Abstractions for, and Generation of, Semi-Regular Grid Computations

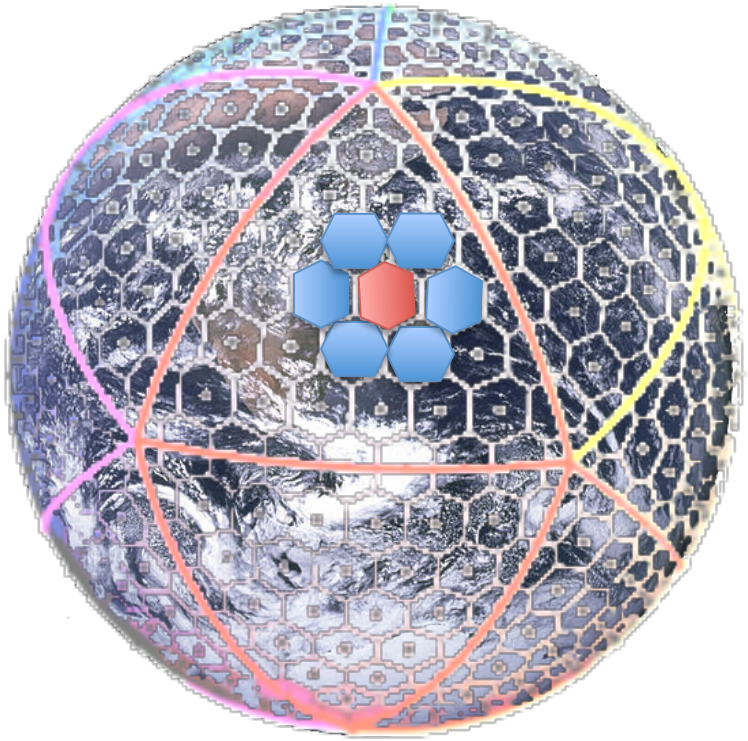
Andy Stone

Aug 15, 2012

Motivation

- Scientists use Earth Simulation programs to:
  - **Predict** change in the weather and climate
  - **Gain insight** into how the climate works
- Components include: ocean, land-surface, sea ice, atmosphere
- Performance is crucial: it impacts **time-to-solution** and **accuracy**

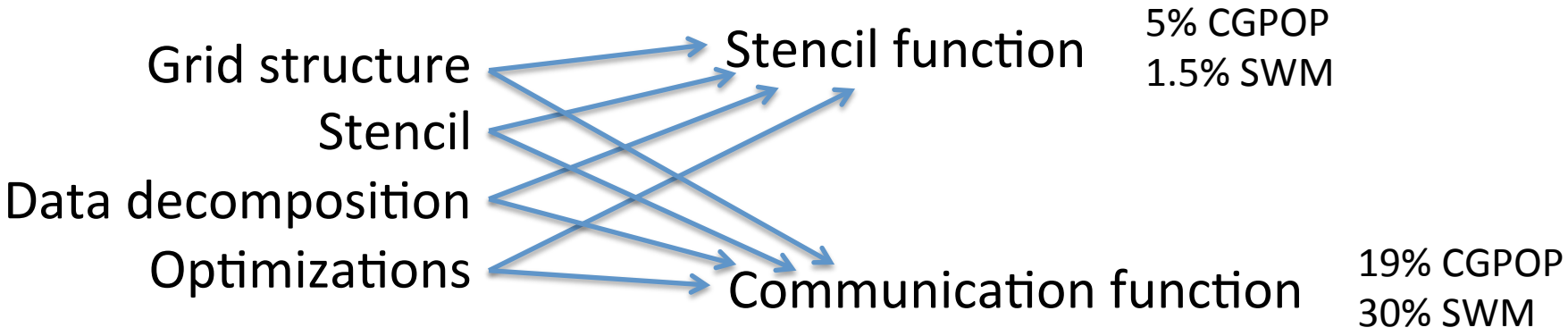
Discretize!



Apply a stencil!

Problem: Implementation details tangle code

```
Loop {
    communicate()
    stencil()
}
```

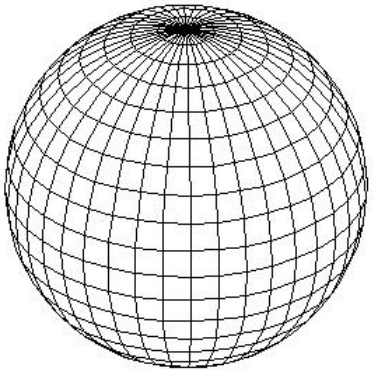


Generic stencil and communication code isn't written due to need for performance.

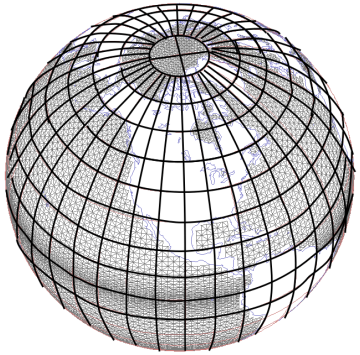
SWM stencil code

```
x2(:,:) = zero
DO j = 2,jm-1
   DO i = 2,im-1
      x2(i,j) = area_inv(i,j)* &
                      ((x1(i+1,j  )-x1(i,j))*laplacian_wghts(1,i,j)+ &
                       (x1(i+1,j+1)-x1(i,j))*laplacian_wghts(2,i,j)+ &

                                                  . . .
    ENDDO
ENDDO

! NORTH POLE
i = 2; j = jm
x2(i,j) = area_inv(i,j)*laplacian_wghts(1,i,j)*((x1(i+1,j)-  . . .

! SOUTH POLE
i = im; j = 2
x2(i,j) = area_inv(i,j)*laplacian_wghts(1,i,j)*((x1(  1, jm)-  . . .
```

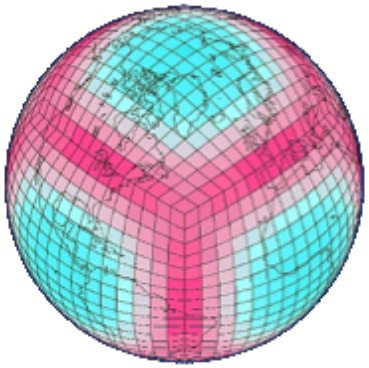
Earth Grids



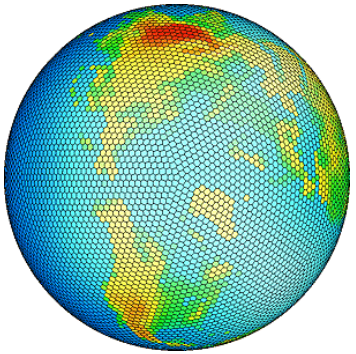
Lat/Lon



Tripole



Cubed Sphere



Geodesic

Proposed solution

- Create set of abstractions to separate details
  - Subgrids, border maps, grids, distributions, communication plans, and data objects
- Provide abstractions in a library: **GridLib**
- Use code generator to remove library overhead and optimize: **GridGen**

Talk outline

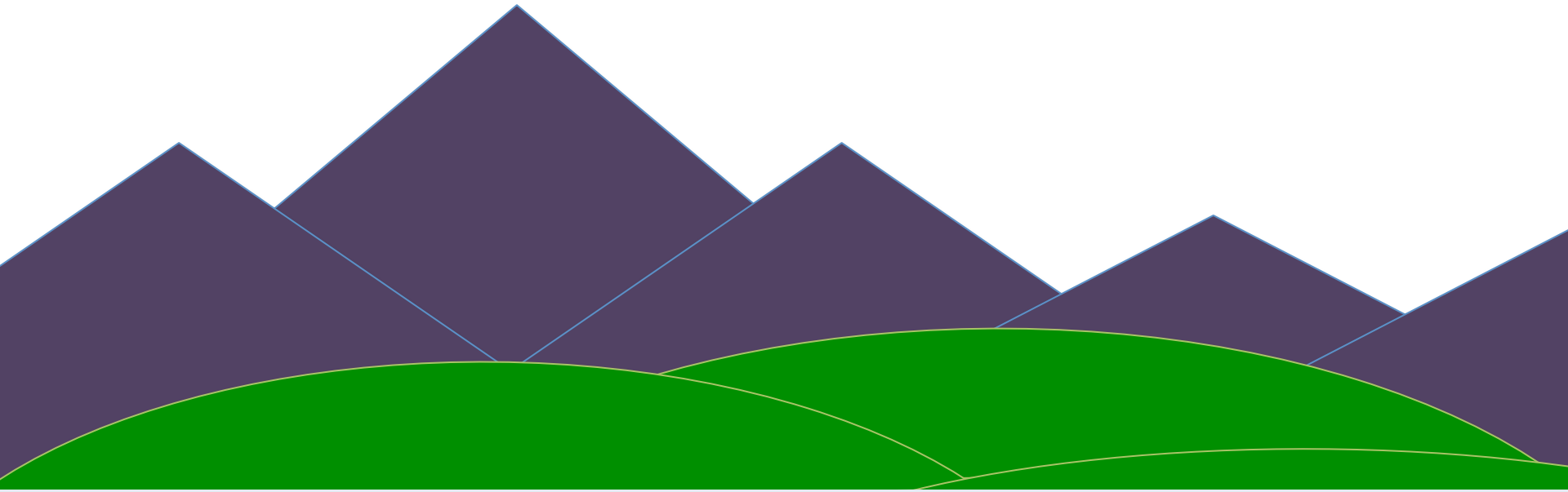
Background

Related work

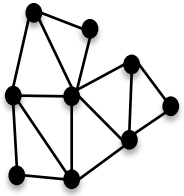
Proposed work

Project organization

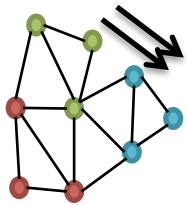
Background



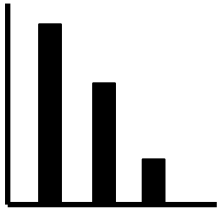
Background:



