Frameworks in Complex Multiphysics HPC Applications

CS267 – Spring 2014
John Shalf
Department Head for Computer Science: Computing Research Division
CTO: National Energy Research Supercomputing Center
Lawrence Berkeley National Laboratory
With contributions from: Gabrielle Allen, Tom Goodale, Eric Schnetter, Ed Seidel (AEI/LSU), Phil Colella, Brian Van Straalen (LBNL)

March 18, 2014

Technology Challenges: Creating Extremely Complex Machine Architectures

Power is leading constraint for future performance growth

Reliability going down for large-scale systems, but also to get more energy efficiency for small systems

By 2018, cost of a FLOP will be less than cost of moving 5 mm across the chip's surface (locality will really matter)

Application Code Complexity

- Application Complexity has Grown
  - Big Science on leading-edge HPC systems is a multi-disciplinary, multi-institutional, multi-national efforts! (and we are not just talking about particle accelerators and Tokamaks)
  - Looking more like science on atom-smashers

- Advanced Parallel Languages are Necessary, but NOT Sufficient!
  - Need higher-level organizing constructs for teams of programmers
  - Languages must work together with frameworks for a complete solution!

Example: Grand Challenge Simulation Science

NASA Neutron Star Grand Challenge
- 5 US Institutions
- Towards colliding neutron stars

Gamma Ray Bursts Core Collapse Supernova
- 10 Inst x 10 years
- Multiple disciplines
- GR
- Hydro
- Chemistry
- Radiation Transp
- Analytic Topology

NSF Black Hole Grand Challenge
- 8 US Institutions, 5 years
- Towards colliding black holes

Examples of Future of Science & Engineering
- Require Large Scale Simulations, at edge of largest computing sys
- Complex multi-physics codes with millions of lines of codes
- Require Large Geo-Distributed Cross-Disciplinary Collaborations
Application Code Complexity

- HPC is looking more and more like traditional “big science” experiments.
- QBox: Gordon Bell Paper title page
  - It’s just like particle physics papers!
  - Looks like discovery of the Top Quark!

Community Codes & Frameworks
(hiding complexity using good SW engineering)

- Clearly separate roles and responsibilities of your expert programmers from that of the domain experts/scientist/users (productivity layer vs. performance layer)
- Define a social contract between the expert programmers and the domain scientists
- Facilitates/Enforces style/discipline for software engineering to ensure correctness
- Hides complex domain-specific parallel abstractions from scientist/users to enable performance (hence, most effective when applied to community codes)
- Allow scientists/users to code nominally serial plug-ins that are invoked by a parallel "driver" (either as DAG or constraint-based scheduler) to enable productivity

Benefits and Organizing Principles

- Other “frameworks” that use same organizing principles (and similar motivation)
  - NEURON (parallel implementation of Genesis neurodyn)
  - SIERRA (finite elements/structural mechanics)
  - UPIC and TechIX (generalized code frameworks for PIC codes)
  - Chombo: AMR on block-structured grids (its hard)
  - Common feature is that computational model is well understood and broadly used (seems to be a good feature for workhorse “languages”)
- Common benefits (and motivations) are
  - Modularity (composition using higher-level semantics)
  - Segmenting expertise / Separation of Concerns
  - Unit Testing: This was the biggest benefit
  - Performance analysis (with data aggregated on reasonable semantic boundaries)
  - Correctness testing (on reasonable semantic boundaries)
  - Enables reuse of "solver" components. Replace “driver” if you have a different hardware platform.
Benefits cont.

Enabling Collaborative Development!

- They enable computer scientists and computational scientists to play nicely together
  - No more arguments about C++ vs. Fortran
  - Easy unit-testing to reduce finger pointing (are the CS weenies "tainting the numerics") (also good to accelerate V&V)
  - Enables multidisciplinary collaboration (domain scientists + computer jocks) to enable features that would not otherwise emerge in their own codes
    - Scientists write code that seem to never use "new" features
    - Computer jocks write code that no reasonable scientist would use
- Advanced CS Features are trivially accessible by Application Scientists
  - Just list the name of the module and it is available
  - Also trivially unit-testable to make sure they don’t change numerics
- Also enables sharing of physics modules among computational scientists
  - The hardest part is agreeing upon physics interfaces (there is no magic!)
  - Nice, but not actually not as important as the other benefits (organizing large teams of programmers along the lines of their expertise is the

Framework vs. Libraries

- Library
  - User program invokes library (imperative execution model offers limited scheduling freedom)
  - User defines presents data layout to library (compiler and system has limited freedom to reorganize to match physical topology of underlying system hardware)
- Framework
  - Framework invokes user plug-in (declarative execution model)
  - Only operation on data given (well defined scope for side-effects)
  - Functional semantics provide more scheduling freedom

Framework Taxonomy

Minimal Component Interoperability: Shallow Component Interoperability: Deep Component Interoperability:

- Structures
- Fluids
- Acoustics
- Electro-Magnetic

- Structures
- Fluids
- Acoustics
- Electro-Magnetic

- Structures
- Fluids
- Acoustics
- Electro-Magnetic

- Common infrastructure

- Physics models are relatively uncoupled
  - May exchange static datasets through flat files.
- Physics models are loosely coupled
  - Data management and parallelization is independent in each module.
  - Exchange common data via wrappers (web services, etc.).
- Physics models are tightly coupled
  - Data exchange across shared service infrastructure.

Integration is invasive: how much will you put up with?

Frameworks vs. Libraries

(Observation by Koushik Sen: view.eecs.berkeley.edu)

- A parallel program may be composed of parallel and serial elements
- Serial code invoking parallel libraries
- Parallel patterns with serial plug-ins
- Composition may be recursive

Parallel Dwarf Libraries
- Dense matrices
- Sparse matrices
- Spectral
- Combinational
- (Un) Structured Grid

Parallel Patterns/Frameworks
- Map Reduce
- Graph traversal
- Dynamic programming
- Backtracking/B&B
- Graphical models
- N-Body
- (Un) Structured Grid
### Separation of Concerns

#### Segmented Developer Roles

<table>
<thead>
<tr>
<th>Developer Roles</th>
<th>Domain Expertise</th>
<th>CS/Coding Expertise</th>
<th>Hardware Expertise</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application:</strong> Assemble solver modules to solve science problems. (e.g. combine hydro+GR +elliptic solver w/MPI driver for Neutron Star simulation)</td>
<td>Einstein</td>
<td>Elvis</td>
<td>Mort</td>
</tr>
<tr>
<td><strong>Solver:</strong> Write solver modules to implement algorithms. Solvers use driver layer to implement “idiom for parallelism”. (e.g. an elliptic solver or hydrodynamics solver)</td>
<td>Elvis</td>
<td>Einstein</td>
<td>Elvis</td>
</tr>
<tr>
<td><strong>Driver:</strong> Write low-level data allocation/placement, communication and scheduling to implement “idiom for parallelism” for a given “dwarf”. (e.g. PUGH)</td>
<td>Mort</td>
<td>Elvis</td>
<td>Einstein</td>
</tr>
</tbody>
</table>

### Observation on Domain-Specific Frameworks

- **Frameworks and domain-specific languages**
  - Enforce coding conventions for big software teams
  - Encapsulate a domain-specific “idiom for parallelism”
  - Create familiar semantics for domain experts (more productive)
  - Clear separation of concerns (separate implementation from specification)

- **Common design principles for frameworks from SIAM CSE07 and DARPA Ogden frameworks meeting**
  - Give up main(): schedule controlled by framework
  - Stateless: Plug-ins only operate on state passed-in when invoked
  - Bounded (or well-understood) side-effects: Plug-ins promise to restrict memory touched to that passed to it (same as CILK)
Cactus

- Framework for HPC: code development, simulation control, visualisation
- Manage increased complexity with higher level abstractions, e.g. for inter-node communication, intra-node parallelisation
- Active user community, 10+ years old

Many of these slides are almost 10 years old!
- Supports collaborative development
- Is this a language or just structured programming? (Why is it important to answer this question?)

