Cutter demo plan

- Start by touring `tstthrust_simplified.cc`
  - This is a mildly simplified version of the original program
    - Removed random input generation
    - Removed output validation & emission
    - Added optimization barriers so the compiler doesn't take notice
    - Computation itself is left untouched.
  - Idea: those are the kind of simplifications that can be carried out in a few minutes, not invasive modifications that affect compiler output.
- Open `-O2` version with Cutter
  - This is not the optimization level where the problem was observed
  - But it has cleaner assembly, so it's an easier introduction to the tool
- Tour the disassembly tab
  - Unlike with objdump, we…
    - Can easily pick which function we want to disassemble
    - Get syntax highlighting & some flow control analysis
  - However, there's still too much code for comfortable reading
- Open the decompiler tab
  - Official builds of Cutter integrate a version of NSA's ghidra decompiler
  - Decompilation is pattern-matching classic compiler output back into C code
  - Can be an easier introduction for those who are not used to reading asm
  - But quite slow and still requires a fair bit of post-processing
- Open the control flow graph tab
  - Explain the visual basic block graph representation
  - Show how it makes loops stand out with their backwards arrows
  - Zoom out, show how easily we locate the two compute loops
  - Zoom in on each loop in turn and discuss the difference
    - `thrust::complex` based loop gets an inline implementation
    - `std::complex` based loop calls `mulsc` libm function, which means…
      - Some function call overhead (mostly register save/reload)
      - Loss of loop optimizations, especially auto-vectorization
- Open `-O3` version with Cutter
  - Indeed, `thrust::complex` version received many extra code optimizations
  - Vectorization is probably responsible for most of the benefit here
- Keeping `-O3` version opened, open `-Ofast` version in another window
  - This time, `std::complex` version received similar code optimizations
  - In which way did `-Ofast` help here?
    - A hint is provided in the notes at the end of https://en.cppreference.com/w/cpp/numeric/complex/operator_arith3
    - Basically a result of tragic IEEE-754 error handling design decisions leading to lots of special values that require special handling
    - In particular, C++11 mandates that there be only one complex infinity (inf, 0). GCC tries to honor that standard, thrust doesn't.
    - Fast-math tells GCC to assume there will be no special float values
- So far, we've reached this conclusion via pure static analysis
  - No need to run the program, unlinke in dynamic analysis like gdb & perf
  - Can enable faster diagnosis when program is built for a special dev machine whose setup (e.g. libraries) takes time to replicate locally
- To motivate dynamic analysis, take a tour of `tstcpu.cc`
  - An intermediate stage of me optimizing the aforementioned program
  - Underwent more invasive modifications, including re/im split
  - Has no external dependency, so easy to build and run locally
- Open `-Ofast` version of `tstcpu.cc` with Cutter
  - Show that GCC split the computation into a vectorized and scalar part
  - Use debugger to show that the vectorized part is used and that program stays a long time in that loop -> no significant overhead!