Accelerating Kinetic Low-Temperature Plasma Simulations via *OpenACC*

Andrew Tasman Powis[1,2], Johan Carlsson[2], Stéphane Ethier[2], Alex Khaneles[2], Arjun Agarwal[2], Igor D. Kaganovich[2]

[1]*Princeton University, Princeton, New Jersey*

[2]*Princeton Plasma Physics Laboratory, Princeton, New Jersey*

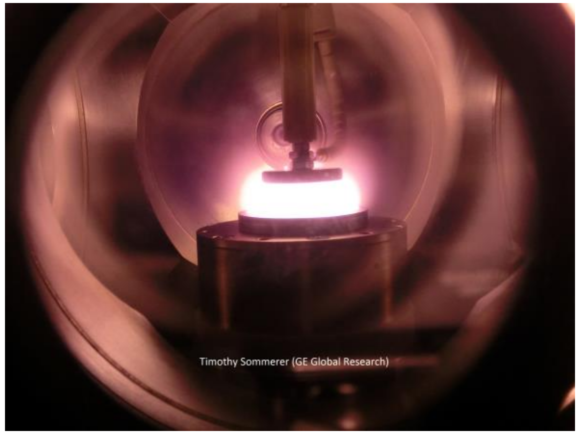
Low-Temperature Plasmas

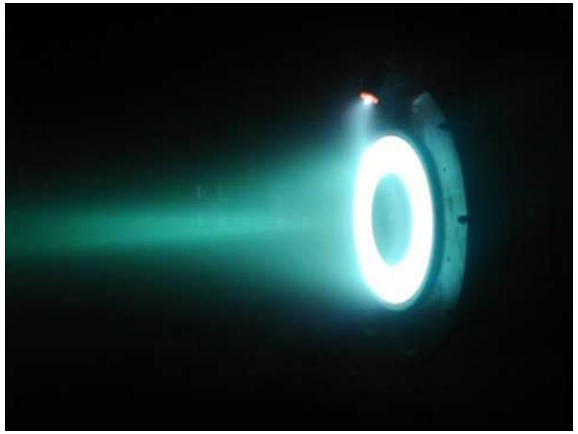
Low-Temperature Plasmas (LTPs) are ubiquitous in industrial applications of plasma physics:

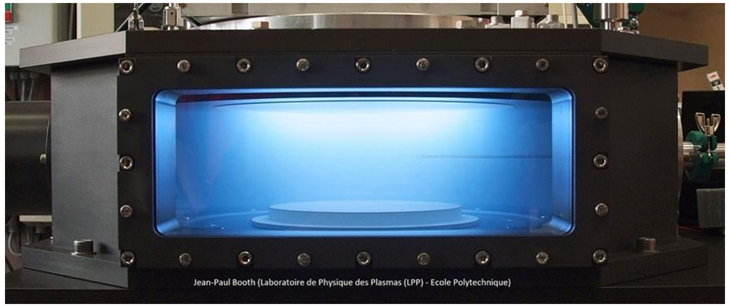
- Materials processing (e.g. silicon etching)
- Power transmission
- Spacecraft propulsion

They also exhibit complex dynamical phenomena:

- Collective behavior
- Long and short range forces
- Non-equilibrium and kinetic species
- Interfaces with solids/liquids
- Chemical and biological interactions





