Using SpF to Achieve Petascale for Legacy Pseudospectral Applications

Dynamics of Planetary and Stellar Interiors Conference
July 28-31, 2014 — La Jolla, CA

Thomas Clune
Weiyuan Jiang

NASA Goddard Space Flight Center

July 29, 2014
Background/Motivation

NASA HEC supports at least 5 pseudospectral applications:

**Spherical Geometry**
- DYNAMO
- MoSST
- ASH

**Cartesian Geometry**
- HPS
- DDSCAT
NASA HEC supports at least 5 pseudospectral applications:

**Spherical Geometry**
- DYNAMO
- MoSST
- ASH

**Cartesian Geometry**
- HPS
- DDSCAT

All except ASH use a 1D decomposition:
**Background/Motivation**

NASA HEC supports at least 5 pseudospectral applications:

<table>
<thead>
<tr>
<th>Spherical Geometry</th>
<th>Cartesian Geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>• DYNAMO</td>
<td>• HPS</td>
</tr>
<tr>
<td>• MoSST</td>
<td>• DDSCAT</td>
</tr>
<tr>
<td>• ASH</td>
<td></td>
</tr>
</tbody>
</table>

All except ASH use a 1D decomposition:
- limits scalability/performance
- constrains **grid resolution**
NASA HEC supports at least 5 pseudospectral applications:

**Spherical Geometry**
- DYNAMO
- MoSST
- ASH

**Cartesian Geometry**
- HPS
- DDSCAT

All except ASH use a 1D decomposition:
- limits scalability/performance
- constrains *grid resolution*

( Mostly) my fault!
Consequences of 1D decomposition

Scaling Legendre Transforms

- 1D Radial (T480x120L)
- 1D Wavenumber (T480x120L)
- 2D Extrapolation (T480x129)
Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective.
Other motivations

Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective

• Alternate between *local* computation and all-to-all communication
Other motivations

Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective

- Alternate between *local* computation and all-to-all communication
- Complicated data structures (harmonic truncation)
Other motivations

Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective

- Alternate between *local* computation and all-to-all communication
- Complicated data structures (harmonic truncation)
- Nontrivial load-balance
Other motivations

Pseudospectral methods have an elegant structure that provides quite interesting challenges from a software design perspective:

- Alternate between *local* computation and all-to-all communication
- Complicated data structures (harmonic truncation)
- Nontrivial load-balance
- Most numerical calculations can be done with vendor-optimized libraries
Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.
SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

- **Support multiple geometries (sphere, box, ...?)**
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- Enable *decomposition independent* formulation of applications
- Allow user extensions/refinements (OO)
- Enable users to focus on *science*
SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)
- Support multiple geometries (sphere, box, ...?)
- **Manage:** domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- Enable *decomposition independent* formulation of applications
- Allow user extensions/refinements (OO)
- Enable users to focus on *science*
SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

• Support multiple geometries (sphere, box, ...?)
• Manage: domain decomposition, transpose, and I/O operations
• Leverage optimized numerical libraries
• Support async communication, hybrid-parallelism and HW accelerators
• Enable decomposition independent formulation of applications
• Allow user extensions/refinements (OO)
• Enable users to focus on science
SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

- Support multiple geometries (sphere, box, ...?)
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- **Support async communication, hybrid-parallelism and HW accelerators**
- Enable *decomposition independent* formulation of applications
- Allow user extensions/refinements (OO)
- **Enable users to focus on science**
SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)

- Support multiple geometries (sphere, box, ...?)
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- **Enable decomposition independent formulation of applications**
- Allow user extensions/refinements (OO)
- **Enable users to focus on science**
SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)
- Support multiple geometries (sphere, box, ...?)
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- Enable decomposition independent formulation of applications
- Allow user extensions/refinements (OO)
- Enable users to focus on science
Solution - SpF

SpF (Spectral Framework) is a software framework tailor designed to maximize the performance and scalability of pseudospectral applications.

Specific design goals: (separation of concerns)
- Support multiple geometries (sphere, box, ...?)
- Manage: domain decomposition, transpose, and I/O operations
- Leverage optimized numerical libraries
- Support async communication, hybrid-parallelism and HW accelerators
- Enable \textit{decomposition independent} formulation of applications
- Allow user extensions/refinements (OO)
- \textbf{Enable users to focus on science}
Benefits of adopting SpF

- Less duplication of effort
  - Parallel “transforms” — Legendre, LU Decomposition, etc.
  - Tedious/fragile transpose implementations
- Reduced effort to exploit new architectures/accelerators
- Readily adopt/share performance innovations within the community
Challenges and Complications

Nonlinear terms

FFT

Legendre

Chebyshev

Implicit Update

T. Clune  SpF - DPSIC  7/31
Challenges and Complications

- Triangular
- Trapezoidal (or pentagonal)
- Rhomboidal
SpF: The Secret Sauce

Each transform can be expressed as a union of independent, atomic (indivisible) numerical ‘kernels’ \( \{K_1, K_2, \ldots, K_n\} \).
SpF: The Secret Sauce

Each transform can be expressed as a union of independent, atomic (indivisible) numerical ‘kernels’ \( \{K_1, K_2, \ldots, K_n\} \).