Grid structure



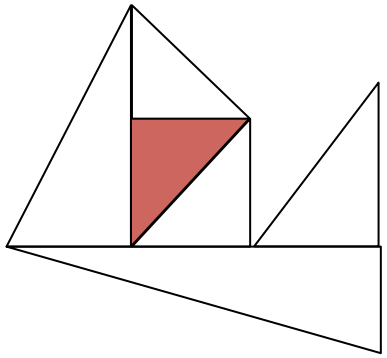
Parallelization and data decomposition

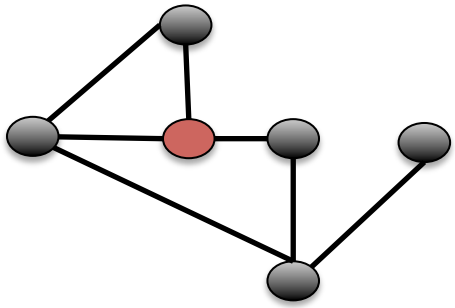


Optimization

Grid abstractions

Irregular





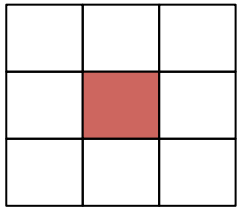
✔ Most general

Structured with a graph

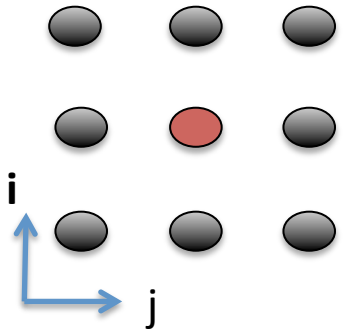
✗ Connectivity is explicitly stored

Regular

✗ Restricted to tessellation of rectangular space with equal sized rectangles



Structured with an array



✔ Storage efficient

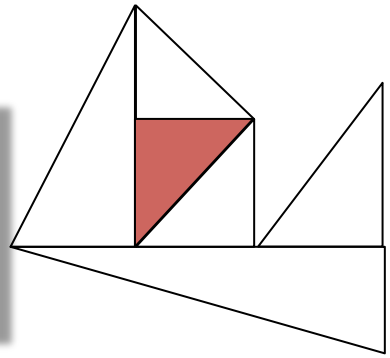
Impact on code

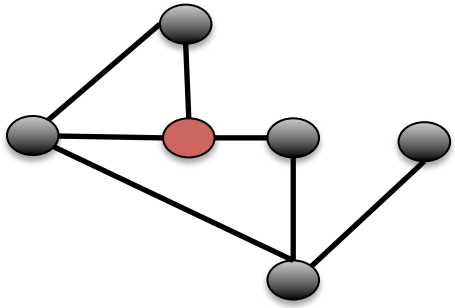
Generic:

```
for each cell c
    c = sum(neighbors(c))
```

Irregular

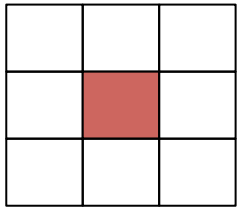
```
for x = 1 to n
  for neigh = 1 to numNeighs(x)
    A(x) += A(neighbor(x, neigh))
```

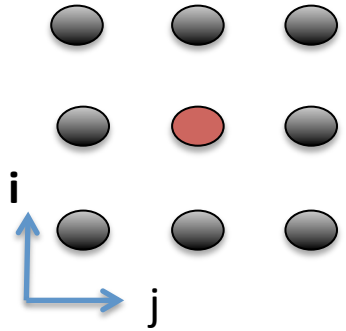




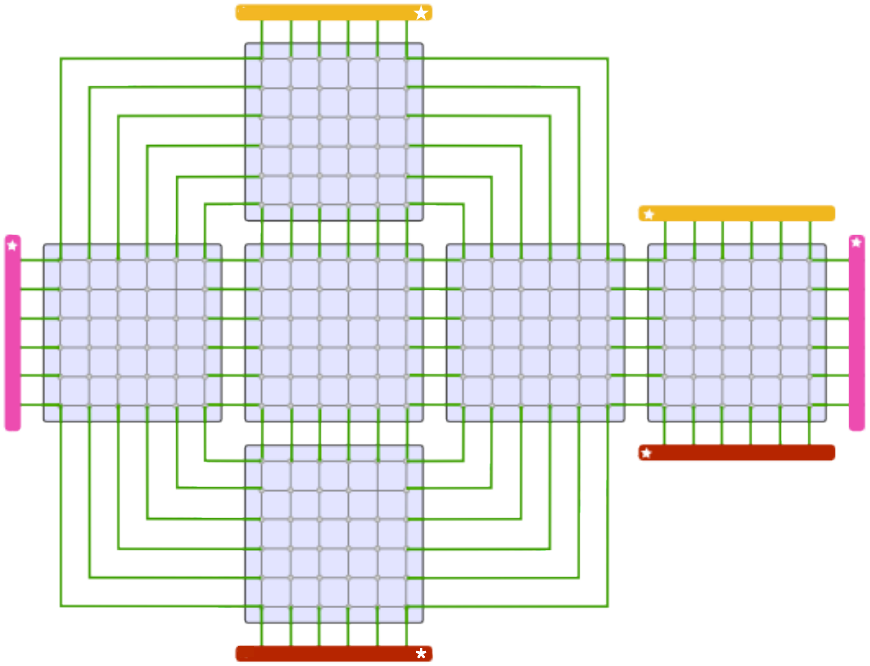
Regular

```
for i = 1 to n
  for j = 1 to m
    A(i,j) = A(i-1, j)   + A(i+1, j) +
             A(i,   j-1) + A(i,   j+1)
```





Many Earth grids are not purely regular



This is the structure of the cubed-sphere grid

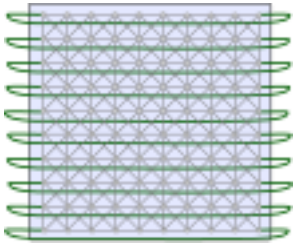
Set of regular grids (arrays)

Connected to one another in an irregular fashion

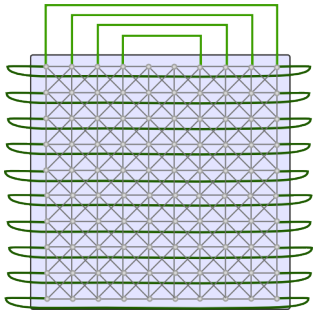
Specialized communication code is needed to handle the irregular structure.