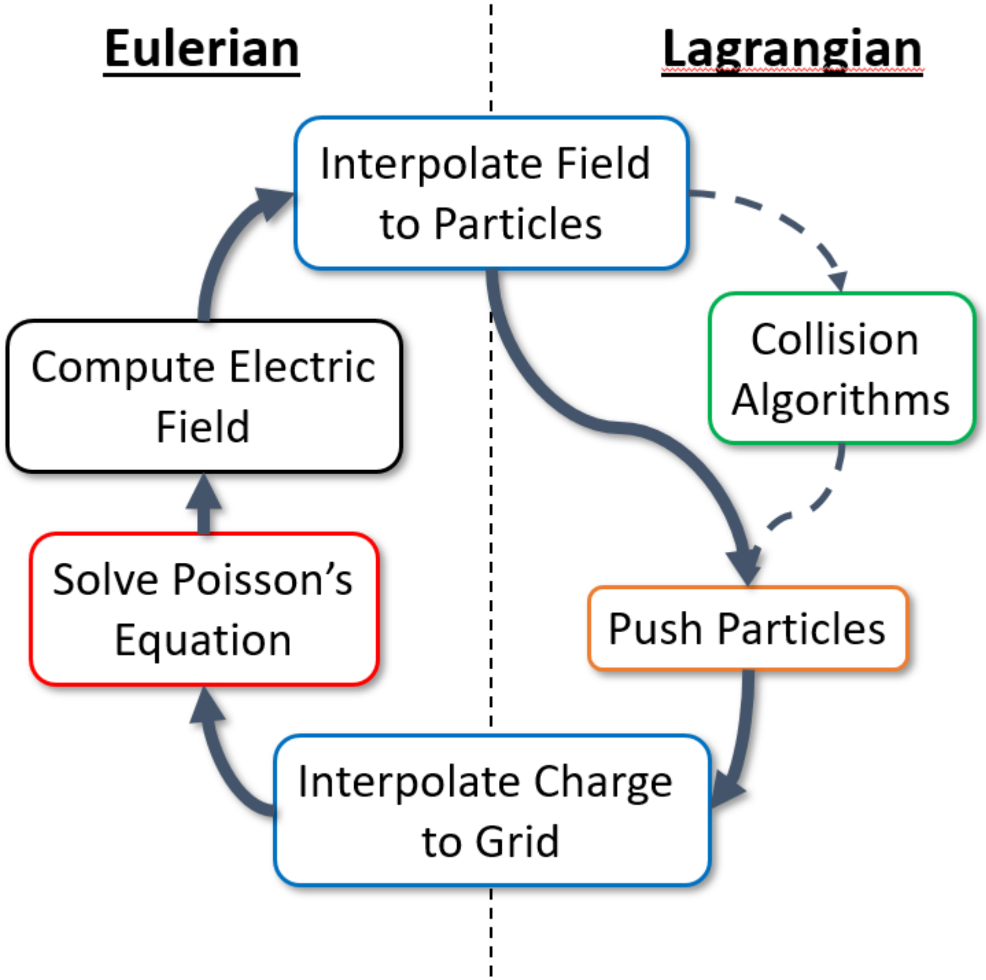


Top left: An experimental plasma switch (General Electric)
Top right: A hall thruster firing in a test chamber (PPPL)
Bottom: A plasma reactor for material processing (Ecole Polytechnique)

Code Overview

- The code, known as *Low-Temperature Plasma Particle-in-Cell* (LTP-PIC), is designed to enable laboratory science and industrial design of low-temperature plasma devices
- Modeling real industrial systems, of kinetic plasmas (i.e. six-dimensional) puts an impetus on ***performance***
- The desire to provide open access to students, researchers, as well as cater to industry puts an impetus on ***portability***

Code Design



- A 2D-3V Particle-in-Cell (PIC) code for modeling low-temperature plasmas (with plans for 3D)
- PIC is a mixed Eulerian/Lagrangian framework which reduces the cost of discretizing 6D phase space
- LTP-PIC is designed from the ground up for scalability
- It is accelerated via *MPI+OpenMP*
- It is coupled with the *Hypre* package for linear algebra

2020 Princeton GPU Hackathon

Team Members

Tasman Powis
Princeton U.

