TECH-X CORPORATION

Challenges in Deployability: Scientific Software on Microsoft Windows

B. M. Cowan, *Tech-X Corporation*

Motivation: Industry participation

- Building a broad base of users and developers for community software is important for sustainability
- Industrial—commercial software developer—contribution to community software can have significant benefits:
  - Can bring developer resources
  - Increases adoption of community software: "Success stories" for further funding
  - Commercial software generally has stricter requirements on robustness and usability than academic software—so commercial developers will contribute to testing

Needs of commercial software

Commercial scientific software requires not just performance portability, but *deployability*: Software must be able to be easily installed, and perform well, on a wide range of customer hardware, without the developer having access to the hardware or even knowing its configuration beforehand. As a commercial scientific software developer:

- We can't assume that the customer
  - Can build software
  - Can install dependencies
  - Can manage drivers/system software
  - "I don't have administrator privileges on my computer." –Magnet engineer at national lab partner
- So we have to
  - Provide installation in user space via installer or tarball
  - Have software perform well on customer machine without access to it
  - Support Windows

Why Windows?

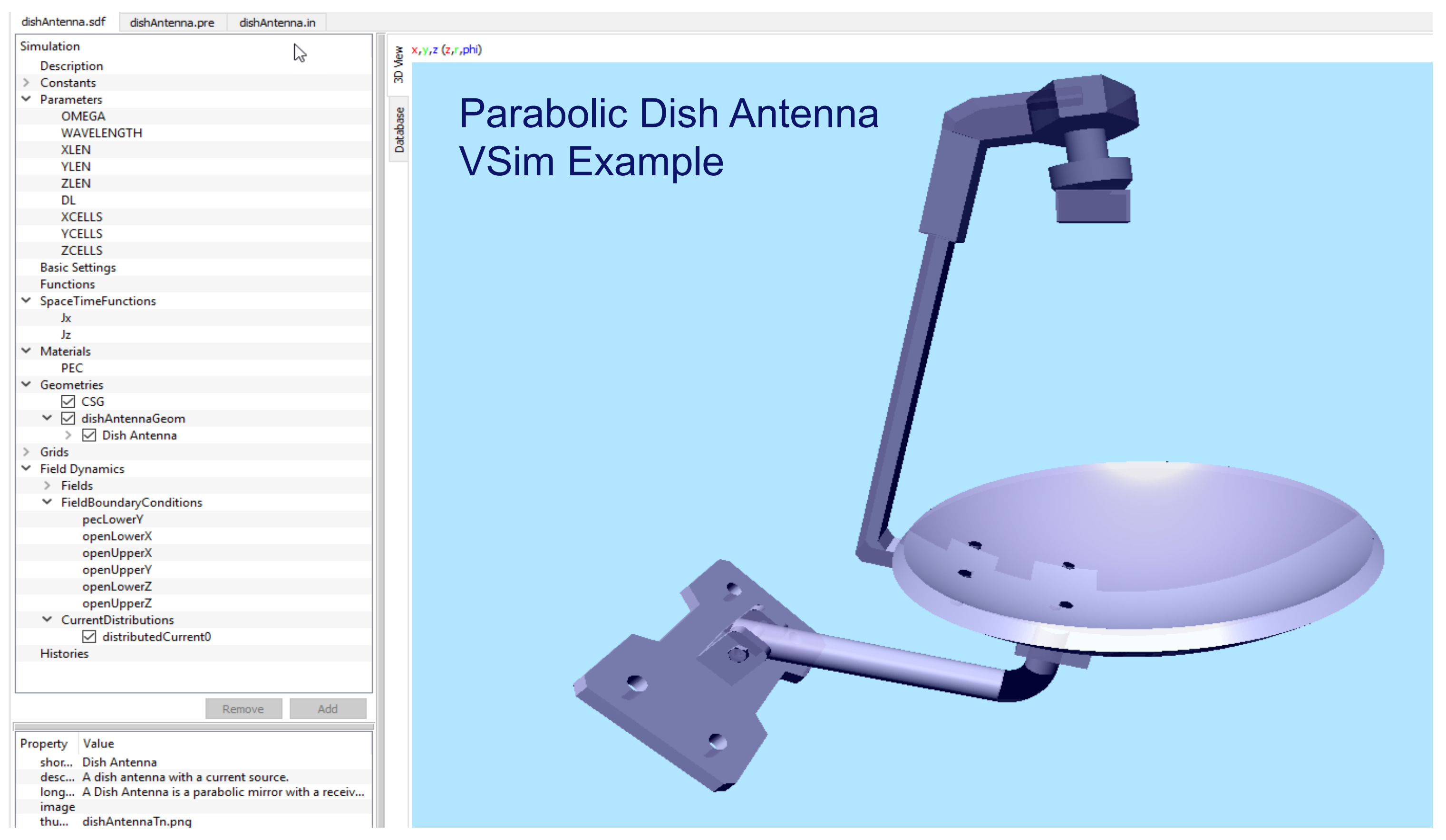
- That's where the market is
  - HPC users might be big customers
  - But there are far more potential users who are engineers with Windows boxes on their desks
- Containers/VMs not (currently) deployable enough that we can just ship Linux software

