A Software Strategy for Pseudospectral Methods at Petascale
(A Last Hurrah?)

Tom Clune

Software Systems Support Office
Earth Science Division
NASA Goddard Space Flight Center

October 8, 2012
Outline

1. Performance Modeling
2. Software Engineering
3. Status
Performance Requirements for 1 Second per Iteration

- > 1 TF
- > 10 TF
- > 100 TF
- > 1 PF
- > 10 PF

Radial levels vs. $L_{max}$

- T480x129
- T960x257
- T1920x513
- T3840x1025
Scaling Legendre Transforms
Scaling Legendre Transforms

1D Radial
(T480x120L)

TF vs # cores

1D Radial
(T480x120L)
Scaling Legendre Transforms

1D Radial (T480x120L)

TF vs. # cores

1D Radial

1D Wavenumber (T480x120L)

TF vs. # cores
Scaling Legendre Transforms

1D Radial (T480x120L)

1D Wavenumber (T480x120L)
Scaling Legendre Transforms

1D Radial (T480x120L)

1D Wavenumber (T480x120L)

TF vs. # cores

TF vs. # cores
Scaling Legendre Transforms

1D Radial (T480x120L)

1D Wavenumber (T480x120L)

2D Extrapolation (T480x129)
Beyond 2D ...
Beyond 2D ...

Combine Multiple Fields
Legendre Transform (T480x129)
Beyond 2D ...

Combine Multiple Fields
Legendre Transform (T480x129)

2D+ (extrapolation)
2D+ 2D (extrapolation) 2D
1D wavenumber
1D radial

TF
# cores

Tom Clune (SSSO)

SPF - CIG

October 8, 2012
Beyond 2D ...

Combine Multiple Fields
Legendre Transform (T480x129)

Distribute Product Space (RxF)
Legendre Transform (T480x129)
Beyond 2D ...

Combine Multiple Fields
Legendre Transform (T480x129)

Distribute Product Space (RxF)
Legendre Transform (T480x129)
Communication Time = Computation Time
Latency = 1 µsec; T960x257L
Balanced Computation and Communication

Single Field; Latency = 1 μsec

Grouped Fields; Latency = 1 μsec

Grouped Fields; Latency = 10 μsec

Single Field; Latency = 10 μsec

# cores x BW

Communication vs. Computation

1D, 2D, 2 Phase, Recursive, 3D, Naive
Outline

1. Performance Modeling
2. Software Engineering
3. Status
Issues and Options

Maximizing granularity
Balancing loads
Global transpose
Hardware accelerators

▶ GPUs
▶ MIC (Intel)

Robust, parallel I/O

Variant paradigms
▶ Hybrid OpenMP-MPI
▶ Co-Array Fortran

Variants
⋆ All-to-all 1D
⋆ All-to-all 2D
⋆ Recursive
⋆ Overlap with computation?

Arbitrary data layouts?

Variant truncations
▶ Triangular
▶ Rhomboidal
▶ Trapezoidal
▶ Hyperviscosity

Variant radial schemes
Implicit coriolis
Solid inner core
Issues and Options

- Maximizing granularity
Issues and Options

- Maximizing granularity
- Balancing loads
Issues and Options

- Maximizing granularity
- Balancing loads
- **Global transpose**
  - Variants
    - All-to-all 1D
    - All-to-all 2D
    - Recursive
    - Overlap with computation?
  - Arbitrary data layouts?
Issues and Options

- Maximizing granularity
- Balancing loads
- Global transpose
- **Hardware accelerators**
  - GPUs
  - MIC (Intel)
Issues and Options

- Maximizing granularity
- Balancing loads
- Global transpose
- Hardware accelerators
  - GPUs
  - MIC (Intel)
- Robust, parallel I/O
Issues and Options

- Maximizing granularity
- Balancing loads
- Global transpose
- Hardware accelerators
  - GPUs
  - MIC (Intel)
- Robust, parallel I/O
- **Variant paradigms**
  - Hybrid OpenMP-MPI
  - Co-Array Fortran
Issues and Options

- Maximizing granularity
- Balancing loads
- Global transpose
- Hardware accelerators
  - GPUs
  - MIC (Intel)
- Robust, parallel I/O
- Variant paradigms
  - Hybrid OpenMP-MPI
  - Co-Array Fortran

- Variant truncations
  - Triangular
  - Rhomboidal
  - Trapezoidal
  - Hyperviscosity
- Variant radial schemes
- Implicit coriolis
- Solid inner core
Issues and Options