Cactus Features

- Scalable Model of Computation
  - Cactus provides "idiom" for parallelism
  - Idiom for Cactus is parallel boundary exchange for block structured grids
  - Algorithm developers provide nominally "serial" plug-ins
  - Algorithm developers are shielded from complexity of parallel implementation
  - Neuron uses similar approach for scalable parallel idiom
- Build System
  - User does not see makefiles (just provides a list of source files in a given module)
  - "known architectures" used to store accumulated wisdom for multi-platform builds
  - Write once and run everywhere (laptop, desktop, clusters, petaflop HPC)
- Modular Application Composition System
  - This is a system for composing algorithm and service components together into a complex composite application
  - Just provide a list of "modules" and they self-organize according to constraints (less tedious than explicit workflows)
  - Enables unit testing for V&V of complex multiphysics applications
- Language Neutrality
  - Write modules in any language (C, C++, F77, F90, Java, etc...)
  - Automatically generates bindings (also hidden from user)
  - Overcomes age-old religious battles about programming languages

Cactus User Community

- General Relativity: worldwide usage
  - LSU(USA), AEI(Germany), UNAM (Mexico), Tuebingen(Germany), Southampton (UK), Sissa(Italy), Valencia (Spain), University of Thessaloniki (Greece), MPA (Germany), RIKEN (Japan), TAT(Denmark), Penn State (USA), University of Texas at Austin (USA), University of Texas at Brownsville (USA), WashU (USA), University of Pittsburgh (USA), University of Arizona (USA), Washburn (USA), UIB (Spain), University of Maryland (USA), Monash (Australia)
- Astrophysics
  - Zeus-MP MHD ported to Cactus (Mike Norman: NCSA/UCSD)
- Computational Fluid Dynamics
  - KISTI
  - DLR: (turbine design)
- Chemistry
  - University of Oklahoma: (Chem reaction vessels)
- Bioinformatics
  - Chicago

Detecting Gravitational Waves

- Will uncover fundamentally new information about the universe
- LIGO, VIRGO (Pisa), GEO600,… $1 Billion Worldwide
- Was Einstein right? 5-10 years, we’ll see!

GR requires solution of dozens of coupled, nonlinear hyperbolic-elliptic equations with 1000’s of terms (barely have the capability to solve after a century of development)
- Detect GR Waves…pattern matching against numerical templates to enhance signal/noise ratio
- Understand them…just what are the waves telling us?

Hanford Washington Site

Amplitude is at 10^-18 meter

40 seconds 0.01 0.02 seconds

3/18/14 18
Cactus components (terminology)

- **Thorns** (modules):
  - Source Code
  - CCL: Cactus Configuration Language (Cactus C&C description)
    - Interface/Types: polymorphic datastructures instantiated in “driver-independent” manner
    - Schedule: constraints-based schedule
    - Parameter: must declare free parameters in common way for introspection, steering, GUIs, and common input parameter parser.

- **Driver**: Separates implementation of parallelism from implementation of the “solver” (can have Driver for MPI, or threads, or CUDA)
  - Instantiation of the parallel datastructures (control of the domain-decomposition)
  - Handles scheduling and implementation of parallelism (threads or whatever)
  - Implements communication abstraction
  - Drive must own all of these

- **Flesh**: Glues everything together
  - Just provide a “list” of modules and they self-assemble based on their constraints expressed by CCL
  - CCL not really a language

Idiom for Parallelism in Cactus

- The central idiom for the Cactus model of computation is boundary exchange
  - Each module (algorithm plug-in) is passed a section of the global grid.

- The actual parallel driver (implemented in a module)
  - Driver decides how to decompose grid across processors and exchange ghost zone information
  - Each module is presented with a standard interface, independent of the driver
  - Can completely change the driver for shared memory, multicore, message passing without requiring any change of the physics modules

- Standard driver distributed with Cactus (PUGH) is for a parallel unigrid and uses MPI for the communication layer
  - PUGH can do custom processor decomposition and static load balancing

- Same idiom also works for AMR and unstructured grids!!! (no changes to solver code when switching drivers)
  - Carpet (Erik Schnetter’s AMR driver)
  - DAGH/GrACE driver for Cactus
  - SAMRAI driver for Cactus

Scalar Wave Model Problem

Scalar waves in 3D are solutions of the hyperbolic wave equation:

\[-\phi_{tt} + \phi_{xx} + \phi_{yy} + \phi_{zz} = 0\]

**Initial value problem**: given data for \(\phi\) and its first time derivative at initial time, the wave equation says how it evolves with time
Numerical solve by discretising on a grid, using explicit finite differencing (centered, second order)

\[
\phi_{n+1}^{i,j,k} = 2\phi_{n}^{i,j,k} - \phi_{n-1}^{i,j,k} + \frac{\Delta t^2}{\Delta x^2} (\phi_{n}^{i+1,j,k} - 2\phi_{n}^{i,j,k} + \phi_{n}^{i-1,j,k}) + \frac{\Delta t^2}{\Delta y^2} (\phi_{n}^{i,j+1,k} - 2\phi_{n}^{i,j,k} + \phi_{n}^{i,j-1,k}) + \frac{\Delta t^2}{\Delta z^2} (\phi_{n}^{i,j,k+1} - 2\phi_{n}^{i,j,k} + \phi_{n}^{i,j,k-1})
\]

Numerical Method

- Finite grid, so need to apply outer boundary conditions
- Main parameters:
  - grid spacings: \( \Delta t, \Delta x, \Delta y, \Delta z \), which coords?, which initial data?
- Simple problem, analytic solutions, but contains many features needed for modelling more complex problems

Example Stand Alone Code: Main.f

```fortran
program WaveToy
    implicit none
    integer i,j,k
    do i=1,nx
        do j=1,ny
            do k=1,nz
                x(i,j,k) = dx*(i-1) + x_origin
                y(i,j,k) = dy*(j-1) + y_origin
                z(i,j,k) = dz*(k-1) + z_origin
                r(i,j,k) = sqrt(x(i,j,k)**2 + y(i,j,k)**2 + z(i,j,k)**2)
            end do
        end do
    end do
    open(unit=11,file="out.x")
    open(unit=12,file="out.y")
    open(unit=13,file="out.z")
    call InitialData
    call Output
    do iteration = 1,nt
        call Evolve
        if (mod(iteration,10).eq.0) call Output
    end do
    stop
end
```

Standalone Serial Program

- Setting up parameters
- Setting up grid and coordinate system
- Opening output files
- Setting up initial data
- Performing iteration 10
- Performing iteration 20
- Performing iteration 30
- Performing iteration 40
- Performing iteration 50
- Performing iteration 60
- Performing iteration 70
- Performing iteration 80
- Performing iteration 90
- Performing iteration 100
- Done
Making a “Thorn” (a Cactus Module)