Johan Carlsson
Radiasoft LLC

Alex Khaneles
PPPL

Arjun Agarwal
PPPL

Mentors

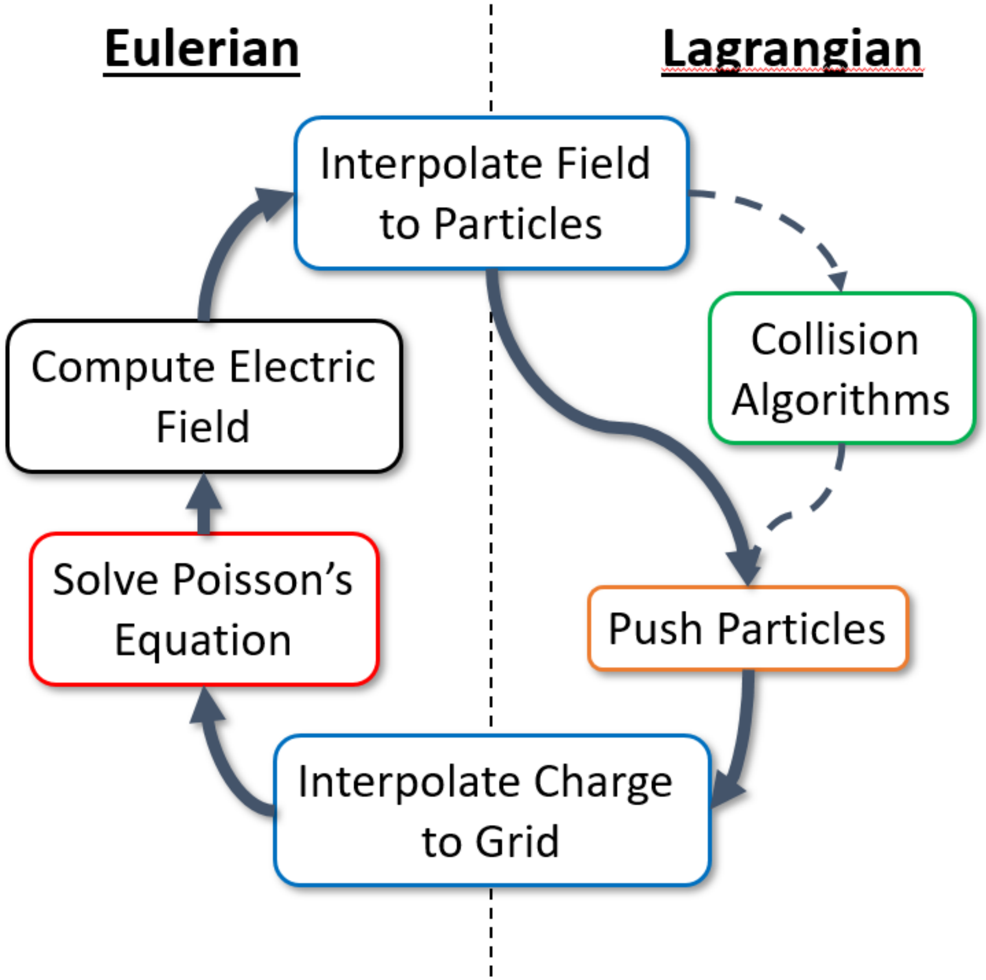
Stéphane Ethier
PPPL

Mathew Colgrove
NVIDIA

Mozhgan Chimeh
NVIDIA

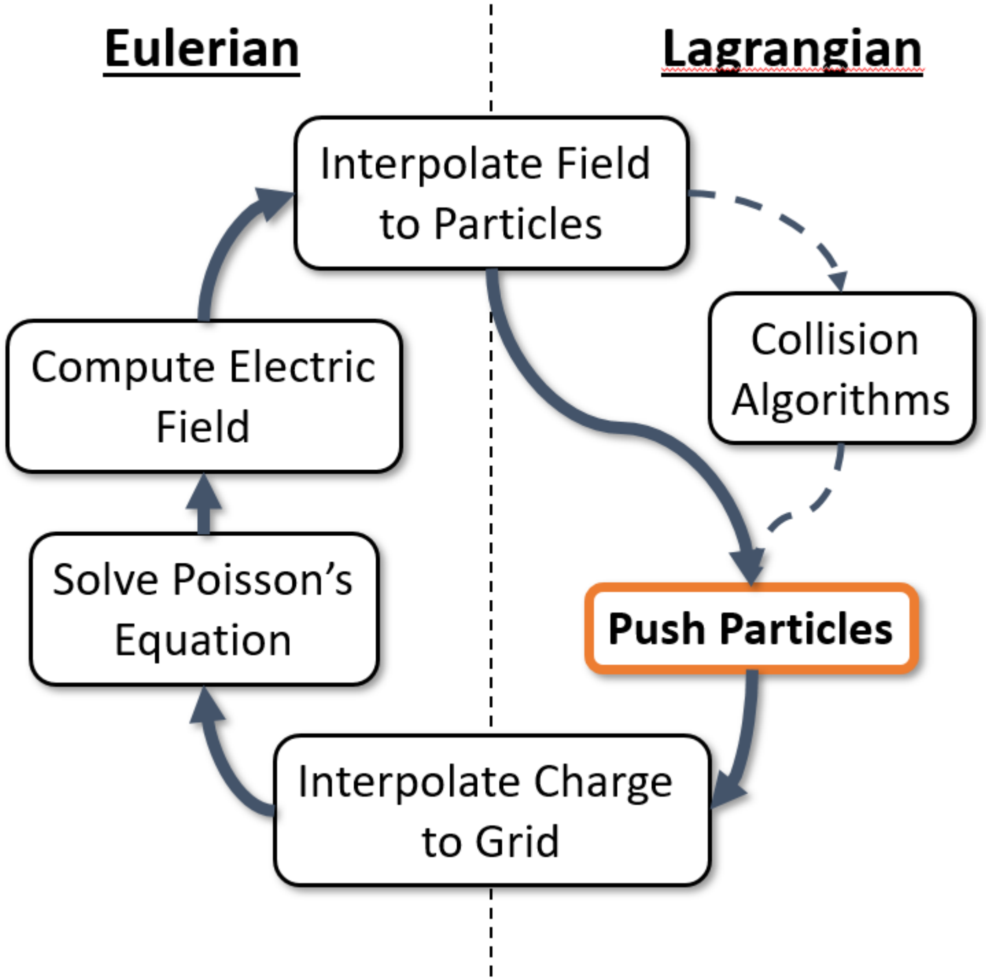
Acceleration Targets

Acceleration Targets



