# Topics of interest

What is the dominant performance bottleneck for you?

- Zach: memory access / cache misses -> see per node optimization, talk to local cluster support team
- Shawn: IO (avoid any 1d and 2d output)
- Roland: load imbalance
- Vassili: MPI communication costs even for unigrid runs
    - Check if this is MPI due to ghosts or due to Slab being called on only a few ranks
    - Similar issue exists in the PreSync branch since BC application and GZ synchronization happen one after the other

What are willing to do about it?

- Eric West willing to test optimizations, also to look into Carpet::commstate OpenMP tasks
- Deborah Ferguson willing to test optimizations
- TODO: Roland / Ian to add Eric and Deborah to Ian's benchmarking repo, add initial Readme file on how to benchmark.
- TODO: see if accounts on important clusters can be provided for this

## Requirements:

- Have a nsns benchmarking file: Shawn Rosofsky
- Have a BBH benchmarking file: https://bitbucket.org/einsteintoolkit/performanceoptimisationwg/

### How to benchmark: Timers

- Reduce IO that would dominate in short runs
- Often enough to run for 2 coarse timesteps
- Files to look at:
    - Carpet::timing has a column physical_time_per_hour
    - TimerReport gives some human readable overview. Does not count IO however.
        - Has options all_timers and all_timers_xml that produce machine readable output -> Ian's SimulationTools can read and present a nice graph
    - Carpet-timing-statistics contains all timers that Carpet knows about incl. IO and SYNC and prolongation. Ask for one file per rank!
    - Use CarpetLib::barrier = yes option which lock-steps SYNC calls
- Generic timer: clock_gettime(CLOCK_REALTIME, [output pointer])

### How to benchmark: Tracking memory accesses / cache misses

- Valgrind's (http://valgrind.org/) cachegrind/callgrind
    - Requires that code be compiled with "-g" (debugging enabled)
    - Sometimes valgrind results can be difficult to interpret if compiler does lots of function inlining, or if many external library calls
        - Useful to create static executable, then *all* function calls are reported
- Linux perf tool lets you do similar things
- Use PAPI itself
    - Talk to your local cluster's support team
- Use "roofline analysis"
    - Nice presentation from NCSA (scroll to JaeHyuk Kwack, NCSA's presentation): https://bluewaters.ncsa.illinois.edu/symposium-2018-tutorials
    - Data on CPU cost per SIMD instruction: https://software.intel.com/sites/landingpage/IntrinsicsGuide/

## How to optimize per node performance

- Peak at the HydroToyOpenMP code (https://github.com/eschnett/HydroToyOpenMP)
    - Vectorize your code either explicitly using thorn Vectors or library vecmathlib (on bitbucket) [also SLEEF http://sleef.org] or if simple enough then auto-vectorize asking for a *vectorization* report
    - Compilers only vectorize the innermost loop
    - Tile based parallelization allows you to use multiple passes over data. Tiles are typically small 16x16x16 points or so which is already 32kb of memory per double variable.
    - Second derivatives: Compute & store first derivatives, then perform first derivative on those data.