Our Road to Exascale: Particle Accelerator & Laser-Plasma Modeling

Date: March 15, 2023

Presented by: Axel Huebl (Lawrence Berkeley National Laboratory)

(The slides are available under “Materials from the Webinar” in the above link.)

Q. With such an open development model, and so many users, how do you handle questions / support (and the time / staffing that takes)?

A. We try to include the community in support and Q&A as well. The most important part for this is to create open, easily accessible communication channels for everyone: GitHub issues, open chat like Gitter, etc. Only after that, we add a more internal layer like Slack chat and developer meetings - and are very easy going in also adding contributors there.

For development, we also have an understanding that if you contribute a feature to WarpX and we merge it to mainline, you will also have documented it and developed tests for the feature, so that it can be maintained long term, even without you. And that contributors are responsive when there are follow-up questions/issues on their work. We have a great community and people fully understand this. The workflow is to make many small-medium contributions, which can be individually reviewed and tested, instead of one big code change [as an afterthought at the end of a PhD/postdoc/paper].

But yes, open source community maintenance will need a core team of committed professionals to hold it together. This is something that communities need to fund. As an example, if open source research software engineering & support enables your science, then include it in the reports of your science domain's strategic planning activities and actively communicate the benefits of open science & open source to your colleagues, students, superiors, the public and funding agencies.

Q. How did you improve on-boarding?

A. We improved onboarding by first assessing it via https://rateyourproject.org . This is super easy, well-made and fun. We then discussed in ECP IDEAS productivity meetings and came up with a simple task list. We started this as an interactive google calc/docs that we share with the onboarded person. Columns are status progress (todo/ongoing/blocked/done) - task name - time needed - objective - description - references, such as tutorials or cheat sheets.

That way, we make sure everyone has the same starting conditions and can close gaps, e.g., technical "How did I use GitHub again?" or cultural "What is the best way to ask for help?" and "Where is the hybrid hallway/lunch break for hallway chatter?".

We improve this sheet periodically, add sections, materials people liked, etc. We skip some software engineering items for people that are primarily on the code "user" side and skip some physics deep dives for people primarily on the applied math or computer science side.

The cool thing about this documented workflow is that this scales the possible mentors to more team members - senior personnel can delegate and nothing will be forgotten. Typical tasks include adding to teams, collecting a picture for team slides, adding to events & meetings, creating HPC accounts, running a first example together, doing a first development cycle on a task, etc.

Q. How do you pay for ongoing computing power needed just for the download/install CI tests?

A. We develop fully in the open and are thus eligible for the usual free resources on GitHub actions and other CI cloud providers.

Additionally, DOE facilities like ORNL and NERSC provide now GitLab instances that can be used (usually post-merge, not pre-merge in a PR, which would be even better). We need to use this more and figure out a way to run this already on pull requests in a safe way.

Q. What is your approach to performance testing? We cannot run performance tests on supercomputers the way we do CI. How often do you do performance testing of your code?

A. Two approaches. For ECP, we did regular full scale tests of our key performance metrics (KPP) on full scale machines. At least every few months (see our 2022 Gordon Bell paper at SC22 for the table). This is a super interesting metric across machine evolution (even in the same system) and code evolution. Additionally, we start to do systematic single-node runs to track regressions in kernels, e.g., register pressure from newly added features or compiler releases. I wish we were further advanced there and could reuse something instead of scripting it ourselves.

Q. You said that plasma driven particle accelerators can work on the millimeter scale, right? What kind of electric field gradient can it achieve? In the PIC model, how big is a cell in this case? Will exascale computers be needed?

A. Thank you, this is a very exciting question. In conventional accelerators one can achieve about 100-200 MV/m electric fields for acceleration. Above that, they essentially short-cut and create electron cascades that break down the cavities (each about 0.2-0.5m in size). In plasmas, we can create accelerating fields that are three to four orders of magnitude higher, by creating accelerating geometries with laser pulses or particle beams. This is exciting, because it carries the potential to shrink accelerating

lengths proportionally if we can control the significantly smaller spatial and time scale in plasmas sufficiently (10s of microns in structure size and smaller).

