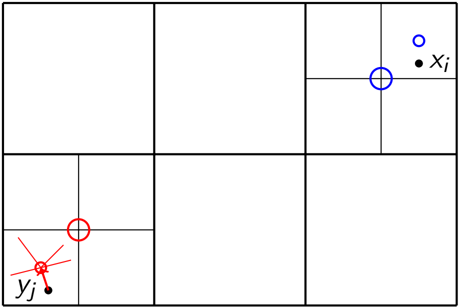


Another View

Using a low rank approximation,

$$K(\mathbf{r}_i + \mathbf{r}_0 + \mathbf{r}_j) = \sum_{p=1}^{r} \sum_{q=1}^{r'} U_p(\mathbf{r}_i)\, \widetilde{K}_{pq}(\mathbf{r}_0)\, V_q^T(\mathbf{r}_j) + \varepsilon$$

We define "multipole" and "local" fields at the nodes of a tree:

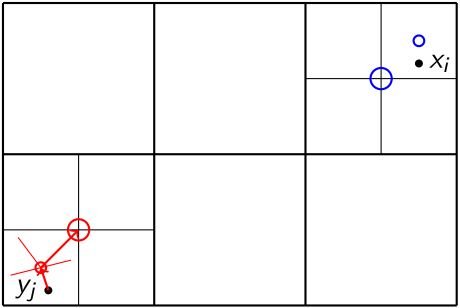


Another View

Using a low rank approximation,

$$K(\mathbf{r}_i + \mathbf{r}_0 + \mathbf{r}_j) = \sum_{p=1}^{r} \sum_{q=1}^{r'} U_p(\mathbf{r}_i)\, \widetilde{K}_{pq}(\mathbf{r}_0)\, V_q^T(\mathbf{r}_j) + \varepsilon$$

We define "multipole" and "local" fields at the nodes of a tree:

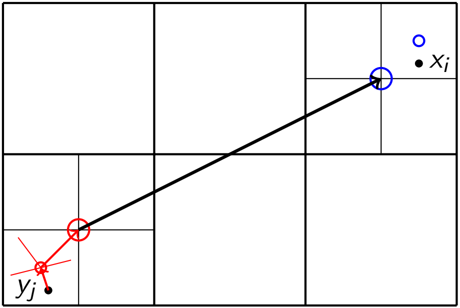


Another View

Using a low rank approximation,

$$K(\mathbf{r}_i + \mathbf{r}_0 + \mathbf{r}_j) = \sum_{p=1}^{r} \sum_{q=1}^{r'} U_p(\mathbf{r}_i)\, \widetilde{K}_{pq}(\mathbf{r}_0)\, V_q^T(\mathbf{r}_j) + \varepsilon$$

We define "multipole" and "local" fields at the nodes of a tree:

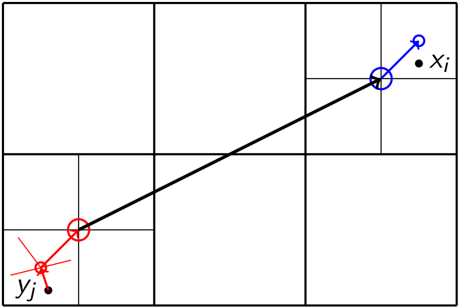


Another View

Using a low rank approximation,

$$K(\mathbf{r}_i + \mathbf{r}_0 + \mathbf{r}_j) = \sum_{p=1}^{r} \sum_{q=1}^{r'} U_p(\mathbf{r}_i)\, \widetilde{K}_{pq}(\mathbf{r}_0)\, V_q^T(\mathbf{r}_j) + \varepsilon$$

We define "multipole" and "local" fields at the nodes of a tree:

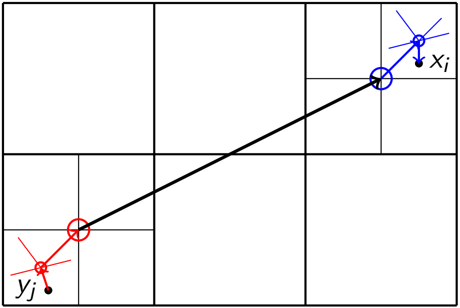


Another View

Using a low rank approximation,

$$K(\mathbf{r}_i + \mathbf{r}_0 + \mathbf{r}_j) = \sum_{p=1}^{r} \sum_{q=1}^{r'} U_p(\mathbf{r}_i)\, \widetilde{K}_{pq}(\mathbf{r}_0)\, V_q^T(\mathbf{r}_j) + \varepsilon$$

