Leveraging Actors and SUNDIALS for Enhanced Performance in Process-based Hydrological Models

Kyle Klenk, Raymond Spiteri
Department of Computer Science

June 1, 2024

Introduction

The Actor Model

- Efficient parallelism of embarrassingly parallel problems
- Fault tolerance
- Distributed computing

SUNDIALS

- Suite of nonlinear and differential-algebraic equation solvers
- Adaptive time integration
- Free and Open Source

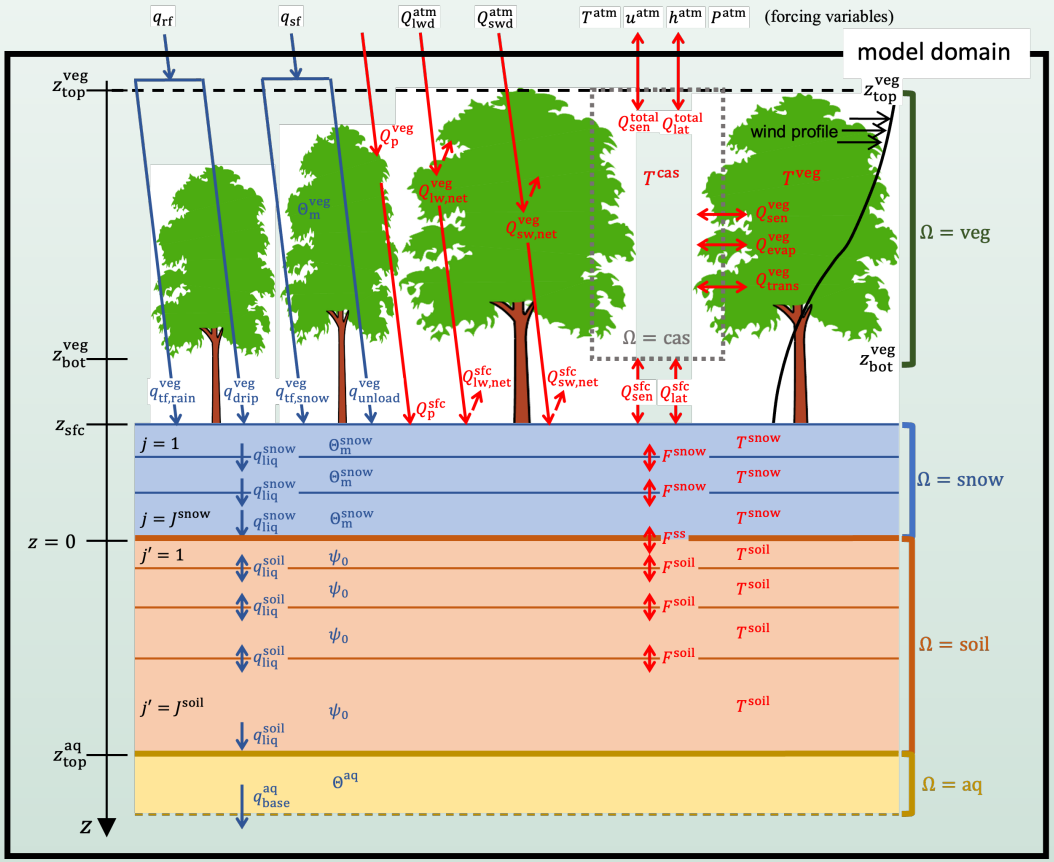
Outline

Section 1

A Motivating Example with SUMMA

SUMMA

SUMMA simulates the following processes (left), and decomposes the spatial domain into Grouped Response Units (GRUs) and Hydrological Response Units (HRUs) (right)



SUMMA

Coupled conservation equations for thermodynamics and hydrology:

$$\frac{\partial H^{\Omega}}{\partial t} = -\frac{\partial F^{\Omega}}{\partial z} + \mathcal{F}^{\Omega}_{\text{sink}}, \qquad \Omega = \text{cas, veg, snow, soil,}$$

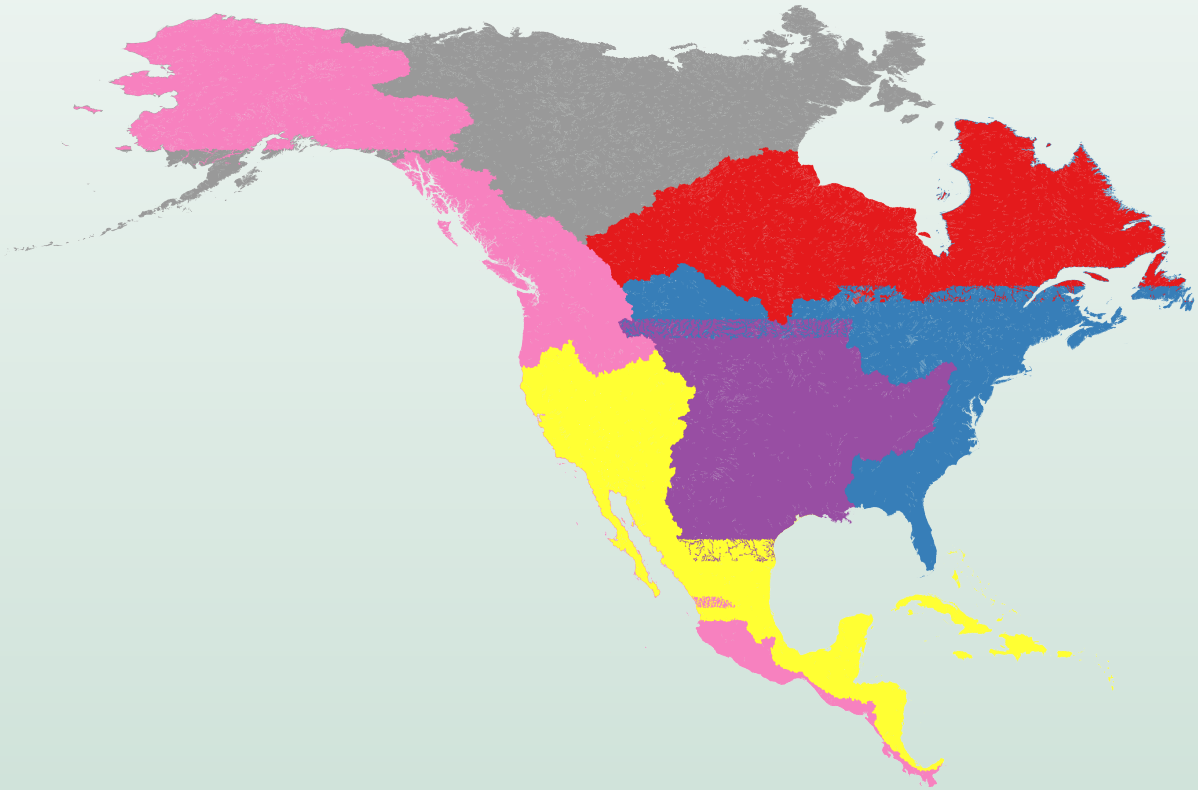
$$\frac{\partial \Theta^{\Omega}_{\text{m}}}{\partial t} = -\frac{\partial q^{\Omega}_{\text{ice}}}{\partial z} - \frac{\partial q^{\Omega}_{\text{liq}}}{\partial z} + \mathcal{M}^{\Omega}_{\text{sink}}, \qquad \Omega = \text{veg, snow, soil, aq.}$$

Modular design of SUMMA allowed for inclusion of SUNDIALS.

