A Parallel Scattered Node Finite Difference Scheme for the Shallow Water Equations on a Sphere

Martin Tillenius Elisabeth Larsson
Uppsala University, Sweden

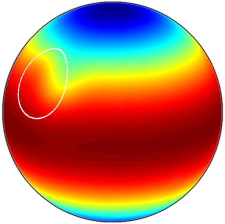
Erik Lehto Natasha Flyer
National Center for Atmospheric Research, CO, USA

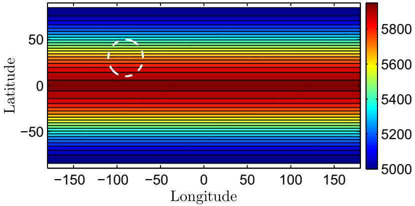
SIAM PP14

Application: Global Climate Simulations

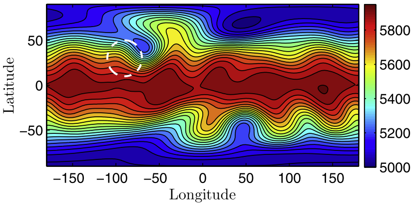
Find **geopotential height**: Height above sea level where pressure is 500 mb.

Standard Test Case: Zonal flow over an isolated mountain (cone)





Day 0



Day 15

Discrete Shallow Water Simulations on a Sphere

We use the method and MATLAB code developed in [1]

Governing Equations

$$RHS = \begin{bmatrix} u \circ D_x u + v \circ D_y u + w \circ D_z u \\ u \circ D_x v + v \circ D_y v + w \circ D_z v \\ u \circ D_x w + v \circ D_y w + w \circ D_z w \end{bmatrix} + f \begin{bmatrix} y \circ w - z \circ v \\ z \circ u - x \circ w \\ x \circ v - y \circ u \end{bmatrix} + g \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} h$$

$$\partial u / \partial t = -p_x \cdot \text{RHS}$$

$$\partial v / \partial t = -p_y \cdot \text{RHS}$$

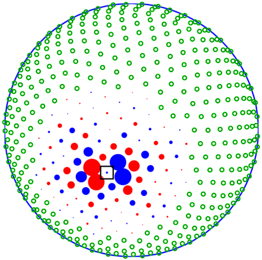
$$\partial w / \partial t = -p_z \cdot \text{RHS}$$

$$\partial h / \partial t = u \circ D_x h + v \circ D_y h + w \circ D_z h \quad + \quad h \circ (D_x u + D_y v + D_z w)$$

(u, v, w): velocity
f: Coriolis force
g: Gravity
h: Geopotential height
p: Projection onto sphere
D: Projected diff. matrix

Method

- Approximate operator D applied to u at x_c using the n nearest nodes: $Du(x_c) \approx \sum_{k=1}^{n} w_k u(x_k)$
- Apply hyper-viscosity (Δ^4) to stabilize
- Use classic 4th-order Runge-Kutta for time stepping



Example stencil

[1] N. Flyer, E. Lehto, S. Blaise, G.B. Wright, A. St-Cyr, **A guide to RBF-generated finite differences for nonlinear transport: Shallow water simulations on a sphere**, J. Comput. Phys. 231 (2012) 4078-4095.

Implementation

Original MATLAB Implementation

MATLAB Code

```matlab
% Build differentiation matrices and hyperviscosity operator
[Dx, Dy, Dz, L] = rbf_matrix_fd(nodes);

% Runge-Kutta time stepping. H = [u v w h]
for i=1:timesteps
  F1 = dt*f( H );
  F2 = dt*f( H + 0.5*F1 );
  F3 = dt*f( H + 0.5*F2 );
  F4 = dt*f( H + F3 );
  H = H + 1.0/6.0*(F1 + 2.0*F2 + 2.0*F3 + F4);
end
```

```matlab
% Evaluate time derivatives
function dH = f(H)

Tx = Dx*H;       % Apply differentiation matrices
Ty = Dy*H;
Tz = Dz*H;

RHS = ...        % RHS = [...] + f [...] + g [...]
dH(:,1) = ...    % ∂u/∂t = -p_x · RHS
dH(:,2) = ...    % ∂v/∂t = -p_y · RHS
dH(:,3) = ...    % ∂w/∂t = -p_z · RHS
dH(:,4) = ...    % ∂h/∂t = u∘Tx_h + v∘Ty_h + w∘Tz_h + h∘(Tx_u + Ty_v + Tz_w)

dH = dH + L*H;   % Apply hyper-viscosity
```

$$RHS = \begin{bmatrix} u \circ \mathrm{Tx}_u + v \circ \mathrm{Ty}_u + w \circ \mathrm{Tz}_u \\ u \circ \mathrm{Tx}_v + v \circ \mathrm{Ty}_v + w \circ \mathrm{Tz}_v \\ u \circ \mathrm{Tx}_w + v \circ \mathrm{Ty}_w + w \circ \mathrm{Tz}_w \end{bmatrix} + f \begin{bmatrix} y \circ w - z \circ v \\ z \circ u - x \circ w \\ x \circ v - y \circ u \end{bmatrix} + g \begin{bmatrix} \mathrm{Tx}_h \\ \mathrm{Ty}_h \\ \mathrm{Tz}_h \end{bmatrix}$$

