Dissertation Proposal: Abstractions for, and Generation of, Semi-Regular Grid Computations

Submitted by:
Andrew Stone
Computer Science Department

In partial fulfillment of the requirements
for the Degree of Doctor of Philosophy
Colorado State University
Fort Collins, Colorado
Summer 2012

August 1, 2012

Abstract

In various applications including atmospheric and ocean simulation programs, stencil computations occur on grids where sub-domains of the grid are regular (e.g., can be stored in an array) but boundaries between sub-domains connect in an irregular way. We call this class of grids semi-regular. Implementations of stencils on semi-regular grids often have grid-structure details tangled with the stencil computation code. This tangling of details requires programmers to have full knowledge of the current grid structure to make changes to the stencil computations and makes changing the grid structure difficult.

Our research objective is to separate algorithm from grid structure and parallelization details. Existing approaches to separate stencil and grid specification include Domain Specific Languages and program generators, which focus on regular or irregular grids. These tools produce efficient, parallel, code, but have not focused on the class of semi-regular grids.

We propose to introduce abstractions for specifying the structure of semi-regular grids. We also introduce abstractions for specifying data distribution, communication, and stencils. We plan to include these abstractions in an active library called GridLib and include a code generation tool called GridGen will replaces library calls with efficient, parallel, code. We will evaluate how this approach impacts the performance and programmability of stencil applications by applying these techniques on CGPOP and the Shallow Water Model (SWM) miniapps. We will also include optimizations in the GridGen tool that will improve the performance of the generated code beyond what is seen in the hand-written code.

Contents

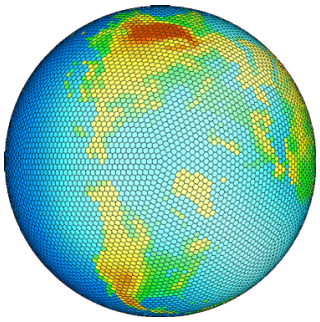


Figure 1: Discretization of the Earth into a grid (picture from [2])

```
subroutine Stencil(array)
  ! ** Input parameters and local variables: **
  real, intent(inout) :: array(:, :)

  do j=lbound(array,2), ubound(array,2)
    do i=lbound(array,1), ubound(array,1)
      array(i, j) = 0.2 * (array(i, j) + array(i-1, j) +      &
                           array(i+1, j) + array(i, j-1) +    &
                           array(i, j+1))
    enddo
  enddo
end subroutine Stencil
```

Figure 2: Fortran code for a simple five point stencil on a regular grid with grid values stored in an array. In this example the stencil coefficients are constant values.

1 Introduction

Computational modeling and simulation play critical roles in contemporary scientific research; they form a third pillar of scientific inquiry alongside the traditional approaches of experimentation and theory [17]. Simulation is used where traditional approaches would be impractical, expensive, slow, or dangerous. Simulation programs model an object's changing physical properties over time. These programs calculate changing properties by solving partial differential equations over discretized grids.

In fields such as Geophysics, Climatology, and Meteorology, programs are used to simulate changes over the Earth's surface and atmosphere. To represent the surface or atmosphere, the Earth is modeled as a grid of connecting cells. In Figure 1 the Earth is discretized into hexagons and pentagons [2].

1.1 Research Problems

Due to the lack of well-defined grid abstractions and the need for performance, details about the grid are often tangled with a simulation program's algorithms. For example, consider a relatively untagled stencil code such as that shown in Figure 2. The stencil computation iterates through all points of the grid and updates them using neighboring values. The simplicity seen in the example comes from the fact that the grid maps to a single array: such grids are said to be *regular*.

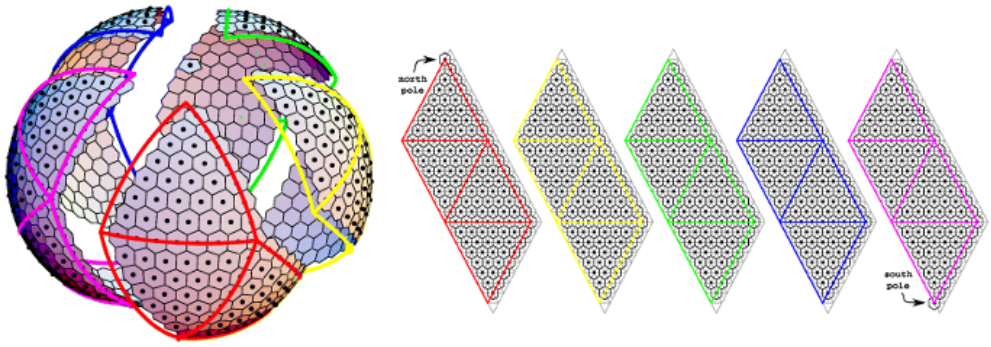


Figure 3: Flattening of Icosahedral grid into five arrays (picture from [4])

Updating the code in Figure 2 to operate on the grid in Figure 1 should not be trivial. Since Figure 1 is not fully regular, values on the grid can not be stored in a single array. In *irregular* grids data can be stored using graph data-structures. However, storing grid data with a graph representation requires that the grid's connectivity by explicitly represented, thus requiring more memory than with an array representation. Furthermore, accessing a node's neighbors in a graph representation requires indirect access through an adjacency list or matrix. Note that regular grids can be stored using the data-structures for irregular grids but would incur a performance penalty due to the need for indirect access.

Although the grid in Figure 1 can not be flattened into a single array, it can be cut and flattened into multiple arrays as illustrated in Figure 3. Grids that can be cut and flattened into a finite number of arrays we term *semi-regular*. Updating the code in Figure 2 to operate on a semi-regular grid involves having to iterate across multiple arrays and account for the connectivity between the arrays.

In practice, Earth simulation stencil codes are often parallelized to work on large supercomputers. Parallelization further complicates stencil algorithm code. In a typical SPMD style parallel implementation grid data is decomposed across processes and stencil code is modified to iterate over locally stored portions of the grid; communication code is added so that all remote data referenced by the algorithm is copied to locally accessible ghost-cells before the stencil is called. The communication code must be cognizant for the grid's topology.

Accounting for grid and parallelization details not only complicate a simulation's initial implementation but also make maintenance more difficult. Parallelization details are often modified when transferring an application to run on a new system. Grid details may be modified as new types of grids are developed. Modifying application code to use a new grid would require large rewrites in the grid code that is tangled with algorithm and communication code.

1.2 Research Approach

Programming abstractions can be used to reduce complexity in software. Thus, we propose to address issues with implementing semi-regular grid stencil computations by creating abstrac-

tions that represent semi-regular grids. We also introduce operations that can be conducted on semi-regular grids, and introduce abstractions for representing parallelization details on semi-regular grids. In a preliminary paper [36] we introduce several such abstractions. These abstractions are subgrids, grids, communication plans, and data distributions. With these abstractions programmers can specify a grid's topology seperately from the algorithms that operate on it.