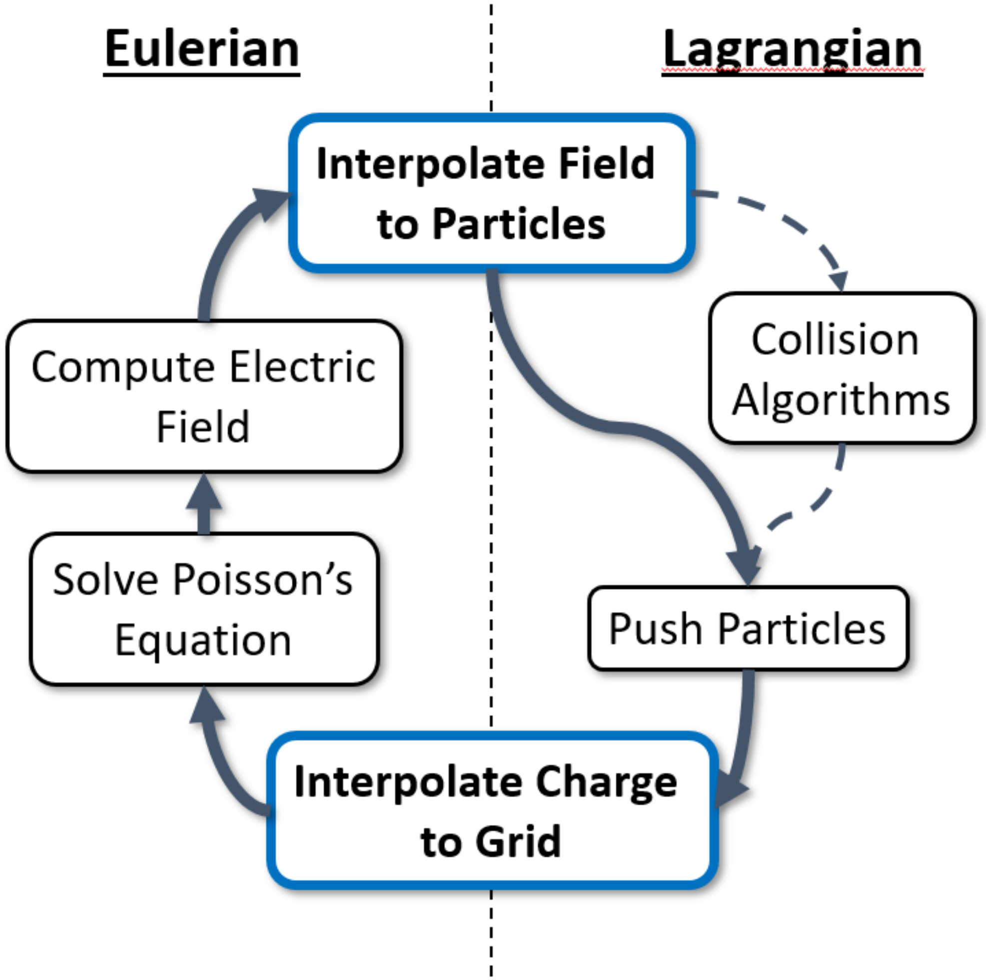
- Nearly entire code!
- We load all large memory structures on to the GPU
- Some functions remain on CPU:
  - Field solver can optionally remain on the CPU
  - Particle boundary communication
  - Diagnostic I/O