Batch Job Example

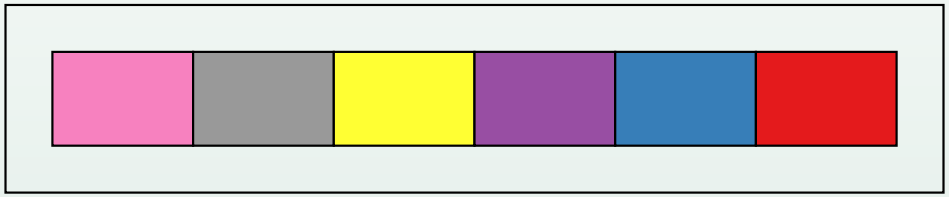
Step 1: Batch GRUs into an array job

- For single node or local machine: Batch GRUs and submit to individual CPUs in background



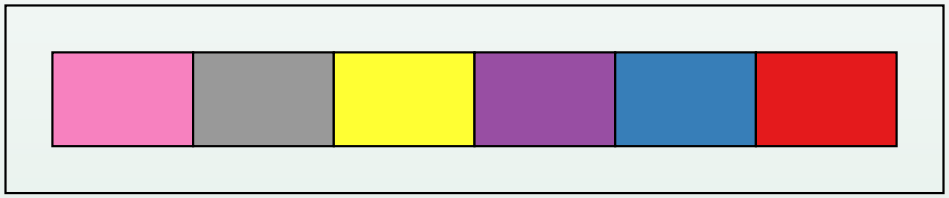
Batch Job Example

Step 1: Batch GRUs into an array job

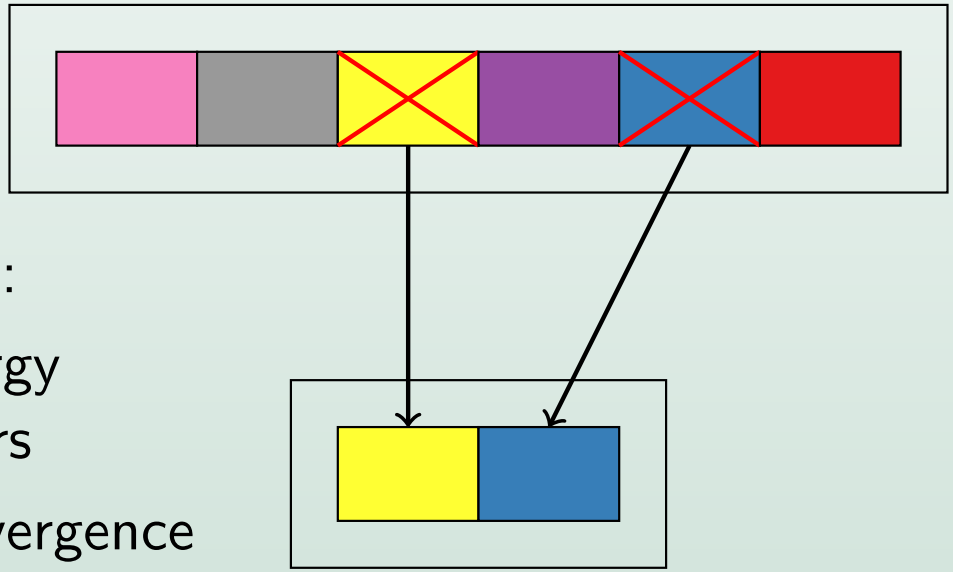


Batch Job Example

Step 1: Batch GRUs into an array job



Step 2: Find and re-submit failed jobs

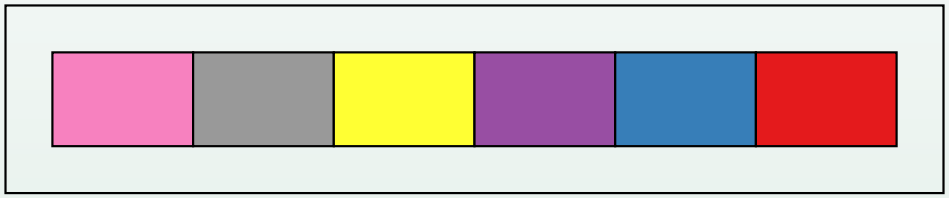


Potential Failures:

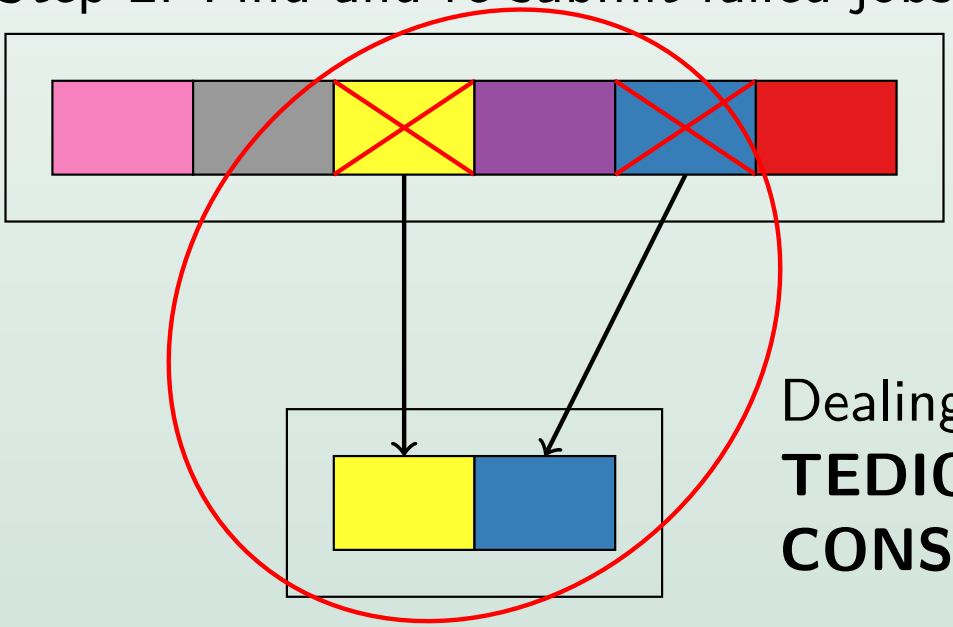
- Mass & energy balance errors
- Lack of convergence
- Hardware failures