Our perspective

- Tech-X develops and markets several commercial scientific software codes

VSim Electromagnetics and Plasma Simulation
RSim Radiation Transport Modeling
USim Fluid Plasma Modeling
PSim Complex Block Copolymer Modeling

- Software provides unique physics and computational capability
- Works on multiple platforms, scales from laptops to supercomputers
- GUI that integrates problem setup, execution, visualization and analysis



- Application areas (for VSim):
  - Antennas; Waveguides
  - Microwave devices (e.g. klystrons, traveling-wave tubes)
  - Magnetron sputtering; RF-driven plasmas for semiconductor processing
  - Optical fibers; Silicon photonics
  - Ion thrusters
  - Plasma-based particle accelerators
- Adding performance portability with CUDA GPUs and vector instructions on CPUs
- Closed-source...
- ...But we use, and contribute to, community software:
  - Trilinos: Linear algebra/solvers
    - Also SuperLU, HYPRE
  - VisIt: Embedded visualization
  - HDF5: I/O
  - CMake: Build system generator

Observations about Windows

Scripting environment

Windows' native scripting environment is DOS, which is fundamentally different from Unix shells

- Paths, command-line argument conventions are different
- Few scientific software developers are conversant in DOS
- But some required Windows development tools adhere to DOS conventions

There are some ways around this:

- Cygwin provides Windows executables that mimic standard Unix equivalents
  - Start in bash shell
  - Some tools can use Unix or Windows path conventions, and convert between them
  - Environment variables set from Windows environment and visible in Windows programs
- The Windows Subsystem for Linux (WSL) provides a complete Linux distribution (e.g. Ubuntu) within Windows 10
  - Can run Windows executables from within Linux environment
  - But programs not necessarily WSL-aware: For instance, CMake for Linux running in WSL assumes Unix-style command-line arguments, even for Visual C++ compiler for Windows

Compilers

Compiler must be able to generate Windows code (except for build-only dependencies). There are several options:

- Microsoft Visual C/C++ (MSVC)
  - Generally lags behind other compilers in support for HPC features (e.g. OpenMP), but latest MSVC 2019 is an improvement
  - Only supported CUDA host compiler for `nvcc` on Windows
  - Required for GUI code (e.g. Qt)
  - Basic command-line arguments don't conform to conventions of normal Unix compilers—so most Linux build tools won't work, even under WSL
- LLVM Clang
  - More Unix-like
  - Also has `clang-cl` executable that uses MSVC command-line argument syntax

Build systems

- Modern CMake (target-based dependencies, CUDA-as-language, etc.) works really well
  - No special cases for Windows needed, even in large mixed C++/CUDA code base
  - But lots of legacy CMake code out there, and updating is often an all-or-nothing affair
  - Still evolving, especially in CUDA features
- Autotools: Not really an option
  - Doesn't work with MSVC-style command-lines
  - On Cygwin, links with Cygwin environment, which is GPL, so can't be linked to commercial software
- MSVC cumbersome in Unix-like environments
  - Uses `nmake` and `jom` instead of `make`
  - Requires execution of a DOS batch script to set up environment
  - We kluge this for our bash-based package management system; also needed for Spack