

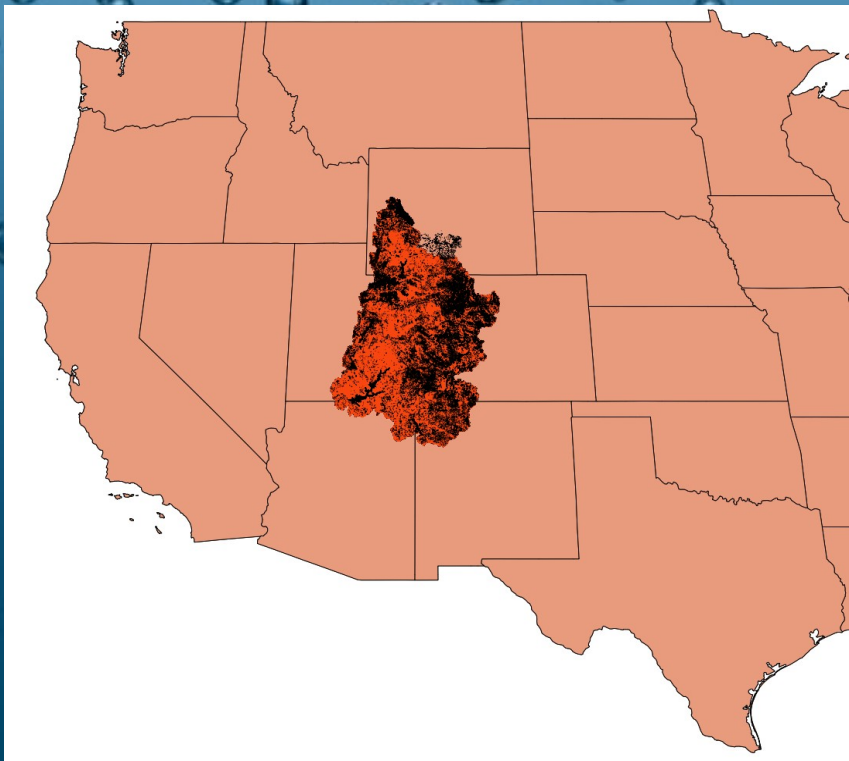


ADHydro

A Large-Scale Multi-Physics Hydrology Simulation Implemented with
Charm++

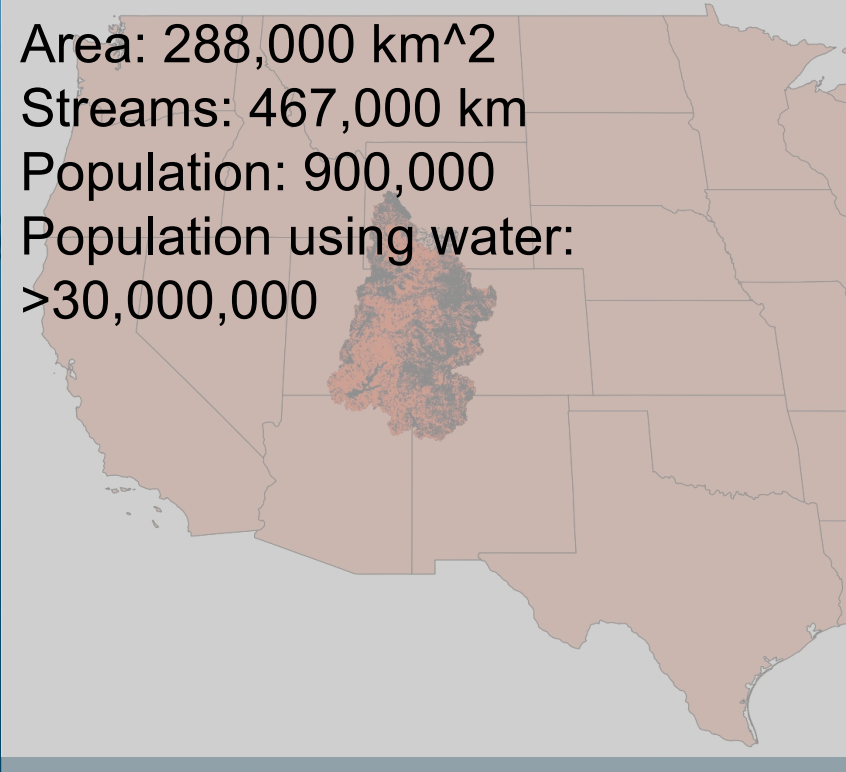


Goal: Upper Colorado Watershed on Yellowstone Supercomputer

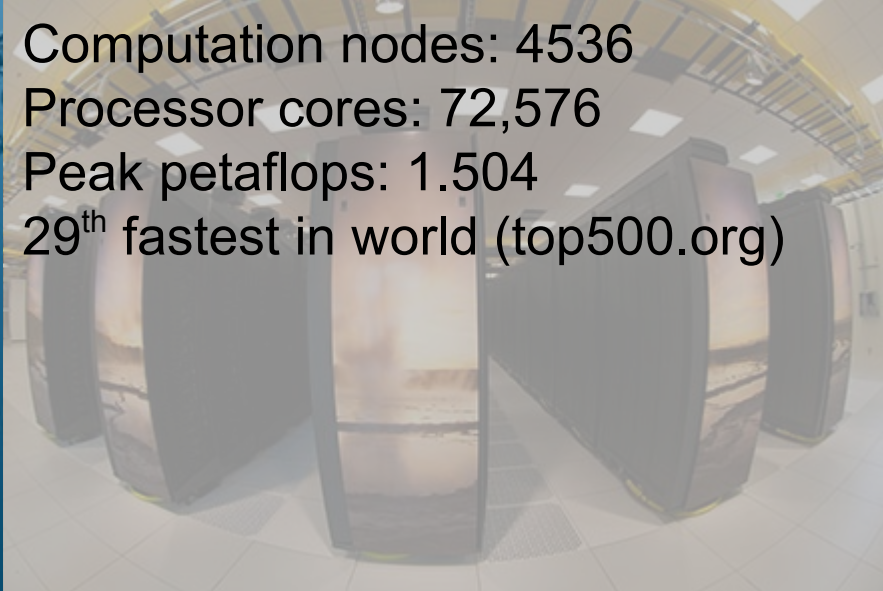




Goal: Upper Colorado Watershed on Yellowstone Supercomputer



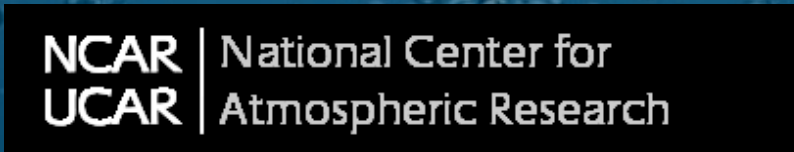
Area: 288,000 km²
Streams: 467,000 km
Population: 900,000
Population using water:
>30,000,000



Computation nodes: 4536
Processor cores: 72,576
Peak petaflops: 1.504
29th fastest in world (top500.org)