$\partial u/\partial t = -p_x \cdot RHS$

$\partial v/\partial t = -p_y \cdot RHS$

$\partial w/\partial t = -p_z \cdot RHS$

$\partial h/\partial t = u \circ \mathrm{Tx}_h + v \circ \mathrm{Ty}_h + w \circ \mathrm{Tz}_h + h \circ (\mathrm{Tx}_u + \mathrm{Ty}_v + \mathrm{Tz}_w)$

MATLAB Code

```
% Build differentiation matrices and hyperviscosity operator
[Dx, Dy, Dz, L] = rbf_matrix_fd(nodes);
```

Ignored

```
% Runge-Kutta time stepping. H = [u v w h]
for i=1:timesteps
  F1 = dt*f( H );
  F2 = dt*f( H + 0.5*F1 );
  F3 = dt*f( H + 0.5*F2 );
  F4 = dt*f( H + F3 );
  H = H + 1.0/6.0*(F1 + 2.0*F2 + 2.0*F3 + F4);
end
```

1 %

```
% Evaluate time derivatives
function dH = f(H)

Tx = Dx*H;      % Apply differentiation matrices
Ty = Dy*H;
Tz = Dz*H;
```

70 %

```
RHS = ...
dH(:,1) = ...
dH(:,2) = ...
dH(:,3) = ...
dH(:,4) = ...
```

$$\% \; RHS = \begin{bmatrix} u \circ \mathrm{Tx}_u + v \circ \mathrm{Ty}_u + w \circ \mathrm{Tz}_u \\ u \circ \mathrm{Tx}_v + v \circ \mathrm{Ty}_v + w \circ \mathrm{Tz}_v \\ u \circ \mathrm{Tx}_w + v \circ \mathrm{Ty}_w + w \circ \mathrm{Tz}_w \end{bmatrix} + f \begin{bmatrix} y \circ w - z \circ v \\ z \circ u - x \circ w \\ x \circ v - y \circ u \end{bmatrix} + g \begin{bmatrix} \mathrm{Tx}_h \\ \mathrm{Ty}_h \\ \mathrm{Tz}_h \end{bmatrix}$$

$\% \; \partial u / \partial t = -p_x \cdot RHS$

$\% \; \partial v / \partial t = -p_y \cdot RHS$

$\% \; \partial w / \partial t = -p_z \cdot RHS$

$\% \; \partial h / \partial t = u \circ \mathrm{Tx}_h + v \circ \mathrm{Ty}_h + w \circ \mathrm{Tz}_h \quad + \quad h \circ (\mathrm{Tx}_u + \mathrm{Ty}_v + \mathrm{Tz}_w)$

6 %

```
dH = dH + L*H;  % Apply hyper-viscosity
```

22 %

C++ Implementation

Serial Implementation and Optimization

C++ Implementation

- Using row-major (MATLAB uses col-major) (CSR)
- Hardcoded SpMV for 4-element wide vectors, using SIMD
- Reuse sparsity pattern: Combine `Tx = Dx*H`, `Ty = Dy*H`, `Tz = Dz*H` into `T = D*H`
- Memory layout: Single array of structs instead of many arrays

Execution Time Comparison (million cycles)

	MATLAB	**C++**	Speedup
D_x, D_y, D_z	8186	1441	5.7 x
RHS	790	200	4.0 x
Hyper-viscosity	2606	679	3.8 x
Total	12062	2402	**5.0 x**

Parallelization

Parallelization Strategy

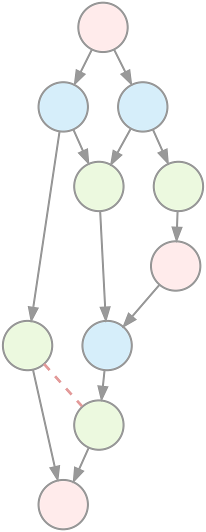
Task-Based Approach

- Fine-grained synchronization
- Avoid global barriers

Parallelized using the SuperGlue library

- Data-dependency driven
  - Programmer divides software into tasks
  - Specifies which data each task reads & writes
  - Submits tasks to SuperGlue
- SuperGlue manages dependencies and maps tasks to cores

