

SpF: Enabling Petascale Performance for Pseudospectral Dynamo Models

T. Clune¹, W. Jiang^{2,1}, J. Vriesema³ and G. Gutmann⁴

¹ NASA Goddard Space Flight Center, ² SGT, ³ University of Arizona, ⁴ Tokyo Institute of Technology

Introduction

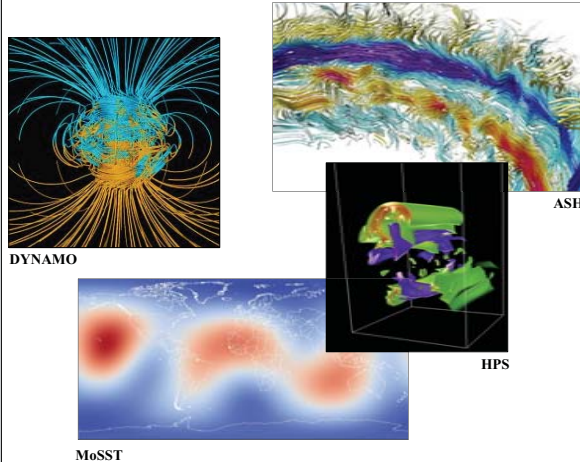
Pseudospectral (PS) methods possess a number of characteristics (e.g., efficiency, accuracy, natural boundary conditions) that are extremely desirable for dynamo models. Unfortunately, dynamo models based upon PS methods face a number of daunting challenges, which include exposing additional parallelism, leveraging hardware accelerators, exploiting hybrid parallelism, and improving the scalability of global memory transposes. Although these issues are a concern for most models, solutions for PS methods tend to require far more pervasive changes to underlying data and control structures. Furthermore, performance improvements in one model are difficult to transfer to other models, resulting in significant duplication of effort across the research community.

We have developed SpF, an extensible software framework for PS methods, which is intended to enable extreme scalability and optimal performance. High-level abstractions provided by SpF unburden applications of the responsibility of managing domain decomposition and load balance while reducing the code changes required to adapt to new computing architectures. The key design concept in SpF is that each phase of the numerical calculation is partitioned into disjoint numerical “kernels” that can be performed entirely in-processor. The granularity of domain-decomposition provided by SpF is only constrained by the data-locality requirements of these kernels. SpF builds on top of optimized vendor libraries for common numerical operations such as transforms and matrix solvers, but can also be configured to use open source alternatives for portability. SpF includes several alternative schemes for global data redistribution and is expected to serve as a testbed for further research into optimal approaches for advanced network architectures.

Goals

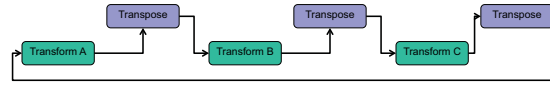
- Scalable and Efficient
 - ✦ Extremely fine-grained parallelism
 - ✦ Multilevel parallelism (MPI+OMP)
 - ✦ Isolate comp. intensive sections
 - ✦ Utilize optimized vendor libraries
 - ✦ Parallel I/O
- Flexible
 - ✦ Variant geometry
 - ✦ Alternative radial schemes
 - ✦ Variant load balance strategies
- Reduced duplication
 - ✦ Shared code infrastructure
 - ✦ Exchange of innovations
- Extensible – OO design
- Robust and Portable
 - ✦ Test driven development
 - ✦ Nightly automated regression tests

Targeted NASA Applications



Design Elements

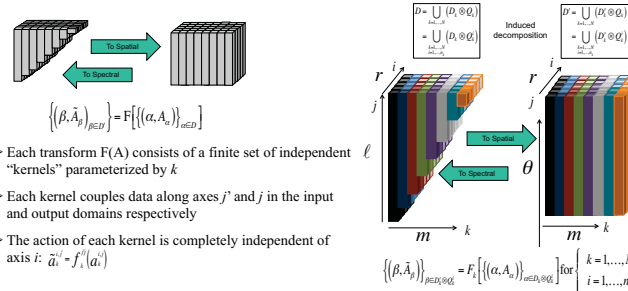
Parallel implementations of PS methods can generally be implemented as a loop consisting of alternating series of numerical transforms and global memory transposes/permutations.



Typical applications use inflexible, parameterized heuristics for decomposing the data across processes for each transform. Transposes are then hardcoded to align with these built-in assumptions. This approach leads to brittle implementations that require significant revision when undergoing further optimization.

The key innovation in SpF is to enable implementation of transforms in a *decomposition-independent*, *communication-free* manner. As demonstrated below for Legendre transforms, locality properties of transforms lead to a natural 3D parameterization of the input and output domains. The numerical operation only couples values along one of these dimensions, which allows a completely arbitrary distribution of the remaining axes.