Batch Job Example

Step 1: Batch GRUs into an array job

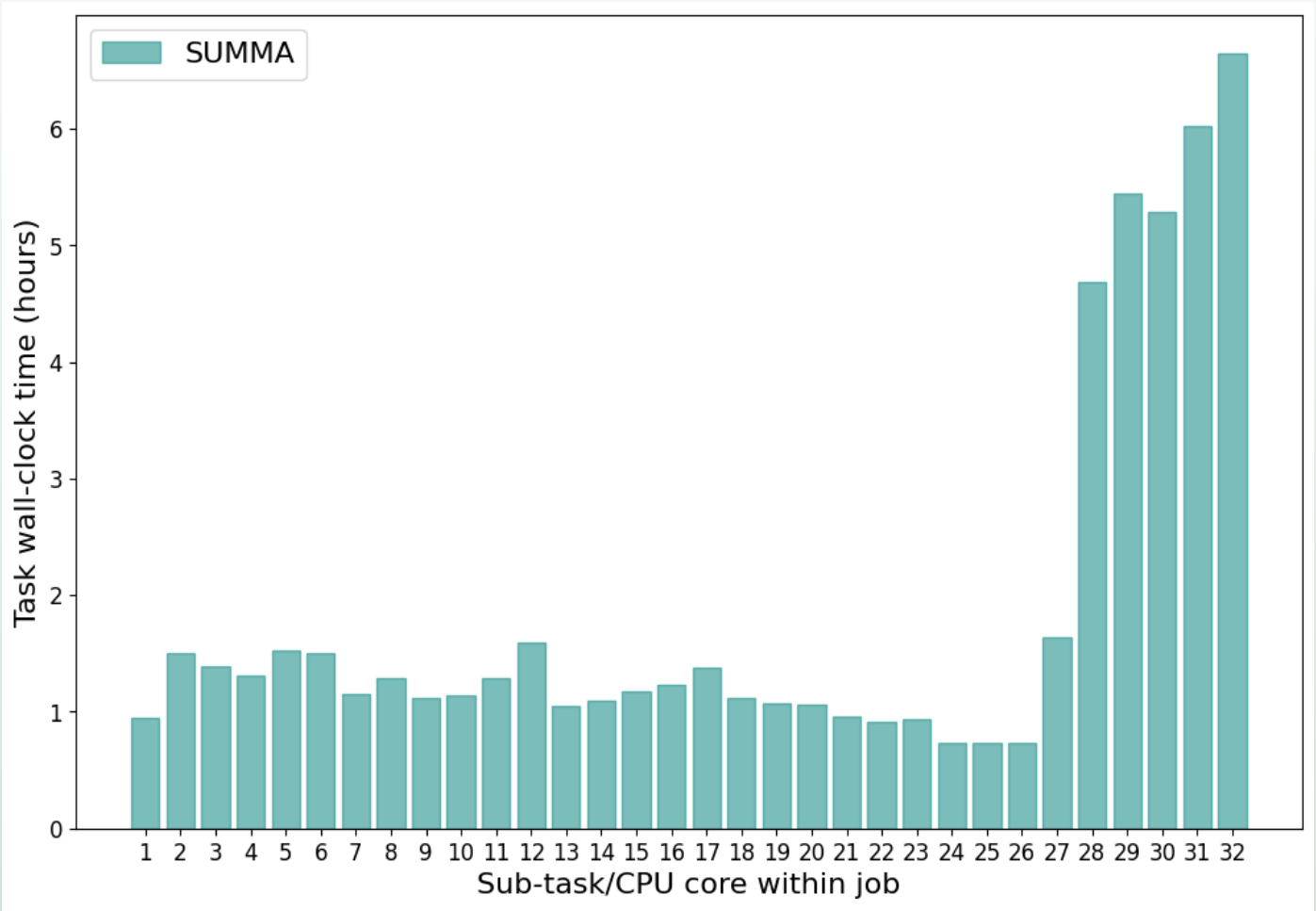


Step 2: Find and re-submit failed jobs



Dealing with failed jobs is **TEDIOUS** and **TIME-CONSUMING**.

Straggler Effect



Demo

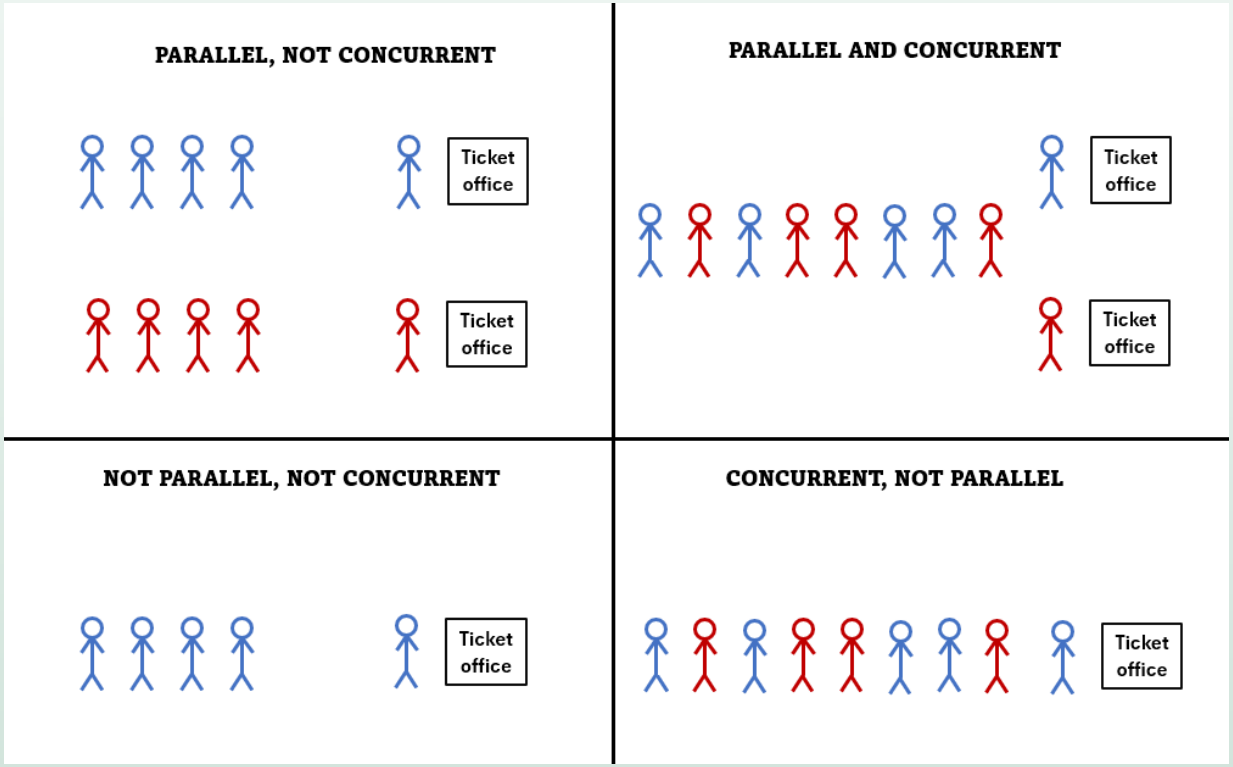
- Setting up SUMMA-SUNDIALS-Actors
- CIROH Cloud
- Docker

Section 2

Actors

Motivation

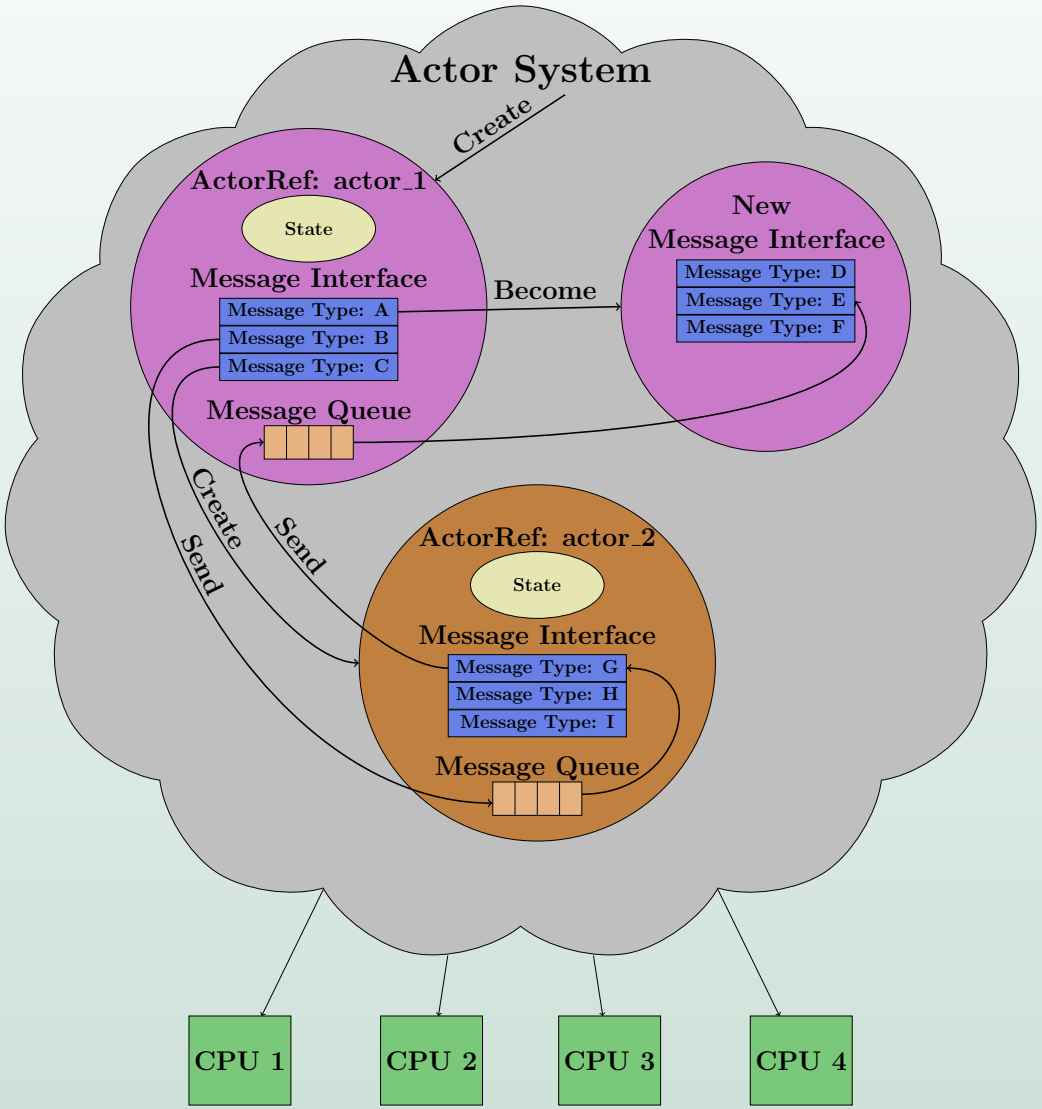
Leverage concurrency to exploit parallelism



What is the Actor Model?