Our simulation resolutions depend on the regime, especially the plasma density at play. For wakefield modeling, which uses low densities similar to gases, we need to resolve usually around 10 to 100nm or above and can apply a few tricks such as the boosted frame method in WarpX, to go in a relativistic reference frame that requires ideal resolution. For acceleration of protons and ions from laser-solid interactions as well as laser-plasma interaction for inertial confinement fusion-relevant scenarios, we need to go way down in cell size, due to the high plasma density. There we talk a few nm and below in cells. High-fidelity modeling of solid-density-like plasma elements, as we did in the Gordon Bell run for our particle injection, is truly heroic in scale. Exascale is just about to give a first foothold into modeling those in full geometry (3D).

Q. Your Gordon Bell prize included Fugaku / A64FX - how did you get support to port/optimize on that?

A. In ECP, we focus on DOE leadership machines, which are all GPUs. We now support three GPU vendors (Nvidia, AMD, Intel) and had HPC machines with two of them available in 2022. For Fugaku, we wanted to really stress-test our performance portability layer and see how much we can achieve with manual tuning. So, we teamed up with colleagues and companies in France (CEA, Atos, Arm) and RIKEN to write a tuned backend for A64FX. The experience we gained with that is now informing what we can improve for CPUs in our performance portability layer. First changes for instance are better generic data structures for particles to enable easier automatic vectorization.

Q. After going through this process of building WarpX are there features/abilities of other performance-portability layers (e.g. Kokkos) which you are now thinking are missing in AMReX?

A. AMReX provides significantly more capabilities to develop portable, block-structured algorithms & applications than the performance portability layer, so focusing only at the kernel generation and performance primitives is always a very interesting aspect for us. We are generally very happy with the domain-specific abstraction AMReX adds on top of the "pure, portable kernels" + local data containers, which Kokkos features. One aspect I personally would like to reuse is the *hierarchical parallelism* concept in Kokkos performance primitives, because it makes cache blocking algorithms easier to implement.

Looking ahead, Kokkos aims to standardize as much as possible in ISO C++, which is fantastic. If we have the funding and time then it would be definitely interesting to implement a Kokkos or ISO C++-next backend in AMReX as well. For now, we want to *move fast and break things*, so we abstract AMReX performance primitives in C++ via "backends" of CUDA/HIP/SYCL/OMP and use lambdas to generate user kernels, similar to how Kokkos is implemented.

Q. Is Fortran 90 the current standard and is it still fast? Is C++ better for some part of the code (e.g. garbage collection and memory)

A. The Fortran ecosystem is evolving as well, but not at prime speed. The industry moved to support C++ first, standardize rapidly in 3-year cycles, and go for other languages next. When we switched from Fortran to C++ for our core kernels in 2019, there was essentially *one* compiler left that could compile our code for Fortran GPU experiments (and it was broken).
It has been shown numerous times that C++ code can be as performant as Fortran code, e.g., when using the (non-standard, sigh) *restrict* keywords and preferring compile-time polymorphism over runtime polymorphism when designing code.

C++ code is super easy to integrate with other languages, e.g., with Python, which we also use extensively for productivity and glue code. (I am a pybind11 co-maintainer.) C++ has no automatic garbage collection like, e.g., Python and Java, but it has a lot of smart containers & pointers & zero-copy views that can clean-up after themselves, exactly when you expect it.

In the end, the huge ecosystem, multi-compiler support, tooling, highly active open standardization, zero-cost composability & modularity where needed, and industry momentum make C++ our top choice. As a full disclaimer, I learned C++ through hard-core template meta-programming, so take my opinion with a grain of salt.

Q. What is the criterion for AMR?

A. At the moment, we are actively researching how to do mesh-refinement in electro-magnetic PIC. This is unique and quite challenging, due to relativistic field retardation effects, moving particles, etc. We are excited to already have a good class of scenarios as the one in Gordon Bell, where we can show that we can refine and save a lot of time.We are not doing the automated, adaptive part yet. Likely criterias for adaptive refinement would be density and field gradients, but due to the long-range potentials in plasmas this will be a fun topic to explore. Maybe predictor-corrector steps when refining, we will see.