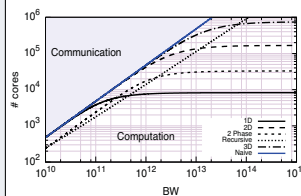
Legendre Transform Example



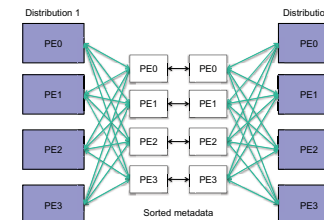
The software framework is free to assign each “pencil” (i, k) to an arbitrary processor. Secondary considerations such as memory footprint and cache-locality lead to prioritizing distribution along the kernel axis. Early parallel implementations of PS applications generally used 1D decompositions corresponding to just one of these axes. Modern implementations often exploit a 2D decomposition, but still cannot fully exploit parallelism when n_k is not constant such as for the Poisson solver illustrated by the figure to the left.

Automated Permutations

To support the arbitrary domain decompositions, SpF must fully automate permutations between alternative layouts.



At extreme scale, the efficient implementation of data permutations is nontrivial. The figure above shows the benefit of multi-staged permutations for a typical petascale dynamo simulation. SpF enables exploring such alternatives with minimal effort.



Naïve initialization requires $O(N^2)$ operations, which is prohibitive for petascale applications. SpF uses a novel approach based upon parallel sort to achieve a complexity of $O(N \log N)$.

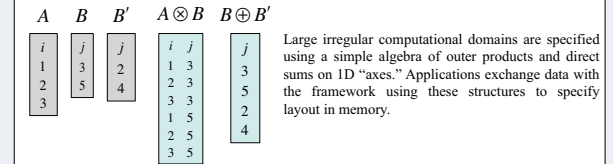
Software Framework Implementation

Key Abstractions

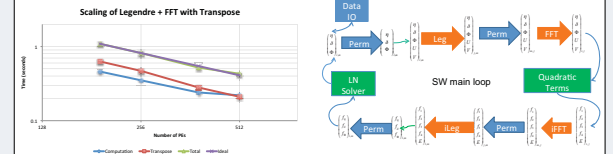
- Algorithms
 - ✦ Kernel – non-decomposable “atom” of a numerical phase
 - ✦ KernelFactory – generates parameterized set of kernels and associated in/out Field objects
- Integrator – performs integration in time (RK, AB, CN, etc.)
- Data
 - ✦ IndexSpace – algebra for composing descriptions of arbitrary domains
 - ✦ Field – container for data associated with an IndexSpace
 - ✦ DataIO – enables multiple parallel I/O formats (initially just p HDF5)
- Communication and Load Balancing
 - ✦ Permutor – redistributes data from one Field to another
 - ✦ Distributor – uses metadata from KernelFactory to perform domain decomposition

Test driven development (TDD) has been routinely applied using pUnit 2.0 to ensure reliability and portability of the basic framework.

Index Space Representation of Computational Domains



Current Status



The above figure demonstrates the scalability of a synthetic SpF application that combines a Legendre transform, an FFT, and a transpose.

In this schematic of an SpF-based SW, customization is only required for the items indicated in green.

Next Steps

The basic SpF framework is now mostly complete, and work has begun on porting legacy applications. Longer term, the primary focus will be to extend SpF in the following ways:

- Advanced algorithms for Distributor and Permutor subclasses for extreme scales
- Factory for constructing linear systems for alternative radial schemes
- Incorporate hardware accelerators
- Distributed Kernels – large kernels with internal parallelism (e.g., implicit coriolis)

References

- Beck, K., *Test Driven Development: by Example*, 2002.
- Brummel, N. et al., “Penetration and Overshooting in Turbulent Compressible Convection”, *ApJ*, 570:825-854, 2002.
- Clune, T., Parallel Fortran Unit Testing Framework (pUnit) http://sourceforge.net/p/punit/_list/git
- Glatzmaier, G. and Clune, T., “Computational Aspects of Geodynamo Simulations”, *Comput. Sci. Eng.* 2:61-67, 2000.
- Hack, J. and Jakob, R., “Description of a Global Shallow Water Model Based on the Spectral Transform Method”, NCAR Technical Note, 1992.
- Jiang, W. and Kuang, W., “An MPI-based MoSST core dynamics model”, *Physics of the Earth and Planetary Interiors*, 170 (2008), 46–51.
- Miesch, M. et al., “Three-dimensional spherical simulations of solar convection. I. Differential rotation and pattern evolution achieved with laminar and turbulent states”, *ApJ*, 2000.