We define "multipole" and "local" fields at the nodes of a tree:



Operators

$$\mathbf{K}_{IJ} = U_{C(I)} \widetilde{U}_I \widetilde{K}_{IJ} \widetilde{V}_J^T V_{C(J)}^T$$

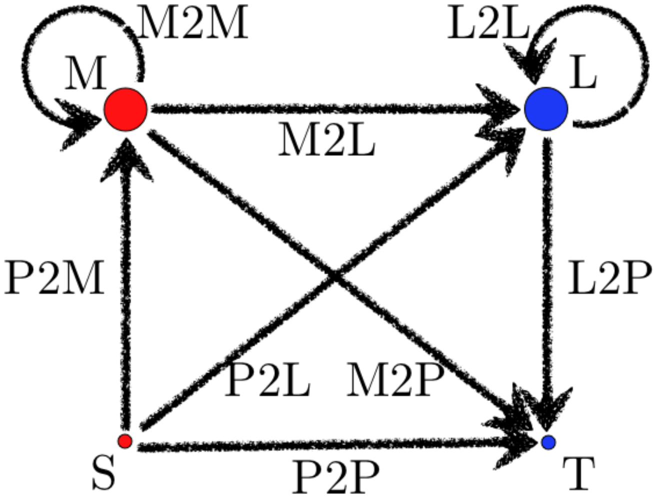
Abstraction of each operator:

- P2P: $K(t, s)$
- P2M: V^T and/or $\widetilde{V}^T V^T$
- M2M: $\widetilde{V}^T$
- M2L: $\widetilde{K}$
- L2L: $\widetilde{U}$
- L2P: U and/or $U\widetilde{U}$
- MAC: Boolean function of two clusters
  - Weak - Adjacent clusters are accepted (are low-rank).
  - Strong - Well-separated clusters are accepted (are low-rank).
  - Dynamic - Function of the multipole values.

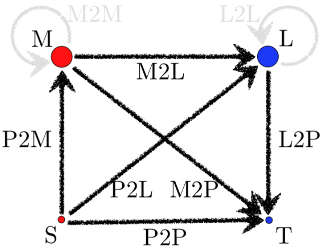
Useful/Common additions:

- P2L: $\widetilde{K}\widetilde{V}^T V^T$
- M2P: $U\widetilde{U}\widetilde{K}$

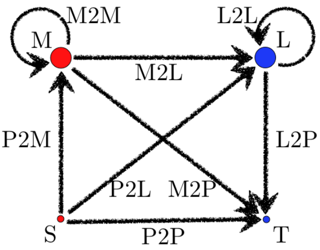
Operator Graph



Special Cases

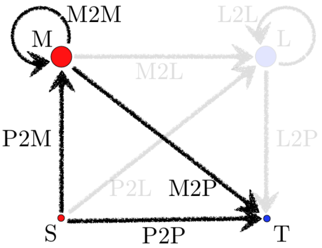


HODLR (Weak MAC)
$\mathcal{H}$ (Strong MAC)



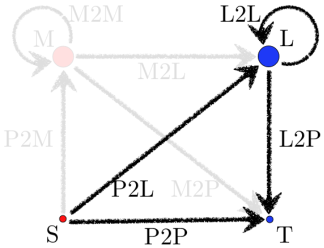
HSS (Weak MAC)
$\mathcal{H}^2$ (Strong MAC)

Special Cases



with strong/dynamic MAC.

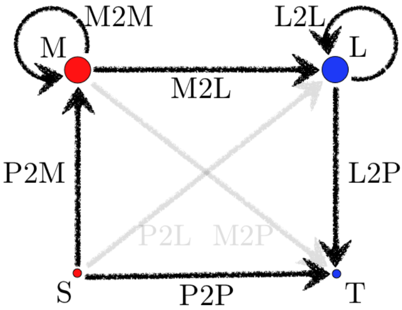
Cluster-Particle Treecode



with strong/dynamic MAC.

