Verifying Concurrency in an Adaptive Ocean Circulation Model

Alper Altuntas [1] John Baugh [2]

altuntas@ucar.edu, jwb@ncsu.edu

[1]National Center for Atmospheric Research, Boulder, CO

[2]North Carolina State University, Raleigh, NC

Correctness'17
November 12, 2017
Denver, CO

An application of lightweight formal methods

- A **model checking** approach for **concurrent** numerical models
  - An abstraction guideline
    - to verify concurrency
    - to generate safe synchronization arrangements
  - An example application:
    - ADCIRC++, an adaptive ocean circulation model
    - Promela (SPIN)

Concurrency in Numerical Modeling

- **Domain decomposition**
  - Data parallelism
- **Coupled modeling**
  - to simulate multiple phenomena
- **Multi-instance modeling**
  - to simulate varying configurations

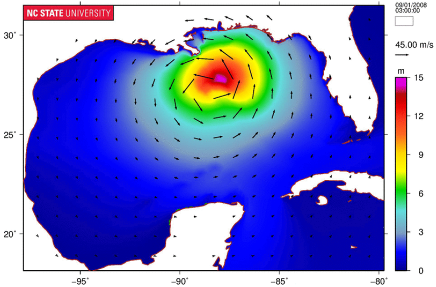
Concurrency in Numerical Modeling

- **Domain decomposition**
  - Data parallelism
- **Coupled modeling**
  - to simulate multiple phenomena
- **Multi-instance modeling**
  - to simulate varying configurations

potentially more asynchronous,
global reductions less common

Storm Surge Modeling

- **Applications:**
  - Forecasting, hindcasting
  - Risk analysis
  - Infrastructure design
- **ADCIRC:**
  - An FE shallow water model.
  - Used by USACE, FEMA, NOAA, and others.
- **ADCIRC++:**
  - A prototype based on ADCIRC to experiment with ASM.



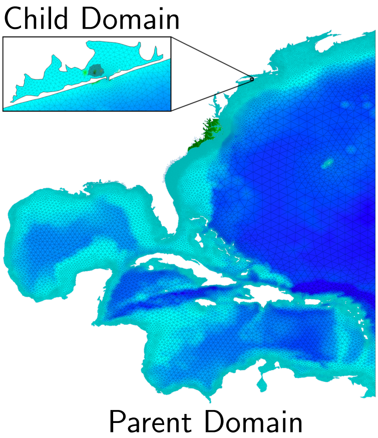
Hurricane Gustav (2008)
(Dietrich et al., 2011)

Outline

Adaptive Subdomain Modeling

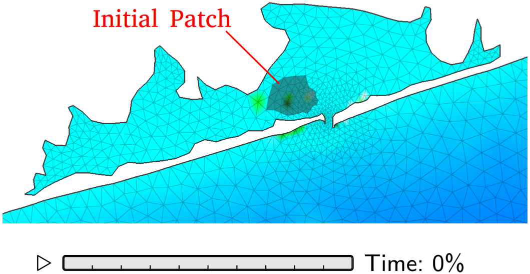
- A computational technique
- Multi-instance concurrency:
  - Parent domain (provides BCs)
  - Child domains
    - alternative design scenarios
    - adaptive spatial extents

- Performance and accuracy:
  - Computational cost of each child: $\sim$2% of cost of full domains
  - Surge height errors: $<$ 1cm



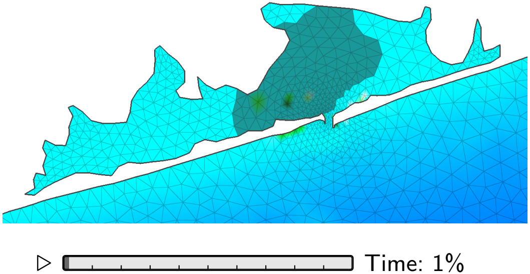
Shinnecock Inlet Child Domain Patch

- The patch **expands** if:
  - Altered hydrodynamics propagate.
- The patch **contracts** if:
  - Altered hydrodynamics recede.



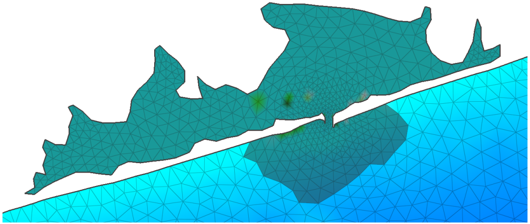
Shinnecock Inlet Child Domain Patch

- The patch **expands** if:
  - Altered hydrodynamics propagate.
- The patch **contracts** if:
  - Altered hydrodynamics recede.



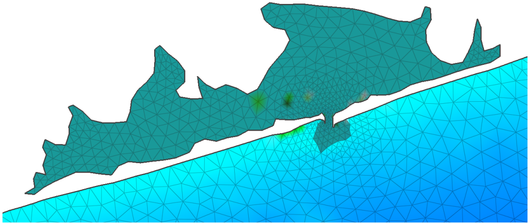
Shinnecock Inlet Child Domain Patch

- The patch **expands** if:
  - Altered hydrodynamics propagate.
- The patch **contracts** if:
  - Altered hydrodynamics recede.



Shinnecock Inlet Child Domain Patch

- The patch **expands** if:
  - Altered hydrodynamics propagate.
- The patch **contracts** if:
  - Altered hydrodynamics recede.