- Maximizing granularity
- Balancing loads
- Global transpose
- Hardware accelerators
  - GPUs
  - MIC (Intel)
- Robust, parallel I/O
- Variant paradigms
  - Hybrid OpenMP-MPI
  - Co-Array Fortran
- Variant truncations
  - Triangular
  - Rhomboidal
  - Trapezoidal
  - Hyperviscosity
- Variant radial schemes
- Implicit coriolis
- Solid inner core
Issues and Options

- Maximizing granularity
- Balancing loads
- Global transpose
- Hardware accelerators
  - GPUs
  - MIC (Intel)
- Robust, parallel I/O
- Variant paradigms
  - Hybrid OpenMP-MPI
  - Co-Array Fortran
- Variant truncations
  - Triangular
  - Rhomboidal
  - Trapezoidal
  - Hyperviscosity
- Variant radial schemes
- Implicit coriolis
- Solid inner core
Issues and Options

- Maximizing granularity
- Balancing loads
- Global transpose
- Hardware accelerators
  - GPUs
  - MIC (Intel)
- Robust, parallel I/O
- Variant paradigms
  - Hybrid OpenMP-MPI
  - Co-Array Fortran
- Variant truncations
  - Triangular
  - Rhomboidal
  - Trapezoidal
  - Hyperviscosity
- Variant radial schemes
- Implicit coriolis
- **Solid inner core**
Definition: A software framework is an abstraction which provides generic functionality to improve efficiency of developing/customizing applications.
Implementation Strategy: SpF (Spectral Framework)

**Definition:** A software *framework* is an abstraction which provides generic functionality to improve efficiency of developing/customizing applications.

**Benefits:**
- Enables extreme limits of granularity
- Reduces duplication of effort across multiple investigation teams
- Efficient propagation of improvements
- Accelerates model extensions/variants
Implementation Strategy: SpF (Spectral Framework)

**Definition:** A software *framework* is an abstraction which provides generic functionality to improve efficiency of developing/customizing applications.

**Benefits:**
- Enables extreme limits of granularity
- Reduces duplication of effort across multiple investigation teams
- Efficient propagation of improvements
- Accelerates model extensions/variants

**What you won’t like:**
- Advanced syntax (F2003)
- Tracing your code
- Debugging
Implementation Strategy: SpF (Spectral Framework)

**Definition:** A software *framework* is an abstraction which provides generic functionality to improve efficiency of developing/customizing applications.

**Benefits:**
- Enables extreme limits of granularity
- Reduces duplication of effort across multiple investigation teams
- Efficient propagation of improvements
- Accelerates model extensions/variants

**What you won’t like:**
- Advanced syntax (F2003)
- Tracing your code
- Debugging

**Other relevant frameworks:**
- **CHOMBO** - adaptive mesh refinement
- **CCA** - parallel interoperability
- **CACTUS** - various; originated with general relativity community
- **ESMF** - Earth system model interoperability
SpF - Primary Abstractions

IndexSpace
- supports arbitrary data layout and processor distribution

Field
- data values on an index space

Operations
▶ Kernel
- indivisible unit of computational work
▶ Operation
- family of related non-overlapping kernels
▶ KernelDriver
- coordinates with Partitioner and Transposer

Transposer
- move data between to distributions of an index space

Partitioner
- subdivides global index space and operations across

Miscellaneous others
▶ IO Manager
- special type of Operation for interacting with filesystem
▶ Logger
- monitor job status and history
▶ Driver
- executes operations and transposes iteratively
SpF - Primary Abstractions

- IndexSpace - supports arbitrary data layout and processor distribution
SpF - Primary Abstractions

- **IndexSpace** - supports arbitrary data layout and processor distribution
- **Field** - data values on an index space
SpF - Primary Abstractions

- IndexSpace - supports arbitrary data layout and processor distribution
- Field - data values on an index space
- Operations
SpF - Primary Abstractions

- IndexSpace - supports arbitrary data layout and processor distribution
- Field - data values on an index space
- Operations
  - Kernel - indivisible unit of computational work
SpF - Primary Abstractions

- IndexSpace - supports arbitrary data layout and processor distribution
- Field - data values on an index space
- Operations
  - Kernel - indivisible unit of computational work
  - Operation - family of related non-overlapping kernels
SpF - Primary Abstractions

- IndexSpace - supports arbitrary data layout and processor distribution
- Field - data values on an index space
- Operations
  - Kernel - indivisible unit of computational work
  - Operation - family of related non-overlapping kernels
  - KernelDriver - coordinates with Partitioner and Transposer
SpF - Primary Abstractions