Semi-regular grid examples

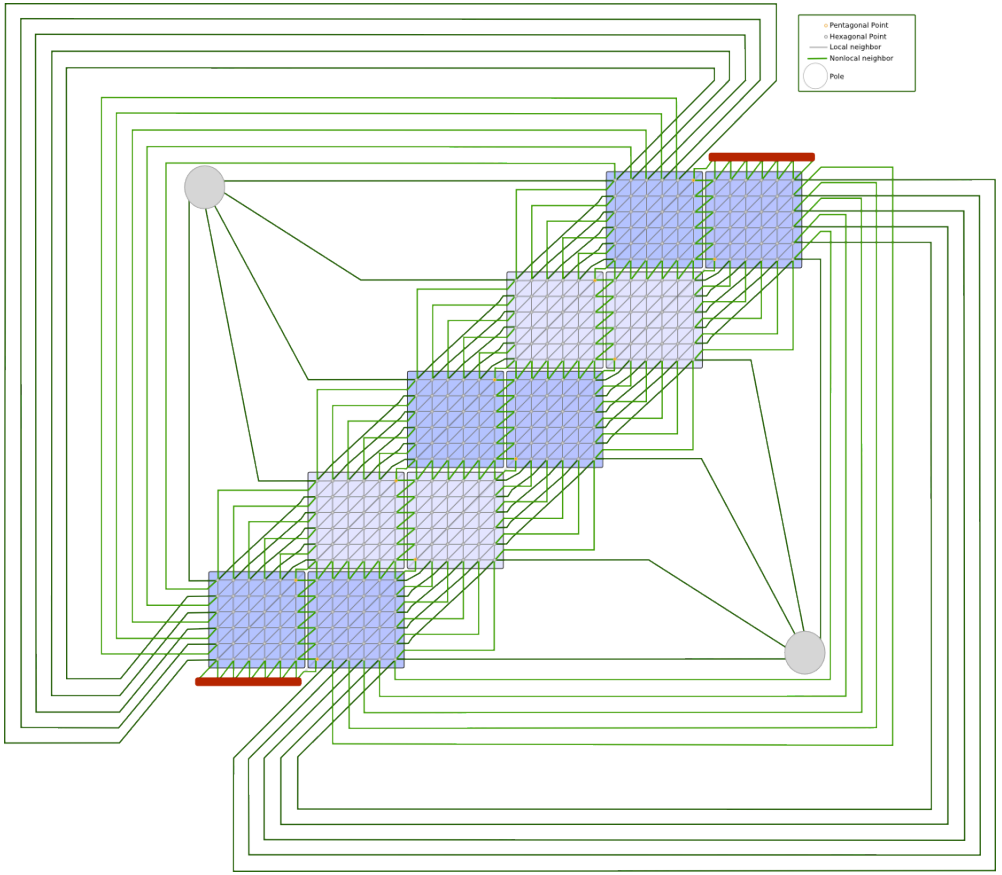
Dipole



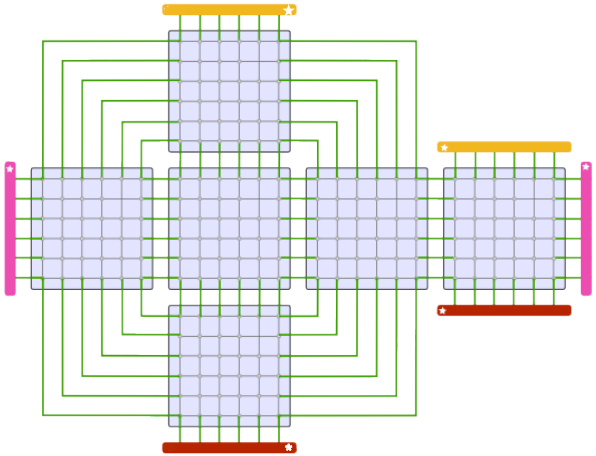
Tripole



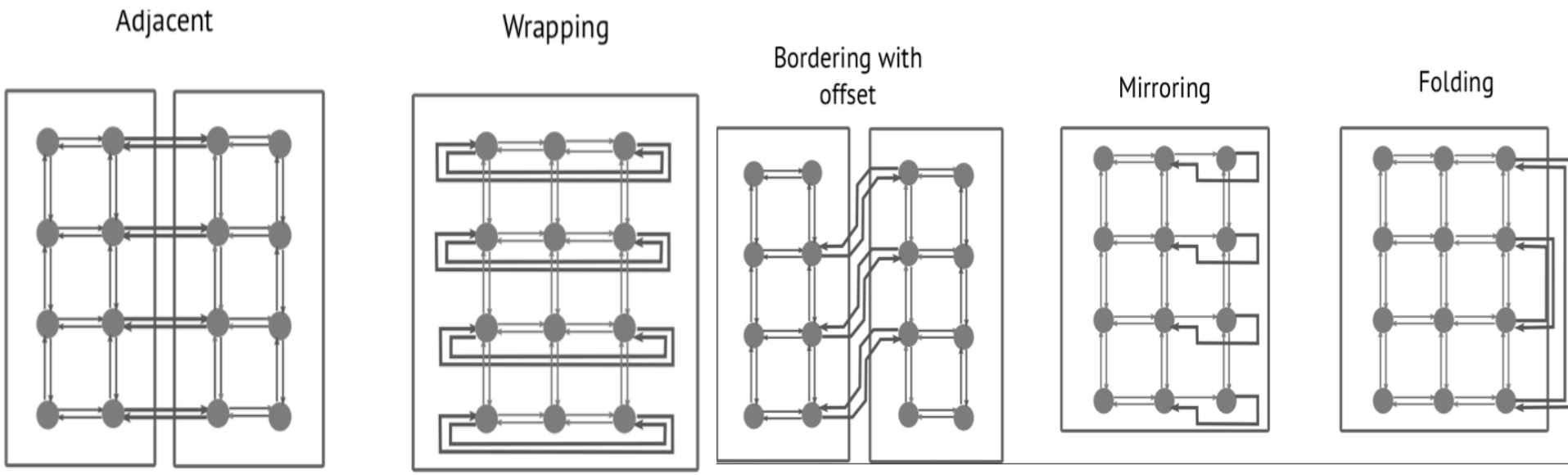
Icosahedral

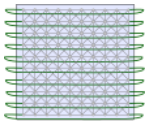


Cubed Sphere

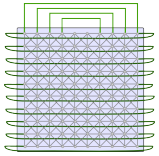


Connectivity between subgrids

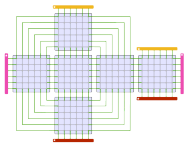




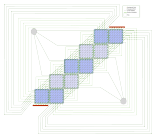
Dipole: Wrapping



Tripole: Wrapping, folding



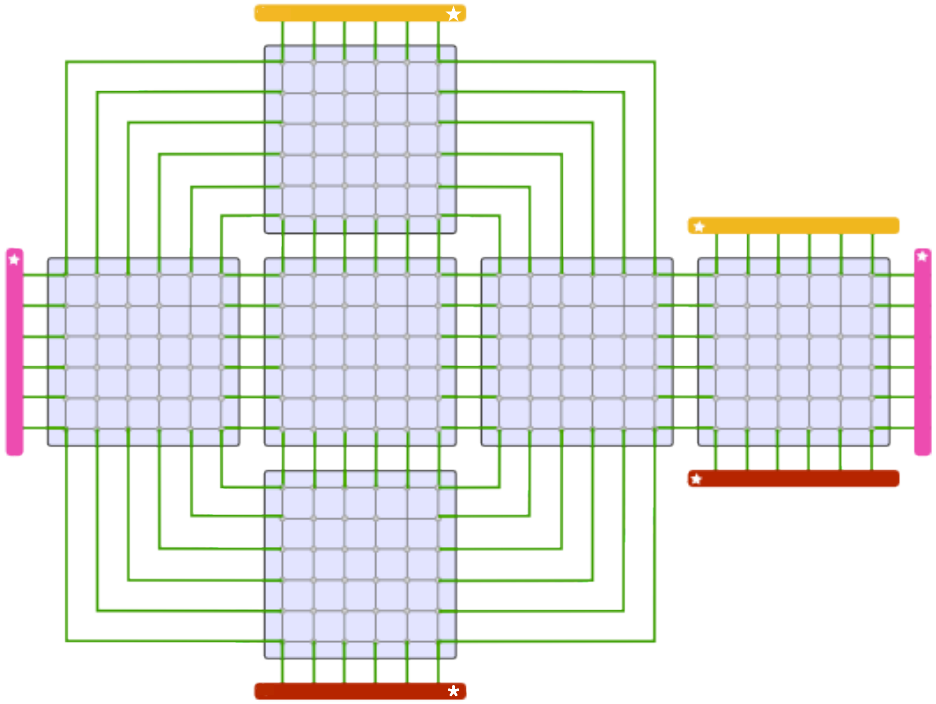
Cubed Sphere: Adjacent



Icosahedral: Adjacent, offset

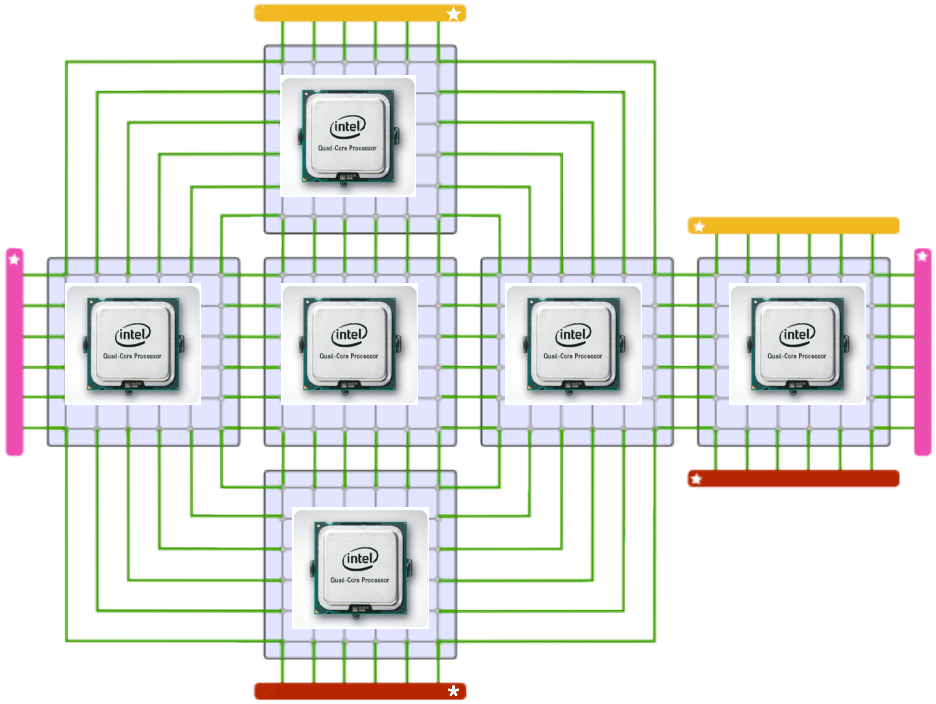
Domain decomposition for parallelization

How to map data and computation onto processors?