---

```
program WaveToy
  implicit real*8 (a-h,o-z)
  integer n, m, p, i, j, k, nx, ny, nz
  real*8 dx, dy, dz, x_origin, y_origin, z_origin
  real*8 x(ny,nz), y(ny,nz), z(ny,nz)
  real*8 r(ny,nz), t(ny,nz)
  real*8 a(10, 10), x0, x1, x2, x3
  integer i, j, k

  nx = 30
  ny = 30
  nz = 30
  dx = 0.02
  dy = 0.02
  dz = 0.02
  x_origin = 0.5
  y_origin = 0.5
  z_origin = 0.5

  do i = 1, nx
    do j = 1, ny
      do k = 1, nz
        x(i, j, k) = (i-1) * dx + x_origin
        y(i, j, k) = (j-1) * dy + y_origin
        z(i, j, k) = (k-1) * dz + z_origin
      end do
    end do
  end do

  open (unit=11, file="data.xl")
  open (unit=12, file="data.yl")
  open (unit=13, file="data.zl")

  call InitialData

  call Evolve
  if (mod(iteration,10).eq.0) call Output

end program WaveToy
```

---

**Throw the rest of this stuff away (less writing)**

And get parallelism, modularity, and portability for free

Abstraction Enables Auto-Tuning

- The following example shows how the framework abstractions enable auto-tuning of the parallel performance of a code without any change to the higher-levels of the framework
  - Normally people accuse abstractions of reducing performance
  - Framework abstractions *enable* performance tuning!!!

Dynamic Adaptation (auto-tuning)

- Automatically adapt to bandwidth latency issues
- Application has NO KNOWLEDGE of machines(s) it is on, networks, etc
- Adaptive techniques make NO assumptions about network
- Adaptive MPI unigrid driver required NO changes to the physics components of the application!! (plug-n-play!)
- Issues:
  - More intelligent adaption algorithm
  - Eg if network conditions change faster than adaption…

Throw the rest of this stuff away (less writing)

And get parallelism, modularity, and portability for free

Making a “Thorn” (a Cactus Module)

---

```
program WaveToy
  implicit real*8 (a-h,o-z)
  integer n, m, p, i, j, k, nx, ny, nz
  real*8 dx, dy, dz, x_origin, y_origin, z_origin
  real*8 x(ny,nz), y(ny,nz), z(ny,nz)
  real*8 r(ny,nz), t(ny,nz)
  real*8 a(10, 10), x0, x1, x2, x3
  integer i, j, k

  nx = 30
  ny = 30
  nz = 30
  dx = 0.02
  dy = 0.02
  dz = 0.02
  x_origin = 0.5
  y_origin = 0.5
  z_origin = 0.5

  do i = 1, nx
    do j = 1, ny
      do k = 1, nz
        x(i, j, k) = (i-1) * dx + x_origin
        y(i, j, k) = (j-1) * dy + y_origin
        z(i, j, k) = (k-1) * dz + z_origin
      end do
    end do
  end do

  open (unit=11, file="data.xl")
  open (unit=12, file="data.yl")
  open (unit=13, file="data.zl")

  call InitialData

  call Evolve
  if (mod(iteration,10).eq.0) call Output

end program WaveToy
```

---

**Throw the rest of this stuff away (less writing)**

And get parallelism, modularity, and portability for free

Making a “Thorn” (a Cactus Module)

---

```
program WaveToy
  implicit real*8 (a-h,o-z)
  integer n, m, p, i, j, k, nx, ny, nz
  real*8 dx, dy, dz, x_origin, y_origin, z_origin
  real*8 x(ny,nz), y(ny,nz), z(ny,nz)
  real*8 r(ny,nz), t(ny,nz)
  real*8 a(10, 10), x0, x1, x2, x3
  integer i, j, k

  nx = 30
  ny = 30
  nz = 30
  dx = 0.02
  dy = 0.02
  dz = 0.02
  x_origin = 0.5
  y_origin = 0.5
  z_origin = 0.5

  do i = 1, nx
    do j = 1, ny
      do k = 1, nz
        x(i, j, k) = (i-1) * dx + x_origin
        y(i, j, k) = (j-1) * dy + y_origin
        z(i, j, k) = (k-1) * dz + z_origin
      end do
    end do
  end do

  open (unit=11, file="data.xl")
  open (unit=12, file="data.yl")
  open (unit=13, file="data.zl")

  call InitialData

  call Evolve
  if (mod(iteration,10).eq.0) call Output

end program WaveToy
```

---

**Throw the rest of this stuff away (less writing)**

And get parallelism, modularity, and portability for free

Making a “Thorn” (a Cactus Module)

---

```
program WaveToy
  implicit real*8 (a-h,o-z)
  integer n, m, p, i, j, k, nx, ny, nz
  real*8 dx, dy, dz, x_origin, y_origin, z_origin
  real*8 x(ny,nz), y(ny,nz), z(ny,nz)
  real*8 r(ny,nz), t(ny,nz)
  real*8 a(10, 10), x0, x1, x2, x3
  integer i, j, k

  nx = 30
  ny = 30
  nz = 30
  dx = 0.02
  dy = 0.02
  dz = 0.02
  x_origin = 0.5
  y_origin = 0.5
  z_origin = 0.5

  do i = 1, nx
    do j = 1, ny
      do k = 1, nz
        x(i, j, k) = (i-1) * dx + x_origin
        y(i, j, k) = (j-1) * dy + y_origin
        z(i, j, k) = (k-1) * dz + z_origin
      end do
    end do
  end do

  open (unit=11, file="data.xl")
  open (unit=12, file="data.yl")
  open (unit=13, file="data.zl")

  call InitialData

  call Evolve
  if (mod(iteration,10).eq.0) call Output

end program WaveToy
```

---

**Throw the rest of this stuff away (less writing)**

And get parallelism, modularity, and portability for free

Making a “Thorn” (a Cactus Module)

---

```
program WaveToy
  implicit real*8 (a-h,o-z)
  integer n, m, p, i, j, k, nx, ny, nz
  real*8 dx, dy, dz, x_origin, y_origin, z_origin
  real*8 x(ny,nz), y(ny,nz), z(ny,nz)
  real*8 r(ny,nz), t(ny,nz)
  real*8 a(10, 10), x0, x1, x2, x3
  integer i, j, k

  nx = 30
  ny = 30
  nz = 30
  dx = 0.02
  dy = 0.02
  dz = 0.02
  x_origin = 0.5
  y_origin = 0.5
  z_origin = 0.5

  do i = 1, nx
    do j = 1, ny
      do k = 1, nz
        x(i, j, k) = (i-1) * dx + x_origin
        y(i, j, k) = (j-1) * dy + y_origin
        z(i, j, k) = (k-1) * dz + z_origin
      end do
    end do
  end do

  open (unit=11, file="data.xl")
  open (unit=12, file="data.yl")
  open (unit=13, file="data.zl")

  call InitialData

  call Evolve
  if (mod(iteration,10).eq.0) call Output

end program WaveToy
```
**Fault Tolerance**

- Need checkpointing/recovery on steroids, need to cope with partial failure
- Checkpoint is transparent to application (uses introspection)
  - architecture independent (independent of system HW and SW)
- Able to change number of active nodes
- Example: keep log of inter-processor messages, so that a lost node can be replaced
- Contain failure, continue simulation

Regular checkpointing

“Localized” checkpointing

**Nomadic Application Codes** (Foster, Angulo, Cactus Team…)

- Running At UC
- Load applied
- 3 successive Resource contract discovery violations & migration
- Running At UIUC

**Hybrid Communication Models**

- New “multicore” driver required no changes to physics components!
- Use MPI between nodes, OpenMP within nodes

- Common address space enables more cache optimisations
- Cactus framework offers abstraction layer for parallelisation: basic OpenMP features work as black box (central idiom)
Remote Monitoring/Steering:
Thorn HTTPD and SMS Messaging

- Thorn which allows simulation any to act as its own web server.
- Connect to simulation from any browser anywhere ... collaborate.
- Monitor run: parameters, basic visualization, ...
- Change steerable parameters.
- Get Text Messages from your simulation or chat with it on IM!