Particle Push



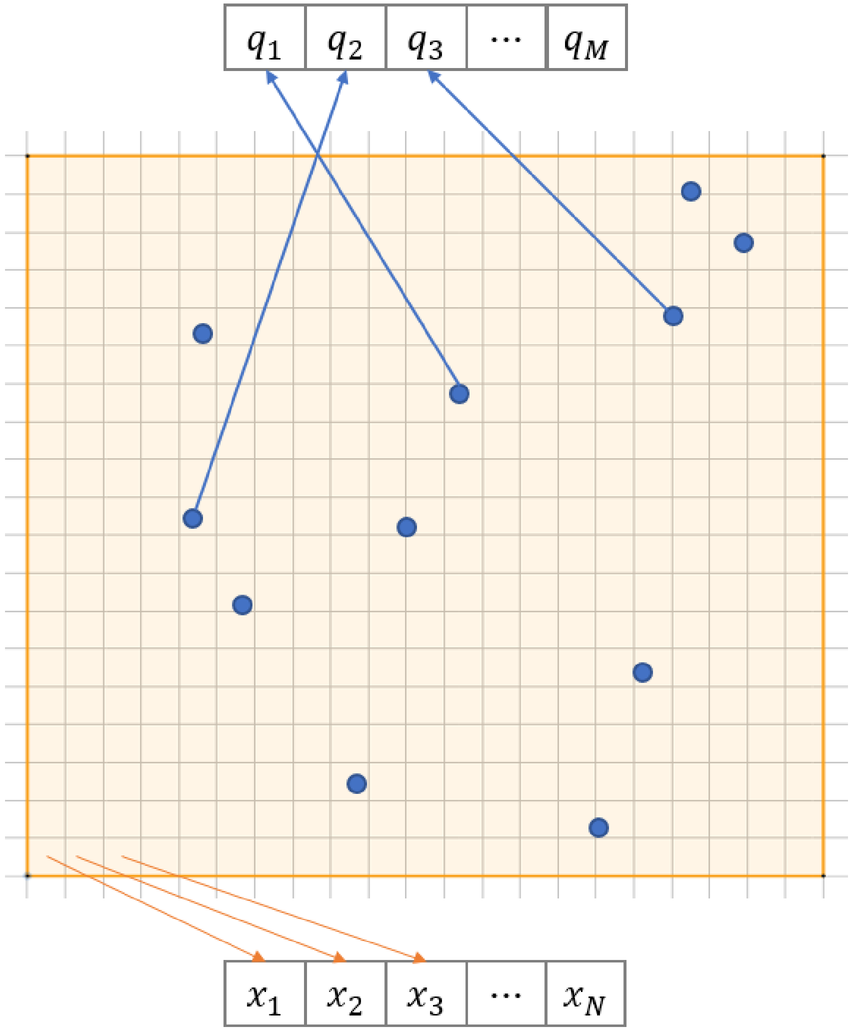
- Embarrassingly parallel
- ~100x speedup
- A large speedup, but it was expected