To apply these abstractions we propose integrating them into a library we call *GridLib*. By introducing these abstractions in a library, as opposed to creating a new language, we will be able apply the abstractions to existing software. In particular, we will apply GridLib to existing Earth Simulation miniapps such as CGPOP [35]. GridLib is implemented in Fortran and conducts parallel computation and communication via MPI.

To avoid library overhead and enable further optimization we also propose developing a tool called *GridGen* that will parse Fortran code, extract GridLib library calls, and replace GridLib calls with more efficient code. To parse and inject Fortran code we will use the ROSE compiler infrastructure [30, 3].

1.3 Contributions

Specifically, we intend to make the following contributions with this research:

1. The introduction of semi-regular grid abstractions and abstractions to represent parallelization and algorithm details. We will evaluate how these abstractions impact programmability and performance in two miniapps: CGPOP [35] (a miniapp of the Parallel Ocean Program [22]) and the Shallow Water Model (SWM) code (a miniapp for GCRM [6]). We will evaluate programmability by examining the impact of using GridLib on the miniapps source lines-of-code.

2. An algorithm that will construct a communication plan for a semi-regular grid. The communication plan specifies how nodes in a distributed memory system transfer data from one-another so that algorithms that operate on the grid can access data stored remotely. Communication is necessary in stencil computations so that when a process updates its local data it has access to neighboring remote vaues.

3. The development of the GridLib active library and the GridGen tool that replaces GridLib calls with stencil and communication code.

4. A performance evaluation of GridLib/GridGen when applied to the CGPOP and SWM miniapps.

5. An evaluation of changing the grid in CGPOP from the dipole [33] to the triople grid with the GridGen package [29].

6. Generation of communication/computation overlap, array of structs, and for communication avoidance with GridGen. With these optimization we are aiming to improve the performance of CGPOP and SWM. We discuss these optimizations in more detail in Section 2.2

With regards to contributions 1, 2, and 3 we present preliminary work in an unpublished paper included with this proposal [36]. We intend to submit this paper to the 27th International Parallel and Distributed Processing Symposium (IPDPS) on October 1, 2012.

Our proposed contributions are limited to operating on two-dimensional, statically defined, semi-regular grids. We focus on these grids since they are seen in Earth simulation applications such as POP [22] and GCRM [6]. Future work could focus on expanding our work to operate on N-dimensional grids (a step that we believe would be a straight-forward addition to our proposed work), that can adaptively refine, and that could generate code to use GPUs.

In the remaining sections of this proposal we discuss how this proposed work relates to existing work and discuss these contributions in greater detail.

2 Background

Stencil computations like that seen in Figure 2 are commonly used to solve partial differential equations [8]. As previously discussed, partial differential equations are used to model changing physical properties on some entity over time, and physical entities can be modeled using regular, semi-regular, or irregular grids.

Given the usefulness of grids and stencil computations, several tools have been developed to model grids and conduct stencil computations. In Section 2.1 discuss and compare these tools, and in Section 2.2 we discuss optimizations that are commonly applied to stencil computations and appear in stencil generation tools. However, despite the prevalence of stencil generation tools, none of these tools none address the class of grids we identify as semi-regular.

In Section 2.3 we discuss several semi-regular grids and connectivity patterns that are seen among them. In Section 2.4 we discuss the CGPOP and SWM, which operate on semi-regular grids.

2.1 Related Work

Several tools have been developed that aid with the implementation of stencil computations that work well either on structured or unstructured grids. We compare several of these existing tools in Table 1.

These tools differ in terms of what types of grids they operate on, how grids and stencils are specified, and what platform and languages they target. The framework from Kamil et al. [23], Mint [38], Patus [10], Physis [28], and Pochoir [26] work on regular grids; Liszt [13], MPAS [5], and OP2 [18] work on irregular grids. Chombo [12] works on grids that can be adaptively refined. Adaptive Mesh Refinement is a method where cells in a grid can been split into sets of finer resolution cells. To the best of knowledge no existing tool targets semi-regular grids, which will be the aim of our proposed work with the GridGen package.

The tools in Table 1 interface with users either via a library, domain specific language (DSL), or set of annotations. Library approaches may be active, active libraries include tools that preprocess source code replacing library calls with generated code. DSLs may either be implemented or within the context of a larger language (such as C or Fortran).

Table 1: Comparison of stencil generation tools and DSLs

	Kamil et al. [23]	**Mint** [38]	**Patus** [10]
Grids	Regular	Regular 3D	Regular
Specification	Annotated Fortran loops	C with pragmas	DSL with C syntax
Targets	MultiCore C and pthreads, Fortran and pthreads	GPU C and CUDA	Multicore and GPU: C and OpenMP, CUDA
Optimizations	Autotunes a decomposition of node, core, thread, and register blocks	2D or 3D tiling, GPU thread aggregation	Autotunes, SIMDization, loop unrolling, NUMA-aware data initialization

	Physis [28]	**Pochoir** [26]	**Chombo** [12]
Grids	Regular	Regular	Adaptively Refined
Specification	DSL embedded in C	DSL embedded in C++	C++ Library
Targets	Clusters with GPUs C and CUDA	Multicore C++ and Cilk	Clusters, C++ and MPI
Optimizations	Overlapping computation and communication,	Trapazoidal decomposition	Load balancing function

	MPAS [5]	**Liszt** [13]	**OP2** [18]
Grids	Unstructured Voroni Meshes	Irregular Scala based	Irregular
Specification	Collection of software components in Fortran	Scala based DSL	Active C++ Library
Targets	Clusters	Clusters with GPUs MPI and C++ or CUDA	Clusters, multicore systems, GPUs C++ with MPI, OpenMP, or CUDA,
Optimizations	-	Partioning and coloring	Autotunig of GPU parameters,

Annotation add syntax to an existing language that specify where some section of code should be transformed, annotations are typically specified using pragmas.

Libraries and annotation based approaches benefit from the fact that they can be applied to existing code. Libraries have the further benefit of interacting with debugging tools. Inactive libraries incur a performance penalty due to library overhead, however, active libraries use code-generation techniques to eliminate this overhead. Languages based approaches have the benefit of being able to introduce new syntax.

The tools also differ from eachother in terms of what platforms and languages they target. For active library and annotation based approaches a preprocessor reads specifications and generates source code in some language. Parallelization details are typically handled with third-party libraries such as MPI or OpenMP. Code intended to run on accelerators is typically generated for CUDA or OpenCL. In our proposed work the GridGen package will target MPI.