Remote Visualization

www.cactuscode.org/VizTools

OpenDX
IsoView
gnuplot
xgraph
Amira
LCAVision

Another Framework Example

PETSc

Slides from: Barry Smith, Jed Brown, Karl Rupp, Matthew Knepley

Argonne National Laboratory

PETSc Software Interfaces and Structure
Computation and Communication Kernels
- MPI, MPI-IO, BLAS, LAPACK

Profiling Interface

PETSc PDE Application Codes
- Object-Oriented Matrices, Vectors, Indices
- Grid Management

Linear Solvers
- Preconditioners + Krylov Methods

Nonlinear Solvers, Unconstrained Minimization

ODE Integrators

Visualization Interface

Data Objects

How to specify the mathematics of the problem?

How to handle Parallel computations?

Support for structured and unstructured meshes

Correctness and Performance Debugging

What debugging and monitoring aids it provides?

PETSc Software Interfaces and Structure

## Some Algorithmic Implementations in PETSc

### Nonlinear Solvers
- Newton-based Methods
- Line Search
- Trust Region
- Line Search
- Trust Region

### Time Steppers
- Euler
- Backward Euler
- Penalty-Time Stepping
- Other

### Krylov Subspace Methods
- GMRES
- CG
- CGS
- Bi-CG-STAB
- TFQMR
- Richardson
- Chebychev
- Other

### Preconditioners
- Additive Schwarz
- Block Jacobi
- Jacobi
- ILE
- ICC
- EIJ
- (Sequential only)
- Others

### Matrices
- Compressed Sparse Row (AIJ)
- Blocked Compressed Sparse Row (BAIJ)
- Block Diagonal (BDIAG)
- Dense
- Matrix-free
- Other

### Distributed Arrays

### Index Sets
- Indices
- Block Indices
- Stride
- Other

## Basic Program setup in PETSc

```c
#include "petsc.h"
int main( int argc, char *argv[] )
{
PetscInitialize(&argc,&argv);
    PetscPrintf(PETSC_COMM_WORLD,"Hello World
";
PetscFinalize();
    return 0;
}
```

## Basic Program Setup in PETSc (Fortran)

```fortran
program main
    integer ierr, rank
    #include "include/finclude/petsc.h"
    call PetscInitialize( PETSC_NULL_CHARACTER, ierr )
    call MPI_Comm_rank( PETSC_COMM_WORLD, rank, ierr )
    if (rank .eq. 0) then
        print *, 'Hello World'
    endif
    call PetscFinalize(ierr)
end
```

## Vectors and Matrices in PETSc

### VECTORS
Fundamental objects to store fields, right-hand side vectors, solution vectors, etc...

### Matrices
Fundamental Objects to store Operators
• PETSc vectors can be sequential (full vector is created in every process) or parallel (every process contains a part of the vector).

- Create a PETSc Vector
  VecCreate(MPI_Comm Comm, Vec *v)
  `v` vector
- Set the PETSc Vector type:
  VecSetType(Vec, VecType)
  `VecType` can be:
  - VEC_SEQ, VEC_MPI, or VEC_SHARED
- Set the PETSc vector size:
  VecSetSizes(Vec *v, int n, int N)
  Where `n` or `N` (not both) could be PETSC_DECIDE
- Destroy a PETSc Vector (important for storage)
  VecDestroy(Vec *v)

PETSC: Some Basic Vector Operations

```c
#include petscvec.h

int main(int argc,char **argv)
{
  Vec x;
  int n = 20, m = 4, ierr;
  VecCreateMPI(PETSC_COMM_WORLD, m, n, x);
  VecSetFromOptions(x);
  /* perform some vector operations */
  PetscFinalize();
  return 0;
}
```

PETSC: Some Basic Matrix Operations

- Create a PETSc Matrix
  MatCreate(MPI_Comm comm, Mat *A)
- Set the PETSc Matrix type
  MatSetType(Mat *A, MatType matype) (see next slides for types of matrices)
- Set the PETSc Matrix sizes
  MatSetsizes(Mat *A, PetscInt m, PetscInt n, PetscInt M, PetscInt N)
  Where `m, n` are the dimensions of local sub-matrix. `M, N` are the dimensions of the global matrix `A`
- Destroy a PETSc Matrix
  MatDestroy(Mat *A)
PETSc Matrix Types:
- default sparse AIJ (generic), MPIAIJ (parallel), SEQAIJ (sequential)
- block sparse AIJ (for multi-component PDEs): MPIAIJ, SEQAIJ
- symmetric block sparse AIJ: MPISBAIJ, SAEQSBAIJ
- block diagonal: MPIBDIAG, SEQBDIAG
- dense: MPIDENSE, SEQDENSE
- matrix-free
- many more formats (check documentation)

PETSC: Some Basic Matrix Operations

Every process will receive a set of consecutive and non-overlapping rows, the columns are determined by the matrix non-zero structure (max(ni) = N)

PETSC: Some Basic Vector Operations

- Input values to the matrix
  In PETSc a process can input values for blocks of the matrix that are not in its local matrix. PETSc makes sure these values get to the right places and corresponding processes.
  
  ```c
  MatSetValues(Mat mat,
                PetscInt m, PetscInt idxm[],
                PetscInt n, PetscInt idxn[],
                PetscScalar v[], InsertMode addv)
  ```

  - `idxm` is a vector of global row indices and `m` is the number of rows in `idxm`
  - `idxn` is a vector of global column indices and `n` is the number of columns in `idxn`
  - `v` is an array of `m X n` values
  - `addv` is either `ADD_VALUES` (accumulates) or `INSERT_VALUES` (sets)

PETSC: Some Basic Matrix Operations

- Assembling the parallel matrix
  (must do before calling solvers and other operations!)
  