Particle-Cluster Treecode

Special Cases

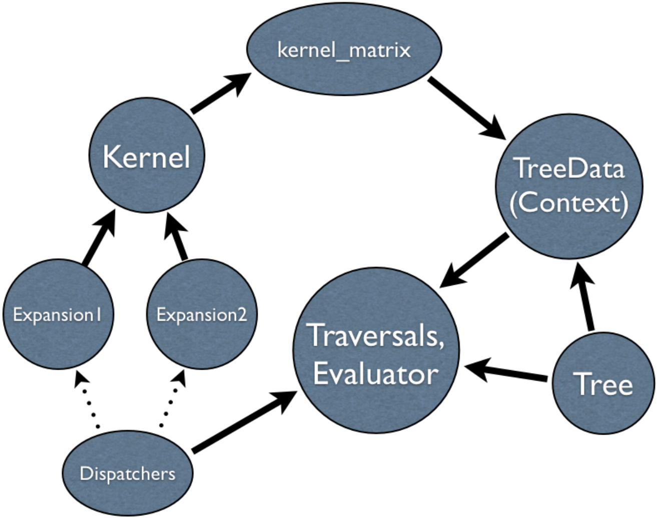


with strong MAC.

Classic FMM.

Code Design

Separate the components of a hierarchical code



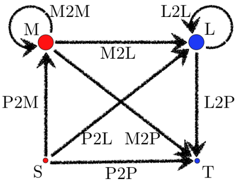
MyKernel

```
struct MyKernel
 : public fmmtl::Kernel<MyKernel> {
    typedef Vec<3,double> source_type;
    typedef double        charge_type;
    typedef Vec<3,double> target_type;
    typedef double        result_type;

    typedef double        kernel_value_type;

    kernel_value_type operator()(const target_type& t,
                                 const source_type& s) const {
        // Compute K(t,s)
    }
    /** OPTIONAL! **/
    kernel_value_type transpose(const kernel_value_type& kts) const {
        // Compute K(s,t) from K(t,s) if possible
    }
};
```

MyExpansion

```
struct MyExpansion
 : public fmmtl::Expansion<MyKernel, MyExpansion> {

    /** For use in defining and building the tree
     * source_type and target_type must be convertible to a point_type:
     *   static_cast<point_type>(source_type)
     *   static_cast<point_type>(target_type) */
    static constexpr unsigned dimension = 3;
    typedef Vec<dimension,double> point_type;

    typedef std::vector<double> multipole_type;
    typedef std::vector<double> local_type;

    ...

    void P2M(const source_type& s, const charge_type& c,
             const point_type& center, multipole_type& M) const {
        // Compute M += V^T(s) * c
    }

    ...
};
```

Kernel/Expansion



- Kernel stored in `.kern` file.
  - May be templated and/or contain state (e.g. κ).
  - Multiple architecture support: CPU/GPU compilation.
  - Optional `transpose` and vectorized P2P methods.
- Expansion stored in `.hpp` file.
  - Implements pathway(s) through the operator graph.
  - Optional vectorized P2X and X2P methods.
- Statically detect available computational pathways (SFINAE).

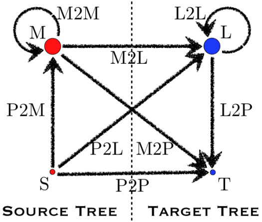
Multipole Acceptance

In development...

```
/** Strong/Weak cluster interactions - well-separated criteria */
bool MAC(const point_type& tbox_extents, const point_type& tbox_center,
         const point_type& sbox_extents, const point_type& sbox_center) const {
}

/** Dynamic criteria */
bool MAC(const multipole_type& M, const local_type& L) const {
}
```

Tree(s)

- $\{s_j\} \equiv \{t_i\}$
  - Single Tree
- $\{s_j\} \not\equiv \{t_i\}$
  - Dual Tree



- Whether single tree or dual tree, a `TreeContext` shall provide

```
concept TreeContext {
    typename source_tree_type;
    typename target_tree_type;

    [const] source_tree_type& source_tree() [const];
    [const] target_tree_type& target_tree() [const];

    permuted_iterator source_tree_permute(data_iter di,
                                          source_body_iter sbi) const;
    permuted_iterator target_tree_permute(data_iter di,
                                          target_body_iter sbi) const;
};
```

- Traversals and evaluators conceptually work with dual trees.

