Tech-X Corporation

Considerations for performance portability in a commercial particle-in-cell code

B. M. Cowan, S. Averkin, J. R. Cary, J. Leddy, S. W. Sides, and I. Zilberter, *Tech-X Corporation*

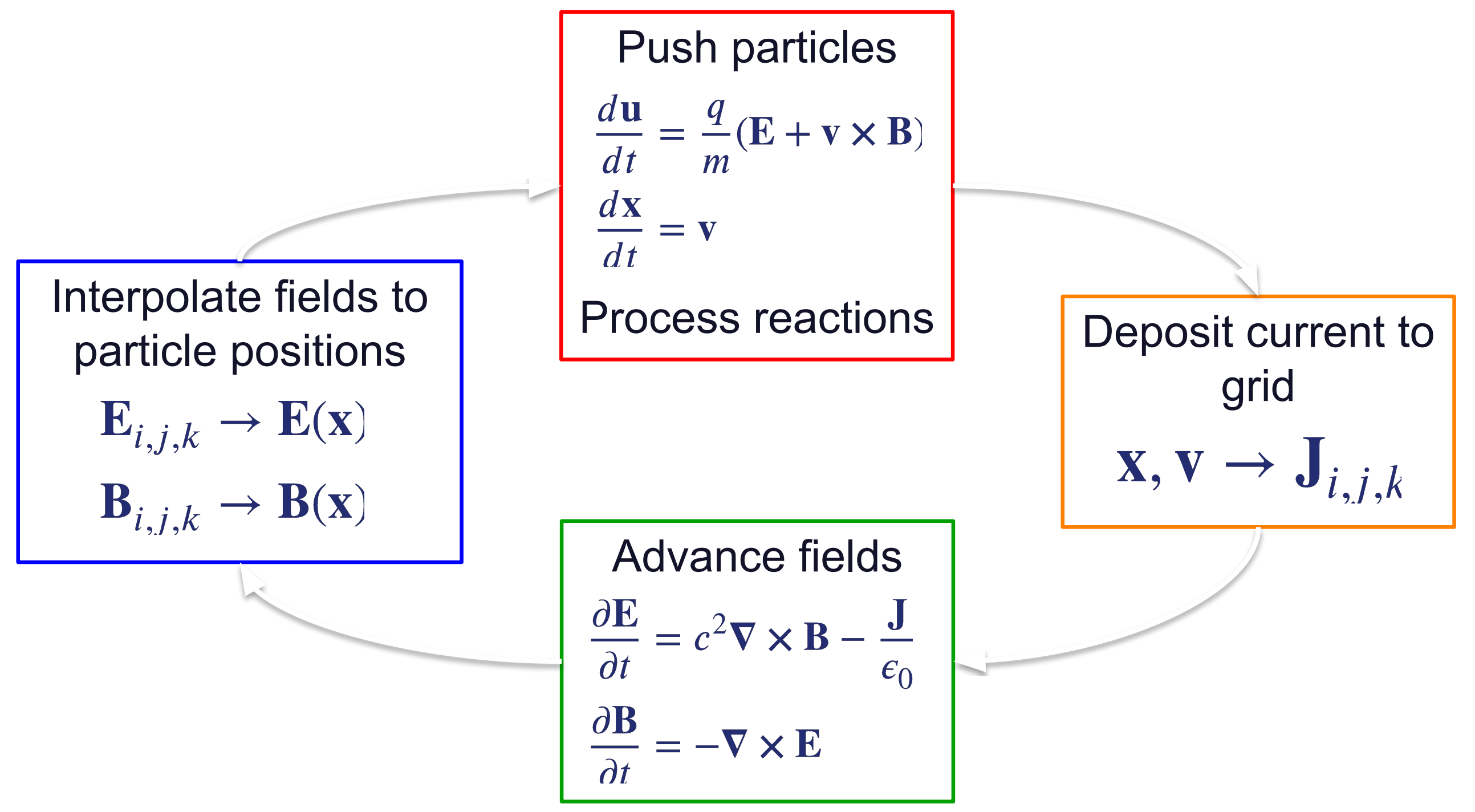
Abstract

We describe the development of performance portability features in the commercial multiphysics particle-in-cell code VSim. VSim was designed nearly 20 years ago (originally as Vorpal), with its first applications roughly four years later. Using object-oriented methods, VSim was designed to allow runtime selection from multiple field solvers, particle dynamics, and reactions. It has been successful in modeling for many areas of physics, including fusion plasmas, particle accelerators, microwave devices, and RF and dielectric structures. Now it is critical to move to exascale systems, with their compute accelerator architectures, massive threading, and advanced instruction sets. Here we discuss how we are moving this complex, multiphysics computational application to the new computing paradigm, and how it is done in a way that keeps the application producing physics during the move. We present performance results showing significant speedups in all parts of the PIC loop, including field updates, particle pushes, and reactions.

We also describe considerations particular to commercial codes, most significantly the need to support Microsoft Windows. Customers of commercial scientific software often use Windows workstations on their desks, so the software must be built to run natively on Windows. The challenge here is that the Windows build environment is markedly different from those on Unix-like systems. Many packages require Microsoft Visual Studio—it is the only supported host compiler for CUDA, for instance—which has different semantics than commonly used compilers on other platforms. It also tends to lag in performance features such as OpenMP, necessitating the use of mixed-compiler toolchains. In addition, the Windows scripting environment and path name conventions are different, complicating package management scripts. Since VSim relies on community codes for some of its features, we have encountered these issues in our efforts to maintain Windows compatibility for our toolchain. We have also explored the possibility of bringing Windows support to the performance portability library Kokkos, and report on some of the issues therein.

Background: Particle-in-cell

- Lorentz force interpolated from gridded fields
- Currents deposited to grid from particles


Push particles
$\frac{d\mathbf{u}}{dt} = \frac{q}{m}(\mathbf{E} + \mathbf{v} \times \mathbf{B})$