  ```c
  MatAssemblyBegin(Mat mat, MatAssemblyType type)
  ```

  ```c
  MatAssemblyType:
  ```

  - `MAT_FLUSH_ASSEMBLY` use between `ADD_VALUES` and `INSERT_VALUES` in `MatSetValues`
  - `MAT_FINAL_ASSEMBLY` use after setting all the values in the matrix and before the matrix is used in the code

  ```c
  MatAssemblyEnd(Mat mat, MatAssemblyType type)
  ```
PETSC: Some Basic Matrix Operations

- Matrix vector multiplication
  \[ \text{MatMult(Mat } A, \text{Vec } y, \text{Vec } x) \ (y \neq x) \]
- Matrix viewing
  - \text{MatView(Mat } \text{mat, PetscViewer } \text{viewer})
  - \text{PetscViewer } \text{some viewer options:}
    - \text{PETSC_VIEWER_STDOUT_SELF} standard output (default)
    - \text{PETSC_VIEWER_STDOUT_WORLD} synchronized standard output, only rank 0 prints - others send to rank 0
    - \text{PETSC_VIEWER_DRAW_WORLD} graphical display of nonzero structure

PETSC: Some Basic Viewer Operations

- VIEWERS provide information on any PETSc conceptual Object
- VIEWERS can be setup inside the program or at execution time
- VIEWERS provide an interface for extracting data and making it available to other tools and libraries
  - vector fields, matrix contents
  - various formats (ASCII, binary)
  - Visualization
    - simple graphics created with X11.

PETSC: Some Basic Viewer Operations

- \text{MatView(Mat } A, \text{PetscViewer } v);\]
- With \text{PETSC_VIEWER_DRAW_WORLD}
- Other useful viewers can be set through \text{PETScViewerSetFormat:}
  - \text{PETSC_VIEWER_ASCII_MATLAB}
  - \text{PETSC_VIEWER_ASCII_DENSE}
  - \text{PETSC_VIEWER_ASCII_INFO}
  - \text{PETSC_VIEWER_ASCII_INFO DETAILED}

PETSC: Some Vector, Viewer and Matrix Examples

Included in the PETSc Distribution:
1) "$\text{PETSC_DIR/src/mat/tests/ex2.c}$
2) Use of $-$mat-view-info_detailed, etc
3) "$\text{PETSC_DIR/src/mat/tests/ex3.c}$
4) Use of $-$mat-view-draw
PETSc Linear System Solver Interface (KSP)
- Solve: \( Ax = b \)
- Based on the Krylov subspace methods with the use of a preconditioning technique to accelerate the convergence rate of the numerical scheme.

\[ (M_L^{-1}AM_R^{-1})(M_Rx) = M_L^{-1}b \]

For \( M_R = I \)

\[ r_L = M_L^{-1}b - M_L^{-1}Ax = M_L^{-1}r \] *PETSc Default*

To solve a Linear System, \( Ax = b \) in PETSc, one needs:
- Declare \( x \), \( b \) as PETSc vectors, and set the RHS \( b \)
- Declare the matrix \( A \), and explicitly set the matrix \( A \) when appropriate
- Set the Solver KSP:
  - Option 1:
    - Select the base Krylov subspace based solver
    - Select the preconditioner (Petsc PC)
  - Option 2:
    - Set the solver to use a solver from an external library

Schema of the program control flow

KSP Object:
- Is the key element to manipulate linear solver
- Stores the state of the solver and other relevant information like:
  - Convergence rate and tolerance
  - Number of iteration steps
  - Preconditioners
PETSc: Linear Solver - KSP Interface

- Create a KSP Object
  KSPCreate(MPI_Comm comm, KSP *ksp)
- Set KSP Operators
  KSPSetOperators(KSP *ksp, Mat Amat, Mat Pmat, MatStructure flag)
  Amat: is the original matrix from Ax=b
  Pmat: is the place holder for the preconditioning matrix (can be the same as A)
  flag: saves work while repeatedly solving linear systems of the same size using the same preconditioners. Possible values:
  SAME_NONZERO_PATTERN (same pattern for Pmat)
  DIFFERENT_NONZERO_PATTERN (different pattern for Pmat)
  SAME_PRECONDITIONER (identical Pmat)

- Solve Linear System
  KSPSolve(KSP *ksp, Vec b, Vec x)
- Get Iteration Number
  KSPSolve(KSP *ksp, int *its)
- Destroy Solver
  KSPDestroy(KSP *ksp)

- Set the type PETSc KSP solver
  KSPSetType(KSP *ksp, KSPType method)
  -ksp_type [cg,gmres,bicgs,tfqmr,...]
  -pc_type [lu,ilu,jacobi,sor,asm,...]
  More advanced options:
  -ksp_max_it <max_iters>
  -ksp_gmres_restart <restart>
  -pc_asm_overlap <overlap>
  -pc_asm_type [basic,restrict,interpolate,none]
  Many more, use -help to see other options
PETSc: Linear Solver - KSP Interface

- Setting up the Preconditioners
  ```c
  KSPGetPC(KSP ksp, PC *pc);
  PCSetType(PC *pc, const PCType type)
  ```

Use of solvers in external libraries

PETSc: Linear Solver - KSP Interoperable Interface

Table 4: PETSc Preconditioners

<table>
<thead>
<tr>
<th>Method</th>
<th>PCType</th>
<th>Options Database Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi</td>
<td>pcjacob</td>
<td>jacobit</td>
</tr>
<tr>
<td>Block Jacobi</td>
<td>pcblock</td>
<td>block</td>
</tr>
<tr>
<td>SOR (and SOR)</td>
<td>pcosor</td>
<td>sor</td>
</tr>
<tr>
<td>SOR with Element trick</td>
<td>pcosor</td>
<td>sor</td>
</tr>
<tr>
<td>Incomplete Cholesky</td>
<td>pc颐</td>
<td>lrc</td>
</tr>
<tr>
<td>Incomplete LU</td>
<td>pctlu</td>
<td>lts</td>
</tr>
<tr>
<td>Additive Schwarz</td>
<td>pcasm</td>
<td>same</td>
</tr>
<tr>
<td>Linear solver</td>
<td>pcpsp</td>
<td>ksp</td>
</tr>
<tr>
<td>Combination of preconditioners</td>
<td>pccomposite</td>
<td>composite</td>
</tr>
<tr>
<td>LU</td>
<td>pcg</td>
<td>lu</td>
</tr>
<tr>
<td>Cholesky</td>
<td>pccs</td>
<td>cholesky</td>
</tr>
<tr>
<td>No preconditioning</td>
<td>pccnone</td>
<td>none</td>
</tr>
<tr>
<td>Shell for user-defined PC</td>
<td>pccshell</td>
<td>shell</td>
</tr>
</tbody>
</table>

Table 5: Options for External Solvers

<table>
<thead>
<tr>
<th>MatType</th>
<th>PCType</th>
<th>MatSolverPackage</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>baij</td>
<td>cholmod</td>
<td>MAT_SOLVER SimpleName</td>
<td>daiuse pack</td>
</tr>
<tr>
<td>sopij</td>
<td>lu</td>
<td>MAT_SOLVER SimpleName</td>
<td>essl</td>
</tr>
<tr>
<td>sopij</td>
<td>lu</td>
<td>MAT_SOLVER SimpleName</td>
<td>lusol</td>
</tr>
<tr>
<td>sopij</td>
<td>lu</td>
<td>MAT_SOLVER SimpleName</td>
<td>matlab</td>
</tr>
<tr>
<td>sopij</td>
<td>lu</td>
<td>MAT_SOLVER SimpleName</td>
<td>rupsa</td>
</tr>
<tr>
<td>sopij</td>
<td>lu</td>
<td>MAT_SOLVER SimpleName</td>
<td>plpack</td>
</tr>
<tr>
<td>sopij</td>
<td>lu</td>
<td>MAT_SOLVER SimpleName</td>
<td>spoool</td>
</tr>
<tr>
<td>sopij</td>
<td>lu</td>
<td>MAT_SOLVER SimpleName</td>
<td>superlu</td>
</tr>
<tr>
<td>sopij</td>
<td>lu</td>
<td>MAT_SOLVER SimpleName</td>
<td>superlu_dist</td>
</tr>
<tr>
<td>sopij</td>
<td>lu</td>
<td>MAT_SOLVER SimpleName</td>
<td>umfpack</td>
</tr>
</tbody>
</table>

PETSc: Linear Solver Examples

Included in the PETSc Distribution:
1) `$PETSC_DIR/src/ksp/ksp/examples/tests/ex2.c`
2) `$PETSC_DIR/src/ksp/ksp/examples/tests/ex9.c`
(understand the use of multigrid in PETSc)

More Opportunities for Data Abstractions using Frameworks
Multi-Scale Proxy Architecture
(what do we need to reason about when designing a new code?)