2.2 Stencil Computation Optimizations

Many of the tools discussed in Section 2.1 automatically perform optimizations to improve stencil performance. Automatically applied optimizations improve programmer productivity (by eliminating the need to manually perform the optimization) and help code maintainability (by avoiding the code obfuscation that often occurs when an optimization is applied).

In this section we list a set of optimizations that have been performed manually and automatically on stencil computations. One aspect of our work with the GridGen package will be to (1) determine a set of optimizations that improve will improve the performance of the SWM and CGPOP miniapps when running on distributed memory machines, and (2) integrating these optimizations into GridGen so that they are automatically applied.

Optimizations commonly applied to stencil computations include:

- **Tiling** [39]: Reorganize a loop-nest that iterates through grid points so that when iterates through blocks of data that fit into cache. **Multi-level tiling** embeds multiple levels of tiles inside a loop nest to address multiple levels of the memory hierarchy.

- **SIMDization** [25]: Parallelizes stencil code through SIMD vectorization calls. These calls use special SIMD CPU registers to perform the same operation on multiple pieces of data simultaneously.

- **Option to Store Data as Array of Structs or Struct of Arrays** [14]: When multiple values are assigned to each node in a grid these values can either be stored in seperate arrays (as a struct of arrays) or stored in a single array where each element is a structure holding these multiple properties (as an array of structs). The struct of array style exposes opportunities for vectorization while struct-of-array style may allow for better cache behavior depending on what values are accessed by a stencil computation.

- **Array padding** [24]: Pad the ends arrays in order to ensure that their data is stored on separate cache-lines. This is done in order to avoid data cache thrashing (cache conflicts).

- **Overlapping tiles (communication avoiding algorithms)** [40, 15]: Perform the same computation locally that was performed on remote node in order to avoid communicating data from that remote node.
- **Overlapping computation and communication** [7]: Perform computation that does not require access remote data while waiting for communication to complete. In the context of stencil computation the interior points (those points not bordering the communication halo) can be updated while waiting for the halo to be populated.
- **Software Controlled Prefetching** [7]: Some architectures include explicit fetch instructions to control prefetching of data into cache. Prefetching avoids memory-fetch latency by moving data to cache prior to it being referenced.
- **Loop unrolling** [16]: Unroll a loop so that its body includes multiple iterations of the previous version of the loop. Loop unrolling is performed in order to reduce loop overhead. Compilers are often able to perform this optimization automatically.

2.3 Semi-regular Grids

One limitation of the tools presented in Section 2.1 is that none of them address semi-regular grids. We define semi-regular grids to be unions of two-dimensional, regular, sub-grids that

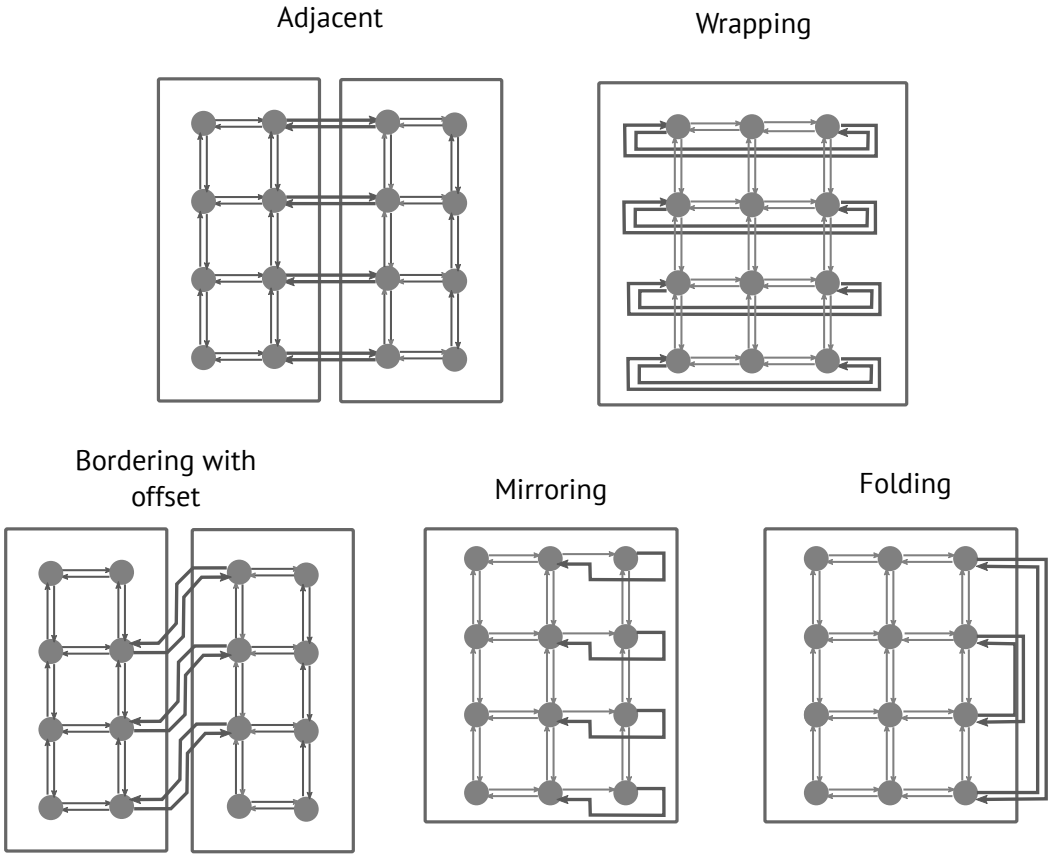


Figure 4: Connectivity patterns seen in semi-regular grids.

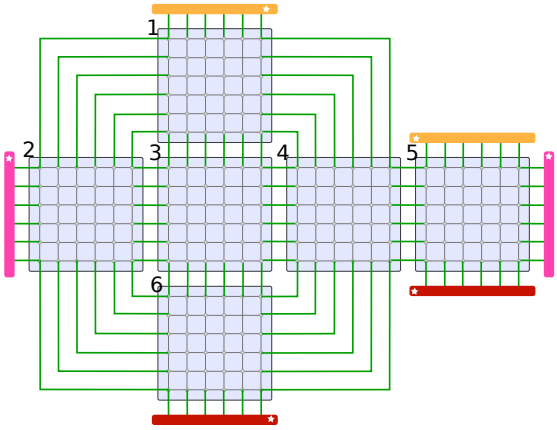


Figure 5: Connectivity of the cubed-sphere grid. The number next to each-subgrid is a value we can use to identify it.

connect to one-another.

Several types of semi-regular grids appear in Earth simulation applications. Examples include the torus, dipole [33], tripole [29], icosahedral [19, 37] (see Figure 3, and cubed-sphere [32] grids.