- A computational model for concurrent computation
- Actors as **fundamental units of computation/concurrency**
- Instead of thinking in terms of ranks, think in terms of actors
- Shift from managing individual cores to a task-oriented model

Actors



- Private state
- Communicate only through messages
- Three primitive operations
  - Create
  - Send
  - Become
- Event-driven, work-stealing scheduler

C++ Actor Framework Example

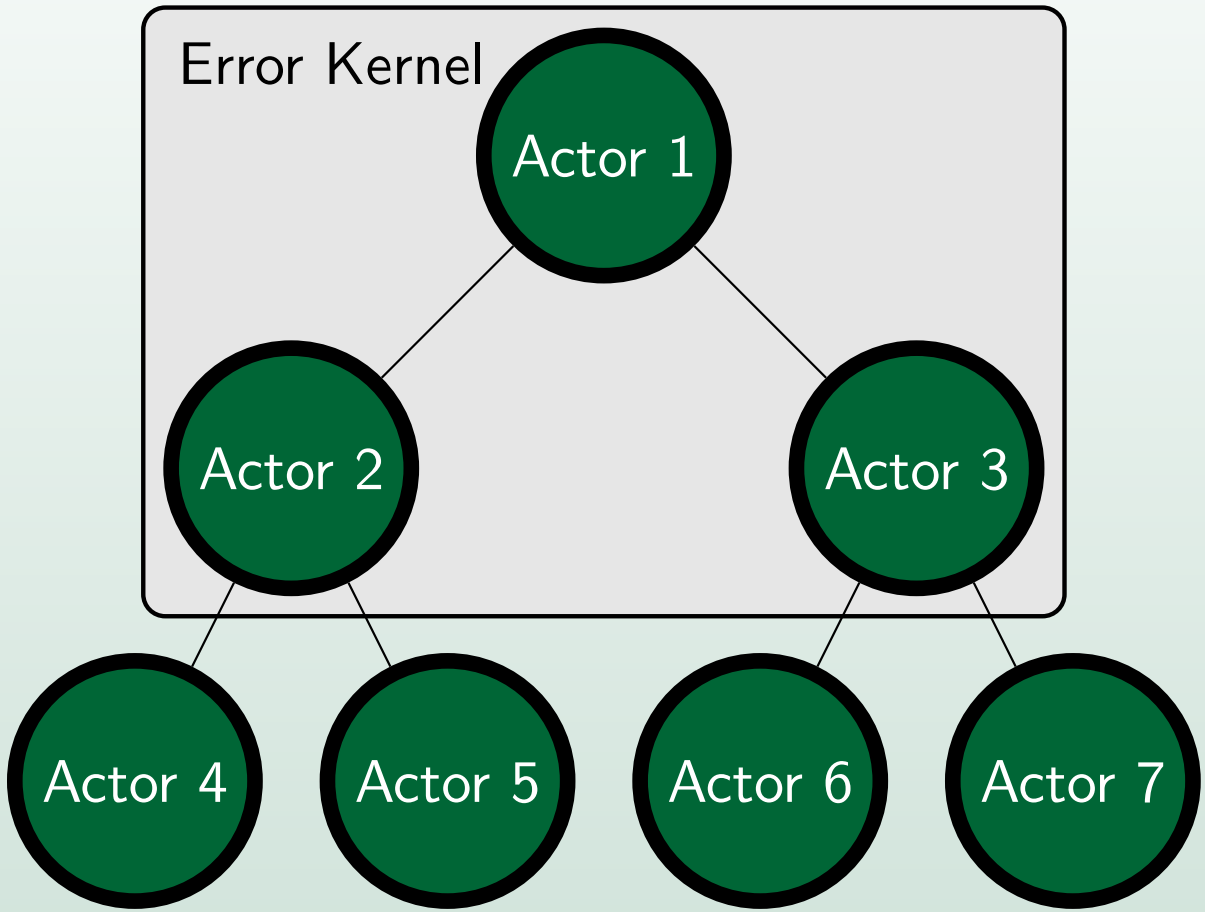
```
#include "caf/all.hpp"
#include "caf/io/all.hpp"
using namespace caf;
behavior hello(event_based_actor *self){
    return {
        [=](std::string name){
            aout(self) << "Hello, " << name <<
                std::endl;
        }
    };
}
void caf_main(actor_system& sys){
    scoped_actor self{sys};
    aout(self) << "Actor System Started" << std
        ::endl;
    auto our_first_actor = self->spawn(hello);
    self->send(our_first_actor, "Actor");
}
CAF_MAIN()
```

Benefits

- Actors are lightweight ($\rightarrow$ extremely scalable)
- Modularity is similar to object-oriented programming
- Network transparent communication:
  Messages appear local, allowing for a **single abstraction** for both **local** and **distributed** computing
- Increased fault tolerance with simplified error handling

Increased fault tolerance

- Isolated Execution
- Supervision
- Error Kernel Pattern



Section 3

SUNDIALS

Time Integration

Time integration with constant-stepsize backward Euler.

Time Integration

Time integration with constant-stepsize backward Euler.

- How do we know whether the answer is right?
- Is there a more efficient way to get this "right" answer?

Time Integration

Time integration with constant-stepsize backward Euler.

- How do we know whether the answer is right? Hint:
- Is there a more efficient way to get this "right" answer?

Time Integration

Time integration with constant-stepsize backward Euler.

- How do we know whether the answer is right? Hint: We don't.
- Is there a more efficient way to get this "right" answer?

Time Integration

Time integration with constant-stepsize backward Euler.

- How do we know whether the answer is right? Hint: We don't.
- Is there a more efficient way to get this "right" answer?

The best we can do is to get
almost the right answer to almost the right question.

Time Integration

Time integration with constant-stepsize backward Euler.

- How do we know whether the answer is right? Hint: We don't.
- Is there a more efficient way to get this "right" answer?

The best we can do is to get

almost the right answer to almost the right question.

- Use adaptive time integration (e.g., SUNDIALS)

Adaptive Time Integration

- We literally don't know how accurate (or not) a one-off constant-stepsize integration is.
  - can be wildly inaccurate
  - can be wildly inefficient
  - can be especially true at low order

Adaptive Time Integration

- We literally don't know how accurate (or not) a one-off constant-stepsize integration is.
  - can be wildly inaccurate
  - can be wildly inefficient
  - can be especially true at low order
- Suite of Nonlinear and Differential-Algebraic Equation Solvers
  - error control through adaptive stepsizes and order

Adaptive Time Integration

- We literally don't know how accurate (or not) a one-off constant-stepsize integration is.
  - can be wildly inaccurate
  - can be wildly inefficient
  - can be especially true at low order
- Suite of Nonlinear and Differential-Algebraic Equation Solvers
  - error control through adaptive stepsizes and order

Caveat Emptor: It's not (exactly) for free!