Cores
• How Many
• Heterogeneous
• SIMD Width
Network on Chip (NoC)
• Are they equidistant or
• Constrained Topology (2D)
On-Chip Memory Hierarchy
• Automatic or Scratchpad?
• Memory coherency method?
Node Topology
• NUMA or Flat?
• Topology may be important
• Or perhaps just distance
Memory
• Nonvolatile / multi-tiered?
• Intelligence in memory (or not)
Fault Model for Node
• FIT rates, Kinds of faults
• Granularity of faults/recovery
Interconnect
• Bandwidth/Latency/Overhead
• Topology
Primitives for data move/sync
• Global Address Space or
•Synchronization primitives/Fences
Interconnect
• Bandwidth/Latency/Overhead
• Topology
Primitives for data move/sync
• Global Address Space or
•Synchronization primitives/Fences
Primitives for data move/sync
• Global Address Space or
•Synchronization primitives/Fences
Interconnect
• Bandwidth/Latency/Overhead
• Topology
Primitives for data move/sync
• Global Address Space or
•Synchronization primitives/Fences
For each parameterized machine attribute, can
• Ignore it: if ignoring it has no serious power/performance consequences
• Abstract it (virtualize): If it is well enough understood to support an automated
mechanism to optimize layout or schedule
  • This makes programmers life easier (one less thing to worry about)
• Expose it (unvirtualize): If there is not a clear automated way of make decisions
  • Must involve the human/programmer in the process (make pmodel more expressive)
  • Directives to control data movement or layout (for example)
Want result to be as simple as possible, but not neglect any
aspects of the machine that are important for performance

Cost of Data Movement Increasing Relative to Ops

FLOPs will cost less than on-chip data movement! (NUMA)

Data Locality
What are the big questions in Fast Forward
### Data Locality Management

**Vertical Locality Management** (spatio-temporal optimization)

**Horizontal Locality Management** (topology optimization)

### Research Thrusts in Data Movement

- **Math:**
  - **Old model:** move data to avoid flops
  - **New model:** use extra FLOPs to avoid data movement
  - **ExaCT Research:** Higher order methods and communication avoiding

- **Models:**
  - **Old model:** Parcel out work on-node and cache-coherence move data (data location follows work). Ignore distance & topology within node and between nodes.
  - **New Model:** Operate on data where it resides (work follows data location).
  - **ExaCT Research:** Tiling abstractions to express data locality info. AMR modeling to study interconnect/box placement interaction

- **SDMA/UQ:**
  - **Old model:** store everything on shared disk and look at it later
  - **New model:** do analysis workflow as much as possible in-situ
  - **ExaCT Research:** Using metaskeleton to evaluate benefits of different workflow approaches and their requirements for system-scale architecture.

### Expressing Hierarchical Layout

- **Old Model (OpenMP)**
  - Describe how to parallelize loop iterations
  - Parallel "DO" divides loop iterations evenly among processors
  - ... but where is the data located?

- **New Model (Data-Centric)**
  - Describe how data is laid out in memory
  - Loop statements operate on data where it is located
  - Similar to MapReduce, but need more sophisticated descriptions of data layout for scientific codes

```c
forall_local_data(i=0;i<NX;i++;A)
C[j]+=A[j]*B[i][j]);
```

### Data-Centric Programming Model

- **Building up a hierarchical layout**
  - Layout block coreblk (blockx,blocky);
  - Layout block nodeblk (nnx,nny,nnz);
  - Layout hierarchy myheirarchy (coreblk,nodeblk);
  - SH:

```c
forall_data threadIdx=( blockIdx[0]%blockx )*blocky + threadIdx[1]
forall_local_data(i=0;i<nx;i++;a)
forall_local_data(j=0;j<ny;j++;a)
forall_local_data(k=0;k<nz;k++;a)
(a00000[k]=C*[a[1]]...)
```

- **And if layout changes, this loop remains the same**

Satisfies the request of the application developers (minimize the amount of code that changes)
Tiling Formulation: abstracts data locality, topology, cache coherence, and massive parallelism

- Expose massive degrees of parallelism through domain decomposition
  - Represent an atomic unit of work
  - Task scheduler works on tiles
- Core concept for data locality
  - Vertical data movement
    - Hierarchical partitioning
  - Horizontal data movement
    - Co-locate tiles sharing the same data by respecting tile topology
- Multi-level parallelism
  - Coarse-grain parallelism: across tiles
  - Fine-grain parallelism: vectorization, instruction ordering within tile
- TiDA: Centralize and parameterize tiling information at the data structures
  - Direct approach for memory affinity management for data locality
  - Expose massive degrees of parallelism through domain decomposition
  - Overcomes challenges of relaxed coherency & coherence domains!!!
Iterate over the tiles by preserving data locality

- Provide a language construct to abstract loop traversal
  - Execute a tile in any order or execute elements in a tile in any order
  - Introduce parallelization strategy for tiles and elements

The new loop construct will
- Respect data layout and topology when we traverse the loop
- Morton order, linear order
- Let compiler and runtime pick the best traversal strategy
- Change parallelization strategy without changing the loop

Related Work:
- C++ lambda func in Raja
- Functors in Kokkos

The prototype for TiDA targets F90 base language

- Native support for multidimensional arrays

Framework
- Minimal invasion to the base language and existing codes
  - We can get quite far without implementing a compiler
  - Have to implement the optimization variants by hand

Directives
- Intermediate step, can be ignored, preferred by apps developers

Language Extension
- Changes the type system in a language
- Provides the compiler more opportunities to perform code transformations
- Our ultimate goal

Library-> Directives->Language

- Heterogeneity / Inhomogeneity
- Async Programming Models?
Assumptions of Uniformity is Breaking (many new sources of heterogeneity)

- Bulk Synchronous Execution
  - Heterogeneous compute engines (hybrid/GPU computing)
  - Fine grained power mgmt. makes homogeneous cores look heterogeneous
    - thermal throttling – no longer guarantee deterministic clock rate
  - Nonuniformities in process technology creates non-uniform operating characteristics for cores on a CMP
    - Near Threshold Voltage (NTV)
  - Fault resilience introduces inhomogeneity in execution rates
    - error correction is not instantaneous
    - And this will get WAY worse if we move towards software-based resilience

1/23/2013

Just Speeding up Components is Design Optimization
The really big opportunities for energy efficiency require codesign!

