ADHydro

A Large-Scale Multi-Physics Hydrology Simulation Implemented with Charm++
Goal: Upper Colorado Watershed on Yellowstone Supercomputer
Goal: Upper Colorado Watershed on Yellowstone Supercomputer

Area: 288,000 km$^2$
Streams: 467,000 km
Population: 900,000
Population using water: >30,000,000

Computation nodes: 4536
Processor cores: 72,576
Peak petaflops: 1.504
36$^{th}$ fastest in world (top500.org)
Research Questions

How will climate and land-use changes affect the availability of water for the Upper Colorado watershed states (CO, WY, UT, NM, AZ) over the coming decades?

How can water management be optimized, and how much benefit can be realized from such optimization?

- E.g. Changing from flood irrigation to center-pivot irrigation
Additional Research Goal

Explore high spatial and temporal resolution from a capability-driven perspective.
Capability-Driven Resolution

High resolution
  - Mesh elements as small as 1/2 acre or 100 meters of stream reach

Variable resolution
  - Unstructured mesh
    - Offline mesh coarsening to explore effects of resolution

Adaptive resolution
  - Implemented for temporal
  - Not yet implemented for spatial
Multi-Physics

We are not a “conceptual” model
- Select abstract coefficients to fit historical data

We are a “physical” model
- All inputs are in theory measurable in the lab
Multi-Physics

We model multiple largely independent physical processes

API for Independent swappable physics modules
  – Some are legacy codes

“Arbitrary” water management
  – Irrigation and reservoir releases based on human decisions, not physics
Multi-Physics

Explicit first-order simulation

Guaranteed to conserve mass

Guaranteed to converge
Simulation Architecture

- Water Management
- Snowmelt
- Rainfall
- Infiltration
- Groundwater
- Channel Network
- Surface Water

State Variables:
- mesh geometry
- groundwater head
- vadose zone state
- surface water depth

Processes:
- Snowmelt
- Rainfall
- Infiltration
- Evapo-Transpiration

Concepts:
- Surface Water
- Channel Network
- Groundwater
- Water Management
Reasons for Using Charm++

Partitioning

Load Balancing

Checkpointing

Easy to overlap computation and communication

Doesn't require learning advanced features of C++
  - Domain experts are not computer scientists
Recent Implementation Work

Regions
- Chare objects that contain multiple mesh elements
- Reduces chare-switching overhead
  Aggregates messages

Heterogeneous timestepping
- Neighbors agree on flow rate and expiration time and can then independently step forward to expiration time with different timesteps
Heterogeneous timestepping

Step 1: Calculate nominal flow rate

Each element makes an agreement with each of its neighbors about the “nominal” flow rate between them and an expiration time for this nominal flow rate
Heterogeneous timestepping

Step 1: Calculate nominal flow rate

This nominal flow rate will not be recomputed until the expiration time

- Although less water than that may flow if the sender runs out of water
- The expiration time must account for both the Courant number of the flow and factors like looking ahead in the forcing file to detect upcoming shocks
Heterogeneous timestepping

Step 1: Calculate nominal flow rate

Both neighbors agree to sync up at the expiration time to recalculate a new nominal flow rate and expiration time

- Both neighbors must end a timestep end at exactly the expiration time
Heterogeneous timestepping
Heterogeneous timestepping

Step 2: Select timestep

Each element selects its timestep end time

- This can be no later than its earliest nominal flow rate expiration time, although it can be earlier
- It does not have to be the same as any other element's timestep end time except when recalculating a nominal flow rate with a neighbor
Heterogeneous timestepping
Heterogeneous timestepping

Step 3: Send outflow water

Each element calculates its actual outflows for its current timestep including flow limiting if it runs out of water

Each element sends messages containing this water to its neighbors
Heterogeneous timestepping
Heterogeneous timestepping

Step 4: Wait for inflow water

Each element waits until it has received water messages for all of its inflows to at least its current timestep end time

- Neighbors might have shorter timesteps so the element may have to wait for multiple water messages from a single neighbor
Heterogeneous timestepping
Heterogeneous timestepping
Step 5: Advance time

Each element updates its water state and advances current time to the end of its timestep

Then the element returns to step 1 if any of its nominal flow rates have expired or step 2 if not
Backup Slides
Heterogeneous timesteping

Guaranteed to conserve water

A quantity of water is removed from one bucket, and the same quantity of water is added to another bucket.

Water cannot be received until the sender removes that water from its bucket.

No element ever sends more water than it has.
Heterogeneous timestepping

Guaranteed Not To Deadlock

If an element is waiting to calculate a nominal flow rate with me its current time must be strictly greater than my current time and its timestep end time must be greater than or equal to my timestep end time

If an element is waiting for a water message from me its timestep end time must be strictly greater than my timestep end time
Heterogeneous timestepping

Guaranteed Not To Deadlock

If an element is waiting on me through a transitive chain that does not include waiting on a water message then its current time is strictly greater than my current time.

If an element is waiting on me through a transitive chain that does include waiting on a water message then its timestep end time is strictly greater than my timestep end time.

There can be no cycles in the waits-for graph.
Collaborators

Utah State University

UW

BYU

University of Utah

NCAR

National Center for Atmospheric Research