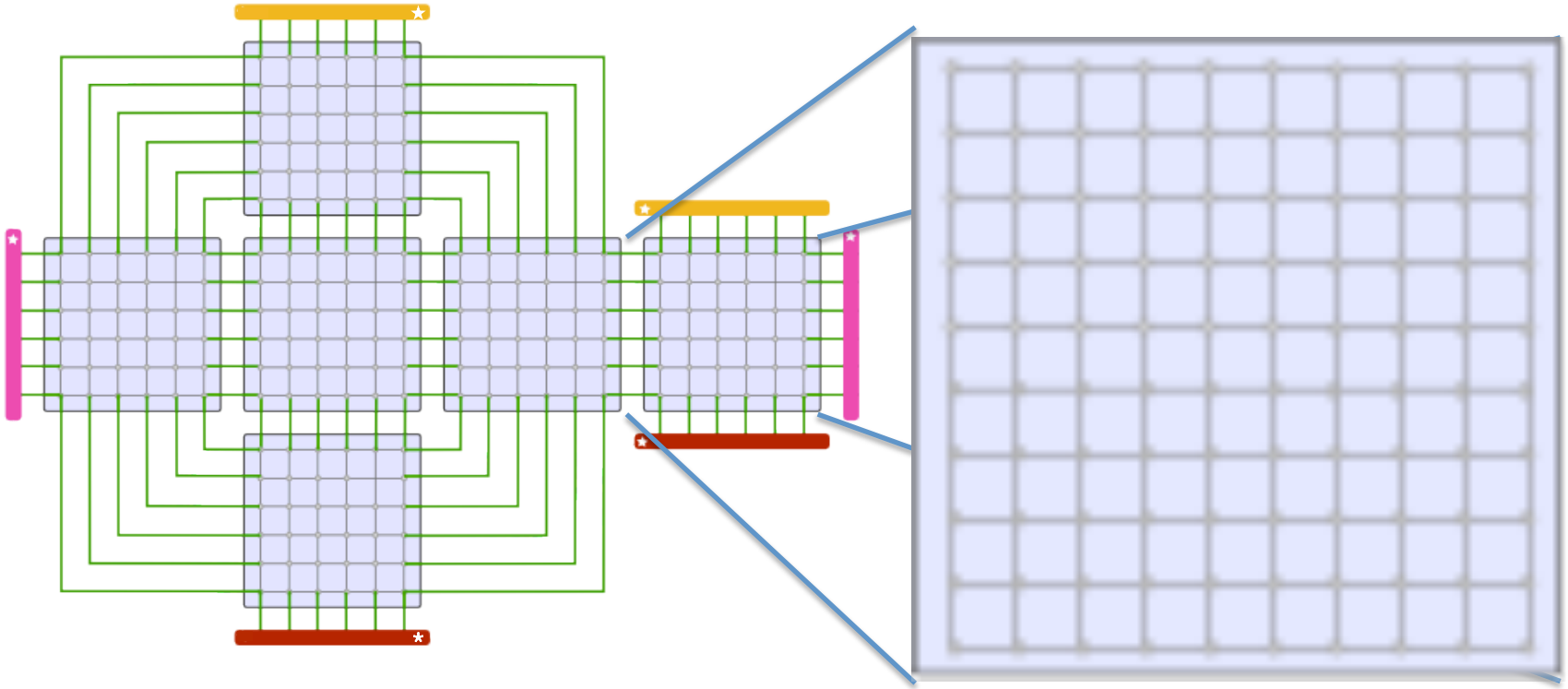
Domain decomposition for parallelization

How to map data and computation onto processors?



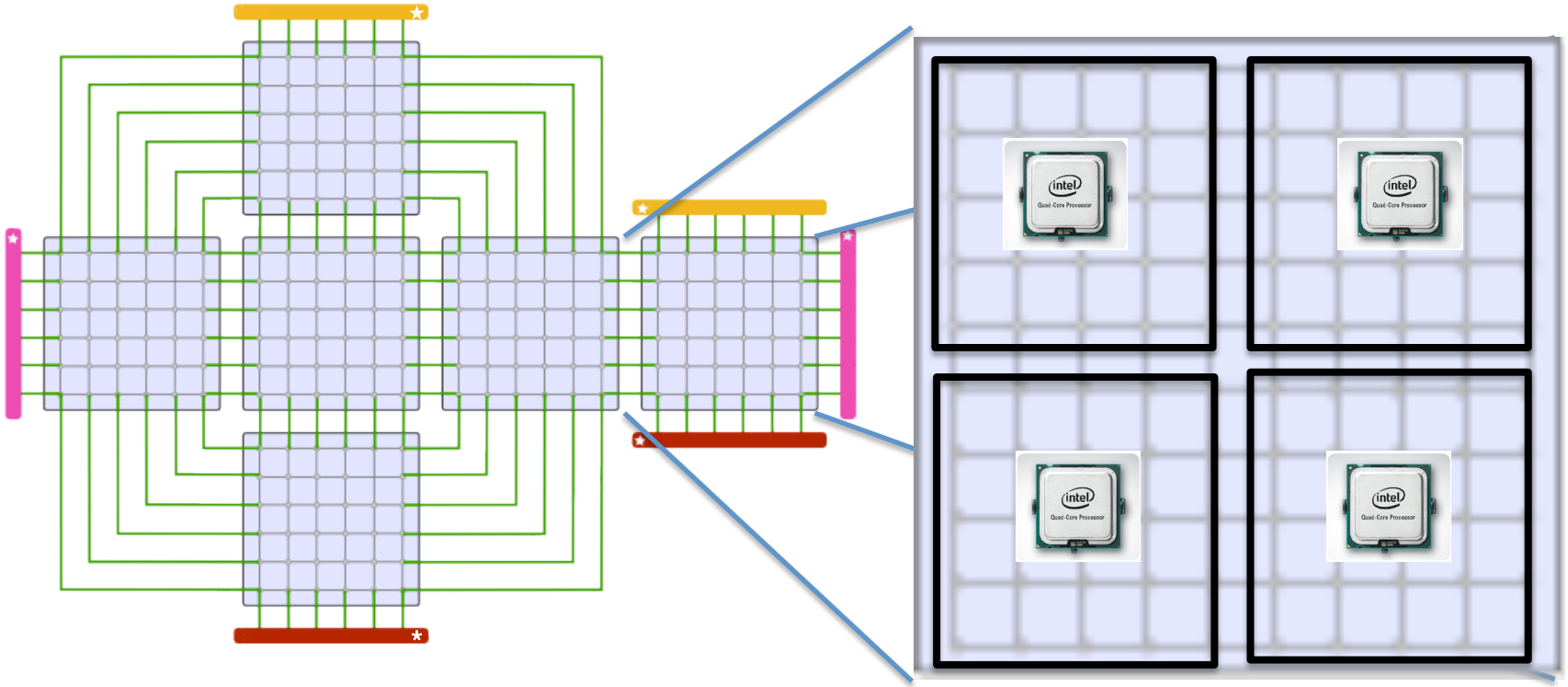
Domain decomposition for parallelization

How to map data and computation onto processors?



Domain decomposition for parallelization

How to map data and computation onto processors?



Common optimizations

Better locality:

Tiling:	Group computation into blocks for locality
Array of structs:	Pack multiple properties of grid together
Array padding:	Start and end arrays on cache lines

Use of special hardware:

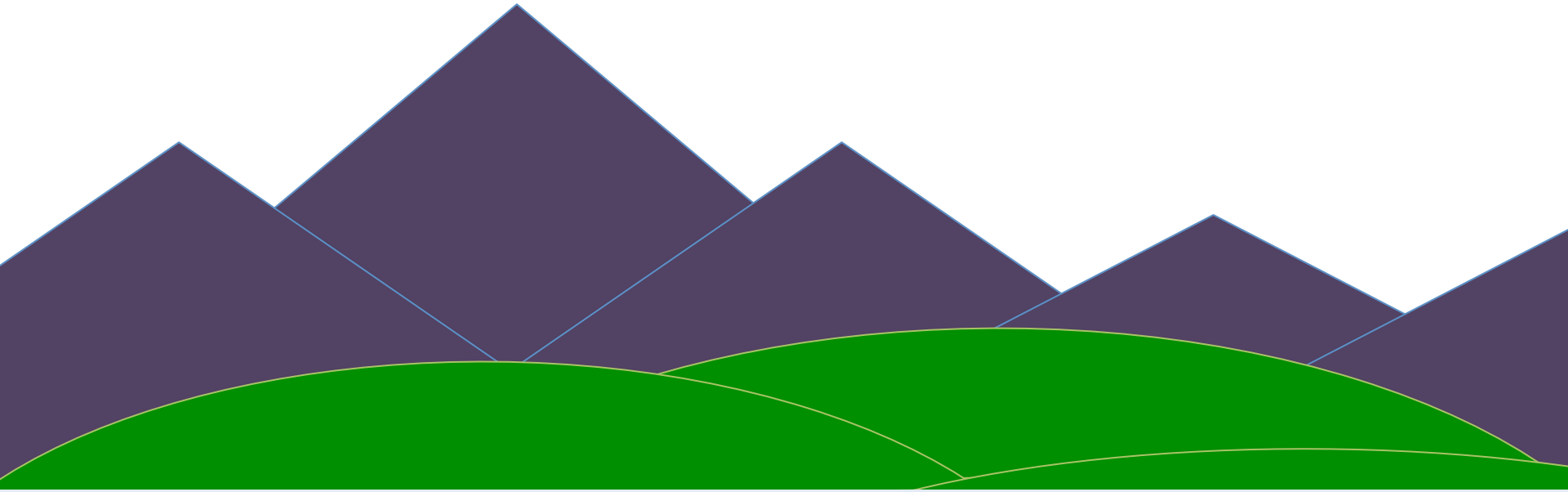
SIMDization: Use special hardware (ex: SSE instructions) to perform the same computation on multiple pieces of data simultaneously