In these grids regular subgrids connect to one another in an unstructured fashion. In Figure 4 we illustrate different connectivity patterns that exist among subgrids. The adjacent connectivity pattern that appears in Figure 4 is seen in several grids including the cubed-sphere and icosahedral grids. The wrapping pattern appears in toroidal grids, the bordering with offset pattern appears in the icosahedral grid, and the folding pattern appears in tripole grids.

The simplest grid used to simulate the Earth is a torus, which can be represented as a single regular, rectangular grid. Specialized communication code is needed to wrap the north boundary to the south boundary and the west boundary to the east boundary. Despite this needed specialization, torus grids are often considered to be purely regular and many tools that generate stencil codes allow for a regular grid to include simple wrap-around boundaries [11, 38, 27]. Parallel array languages such as ZPL [9] also have the ability to express periodic (that is wrapping) boundaries.

In certain applications dipole grids are used. In dipole grids there is no wraparound between the north and south boundaries and the north and south poles of the grids are shifted so that they correspond to points of land on the earth. Simulation around the two poles will not suffer inaccuracy in simulations that do not operate over land-points; for example, in ocean models.

In models that not allow for both poles to be shifted over land-masses a tripole grid may be used. In this grid, west and east boundaries have the same wrap-around properties seen in the torus and dipole grid, but the north-pole includes a fold. The south-pole must be placed over land. The northern fold requires more complicated communication code.

Cubed-sphere and Icosahedral grids can be used when the entire Earth surface (or atmo-

sphere) is to be simulated. In the cube sphere grid the Earth is represented as six regular grids that connect to one-another to form a cube. In the Icosahedral grid twenty triangular regions can form an icosahedron. Pairs of these triangular regions can be stored as regular grids, forming ten arrays. We illustrate the connectivity pattern of the cubed-sphere grid in Figure 5, and we previously illustrated the icosahedral grid in Figure 3.

2.4 Applications and Algorithms of Interest

Two Earth Science applications that make use of semi-regular grids are the Parallel Ocean Program (POP) [22] and the Global Cloud Resolving Model (GCRM) [6]. In this section we briefly discuss POP and GCRM and two miniapps (CGPOP and SWM) that were extracted from important components in these applications. Our evaluation of the GridGen package and its semi-regular grid abstractions will involve using these abstractions in CGPOP and SWM.

The Parallel Ocean Program (POP) was developed at Los Alamos National Laboratory and is an important multi-agency code used for global ocean modeling and is a component within the Community Earth System Model (CESM) [1]. POP has been actively developed for over 18 years, and is nearly 71,000 lines of Fortran 90 and MPI code.

GCRM is a DOE sponsored global atmospheric circulation model capable of simulating at 3 km resolution. GCRM makes use of spherical geodesic grids [4], which are defined by recursively bisecting the icosahedral grid.

Both POP and GCRM are large applications consisting of multiple components. Miniapps are applications on the order of 1000 lines of code that include a simple build system and accurately model the performance of a larger application. In [20], Heroux et al. suggest that miniapps can be used to compare and programming models. We intend to use miniapps to evaluate our abstractions to model grid topology, distribution, and stencil algorithms. We will evaluate the GridGen package with two miniapps: CGPOP and SWM.

CGPOP is a miniapp of POP's Conjugate gradient solver. In [22] we demonstrate that CGPOP accurately models the performance characteristics of POP across a variety of machines and compilers. In in [34] we argue that CGPOP can be used to evaluate a programming model's programmability. CGPOP uses dipole grid [33]. Current implementations of POP use the more complicated tripole grid [29]. Part of research effort will be to use the GridGen package to update CGPOP to use the Tripole grid.

SWM is a miniapp of GCRM's shallow water model component. SWM iteratively applies Jacobi-stle stencil over an icosahedral grid [19, 37]. Part of research effort with the GridGen package will be to represent the icosahedral grid with our abstractions and integrate it into SWM.

3 The GridGen Package

In order to address the research problems raised in Section 1.1 we propose to create a set of tools to define, generate code, and visualize semi-regular grids.

3.1 Software Architecture

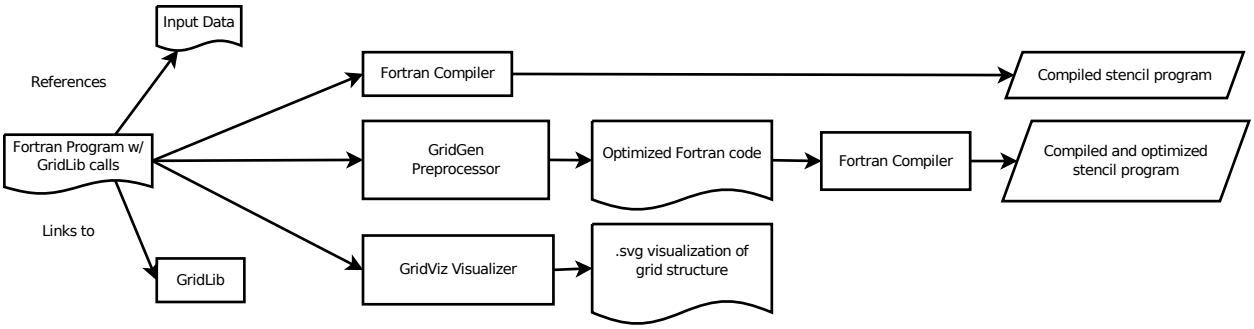


Figure 6: System diagram of the GridGen package

In Figure 6 we illustrate the tools we plan to create: GridLib, GridGen, and GridViz. We refer to the collective of these three tools as *The GridGen Package*.

The GridLib library includes abstractions for specifying grids and functions for applying algorithms. The GridGen tool is used to optimize programs that use the GridLib library. In addition to the GridGen tool and GridLib library we will create a tool called GridViz that can be used to construct visualizations of grid connectivity.

Although programs that use the GridLib library will produce correct results, we do not believe the approach taken by GridLib will produce optimal code. Library approaches introduce overhead from calls to library functions and are unable to take advantage of certain compile-time optimizations; for example, fusing loops from multiple stencil computations together. However, despite the performance limitations of a library approach it does offer some advantages. Code linked to GridLib will work with most debugging tools such as gdb and library calls can be integrated into existing projects.

To address the performance limitations of GridLib, the GridGen project also includes a code generation tool called GridGen. The GridGen tool is passed a Fortran program that uses the GridLib library. The GridGen tool uses Rose (a source-to-source translation tool) to parse the Fortran code and extract library calls. GridGen then replaces the library calls with optimized code. The code produced by GridGen can then be passed to a Fortran compiler to produce an executable.

The GridViz tool produces images for representing semi-regular girds. The purpose of the visualization tool is to help developers determine whether they have specified their grid topology correctly.