Collaborators





Simulation Attributes

Unstructured triangular mesh

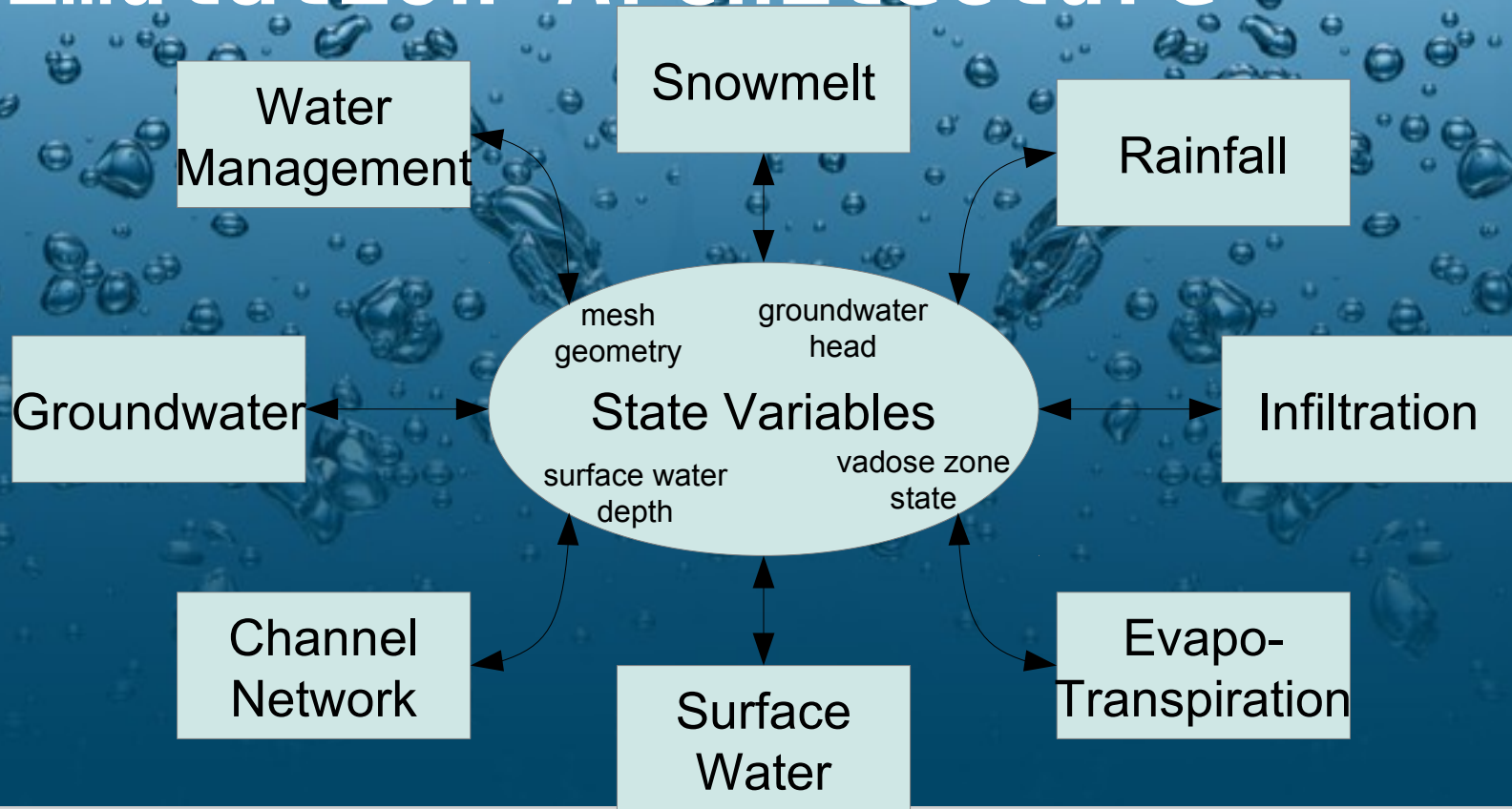
Adaptive mesh geometry

Independent swappable physics modules, some are legacy codes

“Arbitrary” water management, irrigation and reservoir releases
based on human decisions, not physics



Simulation Architecture





Reasons for Using Charm++

Partitioning

Load Balancing

Checkpointing

Easy to overlap computation and communication



Reasons for Using Charm++

Doesn't require learning advanced features of C++

Domain experts are not computer scientists, they know C, not C++

Our code uses no inheritance, no polymorphism, no iterators, operator overloading only for PUP, and templates only for three small helper functions



Charm++ Implementation

Each mesh element is a chare object (~7,000,000 for entire upper Colorado watershed)

Entire mesh is an array of chare objects

State variable store implemented in member variables of chares

Each chare stores the water it has and caches some read-only information about its neighbors



Charm++ Implementation

Physics algorithms implemented/called in methods of chares and some SDAG code

Physics algorithms proceed by rounds of messages to exchange state information, then calculation, then rounds of messages to exchange flow information, then updating state.



Charm++ Performance

We have not yet performed rigorous performance measurement

From preliminary results scaling looks good

Running with only ~100 elements per PE we were still seeing 50% of linear speedup (i.e. 500 cores ran 250 times faster than one core)



Charm++ Debugging

We have not used the Charm debugger

Documentation says it only works with net implementations and we use mpi and have to use mpi for parallel I/O

We have been doing all of our debugging by running on a single PE and using gdb

That's been sufficient so far, but not ideal

It would be nice to watch messages being sent and received and inspect message queues