Adaptive Time Integration

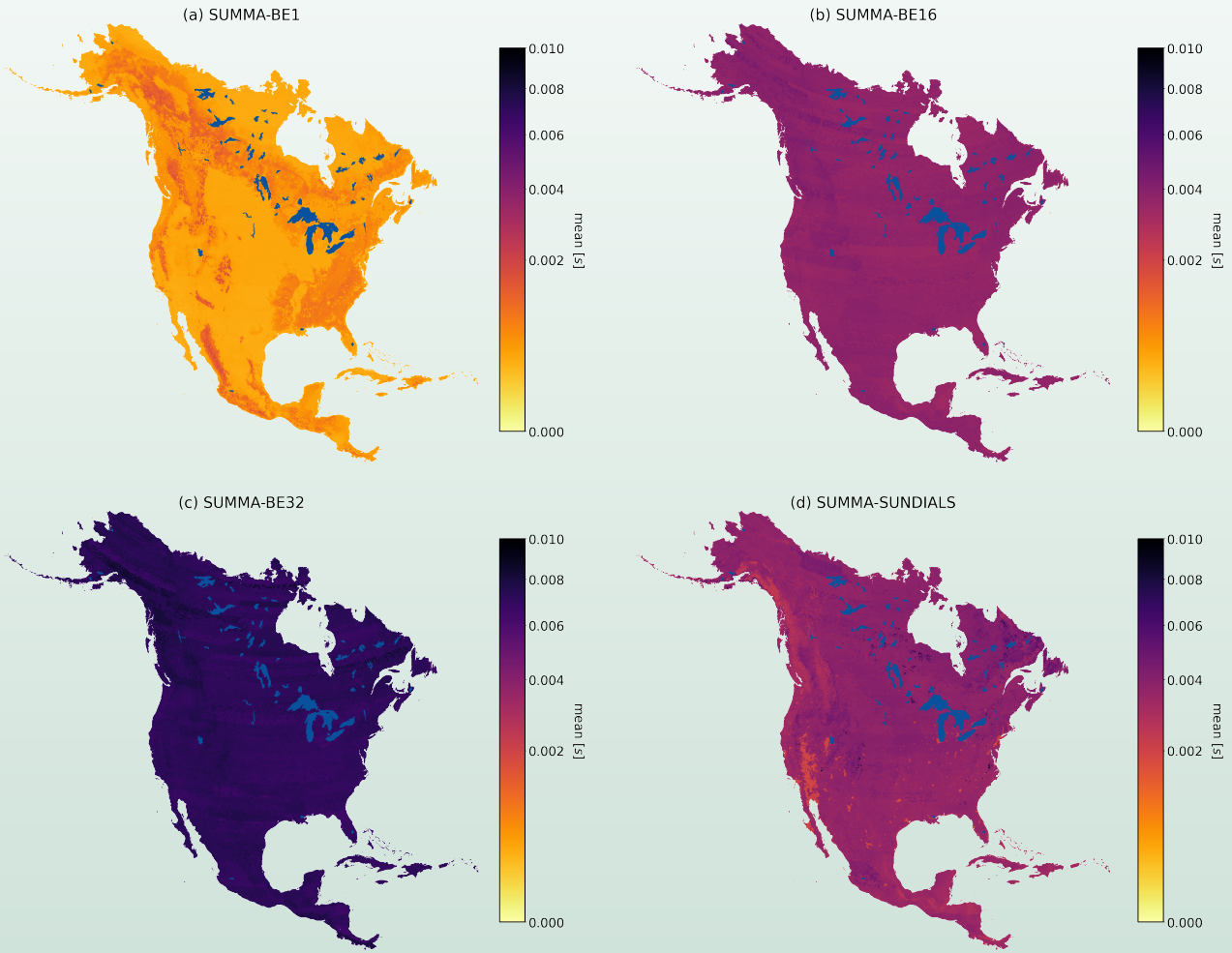
- We literally don't know how accurate (or not) a one-off constant-stepsize integration is.
  - can be wildly inaccurate
  - can be wildly inefficient
  - can be especially true at low order
- Suite of Nonlinear and Differential-Algebraic Equation Solvers
  - error control through adaptive stepsizes and order

Caveat Emptor: It's not (exactly) for free!

Why wait for the right answer when you can get the wrong one now?

Results

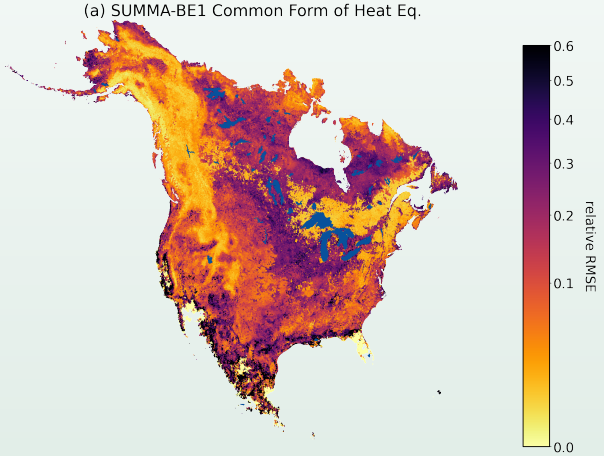
Wall clock time Hourly Statistics



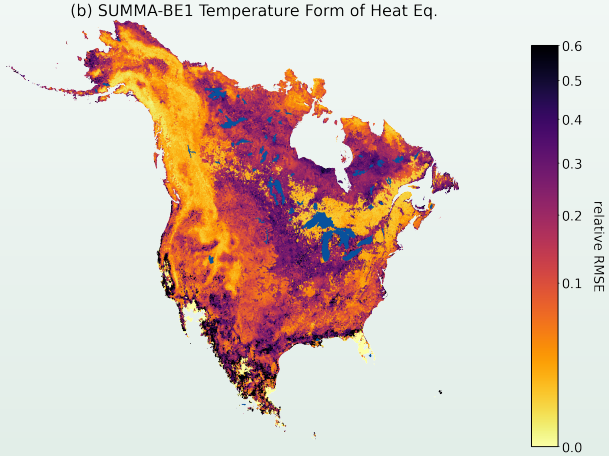
Results

Snow Water Equivalent Hourly Statistics

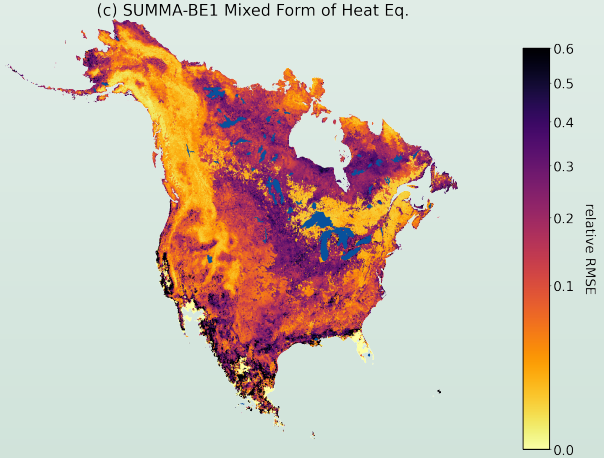
(a) SUMMA-BE1 Common Form of Heat Eq.



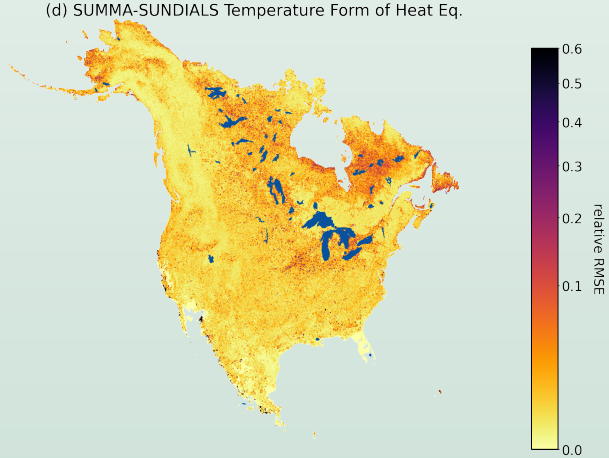
(b) SUMMA-BE1 Temperature Form of Heat Eq.



(c) SUMMA-BE1 Mixed Form of Heat Eq.