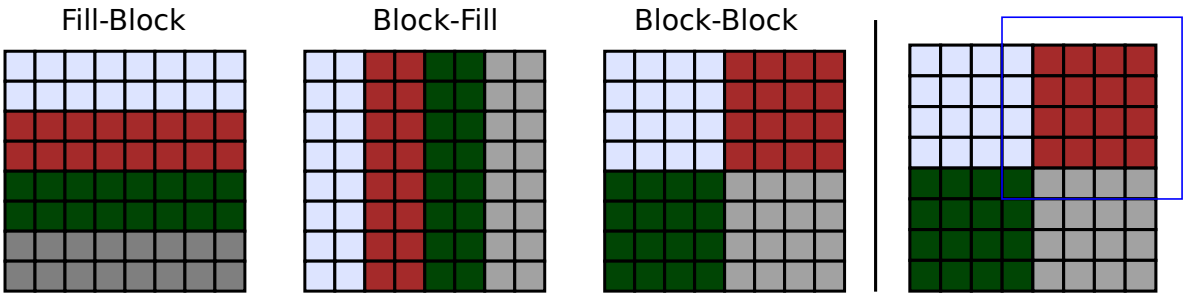
3.2 Grid specification

In the GridGen package grids are specified in an object-oriented manner using the GridLib library. Pertinent objects in the GridLib library include grids, subgrids, distributions, communication schedules, and data objects. Grids, subgrid, and neighbor objects describe how nodes connect to one-another. Data objects are assigned a relevant grid and assign values to each node in the grid. Distribution and schedule objects describe where values in a data object are stored and what communication is needed to update these values respectively. In this section we briefly review these objects.

In Figure 7 we specify the Cubed-Sphere grid in Figure 5 and the stencil in 2. Grids

```
module FivePtStencil
  contains
    real function sumNeighbors(A, i, j)
        sumNeighbors = &
            0.2 * (A(i, j) + A(i-1, j) +      &
                   A(i+1, j) + A(i, j-1) +    &
                   A(i, j+1))
    end function
end module

program Stencil
    implicit none
    include 'gridlib.h'

    type(SubGrid) :: sg1, sg2, sg3, sg4, sg5, sg6
    type(Grid) :: g
    type(Distribution) :: dist
    type(Data) :: input_data, output_data

    integer :: N, M

    ! Specify grid
    N = 6
    M = 6
    call subgrid_new(sg1, N, M)
    call subgrid_new(sg2, N, M)
    call subgrid_new(sg3, N, M)
    call subgrid_new(sg4, N, M)
    call subgrid_new(sg5, N, M)
    call subgrid_new(sg6, N, M)

    call grid_new(g)
    call grid_placeAdjacentHoriz(g, sg2, sg3)
    call grid_placeAdjacentHoriz(g, sg3, sg4)
    call grid_placeAdjacentHoriz(g, sg4, sg5)
    call grid_placeAdjacentHoriz(g, sg5, sg2)

    call grid_placeAdjacentVert(g, sg6, sg3)
    call grid_placeAdjacentVert(g, sg3, sg1)
    call grid_placeAdjacentVert(g, sg1, sg6)

    call grid_placeAdjacentDiagNW(g, sg2, sg1)
    call grid_placeAdjacentDiagSE(g, sg2, sg6)
    call grid_placeAdjacentDiagES(g, sg6, sg4)
    call grid_placeAdjacentDiagNW(g, sg4, sg1)

    call grid_connectAndFlipN(g, sg5, sg1)
    call grid_connectAndFlipS(g, sg5, sg6)

    ! Specify distribution
    call distribution_new_blocked(dist, g, (/ 3, 3/ ))

    ! Initialize data
    call data_new_from_file(input_data, "input.dat", g, dist)
    call data_new(output_data, g, dist)

    ! Perform stencil operation
    output_data = data_apply(input_data, sumNeighbors)
end program Stencil
```

Figure 7: Fortran source using GridLib for a five point stencil applied to a cubed-sphere grid.



(a) Data distributions. (b) Halo for red block.

Figure 8: Different ways of distributing an array across four processes.

are defined in GridLib by: constructing one or more subgrid objects, and describing the connectivity among grids. Data distribution is represented using distribution objects.

The `subgrid_new` function is used to instantiate new subgrids of a specified size. The `grid_new` function creates a new grid, and the various `grid_place` functions are used to place subgrids into the grid according to some connectivity pattern. The make use of adjacency patterns like that seen in Figure 4a. The `grid_placeAdjacentHoriz` and `grid_placeAdjacentVert` respecitvely place two subgrids adjacent to one another horiziontally and vertically. The `grid_placeAdjacentDiag` functions connect one of the four borders (north, south, west, or east) of a subgrid to one of the four borders of another subgrid.

The various `grid_place` functions we use are convenience functions that connect subgrids by constructing border mappings. Border mappings (like those seen in Figure 4 can also be explicitly defined in GridLib.

3.3 Algorithm Specification

Algorithms in GridLib operations are applied to data objects. There are two types of operations in GridLib: (1) point-wise update operations, and (2) reductions.

Pointwise update operations update each value in a data-object with a combination of neighboring values. Stencil computations are a type of point-wise update operation. In Figure 7 the function `sumNeighbors` defines a pointwise operation. The `data_apply` function is used to specify where to perform the operation and what data-objects it should be applied on.

A reduction operation applies an associative and communicative operator to combine all values for a data object over the whole grid. Reductions operations include sum, product, min, max, etc.

More complicated algorithms, such as the conjugate gradient algorithm in CGPOP, can be defined using combinations of stencil and reduction operations.

3.4 Data distribution and Communication

Reductions and pointwise grid operations are data-parallel computations. As such, in order to execute these operations, there needs to be an assignment of data to the process that will operate on it. In order to define this assignment we will include a distribution abstraction in GridLib. Subgrids are blocked into evenly sized rectangle regions called blocks. A mapping exists between blocks and processes. In GridLib users can explicitly control this mapping or

may use higher-level functions to construct commonly used distributions. In Figure 8a we illustrate several distribution patterns that we will create functions for. These distributions are based off of the distribution pragmas seen in High Performance Fortran [21, 31].

When applying stencil computations each proccess updates values within its locally distributed region, however this update process may reference values that are distributed on another process. To address this blocks are allocated a halo region which locally stores topologically values that have been distributed to a different process. In Figure 8b we illustrate the halo region for the red (top-right) block from the block-block distribution. To populate this halo communication must occur the blocks it overlaps with. Note that part of the halo lies off of the subgrid, what data to fill this region of the halo with depends on the topology of the grid. In GridLib and GridGen we include an algorithm that determines what communication must occur to populate each block's halo region. The result of this algorithm is stored in an object we call a communication plan. We discuss this algorithm in more detail in our preliminary paper [36].

