Case Study: Running TauDEM on Blue Waters

James C. Phillips, Ph.D.
Senior Research Programmer, Blue Waters Project Office
Do not expect to learn any hydrology from this talk

- I have never studied hydrology.
- What little I do know was learned out of necessity.
- I only began working with TauDEM in December, 2019.

- I did spend 20+ years developing a popular, highly scalable program for biomolecular simulation despite knowing very little about cellular biology organic chemistry, or computer science.

- As a physicist by training, I assure you that water flows downhill.
Do not expect hydrologists to write highly scalable code

- They haven’t studied high-performance computing.
- What little they know they learned out of necessity.
- They likely have very little experience working with MPI.
- If you are clever and hard-working you can probably write enough MPI to parallelize your program on a desktop or small cluster.
- MPI idioms that work fine on small tests may have bad performance or even deadlock on larger production inputs.
TauDEM is a suite of hydro-conditioning tools

- Acronym of “Terrain Analysis Using Digital Elevation Models”
- Distributed by David Tarboton, Utah State University
  - http://hydrology.usu.edu/taudem/taudem5/
  - https://github.com/dtarb/TauDEM

- “Hydro-conditioning” allows rain to drain from the plain by:
  - Filling depressions (“pits”) to the level of their lowest output
  - Assigning flow directions within flat areas (such as filled pits)
Compiling is easy once you know what modules to load

- module swap PrgEnv-cray PrgEnv-gnu  # GNU tools are more standard than Cray
- module load GDAL  # geospatial I/O library
- module load PROJ  # geospatial projections library
- module swap gcc/7.3.0  # default compiler too old for C++ 11
- module load cmake/3.9.4  # default cmake too old
- module unload darshan  # disable I/O performance monitoring
- export GDAL_ROOT=$EBROOTGDAL  # EasyBuild GDAL root install directory
- export GTEST_ROOT=/u/sciteam/jphillip/gtest  # also need to build Google Test…
- export XTPE_LINK_TYPE=dynamic  # GDAL requires dynamic linking
- export CRAYPE_LINK_TYPE=dynamic  # may not need both of these, but no harm
Git and CMake are standard tools at this point

- git clone https://github.com/dtarb/TauDEM
- mkdir TauDEM/src/build
- cd TauDEM/src/build
- cmake .. -DCMAKE_INSTALL_PREFIX:PATH=/u/sciteam/jphillip/taudem
- make
- make install
TauDEM is run like any other MPI code on Blue Waters

• For example, to run on 16 nodes with 32 processes per node:
  • #PBS -l nodes=16:ppn=32:xe -l walltime=12:00:00
  • …load modules used when compiling…
  • aprun -n 512 -N 32 …
  • ~/taudem/bin/pitremove -z raw_dem -fel filled_dem > pitremove.log
  • ~/taudem/bin/d8flowdir -fel filled_dem
    -p flow_direction_grid -sd8 slope_grid > d8flowdir.log
Just because a code is parallel doesn’t mean it is scalable

- Runs out of memory on last node, more nodes doesn’t help???
  - Decomposition is 1-D, N/P lines/node, last node gets N%P extra
  - Workaround is to select P such that N%P is small
- Still takes forever, adding nodes still doesn’t help???
  - Add timing code for each iteration, print on rank 0 only.
  - Observe that iteration count increases with rank count???
  - Sweeps must propagate full map, inherently serial.
- Big performance win: Only update pixels that might change.
The Blue Waters filesystem is only fast because it is parallel

- Output takes longer than the calculation on a 1 TB/s filesystem???
  - Enabling striping on the output directory doesn’t make any difference.
  - TauDEM uses the GDAL library, which doesn’t support parallel I/O.
    - Rank 0 creates file, writes data, closes, then rank 1 writes, …
- Robust solution is to simultaneously write multiple files.
  - One file per rank is simplest up to a few thousand ranks.
    - Ideally in multiple directories to reduce metadata contention.
  - Then use GDAL VRT “virtual” file format to aggregate the files.
    - Write VRT XML directly to avoid GDAL re-opening all files on rank 0.
Conclusion: Performance improvement is a process

- If it hasn’t been tested at scale, it probably doesn’t work at scale.
- Time each stage of calculation separately, not just total runtime.
- Run experiments on a range of processor counts and input sizes.
- If part of the calculation doesn’t scale, try to understand why.

- Single file, multiple simultaneous readers works great.
- Single file, multiple simultaneous writers likely slow due to locking.
- Many files in many directories, single writer per file is nearly foolproof.
  - Up to a few thousand simultaneous writers.