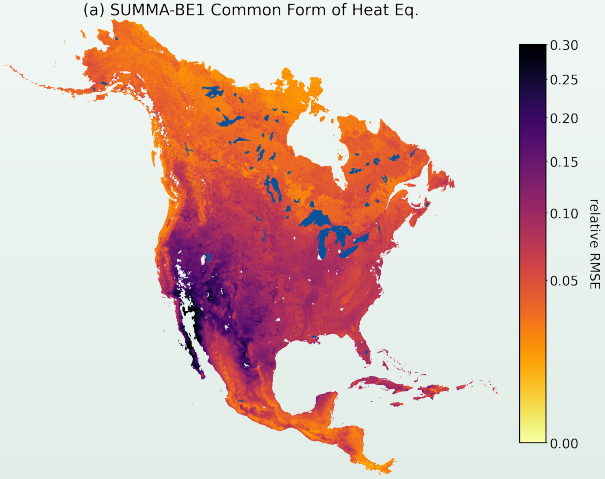
(d) SUMMA-SUNDIALS Temperature Form of Heat Eq.



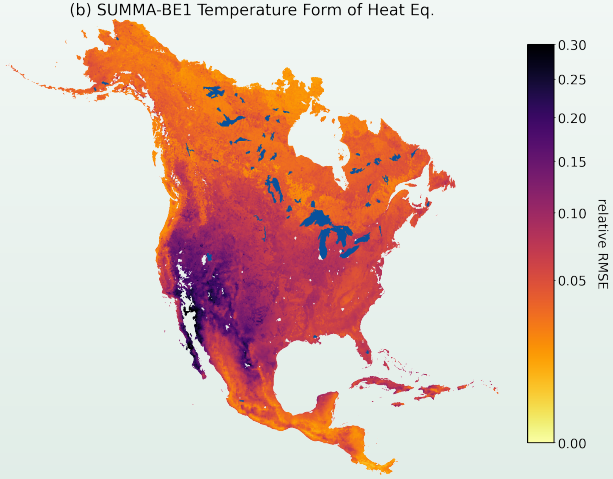
Results

Total water on the vegetation canopy Hourly Statistics

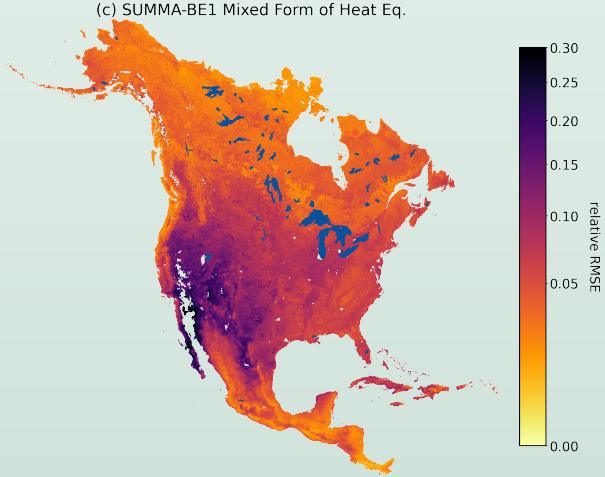
(a) SUMMA-BE1 Common Form of Heat Eq.



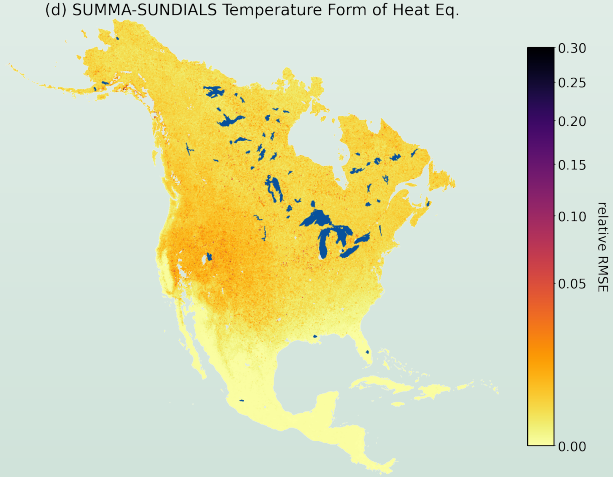
(b) SUMMA-BE1 Temperature Form of Heat Eq.



(c) SUMMA-BE1 Mixed Form of Heat Eq.



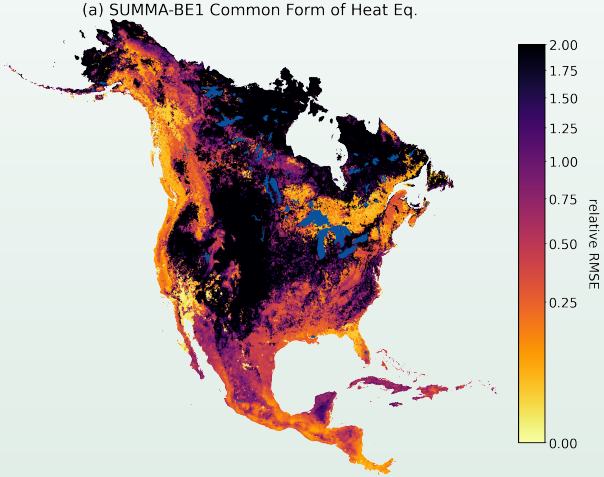
(d) SUMMA-SUNDIALS Temperature Form of Heat Eq.



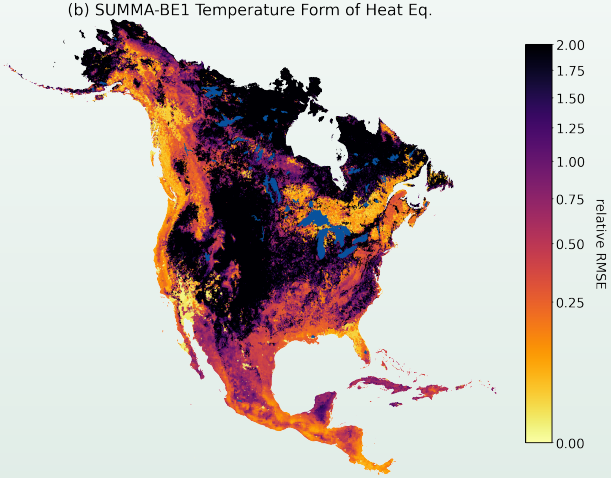
Results

Average routed runoff Hourly Statistics

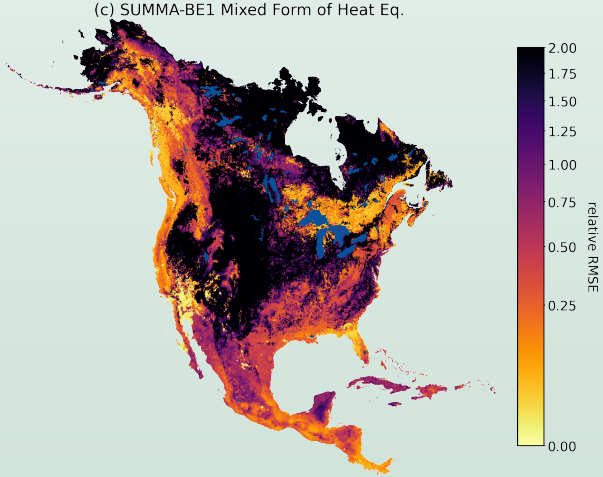
(a) SUMMA-BE1 Common Form of Heat Eq.



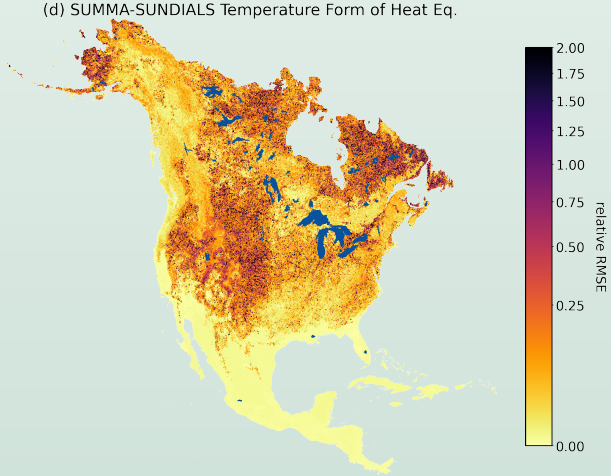
(b) SUMMA-BE1 Temperature Form of Heat Eq.