These provide a natural partition of the computational domain:

\[
(X, d_x) = \left( (\tilde{X}_1^1, d_x^1 \otimes q^1) \oplus (\tilde{X}_2^2, d_x^2 \otimes q^2) \oplus \ldots \oplus (\tilde{X}_n^n, d_x^n \otimes q^n) \right)
\]

\[
(Y, d_y) = \left( (\tilde{Y}_1^1, d_y^1 \otimes q^1) \oplus (\tilde{Y}_2^2, d_y^2 \otimes q^2) \oplus \ldots \oplus (\tilde{Y}_n^n, d_y^n \otimes q^n) \right)
\]

\[
Y = F(X) \implies \tilde{Y}_q^i = K_i(\tilde{X}_q^i), \; i = 1, 2, \ldots, n
\]
Each transform can be expressed as a union of independent, atomic (indivisible) numerical ‘kernels’ \( \{K_1, K_2, \ldots, K_n\} \).

These provide a natural partition of the computational domain:

\[
(X, d_x) = \left( (\tilde{X}^1_q, d^1_x \otimes q^1) \oplus (\tilde{X}^2_q, d^2_x \otimes q^2) \oplus \ldots \oplus (\tilde{X}^n_q, d^n_x \otimes q^n) \right)
\]

\[
(Y, d_y) = \left( (\tilde{Y}^1_q, d^1_y \otimes q^1) \oplus (\tilde{Y}^2_q, d^2_y \otimes q^2) \oplus \ldots \oplus (\tilde{Y}^n_q, d^n_y \otimes q^n) \right)
\]

\[Y = F(X) \implies \tilde{Y}^i_q = K_i(\tilde{X}^i_q), i = 1, 2, \ldots, n\]
SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
  - Most user customization is here
SpF: Key Software Abstractions

• Kernel - Indivisible unit of algorithm
  • Most user customization is here
• Permutor - *Automatic* generation of communication logic for global transposes
SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
  - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)
SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
  - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)
- IndexSpace - Describes *arbitrary* data layout within/across PEs
### SpF: Key Software Abstractions

- **Kernel** - Indivisible unit of algorithm
  - Most user customization is [here](#)
- **Permutor** - *Automatic* generation of communication logic for global transposes
- **Distributor** - Responsible for domain decomposition (including load balance)
- **IndexSpace** - Describes *arbitrary* data layout within/across PEs
- **Field** - Local array and associated IndexSpace
  - Lingua franca within the framework
  - Input/output for kernel
SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
  - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)
- IndexSpace - Describes arbitrary data layout within/across PEs
- Field - Local array and associated IndexSpace
  - Lingua franca within the framework
  - Input/output for kernel
- FieldList - Local collection of Field objects
  - In/out for permutation
SpF: Key Software Abstractions

- **Kernel** - Indivisible unit of algorithm
  - Most user customization is here
- **Permutor** - *Automatic* generation of communication logic for global transposes
- **Distributor** - Responsible for domain decomposition (including load balance)
- **IndexSpace** - Describes arbitrary data layout within/across PEs
- **Field** - Local array and associated IndexSpace
  - *Lingua franca* within the framework
  - *Input/output* for kernel
- **FieldList** - Local collection of Field objects
  - *In/out* for permutation
- **LinearSolver** - Implicit updates
SpF: Key Software Abstractions

- Kernel - Indivisible unit of algorithm
  - Most user customization is here
- Permutor - *Automatic* generation of communication logic for global transposes
- Distributor - Responsible for domain decomposition (including load balance)
- IndexSpace - Describes arbitrary data layout within/across PEs
- Field - Local array and associated IndexSpace
  - Lingua franca within the framework
  - Input/output for kernel
- FieldList - Local collection of Field objects
  - In/out for permutation
- LinearSolver - Implicit updates
- Integrator - time integration (CN, AB, ...)
SpF: Implementation details