- Energy-limited design is a zero-sum-game
  - For every feature you ask for, you need to give something up
  - This is the “ground floor” for Co-Design
- Improving energy efficiency or performance of individual components doesn’t really need co-design
  - Memory is faster, then odds are that the software will run faster
  - If its better, that’s good!

1/23/2013

Example Near Threshold Voltage (NTV): Shekhar Borkar
The really big opportunities for energy efficiency require codesign!

- The really *big* opportunities to improve energy efficiency may require a shift in how we program systems
  - This requires codesign to evaluate the hardware and new software together
  - HW/SW Interaction unknown (requires HW/SW codesign)
- If software CANNOT exploit these radical hardware concepts (such as NTV), then it would be better to not have done anything at all!
Assumptions of Uniformity is Breaking
(many new sources of heterogeneity)

Asynchronous Execution Model

In this situation, AMR might be the solution (not the problem)

Conclusions on Heterogeneity

Sources of performance heterogeneity increasing
- Heterogeneous architectures (accelerator)
- Thermal throttling
- Performance heterogeneity due to transient error recovery

Current Bulk Synchronous Model not up to task
- Current focus is on removing sources of performance variation (jitter), is increasingly impractical
- Huge costs in power/complexity/performance to extend the life of a purely bulk synchronous model

Embrace performance heterogeneity: Study use of asynchronous computational models (e.g., LEGION and Rambutan, and other dataflow concepts from 1980s)

Summary

- Computational Science is increasingly carried out in large teams formed around applications frameworks
- Frameworks enable large and diverse teams to collaborate by organizing teams according to their capabilities
- Frameworks are modular, highly configurable, and extensible
- Isolation of applications, solver, and driver layers enables re-use in different applications domains, and scalability on new parallel architectures

The End
Chapter III

Addressing Petascale and Exascale Challenges

Addressing Petascale Challenges

- Expect ~1 M CPUs, need everything parallel (Amdahl): use performance modelling to improve codes
  - Cactus’ idiom for parallelism is scalable to millions of CPUs
  - Drivers can evolve without changing physics modules

- More cores/node tighten memory bottleneck: use dynamic, adaptive cache optimisations
  - Automatic code generation to select optimal cache strategy
  - Automatic generation for GP-GPU, Cell, and manycore targets

- Probably less memory/processor than today: use hybrid schemes (MPI + OpenMP) to reduce overhead
  - Drivers can be changed dramatically for multicore without requiring changes to physics modules

- Hardware failures “guaranteed”: use fault tolerant infrastructure
  - Cactus integrated checkpoint uses introspection to remain application-independent as well as system independent

XiRel: Improve Computational Infrastructure

- Sponsored by NSF PIF; collaboration between LSU/PSU/RIT/AEI
- Improve mesh refinement capabilities in Cactus, based on Carpet
- Prepare numerical relativity codes for petascale architectures
- Enhance and create new physics infrastructure for numerical relativity
- Develop common data and metadata management methods, with numrel as driver application

Cactus, Eclipse, Blue Waters (NSF Track-1 Supercomputing Project)

Source code
- cvs/svn
- edit
- compile
- debug

Performance data
- gather
- process
- display

Simulations
- submit
- monitor
- steer

Online databases
- Configuration files
- Performance data

Submit
Gather
Edit
Process
Display
Remote
Local

Gather
Process
Display
Application-Level Debugging and Profiling

- Sponsored by NSF SDCI
- As framework, Cactus has complete overview over programme and execution schedule
- Need to debug simulation at level of interacting components, in production situations, at scale
- Grid function declarations have rich semantics -- use this for visual debugging
- Combine profiling information with execution schedule, place calliper points automatically

Remote Visualization

- www.cactuscode.org/VizTools
- OpenDX
- LCAVision
- IsoView
- gnuplot
- xgraph
- Amira
- Soroce
- VolumView
- Visapult

Task Farm/Remote Viz/Steer Capabilities

- Current TFM Status in portal…
- Big BH Sim (LBL, NCSA, PSC, ...)
- Visapult
- BWC
- Baltimore

Cactus/Charm++

- Application
- Cactus Framework
- PUGH
- Carpet
- New Charming Drivers
- Charm++

Also drivers based on SAMRAI, PARAMESH
Summary of Cactus Capabilities

- Variety of science domains (highly configurable)
- Multi-Physics (modular)
- Petascale (tractable programming model for massive concurrency, performance, debugging, reliability)
- Combining HPC (batch systems) and interactivity (GUI), where possible
- Framework -- for any content

Framework Components

- Flesh: The glue that ties everything together (C&C language)
  - Supports composition of modules into applications (targets non-CS-experts)
  - Invokes modules in correct order (baseline scheduling)
  - Implements code build system (get rid of makefiles)
  - Implements parameter file parsing
  - Generates bindings for any language (Fortran, C, C++, Java)
- Driver: Implements idiom for parallelism
  - Implements “dwarf-specific” composite datatypes
  - Handles data allocation and placement (domain decomposition)
  - Implements communication pattern for “idiom for parallelism”
  - Implements thread-creation and scheduling for parallelism
- Solver/Module: A component implementing algorithm or other composable function
  - Can be written in any language (flesh handles bindings automatically)
  - Implementation of parallelism externalized, so developer writes nominally serial code with correct idiom. Parallelism handled by the “driver”.
  - Thorns implementing same functionality derived from same ‘abstract class’ of functionality such as “elliptic solver” (can have many implementations of elliptic solve. Select at compile time and/or at runtime)

More Information

- The Science of Numerical Relativity
  - http://jean-luc.aei.mpg.de
  - http://www.appleswithapples.org/
- Cactus Community Code
  - http://www.cct.lsu.edu
  - http://www.cactuscode.org/
  - http://www.carpetcode.org/
- Grid Computing with Cactus
  - http://www.astrogrid.org/
- Benchmarking Cactus on the Leading HPC Systems
  - http://crd.lbl.gov/~oliker
  - http://www.nerc.gov/projects/SDSA/reports
Examples:
Chombo
AMR

Block-Structured Local Refinement
- Refined regions are organized into rectangular patches.
- Refinement in time as well as in space for time-dependent problems.
- Local refinement can be applied to any structured-grid data, such as bin-sorted particles.

Cartesian Grid Representation of Irregular Boundaries
Based on nodal-point representation (Shortley and Weller, 1938) or finite-volume representation (Noh, 1964).

Efficient Embedded Boundary Multigrid Solvers
- In the EB case, the matrices are not symmetric, but they are sufficiently close to M-matrices for multigrid to work (nontrivial to arrange this in 3D).
- A key step in multigrid algorithms is coarsening. In the non-EB case, computing the relationship between the locations of the coarse and fine data involves simple integer arithmetic. In the EB case, both the data access and the averaging operations are more complicated.
- It is essential that coarsening a geometry preserves the topology of the finer EB representation.

\[ \nabla \cdot F = \frac{1}{x_{ref}} \int \nabla \cdot F \, dx = \frac{1}{x_{ref}} \sum_{\alpha} f_{ref} \cdot \delta_{\alpha} + \alpha_{B} F \cdot \delta_{B} = D \cdot F \]

Advantages:
- Grid generation is easy.
- Good discretization technology (e.g., finite differences on rectangular grids, geometric multigrid)
- Straightforward coupling to AMR (in fact, AMR is essential).
A Software Framework for Structured-Grid Applications

The empirical nature of multiphysics code development places a premium on the availability of a diverse and agile software toolset that enables experimentation. We accomplish this with a software architecture made up of reusable tested components organized into layers.