Hiding or Reducing Communication:

Overlap computation and communication

Perform redundant computation to avoid communication

Related work



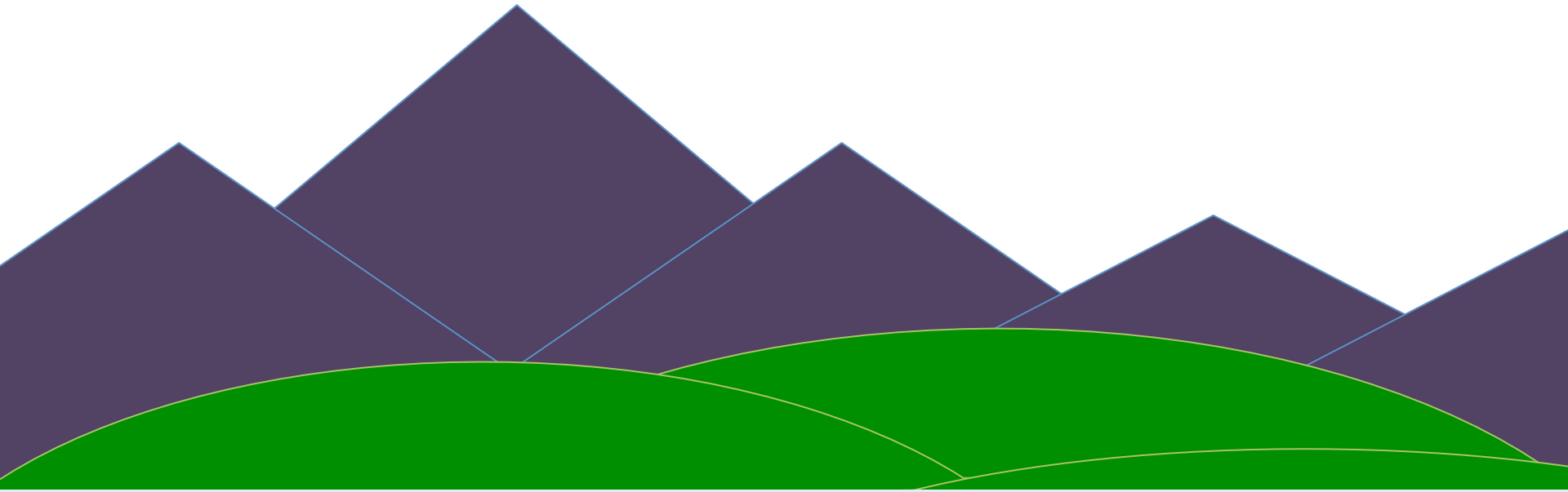
Related work: Grid structure

	Arrays	Graphs	Wrapping	Mirroring	Adjacent	Offset	Folding	Arbitrary
ZPL's regions ('98)	✔		✔	✔				
Chombo ('09)	✔		✔					
Kamil et. al ('09)	✔		✔					
Mint ('11)	✔		✔					
Patus ('11)	✔		✔					
Physis ('11)	✔		✔					
Pocohir ('11)	✔		✔					
OP2 ('10)		✔	✔	✔	✔	✔	✔	✔
Liszt ('11)		✔	✔	✔	✔	✔	✔	✔
Proposed Work	✔		✔	✔	✔	✔	✔	✔

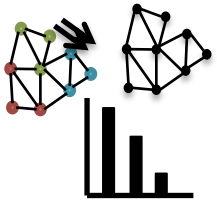
Related work: Optimizations

	Arrays	Graphs	SIMDization	Tiling	Time skewing	Comm. Hiding	Array-of-structs	Redundant Comp.	Autotunes
ZPL ('98)	✔								
Chombo ('09)	✔								
Kamil et al. ('09)	✔			✔					✔
Mint ('11)	✔			✔					
Patus ('11)	✔		✔						✔
Physis ('11)	✔					✔			
Pocohir ('11)	✔				✔				
OP2 ('10)		✔					✔		
Liszt ('11)		✔							
Proposed Work	✔					✔	✔	✔	

Proposed work



Proposed project



Goal:

To enable a **separation of concerns** for semi-regular grid computations between **stencil** algorithm and: (1) **grid structure**, (2) **parallelization** and **decomposition** details, and (3) **optimization** without sacrificing performance.

Miniapps:

CGPOP (ocean simulation)
SWM (shallow water model)

Grids:

Dipole, Tripole, Cubed Sphere, Icosahedral

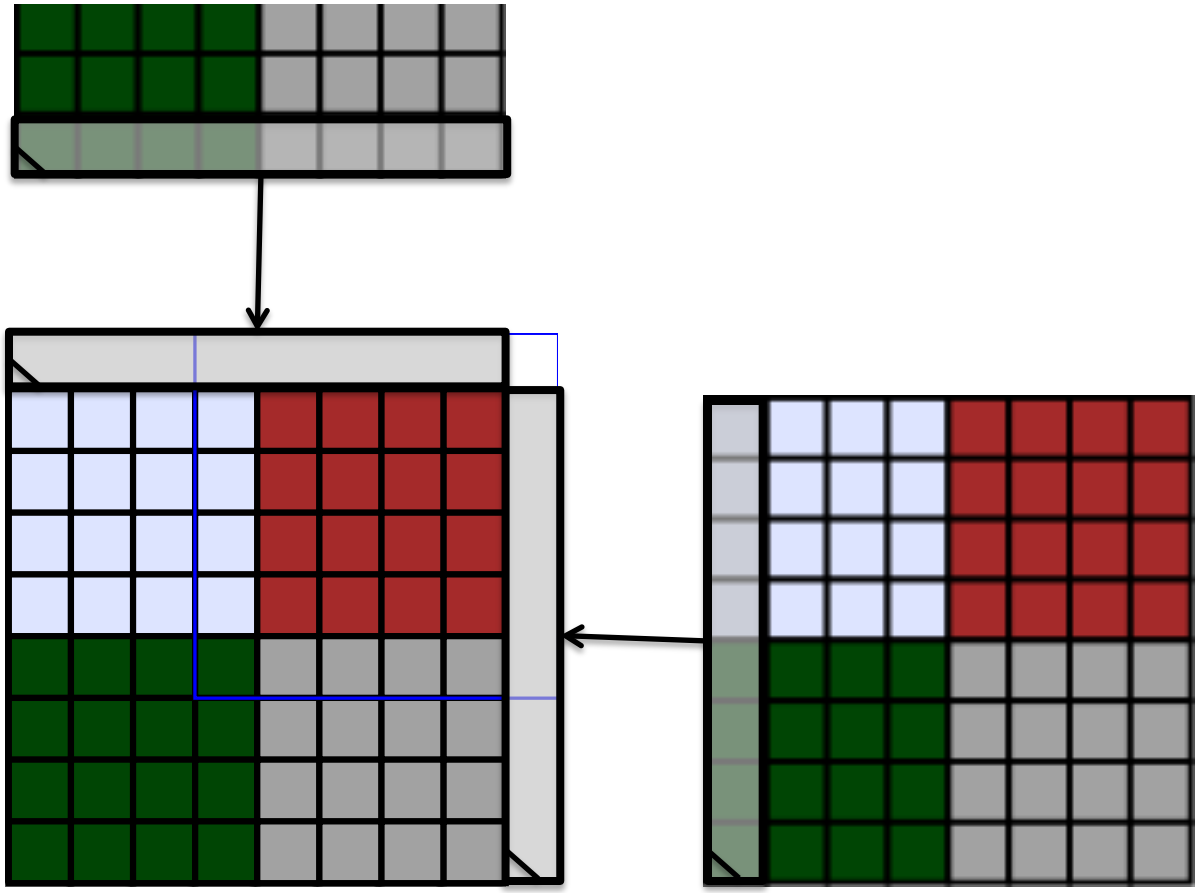
Machines:

Department's Bacon & Eggs (16 cores)
ISTEC Cray (1248 cores)
NCAR YellowStone (72,288 cores)