- Object-oriented design (ala Fortran 2003)
  - Applications built by *extending* SpF abstractions
  - User-extensions that can be shared by community
- Aggressive use of test-driven development (TDD) & pFUnit
  - More that 300 unit tests
  - Runs on at least 3 compilers (Intel, GNU, NAG)

- Demonstrated with multi-layer shallow water
- Not quite ready for distribution
  - Open source release planned (tedious paperwork)
  - Project-level release could be expedited
How SpF sees an application

Transform

p=0

F → T → F

p=1

F → T → F

p=2

F → T → F

p=3

F → T → F

p=4

F → T → F

Permuted

Transform
use SpF_mod

class (IndexSpace) :: cartesian

type (RangeAxis) :: xAxis, yAxis, zAxis

xAxis = RangeAxis(’x’, nx)
yAxis = RangeAxis(’y’, ny)
zAxis = RangeAxis(’z’, nz)

allocate(cartesian, source= xAxis*yAxis*zAxis)
IndexSpace - Cartesian Bundle

```plaintext
use Spf_mod

class (IndexSpace) :: cartesianBundle

type (RangeAxis) :: xAxis, yAxis, zAxis

type (StringAxis) :: qtys

xAxis = RangeAxis(’x’, nx)
yAxis = RangeAxis(’y’, ny)
zAxis = RangeAxis(’z’, nz)
qtys = StringAxis(’qty’, [’W’, ’Z’, ’S’, ’P’])

allocate(cartesianBundle, source= &
    & xAxis*yAxis*zAxis*qtys)
```
use SpF_mod

class (IndexSpace) :: tDomain

class (OuterProductSpace) :: modeAxis

type (RangeAxis) :: rAxis

modeAxis = RangeAxis(’m’, 0, 0) * RangeAxis(’ell’, 0, Lmax)
Allocate(tDomain, source=mode)

do m = 1, mMax
  modeAxis = RangeAxis(’m’, m, m) * RangeAxis(’ell’, m, Lmax)
  allocate(tDomain, source= tDomain + modeAxis)
end do

allocate(tDomain, source= RangeAxis(’r’, nn)*tDomain)
Automating the transpose

First we translate the index space into a labelled table:

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>$m$</th>
<th>$r$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>10</td>
<td>'Z'</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>$m$</th>
<th>$r$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>'W'</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
<td>'W'</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>2</td>
<td>'P'</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Automating the transpose

Then we append process and offset metadata:

<table>
<thead>
<tr>
<th>ℓ</th>
<th>m</th>
<th>r</th>
<th>f</th>
<th>PE</th>
<th>δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>10</td>
<td>'Z'</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ℓ</th>
<th>m</th>
<th>r</th>
<th>f</th>
<th>PE</th>
<th>δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>'W'</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
<td>'W'</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>2</td>
<td>'P'</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Automating the transpose

Then we append process and offset metadata:

<table>
<thead>
<tr>
<th>ℓ</th>
<th>m</th>
<th>r</th>
<th>f</th>
<th>PE</th>
<th>δ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>10</td>
<td>'Z'</td>
<td>8</td>
<td>15</td>
</tr>
</tbody>
</table>

Then we “co-sort” the tables to find source/destination for each element

<table>
<thead>
<tr>
<th>ℓ</th>
<th>m</th>
<th>r</th>
<th>f</th>
<th>ps</th>
<th>δs</th>
<th>pd</th>
<th>δd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>10</td>
<td>'Z'</td>
<td>8</td>
<td>15</td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>
Automating the transpose

For a 2-phase (nested) transpose, we append the rank for each phase.

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>$m$</th>
<th>$r$</th>
<th>$f$</th>
<th>$PE_0$</th>
<th>$PE_1$</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>'S'</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>10</td>
<td>'Z'</td>
<td>8</td>
<td>2</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>$m$</th>
<th>$r$</th>
<th>$f$</th>
<th>$PE_0$</th>
<th>$PE_1$</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
<td>3</td>
<td>'W'</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
<td>'W'</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>2</td>
<td>'P'</td>
<td>8</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Primary configuration

- Azimuthal wavenumbers distributed over PEs
- Constraint $N_p \leq N_m$
- Supports variant spectral truncations and variant hyperviscosity terms
Primary configuration

- One dimensional Distribution over PEs at all stages
- Constraint: $N_p \leq N_m$
- Constraint: $N_p \leq N_r$
- Constraint: All Spherical transforms (Legendre and FFT) are in the same process
Adopting SpF - general strategy

1. **Establish regression tests and data for baseline.**
   - Invest in achieving *strong reproducibility*
   - Turn off optimization and turn on debugging flags