$\frac{d\mathbf{x}}{dt} = \mathbf{v}$
Process reactions
Deposit current to grid
$\mathbf{x}, \mathbf{v} \rightarrow \mathbf{J}_{i,j,k}$
Advance fields
$\frac{\partial \mathbf{E}}{\partial t} = c^2 \nabla \times \mathbf{B} - \frac{\mathbf{J}}{\epsilon_0}$
$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \mathbf{E}$
Interpolate fields to particle positions
$\mathbf{E}_{i,j,k} \rightarrow \mathbf{E}(\mathbf{x})$
$\mathbf{B}_{i,j,k} \rightarrow \mathbf{B}(\mathbf{x})$

Challenges of PIC on GPUs and many-core CPUs:

- Field update can be straightforward: Each thread/vector lane updates a cell
  - But naïve approach does not optimize for memory bandwidth
- Field interpolation and current deposition present problems
  - It's not known a priori which cells particles occupy and hence which field values are needed
  - Naïve one-particle-per-thread memory accesses won't be coalesced
  - Deposition may also experience race conditions: Multiple threads try to write the same current value
- Flexible multiphysics code requires modularity, but modern architectures complicate fine-grained runtime polymorphism

General approaches

- Take advantage of modern hardware
  - Graphics processing units (GPUs) offer tremendous computational performance
  - Much greater processing capability per monetary and power cost
  - Achieved through massive parallelism
  - CPU performance increasing through core count, vector instructions
- Write performance-portable code
  - Write main computational procedures (e.g. cell field update, particle push) in functions that can be executed on both host and device
  - On CPU, function will be executed in a (vectorized) loop
  - On GPU, function will be executed by a thread
- Performance frameworks
  - Use CUDA on GPU
  - Use OpenMP on CPU for multithreading and to trigger generation of SIMD instructions
  - Exploring performance portability libraries (e.g. Kokkos)
- Maintain multiphysics features
  - Design main management routines—*dispatchers*—to work with multiple algorithm variants
  - Different types of field updates, e.g. absorbing boundaries, controlled dispersion
  - High-order particles: Complicates memory management
  - Other physics: Metallic boundaries, dielectric materials, reactions, cut cells…
- Transition to new infrastructure
  - Start with deep infrastructure: Grid objects, multidimensional arrays
  - Proceed with dependent features: fields, particles, collisions
  - Code in new infrastructure encapsulated in separate performance library
  - Interfaces to performance library added to VSim code base