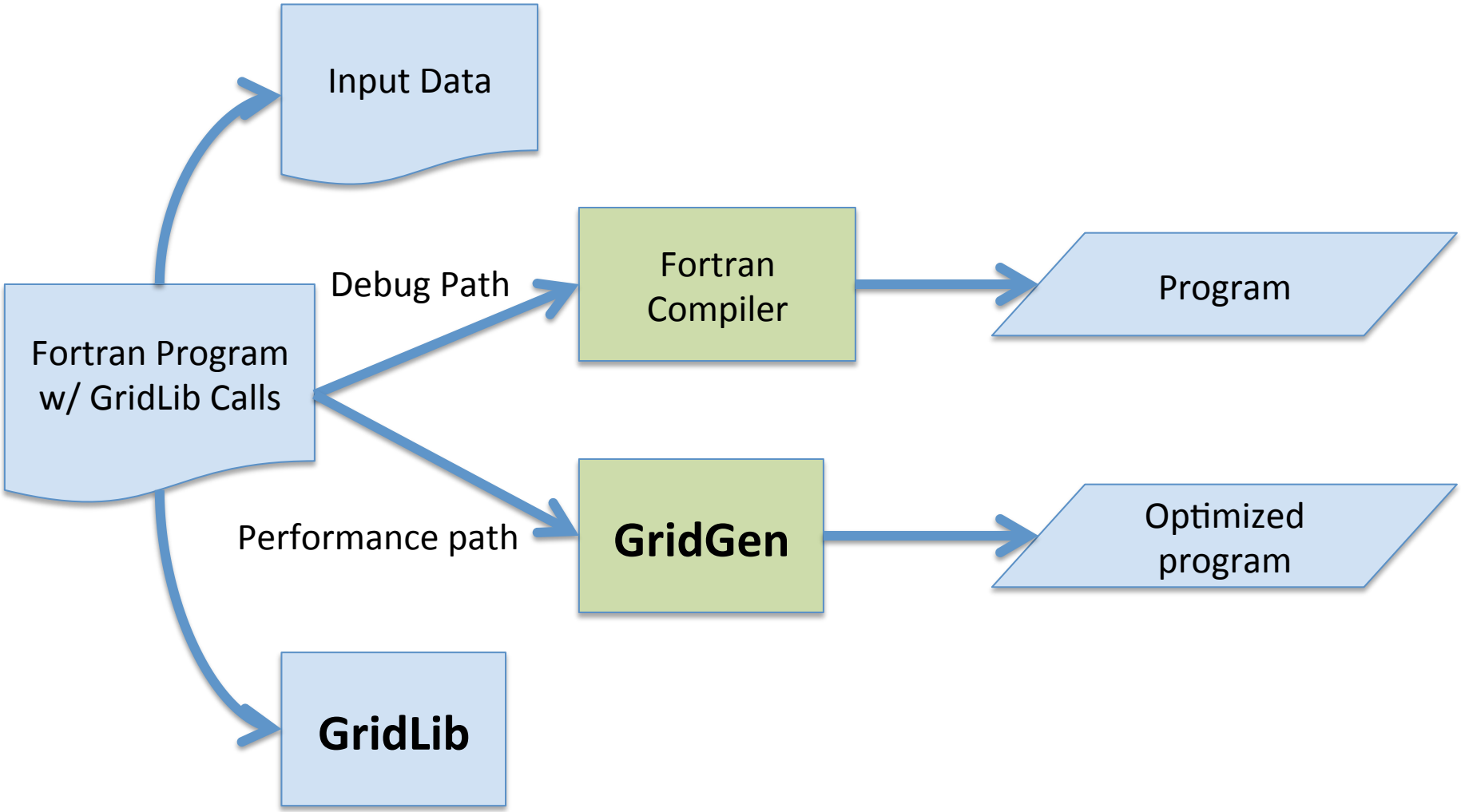
Example Code

```
! Define grid
call sugrid_new(sg1, N, M)
call sugrid_new(sg2, N, M)
call grid_new(g)
call grid_placeAdjacentWE(g, sg1, sg2)


! Define distribution and read in data
call distribution_new_blocked(dist, g, 100, 100)
call data_new_from_file(data_in, “input.dat”, g, dist)


! Apply stencil
data_out = data_apply(data_in, fourPtAvg)

real function fourPtAvg(A, i, j)
    fourPtAv = &
   0.2 * (A(i,j) + A(i-1,j) +  &
         A(i+1, j) + A(i, j-1) +  &
         A(i, j+1))
end function
```

Communication plan



Proposed project

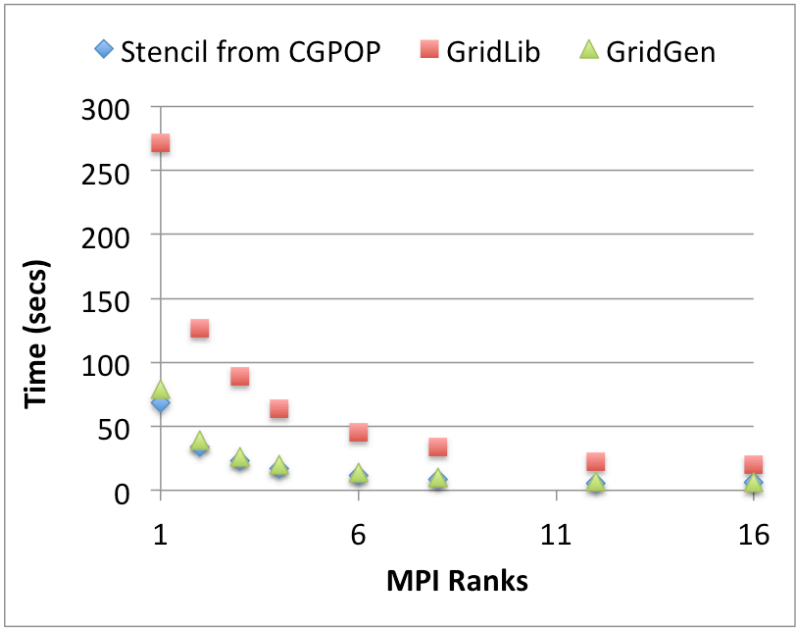


Preliminary results

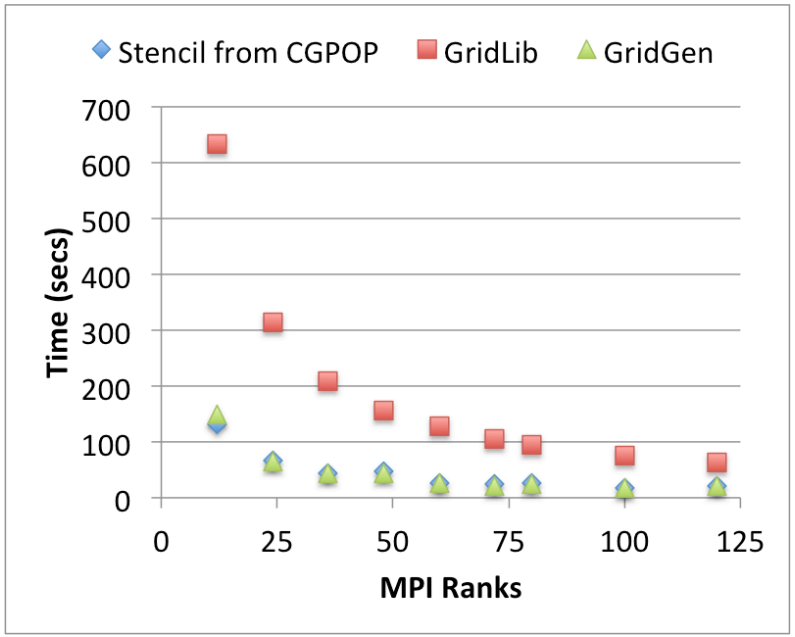
Implementation of stencil from CGPOP on a dipole grid

Goal: To Have GridGen match performance of handwritten code

Shared Memory Machine



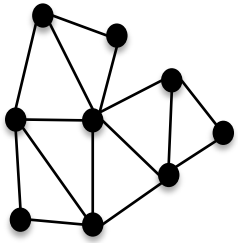
Distributed Memory Machine



Bacon, a 16 core, 2.13 GHz, XeonE7 machine
100 iterations, 3600 x 3600 grid

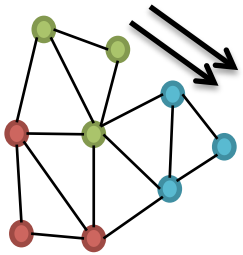
Cray XT6m with 1248 cores across 52 nodes,
24 cores per node
1000 Iterations, 10800 x 10800

Proposed solution:



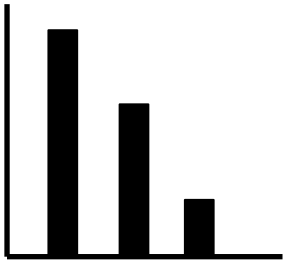
Grid Structure

SubGrids
Border Mappings
Grids
Convenience Functions



Parallelization and data decomposition

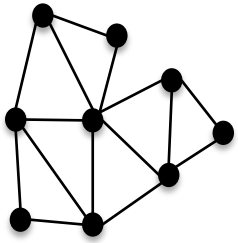
Distribution objects
Convenience Functions
Targets clusters using MPI



Optimization

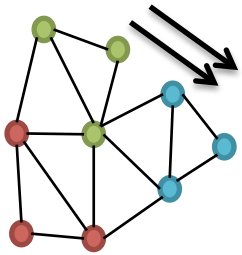
Users specify what optimizations to apply
Optimizing Generators

What we have done:



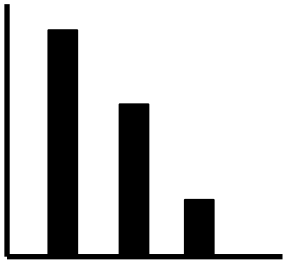
✔Grid Structure

SubGrids
Border Mappings
Grids
Convenience Functions



✔Parallelization and data decomposition

Distribution objects
Convenience Functions
Targets clusters using MPI

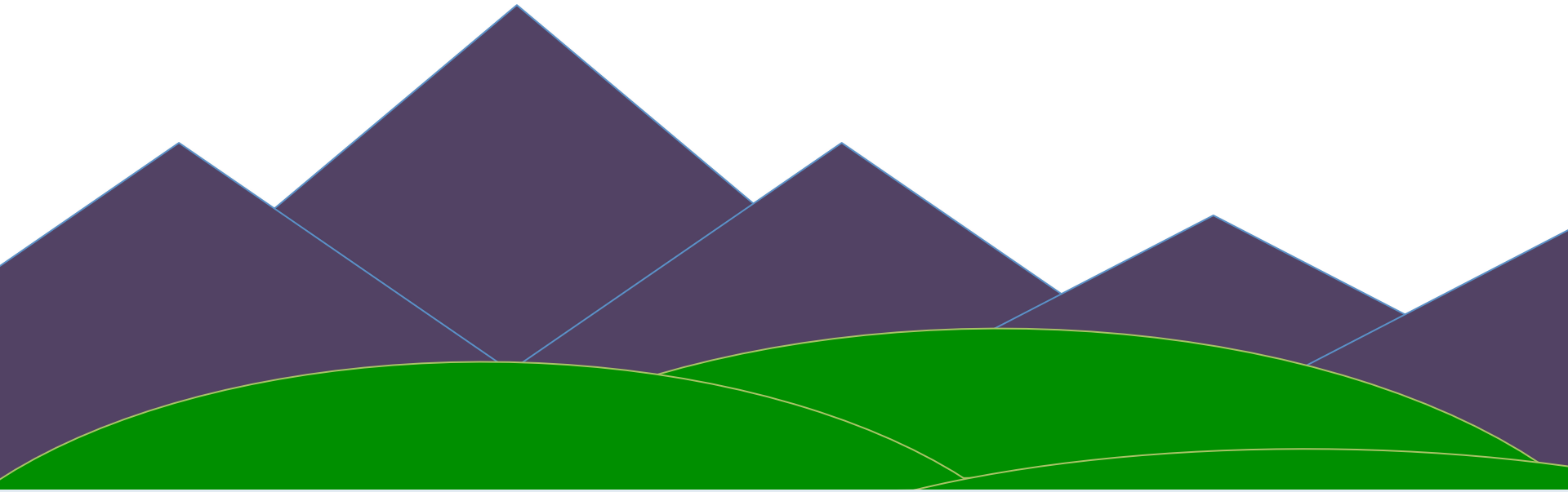


Optimization

Users specify what optimizations to apply
Optimizing Generators

Limitation: Compact stencils only

Project organization



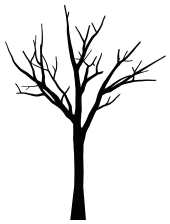
Project organization

Three Papers

Summer (now)

How do we prevent tangling semi-structured grid topology with stencil algorithm?

Submit to: IPDPS (October)



Fall

How do we generalize our previous approach for non-compact stencils?

Submit to: PLDI (November)

Spring

What Optimizations can we automatically apply to semi-regular grid code, and how do we apply these optimizations?

Submit to: SC (April)

Summer 2013

Compile this work into a dissertation

Paper one

Problem: How do we prevent tangling semi-structured grid topology with a stencil algorithm?

Solution: Create semi-regular grid abstractions. Integrate into a library.

Sub problems:

How do we perform communication for any semi-regular grid?

How can we ensure we match hand written performance?

Solution:

- Have a generic communication algorithm to populate halos and generate a communication plan to direct it for a specific grid.
- Aggregate messages
- Use a code generation tool to inline library calls.

Experimentation Goal:

Enable **separation without sacrificing performance** for small, example stencils extracted from CGPOP and SWM on dipole, tripole, and cube-sphere grids when run on a cluster.

Paper two

Problem: How do we generalize our previous approach for non-compact stencils? Does my approach work for real-world applications?

Solution:

New border maps abstraction
Time dimension in data objects
Use tools with SWM and CGPOP

Challenges:

Getting SWM and CGPOP to pass through frontend
Maintaining SWM and CGPOP performance
Developing border map abstraction

Experimentation Goal:
Maintain separation without sacrificing performance when applied to CGPOP and SWM. Replace stencils in CGPOP and SWM with stencils that have greater depth and/or refer to more than one timestep back.

Paper three

Problem: How do we improve the performance of CGPOP and SWM without tangling code or requiring a large rewrite?

Solution: Have a tool automatically apply optimizations when given a grid and algorithm specification.

Challenges: Determining what optimizations would be the most useful to apply

Experimentation Goal:
Improve the performance of CGPOP and SWM without changing the SWM or CGPOP code.

Summary

Goal: To enable a **separation of concerns** for semi-regular grid computations between **stencil** algorithm and: (1) **grid structure**, (2) **parallelization** and **decomposition** details, and (3) **optimization** without sacrificing performance.

Semi-regular grid computations appear in Earth simulation applications.

Existing stencil generators do not focus on semi-regular grids.

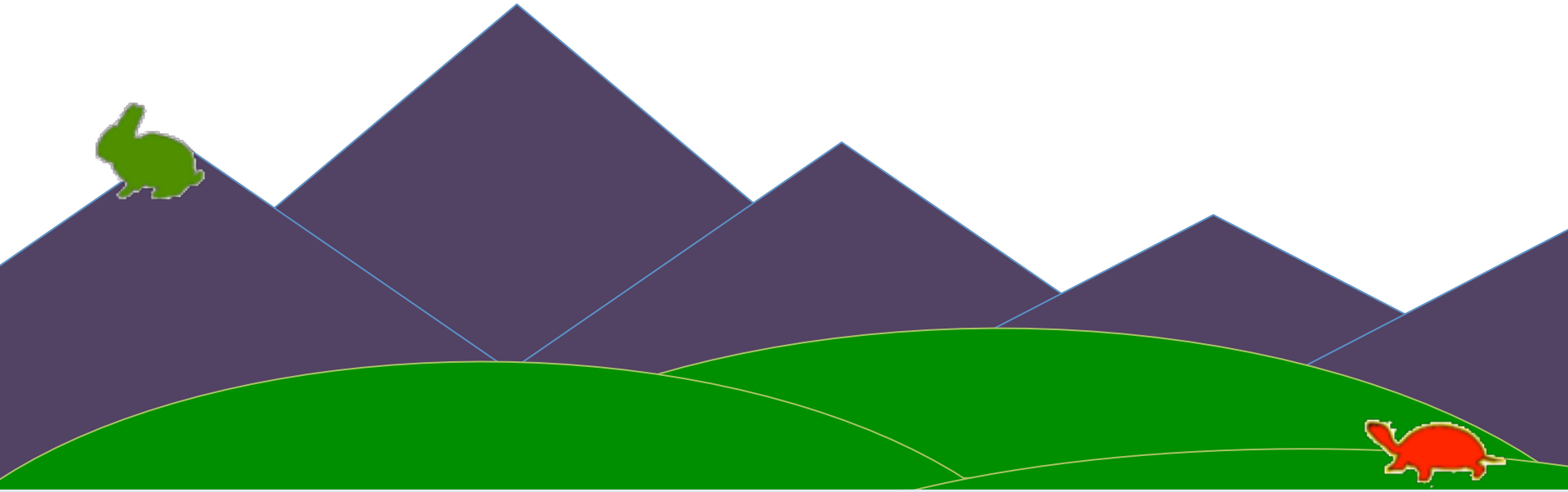
Contributions:

- **Introduction of abstractions**
- **Design of communication plan algorithm**
- **Development GridLib library and GridGen tool**
- **Evaluation with CGPOP and SWM**
- **Adapting optimizations for semi-regular grids and automatically applying them**

Plan:

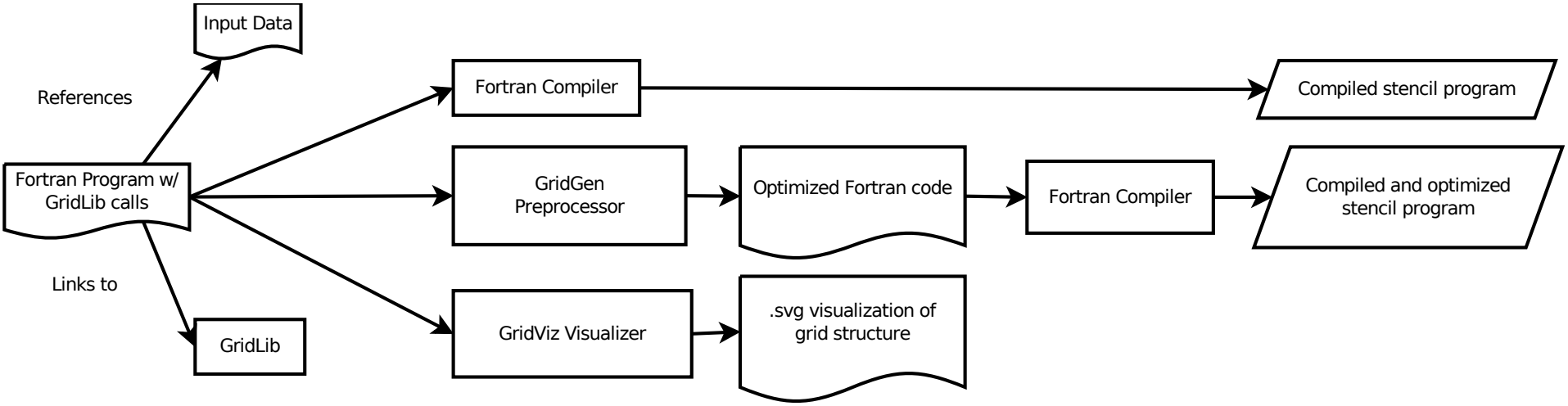
- 3 papers
- Finish Summer 2013

Questions?