The most efficient distribution to use depends on the stencil used and grid topology. For example, with a four-point (north, south, weast, and east neighbor) stencil and a grid where the west and east borders connect the block-fill distribution would require the least amount of communication between processes, in a grid where the north and east border connect the fill-block would perform the least commuication, and in a grid where no borders connect the block-block would perform the least communication. Future work could involve automatically determining how to distribute data.

3.5 Preliminary Evaluation

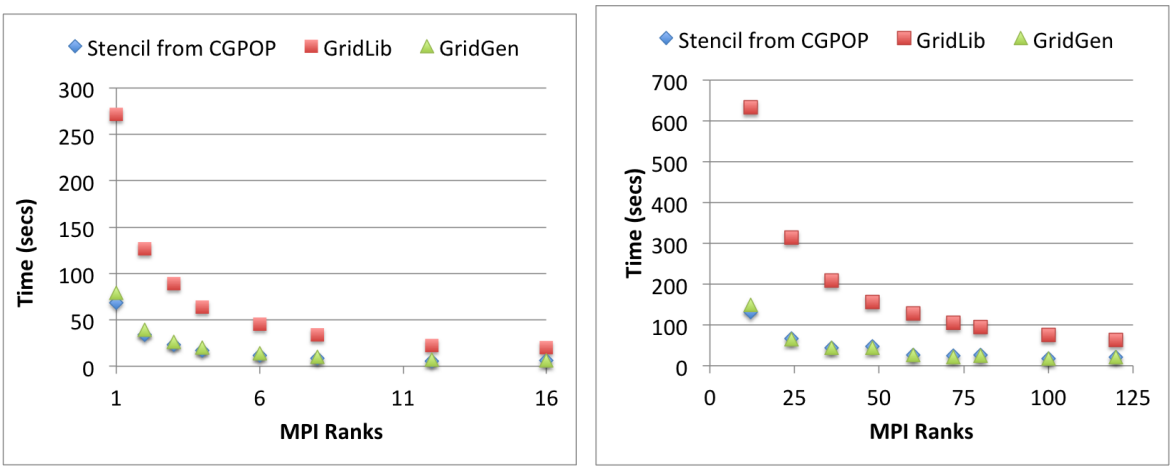
In [36] we present preliminary work where we introduce abstractions for semi-regular grids and present preliminary work on GridGen package. At this time we have working implementations of GridLib and GridGen that work on compact stencils.

In Figure 9 we show a performance comparison of GridLib and GridGen against a stencil computation extracted from CGPOP. The extracted stencil finds a weighted average of the eight neighboring points. The stencil coefficients come from edge weights values stored in arrays.

We performed our experiments on two machines at Colorado State University: Bacon, a 16 core, 2.13 GHz, XeonE7 machine with 128 GB of memory; and the ISTeC Cray HPC system, a Cray XT6m with 1248 cores across 52 compute-node, and 1.6 TB of memory. There are 24 cores per node on the ISTeC Cray.

For the multicore performance experiments we performed the stencil for 100 iterations on a 3600 by 3600 dipole-grid with a row-blocked distribution. For the cluster experiments we performed the stencil for 1000 iterations on a 10800 by 10800 dipole-grid with a row-blocked distribution.

This preliminary evaluation demonstrates that (1) GridLib introduces library overhead that reduces performance, particularly when run on small number of cores, and (2) GridGen is able to produce code that performs similarly to hand-written code.



(a) Performance on multicore

(b) Performance on cluster

Figure 9: Performance comparison of GridLib and GridGen against the stencil extracted from CGPOP

4 Project Organization

We organize our proposed work around three papers that we will write and submit to relevant academic conferences. Each of these papers addresses successive research goals and introduces new functionality into the GridGen package. The timeline for these papers is presented in Table 2.

Table 2: Project timeline

Date	*Category*	*Description*
`Summer 2012`	PAPER	First paper: IPDPS draft [36]
`Summer 2012`	EXAM	Preliminary exam
`Fall 2012`	PAPER	Implementation and writing for second paper
`Spring 2013`	PAPER	Implementation and writing for third paper
`June 2013`	EXAM	Dissertation defense

4.1 Paper Topics

The three papers we will produce focus on the following:

First Paper

Abstractions for Defining Semi-Regular Grids Orthogonally From Stencils. As discussed in Section 1.1, due to the need for performance, grid details are often tangled with algorithm code. This tangling negatively impacts code clarity and maintainability and makes changing grids difficult. To address these issues we propose designing a set of semi-regular grid abstractions enable a separation of concerns between grid topology and algorithm. We integrate these abstractions into a library called GridLib. To address library overhead we create a code preprocessing tool called GridGen that replaces library calls with efficient code. With this tool we are able to

specify grid topology separately from algorithm without negatively impacting performance. We include an unpublished draft of this paper with this proposal in [36] and intend to submit to the IPDPS conference, which has an October 1, 2012 deadline.

Second Paper

Generating code for Non-Compact Semi-Regular Grids. In the first paper we assume that stencil algorithms are compact (have halo depth of one). However, in some applications non-compact stencils appear [41, 42]. In this paper we generalize our previous abstractions to enable larger halos and reference iterations from more than one time-step back. In this paper we will apply the GridGen package to the CGPOP and SWM miniapps. Optimizations seen in CGPOP and SWM will also be integrated (for example: CGPOP includes an optimization to exclude computing over blocks of data that represent land). We intend to send a submission of this paper to the 2013 PLDI conference. The submission date for PLDI is tentatively November 11, 2012.

Third Paper

Optimizing Earth Simulation Applications with an Active Library for Semi-Regular Grids. Manual optimizations of stencil computations obfuscates code and are often difficult to implement. In this paper we define how to automate the implementation of communication/computation overlap, array of structs, and communication avoidance optimizations using our abstractions. We evaluate the performance impact of these optimizations on the CGPOP and SWM miniapps. We intend to send a submission of this paper to the 2014 Supercomputing Conference (SC), which has a tentative deadline in May of 2013.

4.2 Evaluation Approach

Recall that the goal of the first paper is to introduce abstractions for representing semi-regular grid and that by using a code-generation tool (GridGen) we are able to produce stencil code that matches the performance of hand written code. In this paper we will show that this approach can be used to represent dipole, tripole, cube-sphere, and icosahedral grids. To show that generated code matches hand-written code in terms of performance we will show compare the scalability of GridLib and GridGen generated code against small hand-written examples.

In the second paper we will expand our abstractions to allow for non-compact stencils and reconduct our experiments from the first paper with non-compact stencils to demonstrate that we don't lose our matching performance. In this paper we show results for CGPOP and SWM.

In the third paper we will compare the performance of GridGen generated code with and without optimization on the CGPOP and SWM miniapps.

We will run our experiments on our department's multicore machines and the ISTEC Cray machine. We will also apply for small allocations for time on NCAR machines such as YellowStone and perform experiments there if compute time is granted.