2. Proceed with incremental changes that preserve results

3. Commit to repository after each success.

4. Minor roundoff issues may be encountered
   - Isolate cause, then update baseline regression data
   - Bracket change in repository
Adopting SpF - copy to/from legacy data structures

1. Declare a FieldList object
2. Create a procedure that copies an array into a Field
3. For each contiguous array
   1. Define corresponding IndexSpace domain object
   2. Call append() method on FieldList
   3. Insert call to copy procedure just prior to use
Adopting SpF - Kernel Factory

1. Create a new module:
   1. Define a derived type that extends KernelFactory
   2. Implement methods that compute Kernel IndexSpace (I/O)
   3. Define a derived type that extends Kernel
   4. Implement apply() method that wraps actual computation

2. Declare and initialize in main code:
   1. new Factory defined above
   2. Distributor, Permutor
   3. TaskList, and 2 FieldLists (in and out)
   4. Build task list, and field lists using distributor and factory
   5. Build permutor object connecting previous transform to new

3. Use in main loop:
   1. Insert call to apply() method of TaskList object
Adoption status

MoSST
  • Now uses SpF permutations

DYNAMO
  • SpF conversion completed for
    • Legendre transforms
    • Quadratic convolution
    • Stream to vector (i.e. \( \{ W, Z, \ldots \} \rightarrow \{ v_r, v_\theta, \ldots \} \))
    • Permutations (including to/from legacy layout)
  • Took \( \approx 1 \) week for expert (me)
    • Lots of ugly shortcuts
  • Issues encountered with implicit update step
    • Could “cheat”
    • Will use experience to instead improve framework
Example - top declaration

type (SimpleMpiDistributor) :: d

type (FieldList) :: leg_in, leg_out, NL_in, NL_out

type (LegendreFactory) :: legFactory

type (NL_ConvolutionFactory) :: NL_Factory

type (PartitionedAlgorithm) :: legTasks, NL_tasks

type (SimpleMpiIPermutor) :: perm

class (IndexSpace) :: initialDomain


d = SimpleMpiDistributor(MPI_communicator)

legFactory = LegendreFactory(mMax=1023)

NL_Factory = NL_ConvolutionFactory(ni, nk)

initialDomain = ...
Example - initialization

```plaintext
1 legTasks = d%distribute(legFactory, initialDomain)
2 leg_in = FieldList(legTasks,'in')
3 leg_out = FieldList(legTasks,'out')
4
5 NL_tasks = d%distribute(NL_Factory, leg_in)
6 NL_in = FieldList(NL_tasks,'in')
7 NL_out = FieldList(NL_tasks,'out')
8
9 perm = SimpleMpiPermutor(MPI_communicator, leg_out, NL_in)
```
Example - execute

1  ...  
2  call legTasks%apply(leg_in, leg_out)  
3  call perm%permute(leg_out, NL_in)  
4  call NL_tasks%apply(NL_in, NL_out)  
5  ...
## Variations

<table>
<thead>
<tr>
<th></th>
<th>Alternate load balancing strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>type (SimpleMpiDistributor) :: d</td>
</tr>
<tr>
<td>3</td>
<td>type (RoundRobinDistributor) :: d</td>
</tr>
</tbody>
</table>
Variations

1 ! Alternate load balancing strategy
2 ! type (SimpleMpiDistributor) :: d
3 type (RoundRobinDistributor) :: d

5 ! Alternative permutation strategy
6 ! type (SimpleMpiPermutor) :: perm
7 type (SomeOtherPermutor) :: perm
Variations

1  ! Alternate load balancing strategy
2  ! type (SimpleMpiDistributor) :: d
3  type (RoundRobinDistributor) :: d

5  ! Alternative permutation strategy
6  ! type (SimpleMpiPermutor) :: perm
7  type (SomeOtherPermutor) :: perm

9  ! Alternative Legendre implementation
10  ! type (LegendreFactory) :: legFactory
11  type (AltLegFactory) :: legFactory
Next steps

• Finish ports of DYNAMO, MoSST, HPS, DDSCAT
• Improve framework
  • Generalize/optimize Permutor classes
    • Allow for multiple sources
    • Allow for “subsetting”
    • Implement multiphase transpose (ala Nick Featherstone)
• Extend/improve kernels
  • Better mechanism for defining offsets
  • Allow for multiple sources/destinations
  • Allow for “fat” kernels that do internal communication (e.g. implicit treatment of coriolis)
• Release SpF as open source
Credits

- NASA High End Computing program for supporting this work
- Gary Glatmaier - for providing DYNAMO as an interesting challenge
Questions?