Interpolation



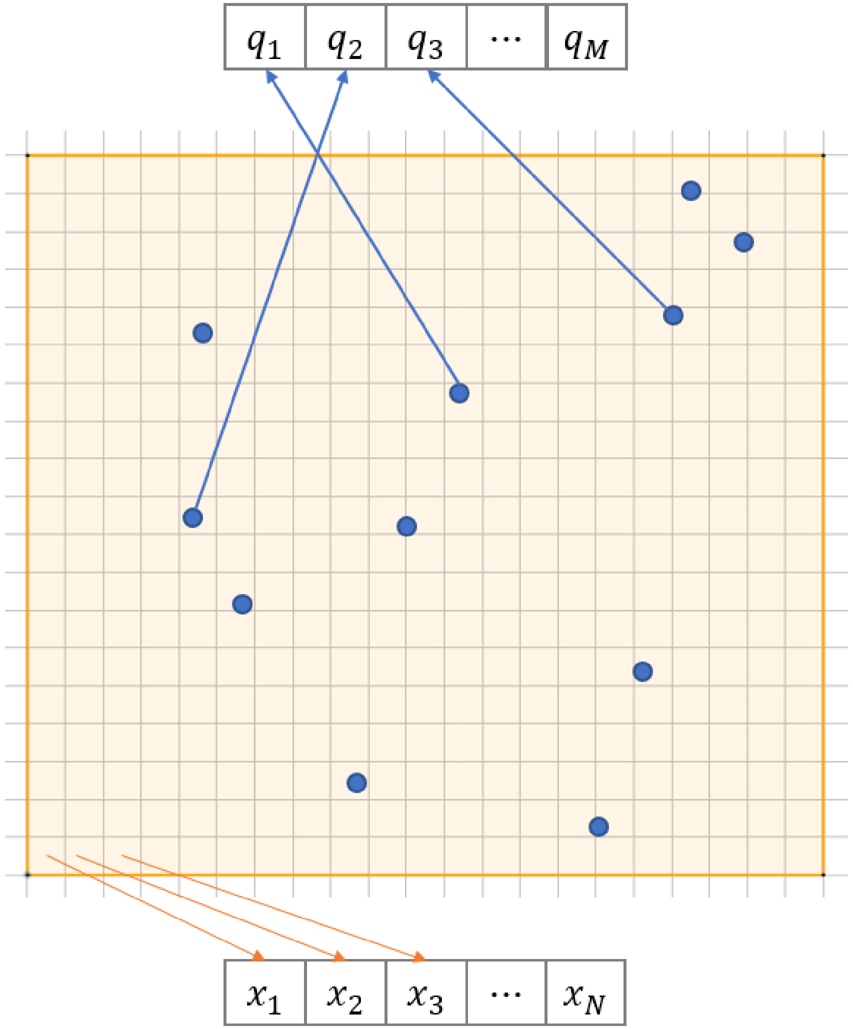
- Nearly embarrassingly parallel except:

Interpolation



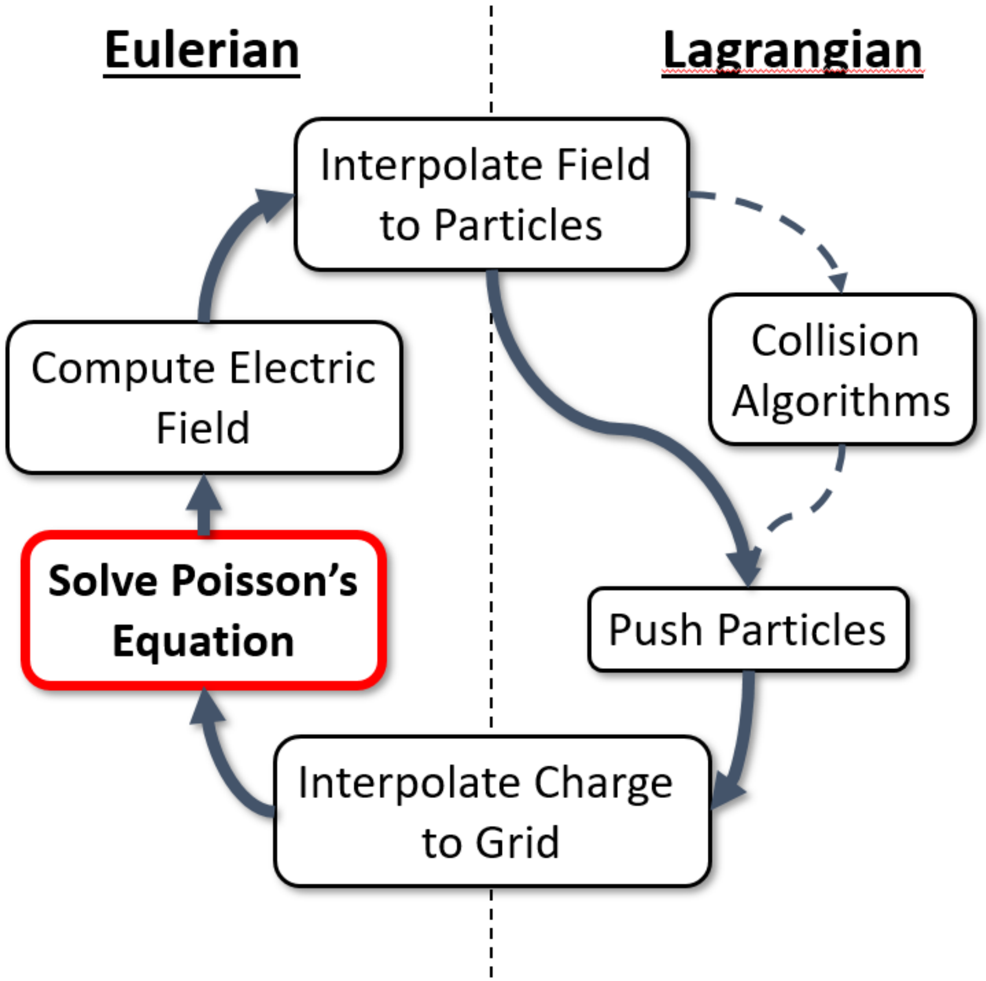
- Nearly embarrassingly parallel except:
  - Requires non-uniform random memory access
  - Requires atomic memory access