`libsuperglue.org`

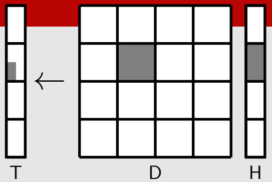


SuperGlue Library

User Interface:

- Create **handles** for all blocks of matrices or vectors
- Create **tasks**, and register which handles are accessed (and how)

Example: Matrix-Vector




```
BlockedMatrix D(n,n);
BlockedVector H(n);
BlockedVector T(n);

// T += D*H
for (int i = 0; i < n; ++i)
  for (int j = 0; j < n; ++j)
    submit(new mult(T(i), D(i,j), H(j)));

struct mult : public Task {
  mult(VectorBlock &T, MatrixBlock &D, VectorBlock &H) {
    registerAccess(add, T.handle);
    registerAccess(read, D.handle);
    registerAccess(read, H.handle);
  }
  void run() { /* T += D*H */ }
};
```

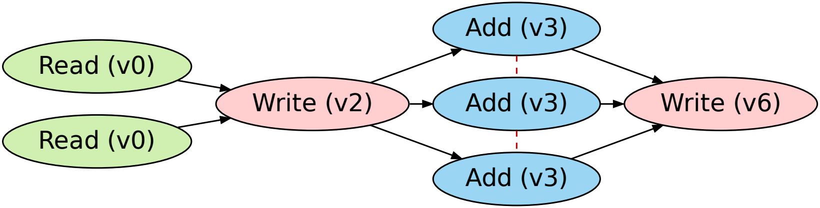
SuperGlue: How It Works

Dependency Management through Data Versioning

- Each handle has a version
- This version is increased after each access
- Tasks must wait for certain handle versions

Example

```
submit(new ReadTask(x));
submit(new ReadTask(x));
submit(new WriteTask(x));
submit(new AddTask(x));
submit(new AddTask(x));
submit(new AddTask(x));
submit(new WriteTask(x));
```



Task-Based Implementation

Parallelization

The Parallel Code

```
// Runge-Kutta step
GenTasks::run() {
  f(F1, H);                             // F1 = f(H)
  add(H1, H, 0.5*dt, F1); f(F2, H1);   // F2 = f(H + 0.5*dt*F1)
  add(H2, H, 0.5*dt, F2); f(F3, H2);   // F3 = f(H + 0.5*dt*F2)
  add(H3, H,     dt, F3); f(F4, H3);   // F4 = f(H + dt*F3)

  step(H, F1, F2, F3, F4);             // H = H + dt/6*( F1 + 2*F2 + 2*F3 + F4 )

  submit(new GenTasks(H));             // Generate new tasks when this step is finished
}

// evaluate ∂H/∂t
void f(dH, H) {
  mult(T, D, H);           // T = D*H
  rhs(dH, H, T);           // dH = ...
  mult(dH, L, H);          // dH = dH + L*H
}
```

Helper Functions to Submit Tasks

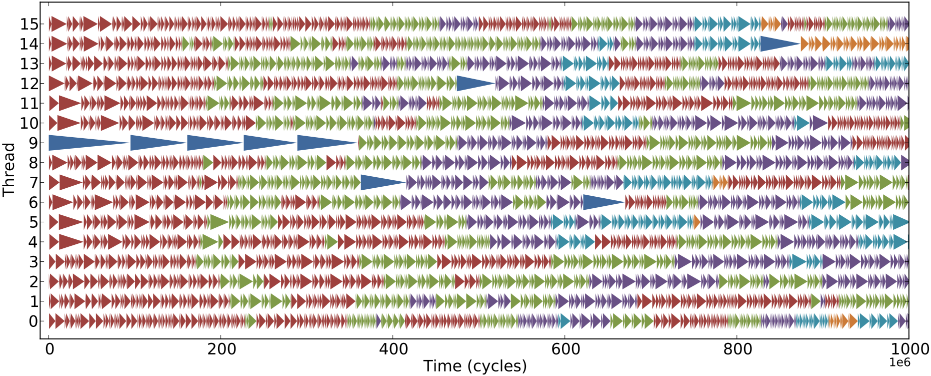
```
mult(T, D, H) {
  for (int r = 0; r < n; ++r)
    for (int c = 0; c < n; ++c)
      submit(mult_task(T(r), D(r, c), H(c)));
}

step(H, F1, F2, F3, F4) {
  for (int r = 0; r < n; ++r)
    submit(step_task(H(r), F1(r), F2(r),
                     F3(r), F4(r)));
}
```

```
add(Htmp, a, H) {
  for (int r = 0; r < n; ++r)
    submit(add_task(Htmp(r), a, H(r));
}

rhs(dH, H, T) {
  for (int r = 0; r < n; ++r)
    submit(rhs_task(H(r), T(r));
}
```