Tree data

- Data to be associated with (not implemented with!) a Tree.
  - Facilitates optimization (data structures/parallelization).
  - Promotes code independence.
- Source Box data: M
- Target Box data: L
- Source Body data: source, charge
- Target Body data: target, result

- Stored independently, but associated with boxes and bodies:

```
concept DataContext {
    multipole_type& multipole(    const source_box_type& sbox);
    local_type&     local(        const target_box_type& sbox);
    ...
    source_iterator source_begin(const source_box_type& sbox) const;
    source_iterator source_end(  const source_box_type& sbox) const;
    ...
    result_iterator result_begin(const target_box_type& tbox) const;
    result_iterator result_end(  const target_box_type& tbox) const;
    ...
};
```

e.g.

```
#incldue <fmmtl/KernelMatrix.hpp>
#include "MyExpansion.hpp"

int main() {
    typedef MyExpansion expansion;
    expansion K(...);                     // Expansion order, error target, etc
    ...
    std::vector<source_type> s = ...
    std::vector<charge_type> c = ...

    std::vector<target_type> t = ...

    fmmtl::KernelMatrix<expansion> M = K(t,s);                 // Construct
    fmmtl::set_options(M, opts);                               // Set options?
    ...
    std::vector<result_type> r_apprx = M * c;                  // FMM/Treecode
    std::vector<result_type> r_exact = fmmtl::direct(M * c);  // O(N^2)
    ...
    M.expansion().set_something(...);    // Mutate the expansion
    ...
}
```

Examples in FMMTL.

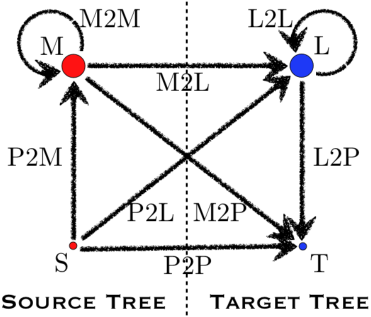
Applications

- `KernelSkeleton.kern` – Documentation
- `UnitKernel.kern`, `ExpKernel.kern` – Testing
- `Laplace.kern` – single layer, double layer, both, BEM
  - `LaplaceSpherical.hpp`
  - `LaplaceCartesian.hpp`
  - `LaplaceSphericalBEM.hpp`
- `Yukawa.kern` – single layer, double layer, both, BEM
  - `YukawaCartesian.hpp`
  - `YukawaSpherical.hpp`
  - `YukawaCartesianBEM.hpp`
- `Stokes.kern` – single layer, double layer, both, BEM
  - `StokesSpherical.hpp`
  - `StokesSphericalBEM.hpp`
- `Helmholtz.kern` – single layer
  - `HelmholtzFourier.hpp`
- Generalized
  - `KIExpansion.hpp` – KIFMM (in progress)
  - `bbExpansion.hpp` – bbFMM (wanted!)

Current Work

- Comparison and research into Expansions and Traversals.
  - Fair comparison between bbFMM and KIFMM.
  - Scheduling and parallelization research.
  - Development/testing of new expansions [Harvard, UMass]
- Collaborations:
  - Hans Johnson, UMass - Development/testing of expansions.
  - Lorena Barba, BU - BEM with inexact GMRES: Laplace, Stokes.
  - Rio Yokota, KAUST - Fast preconditioners using FMM.
  - L Mahadevan, Harvard - Integral polyharmonic PDE solvers.
  - Brian Quaife, UT Austin - Quaterniotic FMM expansions.
- Applications
  - Stokes BEM (Boston University)
  - Biharmonic BEM (Harvard University)
  - Laplace NBody (Harvard University)
  - Wanted: more!

Open Problems



- Swapping operators
  - M2L $\Rightarrow$ M2P
  - M2L $\Rightarrow$ P2L
  - M2L $\Rightarrow$ P2P
- Balancing P2P and M2L.
- Scheduling.

Future Work

- Collect more Kernels/Expansions!
  - If you have one (especially if it's odd...) contact me!
- Continue development:
  - Parallelization – MPI, more OpenMP, more GPU.
  - Classic matrix interface
    - Sugar + interop with (e.g.) Eigen, uBlas, ViennaCL, Python

- Million-dollar scheduling/autotuning questions:
  - Given a subset of the operations

$$\{P2P, P2M, P2L, M2M, M2L, M2P, L2L, L2P\} \quad \text{(Statically)},$$

(+potentially cost estimates/target error) and a

$$MAC \quad \text{(Static/Dynamic)},$$

and a

$$\text{[Dual] Tree} \quad \text{(Dynamically)},$$

find an optimal dispatch such that all source/target pairs interact.