(c) SUMMA-BE1 Mixed Form of Heat Eq.



(d) SUMMA-SUNDIALS Temperature Form of Heat Eq.



Section 4

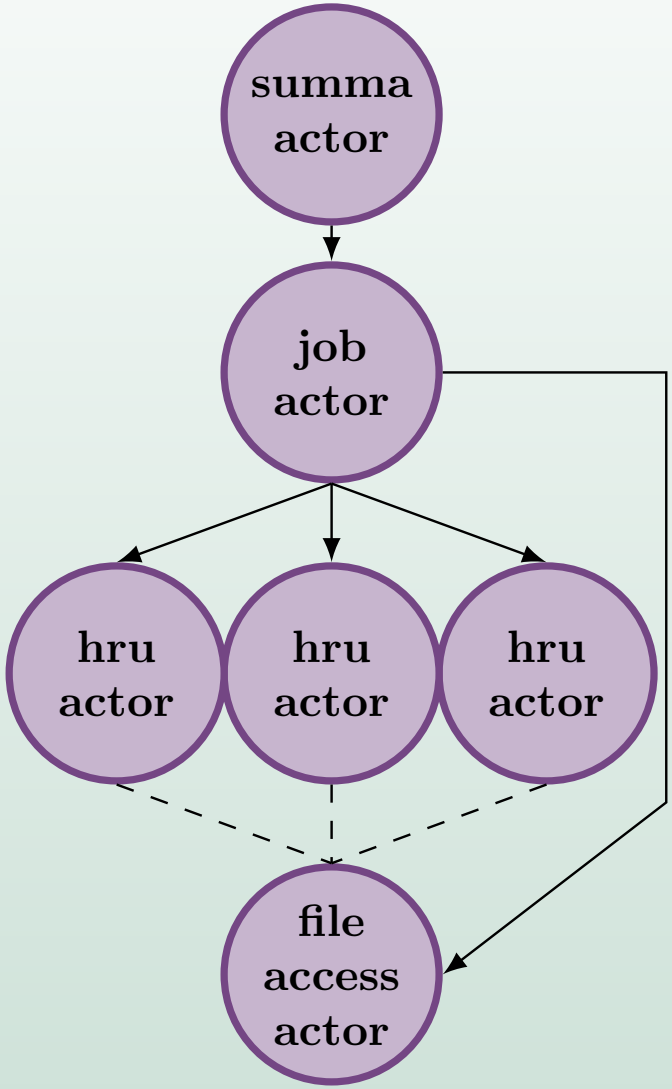
SUMMA-SUNDIALS-Actors

SUMMA-SUNDIALS-Actors

Do it all again with the addition of actors.

- more advantages for resource utilization and fault tolerance
  - more variability in run times = bigger stragglers
- more advantages for workflow
  - more parameters to tweak before re-start
- potential for "computational reasoning"
  - data from finished/running simulations inform next ones

Architecture



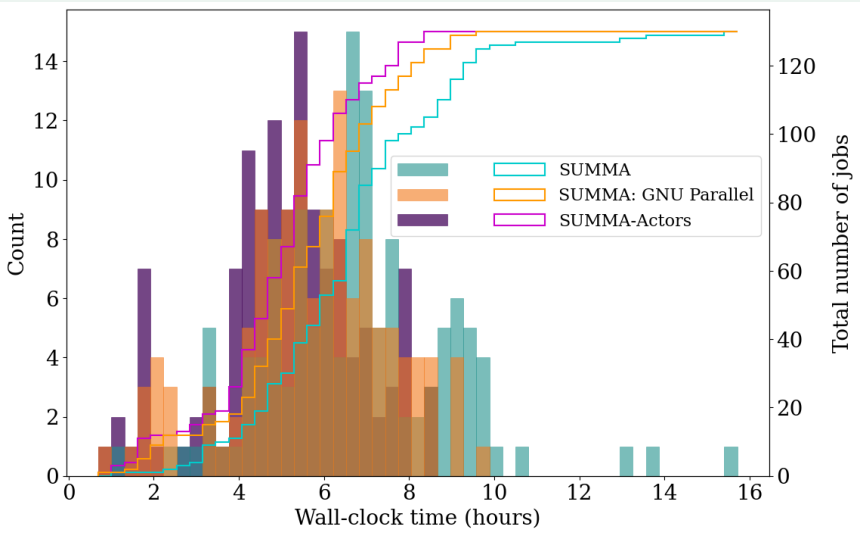
Performance Comparison (SUMMA-Actors)

- Hydrological simulation over North America for 40 years
- 517,315 GRUs
- Submitted as array jobs
- SUMMA-Actors compared against
  - Standard SUMMA batch submission (no load balancing)
  - SUMMA using GNU Parallel for load balancing

Graham Simulation

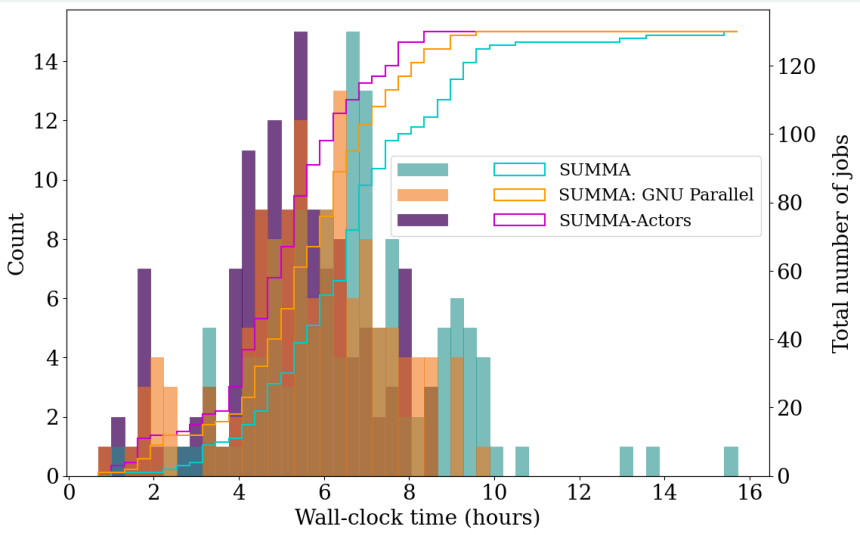
- 4,000 GRUs per job; 130 total jobs
- 32 Broadwell CPUs per job (4,160 total cores)
- SUMMA: divided into 32 processes, 125 GRUs per process
- SUMMA-GNU Parallel: divided into 160 subtasks of 25 GRUs
- SUMMA-Actors: no batching; simply submit all jobs

Graham Simulation: Wall-Clock Time



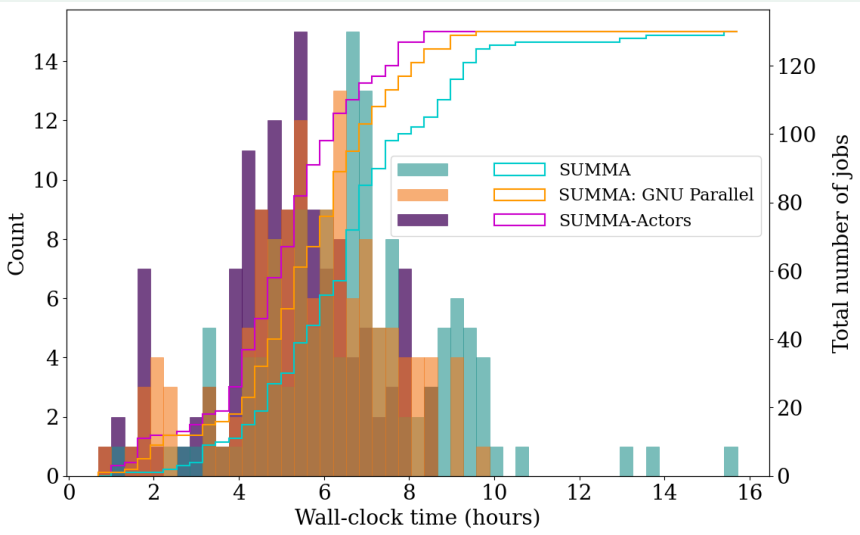
- Totals (hours)
  - SUMMA: 875
  - SUMMA: GNU Parallel: 743
  - SUMMA-Actors: 667
- SUMMA-Actors is faster than
  - SUMMA by 26%
  - SUMMA: GNU Parallel by 10%