Results

Shared-Memory Experiments (16 core AMD Bulldozer, 8 FPUs)

- 655362 nodes, 100 time steps (only first few visible here)
- Same color = Same time-step



Serial	$325 \cdot 10^9$	cycles	
Serial, blocked	$356 \cdot 10^9$	cycles	(9.5 % slower)
Parallel	$63 \cdot 10^9$	cycles	**(5.2 x faster than serial)**

Idle time: 1.28 %. Not closer to 8 x because of shared resources.

Distributed Memory

Extending to MPI

Extending to MPI

User Interface

- Introduce `MPI_Handle` and `MPI_Task`
  - **Associate rank and memory block with each `MPI_Handle`**
- All nodes must submit the same tasks, in the same order
- Data transfers implicit, inserted automatically

Implementation

- Built as a library on top of SuperGlue
- One core dedicated to MPI

See also

"**DuctTeip**: A Task-Based Parallel Programming Framework with Modularity, Scalability, and Adaptability Features"

Friday, February 21, 10:35-10:55, Salon C

SuperGlue MPI Implementation

Handles extended with following fields to track data

- `last_written_rank` – who last wrote to the data
- `copies` – list of nodes that have a copy of the latest version

On task submit, check if need to transfer data

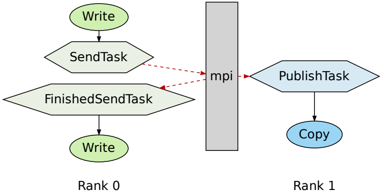
- Send: Submit SendTask { `MPI_Thread.send()` } (and add future read)
- Receive: `MPI_Thread.receive()` (and add future write)

MPI Thread

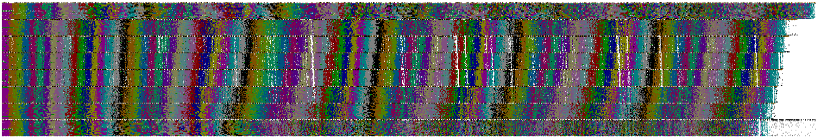
- When data is received: Submit PublishTask { Copy data to handle }
- When data is sent: Submit FinishedSendTask { Nothing }

Example

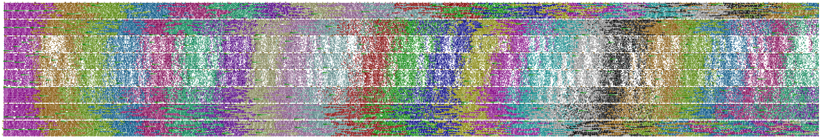
```
x.set_owner_rank(0);
y.set_owner_rank(1);

submit(new Write(x));
submit(new Copy(x, y));
submit(new Write(x));
```



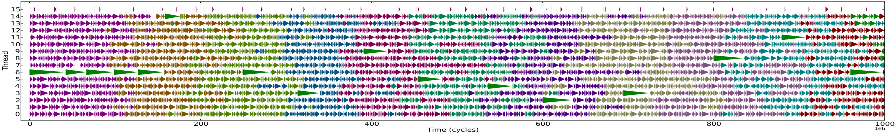
Results on 8 nodes × 16 cores



Full execution: 100 time steps

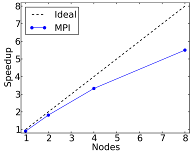


Start zoomed in



First node zoomed in

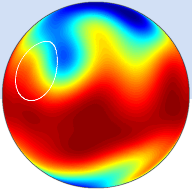
	Time (10^9 cycles)	Speedup over no MPI
No MPI	62.9	
1 node	69.6	(11 % slower)
2 nodes	34.6	1.8 x
4 nodes	18.9	3.3 x
8 nodes	11.4	**5.5 x** (22.6 % idle time)



Conclusion

- Non trivial problem
  - Memory bound
  - Fine-grained
  - Sequential time steps
- High quality sequential implementation
  - 5 times faster than MATLAB implementation
- Successful shared memory parallelization
  - 5 times faster on 16 cores (8 FPUs) (1.28% idle time)
  - Task-based approach was easy and efficient
- Successful MPI version
  - 5 times faster on 8 nodes compared to single node (22.6% idle time)
  - Very few changes in application code

Questions?



Flow over isolated mountain, day 15

`libsuperglue.org`