4.3 Project Tasks

In order to update the GridGen package for each of the three papers there are a number of design and implementation tasks that we must complete. In the remainder of this section we list these tasks:

Summer and Fall 2012 (Work for first paper): Initial design and implementation of the GridGen package

Designing and implementing GridLib

- Implement the region, subgrid, border-mapping, grid, distribution, communication plan, and data object abstractions discussed in [36].
- Design and implement convenience functions for stitching subgrids together using the patterns illustrated in Figure 4. These functions provide a higher level of abstraction for representing grid topology than border mappings.
- Design and implement functions that representing distributions in a high-level fashion We will base these functions on the distribution pragmas available in High Performance Fortran [21, 31].
- Design and implement an algorithm conducting a communication plan. We discuss communication plan abstraction in [36].
- Design and implement an algorithm for generating communication plans given a distribution.
- Implement function for applying pointwise update operations that assume that the halo has a single element depth. This function will be passed a data object to operate on and a function that is applied to each value in the grid (for example: the `sumNeighbors` function in Figure 7).
- Implement function for applying reductions on data-objects.
- Implement functions that read in and write out grid, distribution, and communication plan objects to files.
- Create regression tests that ensure proper implementation of the previous tasks.

Designing and implementing GridGen

- Implement the Region, subgrid, border-mapping, grid, distribution, communication plan, and data object abstractions discussed in the IPDPS paper in GridGen.
- Implement functions for reading in and outputting grid, distribution, and schedule objects to files.
- Create an interface of GridLib that can pass through the ROSE compiler framework frontend.

- Create objects that represent GridLib calls (GridLibCall objects)
- Create a visitor that extracts GridLib calls and wraps them in GridLibCall objects.
- Create a generic visitor class that steps through extracted GridLibCall objects.
- Create regression tests that ensure proper implementation of the previous tasks.

Designing and implementing GridViz

- Implement a visualization tool that outputs .svg files for grids that have been placed using the adjacent pattern without offset.
- Enable visualization for when adjacency has an offset.
- Enable visualization for wrapping, mirroring, and folding patterns.

Fall 2012 (Work for second paper): Extending the GridGen package and its abstractions for non-compact stencils

Updating GridLib

- Update border-map abstraction to allow for maps that have can have greater than single element depth.
- Update communication-plan generation algorithm in GridLib so that it can populate halos of depth-n.
- Lazily expand halo size and redo communication when pointwise-update algorithms access elements outside of halo.
- Expand data-objects to include a time dimension.
- Update pointwise-update algorithm to allow accessing previously updated elements.

Updating GridGen

- Update abstraction in GridGen to account for changes made to GridLib.
- Update communication-plan generation algorithm in GridGen so that it can populate halos of depth-n.
- Update pointwise update algorithm generator.

Spring 2013 (Work for third paper): Automating optimizations in GridGen and implementing CGPOP and SWM with the GridGen package

Updating GridLib

- Update the data-object abstraction (and its implementation) to include blank blocks. A blank block does not store data and is used in CGPOP to represent a block of land (which does not require simulation in an ocean program).
- Enable programmers to choose that data blocks may be represented using a compressed data format like in CGPOP
- Update communication plan generation and conduction algorithms to allow for new data-object format.
- Allow pointwise update algorithm to allow for new data-object format.
- Allow reduction algorithm to allow for new data-object format.
- Create an abstraction in GridLib for storing edge-weights.

Representing CGPOP and SWM

- Structure CGPOP's and SWM's native grids.
- Create data distributions that mirror those seen in CGPOP and SWM.
- Read in initial CGPOP and SWM data.
- Implement the algorithms seen in CGOP and SWM.
- Implement computation/communication overlap.
- Implement overlapping tiles optimization.

Updating GridGen

- Update GridGen to account for new edge weight abstraction.
- Determine from Section 2.2 a set of optimizations that are likely to improve the performance of CGPOP and SWM.
- Have GridGen apply these optimizations to its generated code.

5 Limitations and Future Work

In our proposed work we will construct abstractions for representing semi-regular grids, stencil computations, and data-distributions. These abstractions address the research problem of how to separate algorithm from grid structure and parallelization details. We intend to construct a set of tools in the GridGen package that will operate on semi-regular grids. This tool will initially work on compact stencils and later on noncompact stencils. We will this tool apply optimizations to its generated stencil code.

Our proposed approached is limited to operating on two dimensional, statically defined, semi-regular grids. We choose this class of grids because it is seen in important Earth

science applications such as POP. Future work could focus on abstracting the GridGen package to work will additional types of grids (for example: high-dimensionality grids or adaptively refined grids). Additional work could be done so that GridGen could target other architectures, for example: GPUs.

References

[1] Community Earth System Model.
`http://www.cesm.ucar.edu/`.

[2] GPU parallelization of NIM.
`http://www.esrl.noaa.gov/gsd/ab/ac/GPU_Parallelization_NIM.html`.

[3] Rose compiler project.
`http://www.rosecompiler.org`.

[4] Spherical geodesic grids: A new approach to modeling the climate.
`http://kiwi.atmos.colostate.edu/BUGS/geodesic/`.

[5] MPAS: Model for prediction across scales.
`http://mpas.sourceforge.net/`, December 2011.

[6] Website: Design and testing of a global cloud resolving model.
`http://kiwi.atmos.colostate.edu/gcrm/`, July 2012.

[7] Ramprasad Venkataraman Aaron Becker and Laxmikant V. Kale. Patterns for Overlapping Communication and Computation. In *Workshop on Parallel Programming Patterns (ParaPLOP 2009)*, June 2009.

[8] Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiatowicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick. A view of the parallel computing landscape. *Communications of the ACM*, 52(10):56–67, 2009.

[9] Bradford L. Chamberlain, Sung eun Choi, Steven J. Deitz, and Lawrence Snyder. The high-level parallel language zpl improves productivity and performance. In *In Proceedings of the IEEE International Workshop on Productivity and Performance in High-End Computing*, 2004.

[10] Matthias Christen, Olaf Schenk, and Helmar Burkhart. Automatic code generation and tuning for stencil kernels on modern shared memory architectures. *Comput. Sci.*, 26:205–210, June 2011.

[11] Matthias Christen, Olaf Schenk, and Helmar Burkhart. Automatic code generation and tuning for stencil kernels on modern shared memory architectures. *Computer Science - Research and Development*, 26(3-4):205–210, April 2011.

[12] P Colella, D T Graves, N D Keen, T J Ligocki, D F Martin, P W Mccorquodale,

D Modiano, P O Schwartz, T D Sternberg, and B Van Straalen. Chombo Software Package for AMR Applications: Design Document. Technical report, Lawrence Berkeley National Laboratory, 2009.