- **IndexSpace** - supports arbitrary data layout and processor distribution
- **Field** - data values on an index space
- **Operations**
  - **Kernel** - indivisible unit of computational work
  - **Operation** - family of related non-overlapping kernels
  - **KernelDriver** - coordinates with Partitioner and Transposer
- **Transposer** - move data between to distributions of an index space
SpF - Primary Abstractions

- **IndexSpace** - supports arbitrary data layout and processor distribution
- **Field** - data values on an index space
- **Operations**
  - **Kernel** - indivisible unit of computational work
  - **Operation** - family of related non-overlapping kernels
  - **KernelDriver** - coordinates with Partitioner and Transposer
- **Transposer** - move data between to distributions of an index space
- **Partitioner** - subdivides global index space and operations across processes
SpF - Primary Abstractions

- IndexSpace - supports arbitrary data layout and processor distribution
- Field - data values on an index space
- Operations
  - Kernel - indivisible unit of computational work
  - Operation - family of related non-overlapping kernels
  - KernelDriver - coordinates with Partitioner and Transposer
- Transposer - move data between to distributions of an index space
- Partitioner - subdivides global index space and operations across processes
- Miscellaneous others
  - IO_Manager - special type of Operation for interacting with filesystem
  - Logger - monitor job status and history
  - Driver - executes operations and transposes iteratively
Fundamental Data Structure

Index space:

- Each node in computational domain represented as a vector of coordinates.

<table>
<thead>
<tr>
<th>n</th>
<th>r</th>
<th>θ</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nr</td>
<td>nr</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>nr + 1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>nr + 2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Care must be taken not to represent entire domain on any process.

- Transposer objects are constructed from an input and an output IndexSpace object.

- Numerical operations interact with IndexSpace objects to indicate locality and sequence requirements.
Example - Legendre transform

- Each wavenumber corresponds to a distinct kernel.
- Within a wavenumber, order with increasing harmonic degree.
- All other coordinates are irrelevant.

<table>
<thead>
<tr>
<th>'r_level'</th>
<th>'m'</th>
<th>'ℓ'</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>idx_W</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>idx_W</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>idx_W</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>idx_W</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>idx_W</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>idx_W</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>idx_Z</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
<td>idx_B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>9</td>
<td>idx_S</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example - Legendre transform

- Each wavenumber corresponds to a distinct kernel.
- Within a wavenumber, order with increasing harmonic degree.
- All other coordinates are irrelevant.

<table>
<thead>
<tr>
<th>'r_level'</th>
<th>'m'</th>
<th>'ℓ'</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>idx_W</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>idx_W</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>idx_W</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>idx_W</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>idx_W</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>idx_W</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>idx_Z</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
<td>idx_B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>9</td>
<td>idx_S</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kernel ID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Example - Legendre transform

- Each wavenumber corresponds to a distinct kernel.
- Within a wavenumber, order with increasing harmonic degree.
- All other coordinates are irrelevant.
- Each kernel also indicates output as an IndexSpace.

<table>
<thead>
<tr>
<th>'r_level'</th>
<th>'m'</th>
<th>'ℓ'</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>idx_W</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>idx_W</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>idx_W</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>idx_W</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>idx_W</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>idx_W</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>idx_Z</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
<td>idx_B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>9</td>
<td>idx_S</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kernel ID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>'m'</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Generalized Redistribution

Distribution 1          Distribution 2

Naïve Initialization:
Each processor exchanges *all* metadata with all others and determine overlapping elements. Complexity is O(N²)

Consider a petascale problem:
* T1920x513L
  10 peta-op platform

At 10 ops/comparison and 10% efficiency, initialization requires 10⁵ sec!
Fast Initialization

1. Global (sample) sort is used to order metadata.
   - Complexity $\sim O((N/p) \log_2 p)$
   - Use same pivots for both distributions.
   - Data tagged with original PE
2. Each processor compares sorted metadata to determine overlap.
3. Overlap data is then “unsorted”.

Petascale initialization should require $\sim 10$ minutes.
Outline

1. Performance Modeling
2. Software Engineering
3. Status
SpF Status

- Functioning OO prototype of several key classes
  - IndexSpace (axis, union, product)
  - Operation, Kernel, and KernelDriver
  - Partitioner
  - Transposer

- Functioning hardcoded F90 implementations working in DYNAMO
  - Partitioner - loadbalancing implicit solves
  - Transposer - to/from implicit layout
Next steps

- Complete basic SpF framework
- Implement concrete instances for
  - Legendre transforms
  - FFT - nonlinear - FFT
  - Implicit solve
  - Linear terms
Acknowledgements

- Several helpful conversations with Nick Featherstone.
- Support from the NASA High End Computing Program.
References

- Wikipedia software framework
- pFUnit
- DYNAMO
- ASH