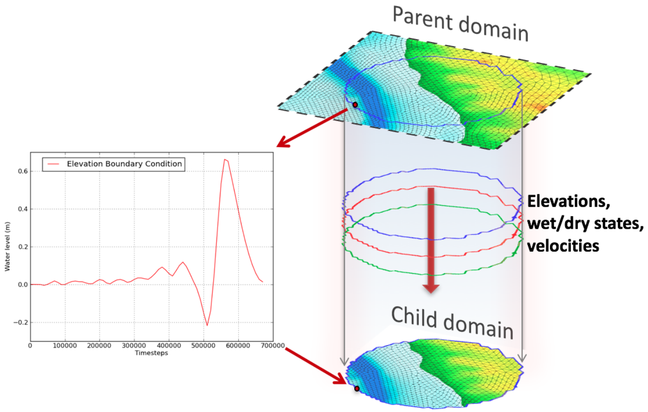


▷ Time: 100%

Adaptive Subdomain Modeling

Enforcing child domain interfaces

- how to **synchronize** concurrent domains sharing the same memory space?



Adaptive Subdomain Modeling

Correctness

- Challenge: multi-instance concurrency
  - Race conditions on critical quantities [1]
- Our solution:
  - Phasing mechanism
- Our verification approach:
  - lightweight model checking

[1] Quantities transferred from parent to children: surge height, velocities, wet/dry states, wind forcing

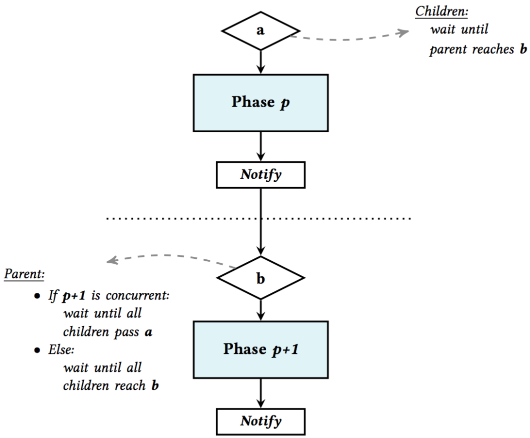
Adaptive Subdomain Modeling

Phasing Mechanism:

1. Group the routines (that constitute a timestep) into phases.
2. Regulate the progression of domains during each timestep.

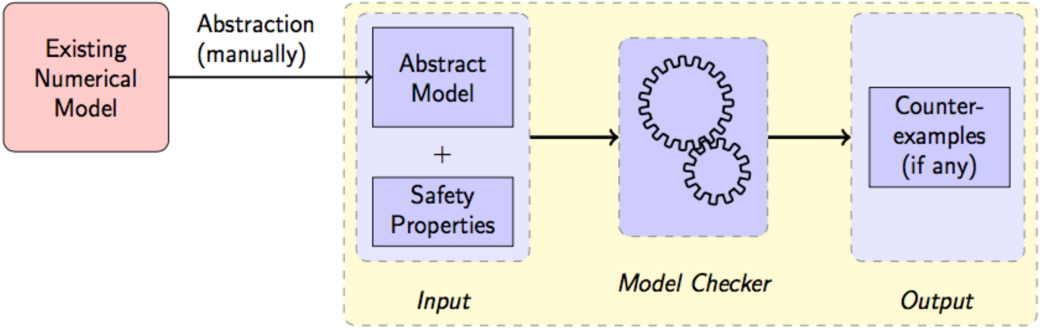
to prevent:

- parent from overwriting data.
- children from using stale data.



Verifying Concurrency in Numerical Models

Model Checking Workflow



Verifying Concurrency in Numerical Models

- **Constituents to be abstracted**
  1. critical quantities
  2. concurrent components/instances
  3. synchronization mechanism

Abstraction

1. **critical quantities:** (e.g., masses, velocities, fluxes)
    - represent with integer variables (denoting the version, or timestamp).
    - a single variable to represent the entire grid.
    - model only two operations:
        - `write` $\rightarrow$ increments variable to designate a new version
        - `copy` $\rightarrow$ a placeholder for safety checks

Abstraction

2. **concurrent components/instances:**
    - represent each as a separate process.
    - incorporate only the synchronization/communication properties.
3. **synchronization mechanism:**
    - use synchronization constructs of the specification language of choice.

Case Study: Verifying Concurrency in ADCIRC++

1. **Critical Quantities**

```
inline write(var){
  if
  :: isParent() -> var++;
  :: else -> var--;
  fi
}
```

```
inline copy(var){
  if
  :: isParent() -> skip;
  :: else -> CHECK_SAFETY;
  fi
}
```

Case Study: Verifying Concurrency in ADCIRC++

2. **Concurrent Domain Instances**
   - In a typical ASM run: 50-100 children
   - In the abstract model: a single child
     - data transfer is one-way from a parent to its children.
     - children do not interfere.

Case Study: Verifying Concurrency in ADCIRC++

3. **Phasing Mechanism**
   - Let SPIN generate phasing arrangements non-deterministically.
   - Call the timestepping routine infinitely many times.

Case Study: Verifying Concurrency in ADCIRC++

- **Verification**
  - The correctness depends on:
    - criteria for entering a phase.
    - arrangement of routines as phases.
  - Thus, verification involves:
    - confirming the correctness of criteria
    - determining safe phasing arrangements
      (that eliminate race conditions on the critical quantities)

Case Study: Verifying Concurrency in ADCIRC++

- **Verification**
  - Safety property: (checked at every copy operation)

```
#define CHECK_SAFETY safe=(var==0)
ltl notsafe {eventually !safe}
```

Interpretation: Child will eventually copy the wrong version of a quantity.
Counterexamples: Safe phasing arrangements

$\Rightarrow$ Looking for phasing arrangements that are never not safe.

Conclusions

- Using SPIN, we found all race-free phasing arrangements in ASM.
- The approach requires only modest levels of human and computer effort:
  - Promela code: 190 lines
  - Initial model put together in less than a day.
- **Future direction:** Using the approach in the context of performance optimization, e.g., optimizing concurrency in coupled climate models.

Thanks