[13] Zachary DeVito, Niels Joubert, Francisco Palacios, Stephen Oakley, Montserrat Medina, Mike Barrientos, Erich Elsen, Frank Ham, Alex Aiken, Karthik Duraisamy, Eric Darve, Juan Alonso, and Pat Hanrahan. Liszt: a domain specific language for building portable mesh-based pde solvers. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 9:1–9:12, New York, NY, USA, 2011. ACM.

[14] Chen Ding and Ken Kennedy. Inter-array data regrouping. In *Twelfth International Workshop on Languages and Compilers for Parallel Computing*. Springer-Verlag, August 1999.

[15] Chris Ding and Yun He. A ghost cell expansion method for reducing communications in solving pde problems. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '01, pages 50–50, New York, NY, USA, 2001. ACM.

[16] J. J. Dongarra and A. R. Hinds. Unrolling loops in fortran. *Software - Practice and Experience*, 9:219–226, 1979.

[17] Jack Dongarra, Dennis Gannon, Geoffrey Fox, and Ken Kennedy. The impact of multicore on computational science software. *CTWatch Quarterly*, 3(1), February 2007.

[18] M. B. Giles, G. R. Mudalige, Z. Sharif, G. Markall, and P. H. J. Kelly. Performance Analysis and Optimization of the OP2 Framework on Many-Core Architectures. *The Computer Journal*, 55(2):168–180, July 2011.

[19] Ross Heikes and David A Randall. Numerical Integration of the Shallow-Water Equations on a Twisted Icosahedral Grid. Part II. A Detailed Description of the Grid and an Analysis of Numerical Accuracy. *Monthly Weather Review*, 123(6):1881–1887, 1995.

[20] Michael A. Heroux, Douglas W. Doerfler, Paul S. Crozier, James M. Willenbring, H. Carter Edwards, Alan Williams, Mahesh Rajan, Eric R. Keiter, Heidi K. Thornquist, and Robert W. Numrich. Improving performance via mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, 2009.

[21] High Performance Fortran Forum. High Performance Fortran language specification, version 1.0. Technical Report CRPC-TR92225, Houston, Tex., 1993.

[22] P. Jones. Parallel Ocean Program (POP) user guide. Technical Report LACC 99-18, Los Alamos National Laboratory, March 2003.

[23] Shoaib Kamil, Cy Chan, Samuel Williams, Leonid Oliker, John Shalf, Mark Howison, and E. Wes Bethel. A generalized framework for auto-tuning stencil computations. In *In Proceedings of the Cray User Group Conference*, 2009.

[24] Markus Kowarschik and Christian Wei. An overview of cache optimization techniques and cache-aware numerical algorithms. In *Algorithms for Memory Hierarchies Advanced Lectures, volume 2625 of Lecture Notes in Computer Science*, pages 213–232. Springer, 2003.

[25] Samuel Larsen and Saman Amarasinghe. Exploiting superword level parallelism with multimedia instruction sets. In *Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*, PLDI '00, pages 145–156, New York, NY, USA, 2000. ACM.

[26] Chi-keung Luk, Bradley C Kuszmaul, and Charles E Leiserson. The Pochoir Stencil Compiler. *Artificial Intelligence*, 36:117–128, 2011.

[27] Naoya Maruyama. Physis : An Implicitly Parallel Programming Model for Stencil Computations on Large-Scale GPU-Accelerated Supercomputers Categories and Subject Descriptors. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 11:1—-11:12, Seattle, Washington, 2011. ACM.

[28] Naoya Maruyama, Tatsuo Nomura, Kento Sato, and Satoshi Matsuoka. Physis: an implicitly parallel programming model for stencil computations on large-scale gpu-accelerated supercomputers. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 11:1–11:12, New York, NY, USA, 2011. ACM.

[29] Ross J Murray. Explicit Generation of Orthogonal Grids for Ocean Models. *Journal of Computational Physics*, 126(2):251–273, 1996.

[30] Dan Quinlan. Rose: Compiler support for object-oriented frameworks. In *Proceedings of Conference on Parallel Compilers (CPC2000), Aussois, France*, volume 10 of *Parallel Processing Letters*. Springer Verlag, 2000.

[31] Harvey Richardson. High performance fortran: history, overview and current developments. Technical report, 1.4 TMC-261, Thinking Machines Corporation, 1996.

[32] Robert Sadourny. Conservative Finite-Difference Approximations of the Primitive Equations on Quasi-Uniform Spherical Grids. *Monthly Weather Review*, 100(2):136–144, February 1972.

[33] R. D. Smith and S Kortas. Curvilinear coordinates for global ocean models. Technical report, Los Alamos National Laboratory, LA-UR-95-1146, 1995.

[34] Andrew Stone, John Dennis, and Michelle Strout. Establishing a miniapp as a programmability proxy. In *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 333–334, New York, NY, USA, 2012. ACM.

[35] Andrew Stone, John Dennis, and Michelle Mills Strout. The CGPOP miniapp, version 1.0. Technical Report Technical Report CS-11-103, Colorado State University, July 1 2011.

[36] Andrew Stone and Michelle Mills Strout. Abstractions for defining semi-regular grids orthogonally from stencils. *Unpublsihed: will be submitted to the 27th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2013.

[37] Hirofumi Tomita, Motohiko Tsugawa, Masaki Satoh, and Koji Goto. Shallow Water Model on a Modified Icosahedral Geodesic Grid by Using Spring Dynamics. *Journal of Computational Physics*, 174(2):579–613, 2001.

[38] Didem Unat, Xing Cai, and Scott B. Baden. Mint: realizing cuda performance in 3d stencil methods with annotated c. In *Proceedings of the international conference on Supercomputing*, ICS '11, pages 214–224, New York, NY, USA, 2011. ACM.

[39] Michael Wolfe. Iteration space tiling for memory hierarchies. In *Proceedings of the Third SIAM Conference on Parallel Processing for Scientific Computing*, pages 357–361, Philadelphia, PA, USA, 1989. Society for Industrial and Applied Mathematics.

[40] Xing Zhou, Jean-Pierre Giacalone, María Jesús Garzarán, Robert H. Kuhn, Yang Ni, and David Padua. Hierarchical overlapped tiling. In *Proceedings of the Tenth International Symposium on Code Generation and Optimization*, CGO '12, pages 207–218, New York, NY, USA, 2012. ACM.

[41] D. W. Zingg. A review of high-order and optimized finite-difference methods for simulating linear wave phenomena. In *AIAA Paper 97-2088, 13th AIAA Computational Fluid Dynamics Conference, Snowmass, June, ABSTRACT MULTI-DIMENSIONAL UPWIND LEAPFROG SCHEMES AND THEIR APPLICATIONS*, 1996.

[42] David W. Zingg. Comparison of high-accuracy finite-difference methods for linear wave propagation. *SIAM J. Sci. Comput*, pages 476–502.