Graham Simulation: Wall-Clock Time



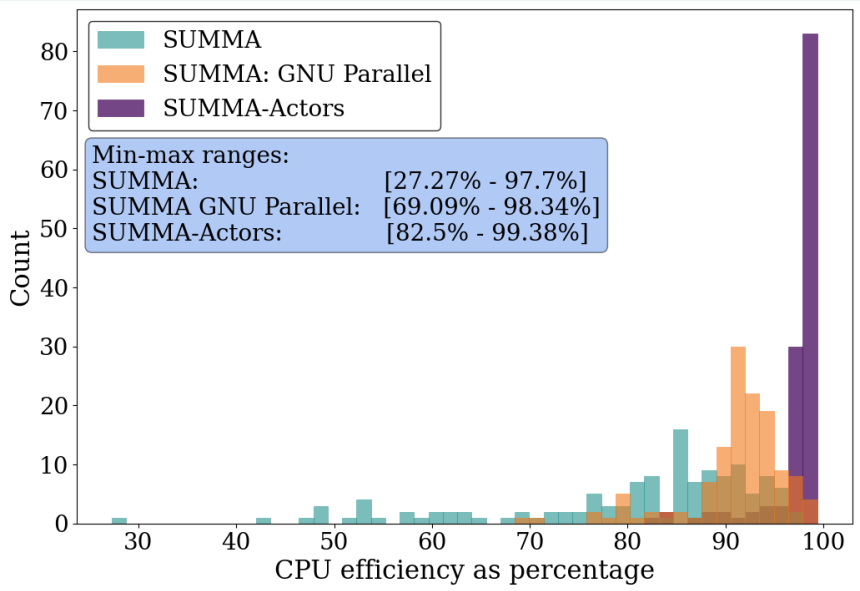
- Totals (hours)
  - SUMMA: 875
  - SUMMA: GNU Parallel: 743
  - SUMMA-Actors: 667
- SUMMA-Actors is faster than
  - SUMMA by 26%
  - SUMMA: GNU Parallel by 10%

Graham Simulation: Wall-Clock Time



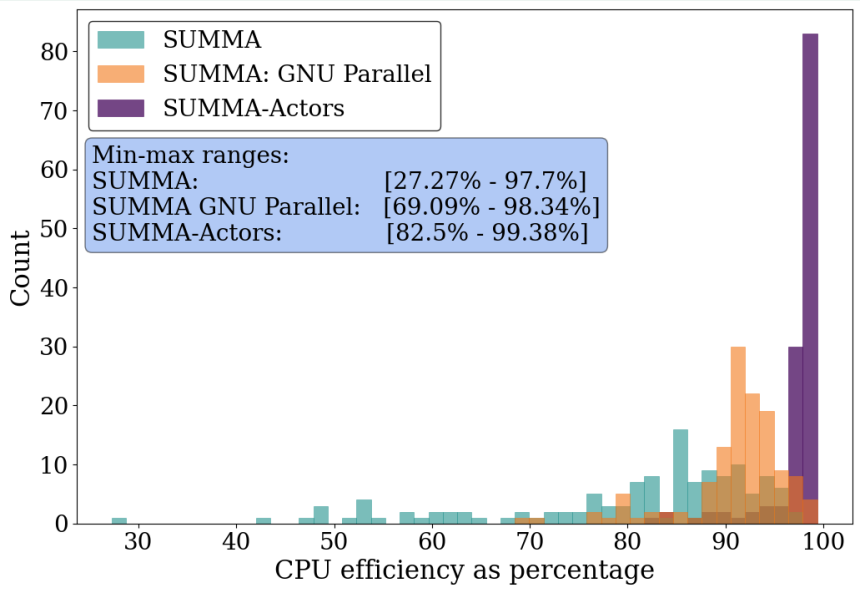
- Totals (hours)
  - SUMMA: 875
  - SUMMA: GNU Parallel: 743
  - SUMMA-Actors: 667
- SUMMA-Actors is faster than
  - SUMMA by 26%
  - SUMMA: GNU Parallel by 10%

Graham Simulation: CPU Efficiency



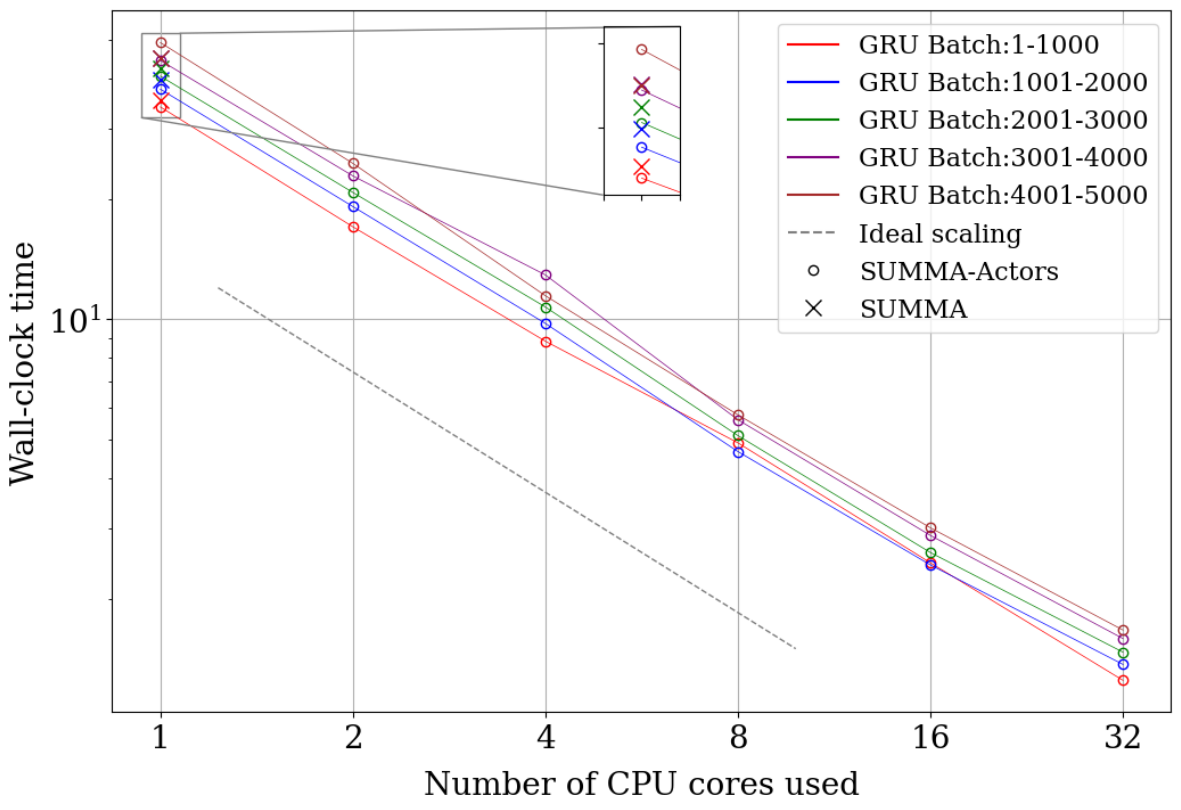
- Average efficiency (Slurm reported)
  - SUMMA: 85%
  - SUMMA: GNU Parallel: 90%
  - SUMMA-Actors: 98%

Graham Simulation: CPU Efficiency



- Average efficiency (Slurm reported)
  - SUMMA: 85%
  - SUMMA: GNU Parallel: 90%
  - SUMMA-Actors: 98%

Results: Scalability



Early SUMMA-SUNDIALS-Actors Performance

North America simulation for 5 years (1979–1984)

Same configuration on Graham as previous slides, no GNU-Parallel