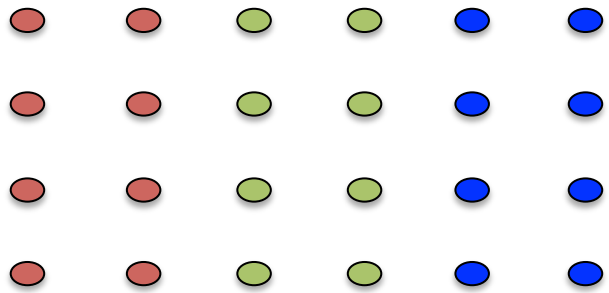
Paper One: Tools

2012

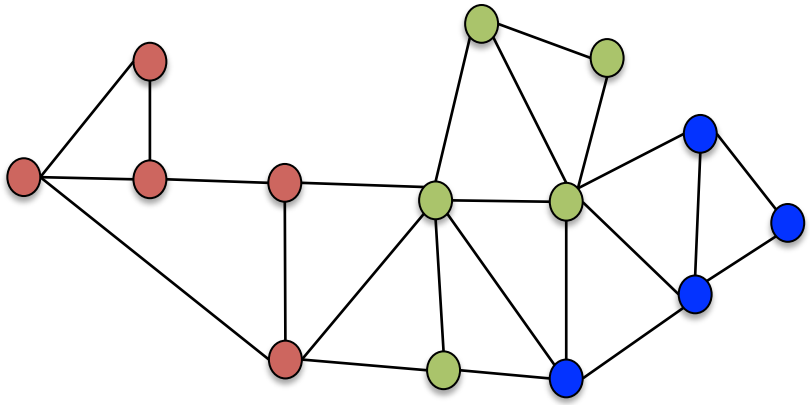


Parallelization Complicates Code

```
// distribute
is = 1;         ie = n
js = pid * 2;   je = js+1

communicate();
for i = is to ie
  for j = js to je
    A(i,j) = A(i-1, j)   + A(i+1, j) +
             A(i,   j-1) + A(i,   j+1)
```



```
distribute()

communicate()
for x = ns to ne
  for neigh = 1 to numNeighs(x)
    A(x) += A(neighor(x, neigh))
```



Resolution to Tangling Details:

For my dissertation:

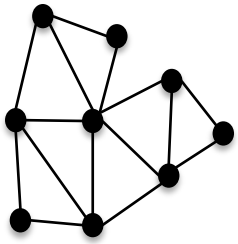
Create a set of abstractions for:

- semi-regular grids
- data decompositions
- algorithms on grids

I will incorporate these abstractions into a library called **GridLib**, and code generation tool called **GridGen**

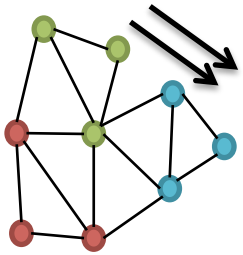
I will target clusters using **MPI**.
And apply optimizations when specified by the user

Methods of Detangeling:



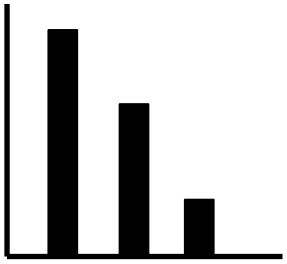
Grid Structure

Abstractions to represent grid: Node, Edge, Cell, Field



Parallelization and data decomposition

Decomposition functions
Generators for: MPI, OpenMP, CUDA



Optimization

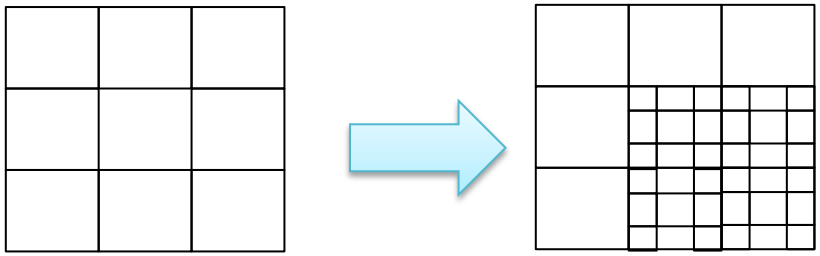
Automatically Apply
Apply via tool
Autotune

Stencil Generation Tools

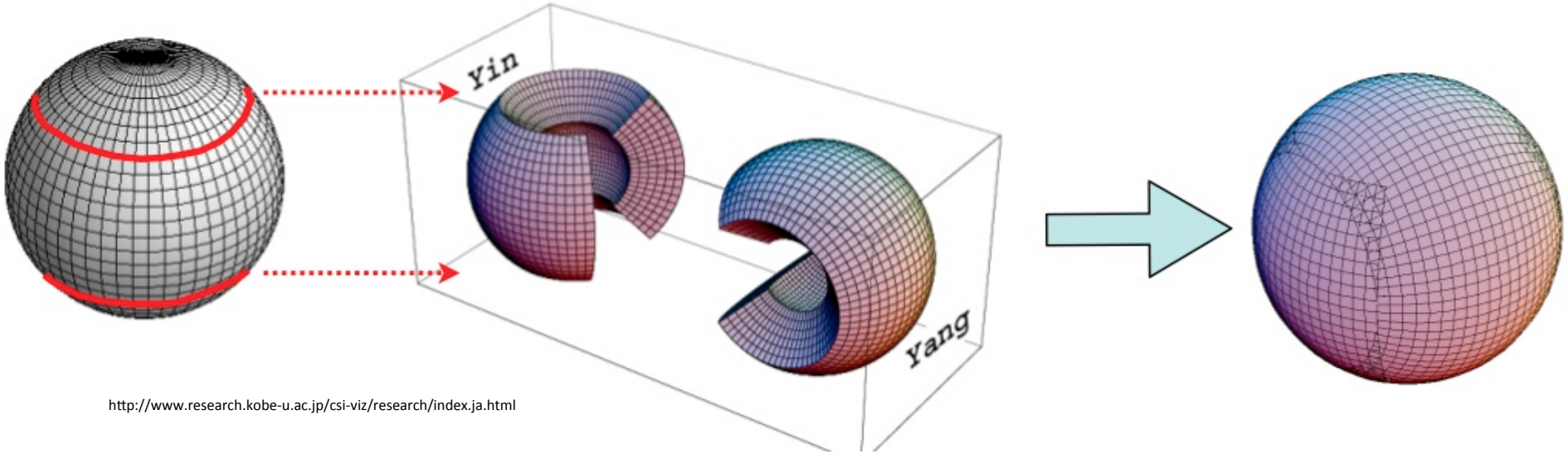
	Regular	Irregular	Output Language	Multicore	Clutser	GPU
Kamil et al.	✔		C, Fortran, pthreads	✔		
Mint	✔		C, CUDA			✔
Patus	✔		C, Openmp, CUDA	✔		✔
Physis	✔		C , CUDA		✔	✔
Pocohir	✔		C++, Cilk	✔		
Chombo	✔		C++, MPI		✔	
Liszt		✔	C++, MPI, CUDA		✔	✔
OP2		✔	C++, MPI OpenMP, CUDA		✔	✔
ZPL	✔		C, PGAS Library		✔	
Proposed Work	✔		MPI		✔	

Limitations of planned research

Adaptive



Yin Yang



http://www.research.kobe-u.ac.jp/csi-viz/research/index.ja.html

Future Work

Other types of grids:

- Higher-dimensionality
- Adaptively Refined
- Yin Yang
- Unstructured

Additional Evaluation:

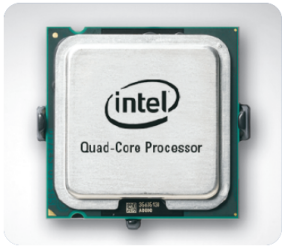
- Applied in larger apps
- Additional miniapps

Other Frontends and backends:

- C/C++, HPCC languages (Chapel, X10)
- GPUs

Additional Optimizations

Parallelization Targets



MultiCore Machines:

Multiple processors with shared memory

OpenMP, Cilk, pThreads

GPUs

Massive SIMD processing

OpenCL, CUDA

Clusters

Networked collection of machines

MPI