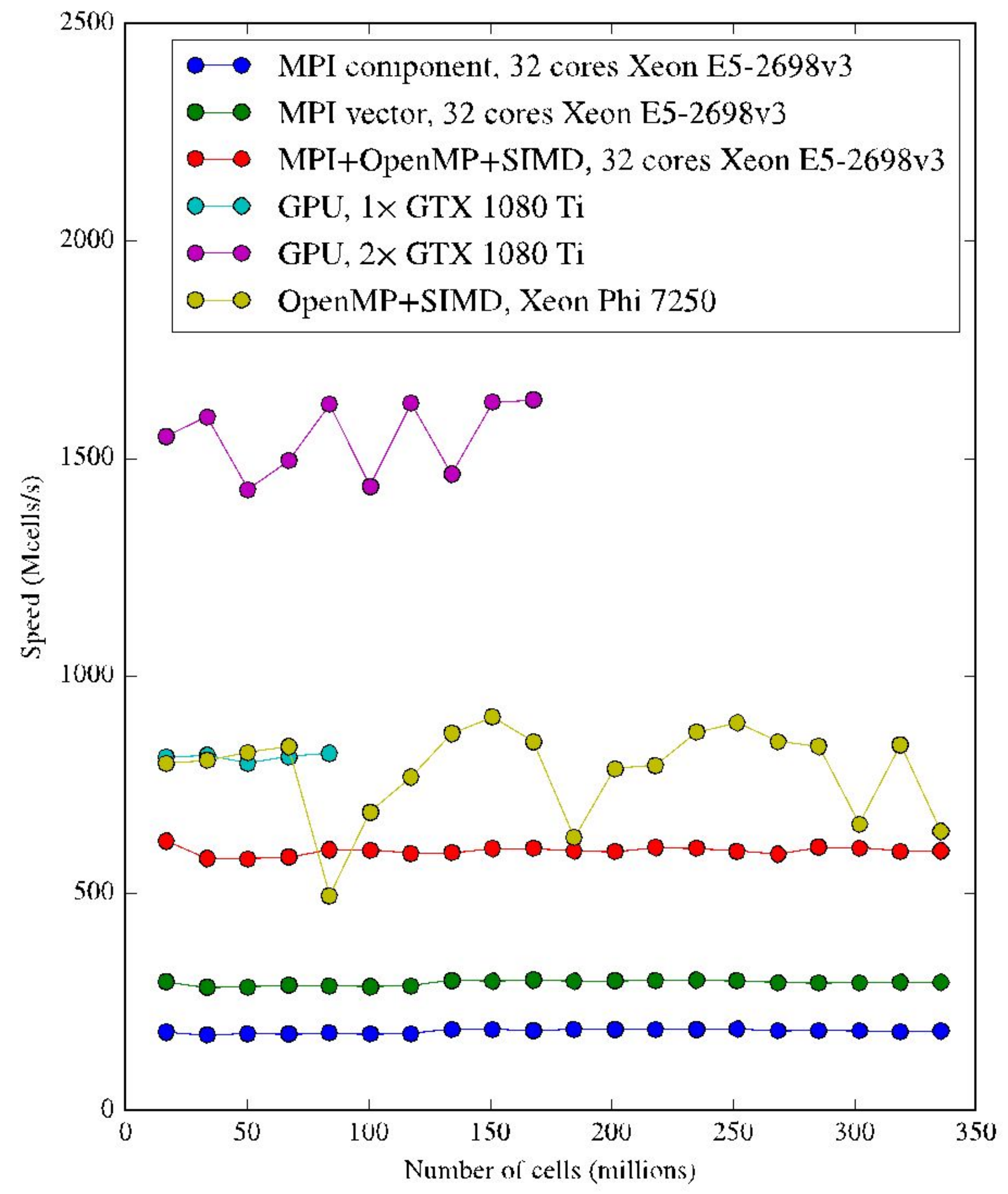
Needs of commercial software

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:

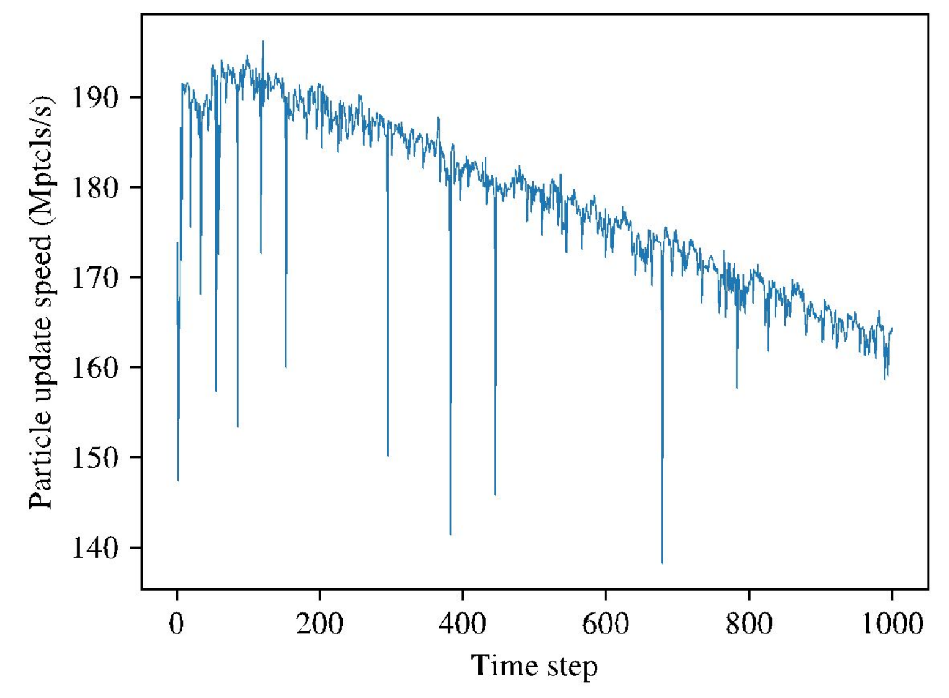
- We can't assume that the customer
  - Can build software
  - Can install dependencies
  - Can manage drivers/system software
  - "I don't have administrator privileges on my computer." –Magnet engineer at national lab partner
- So we have to
  - Provide installation in user space via installer or tarball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

Results

Raw FDTD sees 4 Gcells/s on an NVIDIA GeForce GTX 1080 Ti GPU in single precision. FDTD with embedded boundary dielectrics and absorbing boundaries: Speedup of 2× Cori Haswell node results from software improvement alone. Additional speedup obtained on GPUs and KNL. These developments enable highly performant photonics simulations.



Particle update on the GPU gets nearly 200 M particles/s. This shows a simulation of a hot plasma, with slight decrease in speed as the particle layout loses regularity.



Simulation time for a simulation of Rayleigh-Taylor instability with collisions.

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

2020 Performance, Portability, and Productivity in HPC Forum, Sep. 1–2, 2020

Support provided by U.S. Defense Advanced Research Projects Agency Contract W31P4Q-16-C-0009 (SBIR).
Vorpal simulations used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. DOE under Contract DE-AC02-05CH11231.

DARPA NERSC