- **Layer 1**: Data and operations on unions of rectangles - set calculus, rectangular array library (with interface to Fortran). Data on unions of rectangles, with SPMD parallelism implemented by distributing boxes to processors. Load balancing tools (e.g., SFC).

- **Layer 2**: Tools for managing interactions between different levels of refinement in an AMR calculation - interpolation, averaging operators, coarse-fine boundary conditions.

- **Layer 3**: Solver libraries - multigrid solvers on unions of rectangles, AMR hierarchies, hyperbolic solvers; AMR time stepping.

- **Layer 4**: Complete parallel applications.

- **Utility Layer**: Support, interoperability libraries - API for HDF5 I/O, AMR data alias.

Mechanisms for Reuse

- **Algorithmic reuse.** Identify mathematical components that cut across applications. Easy example: solvers. Less easy example: Layer 2.

- **Reuse by templating data holders.** Easy example: rectangular array library - array values are the template type. Less easy example: data on unions of rectangles - "rectangular array" is a template type.

- **Reuse by inheritance.** Control structures (Iterative solvers, Berger-Oliger timestepping) are independent of the data, operations on that data. Use inheritance to isolate the control structure from the details of what is being controlled (interface classes).

Examples of Layer 1 Classes (BoxTools)

- **IntVect / \mathbb{Z}^d**: Can translate i, §i, coarsen i/s, refine i $\times$ s.

- **Box B \in \mathbb{Z}^d** is a rectangle: $B = [i_{\text{low}}, i_{\text{high}}]$. B can be translated, coarsened, refined. Supports different centerings (node-centered vs. cell-centered) in each coordinate direction.

- **IntVectSet / \mathbb{Z}^d** is an arbitrary subset of \mathbb{Z}^d. I can be shifted, coarsened, refined. One can take unions and intersections, with other IntVectSets and with Boxes, and iterate over an IntVectSet.

- **FArrayBox A(Box B, int nComps)**: multidimensional arrays of doubles or floats constructed with B specifying the range of indices in space; nComps the number of components. Real* FArrayBox::dataPtr returns the pointer to the contiguous block of data that can be passed to Fortran.

Layer 1 Reuse: Distributed Data on Unions of Rectangles

Provides a general mechanism for distributing data defined on unions of rectanglesonto processors, and communication between processors.

- **Metadata of which all processors have a copy**: BoxLayout is a collection of Boxes and processor assignments: $(B_k, p_k)_{k=1}^{n\text{Grids}}$. DisjointBoxLayout:public BoxLayout is a BoxLayout for which the Boxes must be disjoint.

- **template <class T> LevelData<T>** and other container classes hold data distributed over multiple processors. For each $k = 1 \ldots n\text{Grids}$, an "array" of type T corresponding to the box $B_k$ is located on processor $p_k$. Straightforward API's for copying, exchanging ghost cell data, iterating over the arrays on your processor in a SPMD manner.
Example: explicit heat equation solver, parallel case

- `LevelData<T>::exchange()`: obtains ghost cell data from valid regions on other patches
- `DataIterator`: iterates over only the patches that are owned on the current processor.

AMR Utility Layer

- API for HDF5 I/O.
- Interoperability tools. We have developed a framework-neutral representation for pointers to AMR data, using opaque handles. This will allow us to wrap Chombo classes with a C interface and call them from other AMR applications.
- Chombo Fortran - a macro package for writing dimension-independent Fortran and managing the Fortran / C interface.
- `Parmparse` class from BoxLib for handling input files.
- Visualization and analysis tools (VisIt).

First Light on LMC (AMR) Code Control Dependencies

Spiral Design Approach to Software Development

Scientific software development is inherently high-risk: multiple experimental platforms, algorithmic uncertainties, performance requirements at the highest level. The Spiral Design approach allows one to manage that risk, by allowing multiple passes at the software and providing a high degree of schedule visibility.

Software components are developed in phases.

- Design and implement a basic framework for a given algorithm domain (EB, particles, etc.), implementing the tools required to develop a given class of applications.
- Implement one or more prototype applications as benchmarks.
- Use the benchmark codes as a basis for measuring performance and evaluating design space flexibility and robustness. Modify the framework as appropriate.
- The framework and applications are released, with user documentation, regression testing, and configuration for multiple platforms.
Software Engineering Plan

- All software is open source: [http://seesar.lbl.gov/anag/software.html](http://seesar.lbl.gov/anag/software.html).
- Documentation: algorithm, software design documents; Doxygen manual generation; users’ guides.
- Implementation discipline: CVS source code control, coding standards.
- Portability and robustness: flexible make-based system, regression testing.
- Interoperability: C interfaces, opaque handles, permit interoperability across a variety of languages (C++, Fortran 77, Python, Fortran 90). Adaptors for large data items a serious issue, must be custom-designed for each application.

Replication Scaling Benchmarks

- Take a single grid hierarchy, and scale up the problem by making identical copies. Full AMR code (processor assignment, remaining problem setup) is done without knowledge of replication.
  - Good proxy for some kinds of applications scaleup.
  - Tests algorithmic weak scalability and overall performance.
  - Avoids problems with interpreting scalability of more conventional mesh refinement studies with AMR.

Replication Scaling of AMR: Cray XT4 Results

- PPM gas dynamics solver:
  - 97% efficient scaled speedup over range of 128-8192 processors (176-181 seconds).
  - Fraction of operator peak: 90% (480 Mflops / processor).
  - Adaptivity Factor: 16.

- AMR-multigrid Poisson solver:
  - 87% efficient scaled speedup over range of 256-8192 processors (8.4-9.5 seconds).
  - Fraction of operator peak: 45% (375 Mflops / processor).

Embedded Boundary Performance Optimization and Scaling

- Aggregate stencil operations, which use pointers to data in memory and integer offsets, improve serial performance by a factor of 100.
- Template design
  - Implement AMRMultigrid once and re-use across multiple operators.
  - Operator-dependent load balancing
  - Space-filling curve algorithm to order boxes (Morton)
  - Minimization of communication
- Relaxing about relaxation
  - gsrpb vs. multi-color
  - Edge and corner trimming of boxes
- And many many more
Communication Avoiding Optimizations

- Distributing patches to processors to maximize locality. Sort the patches by Morton ordering, and divide into equal-sized intervals.
- Overlapping local copying and MPI communications in exchanging ghost-cell data (only has an impact at 4096, 8192).
- Exchanging ghost-cell data less frequently in point relaxation.

Chombo AMR Capabilities

- Single-level, multilevel solvers for cell-centered and node-centered discretizations of elliptic / parabolic systems.
- Explicit methods for hyperbolic conservation laws, with well-defined interface to physics-dependent components.
- Embedded boundary versions of these solvers.
- Extensions to high-order accuracy, mapped grids (under development).
- AMR-PIC for Vlasov-Poisson.
- Applications:
  - Gas dynamics with self gravity. Coupling to AMR-PIC.
  - Incompressible Navier-Stokes Equations.
  - Resistive magnetohydrodynamics.
- Interfaces to HDF5 I/O, hypre, VisIt.
- Extensive suite of documentation. Code and documentation released in public domain. New release of Chombo in Spring 2009 will include embedded boundary capabilities (google "Chombo").