Interpolation



- Nearly embarrassingly parallel except:
  - Requires non-uniform random memory access
  - Requires atomic memory access
- 100-200x speedup
- Performed better than expected on the GPU!
- Due to low memory latency?

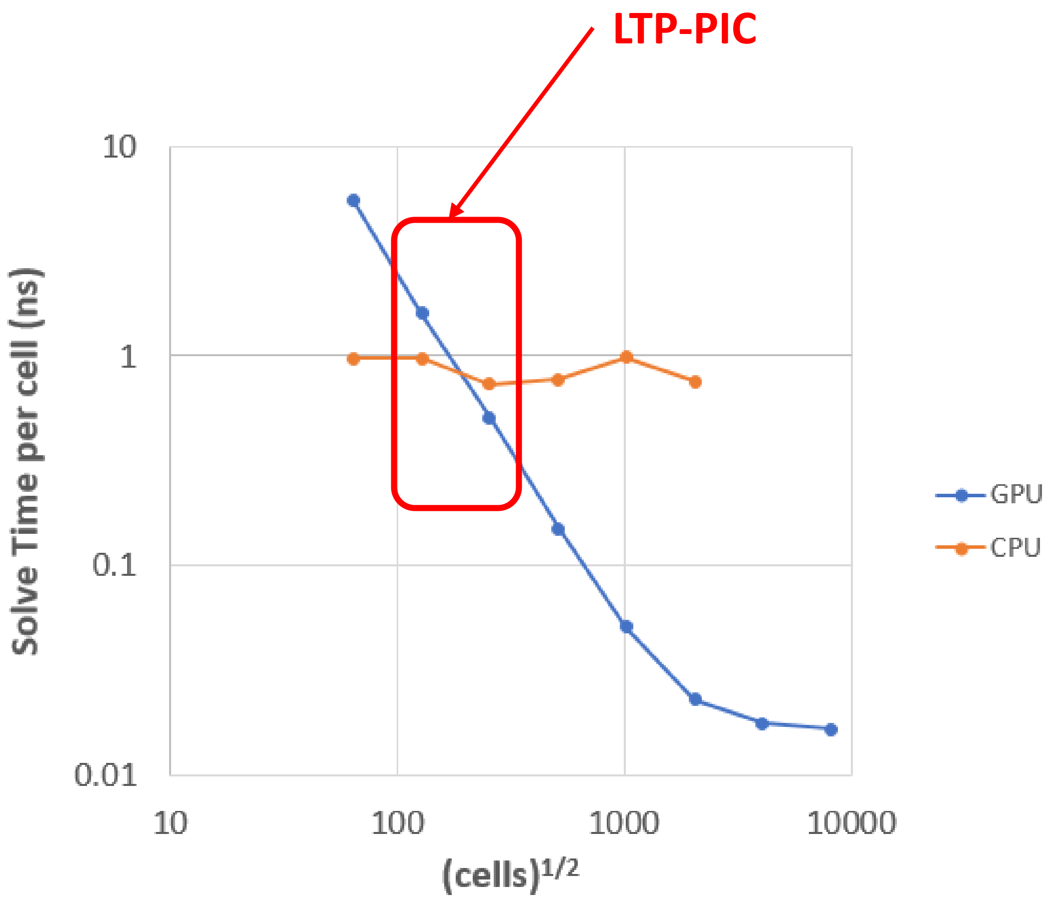
Field Solver



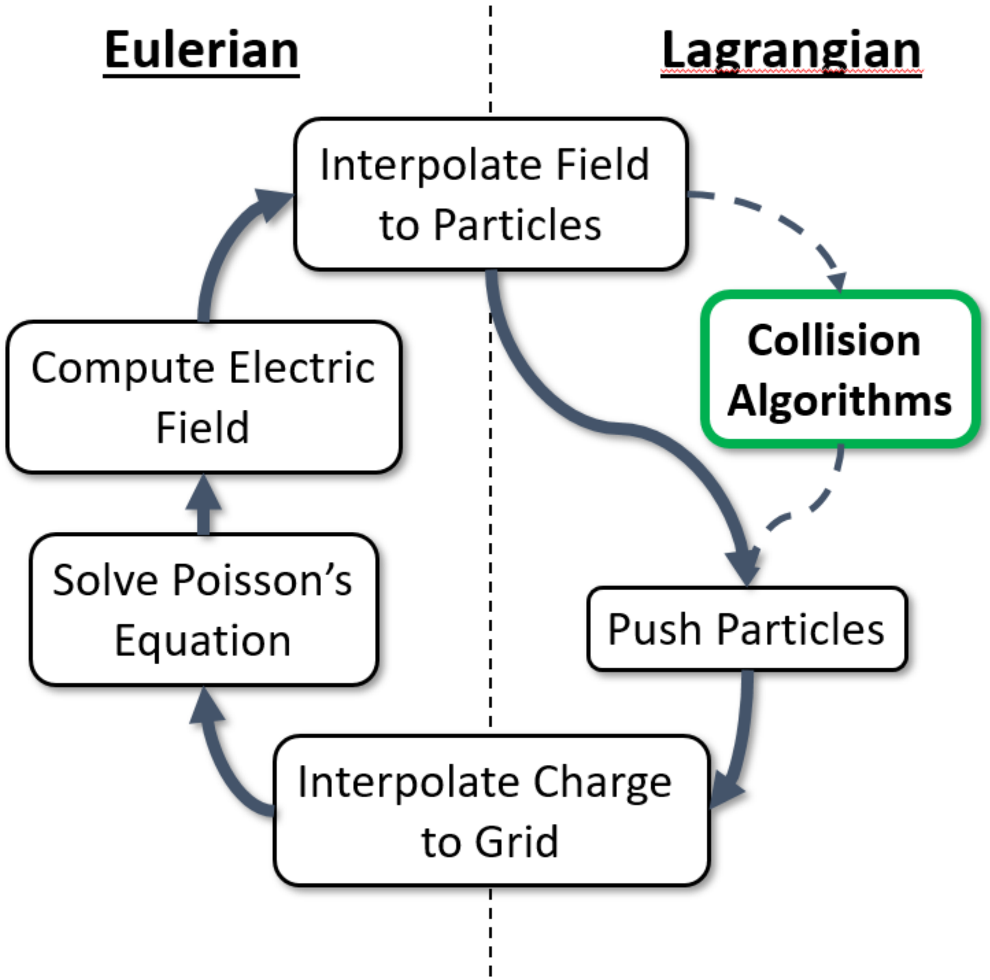
- Poisson's equation is a **global** solve, requiring global communication
- We incorporated a Geometric Multigrid algorithm from *Hypre*
- Runs on the CPU and GPU
- **NOTE:** GPU implementation is a work in progress

Field Solver

- Performance was good on a single GPU *if* the problem is large enough
- Our problem is around the size where we see little difference
- Furthermore, poor scaling was observed when going to multiple GPUs



Collision Module



- Collisions are Monte-Carlo and therefore require (in this case trillions or more) high quality random numbers
- Explored multiple approaches
- Ideally, we want to produce these on the GPU in a portable way (i.e. not using cuRAND)

PRNGs on the GPU

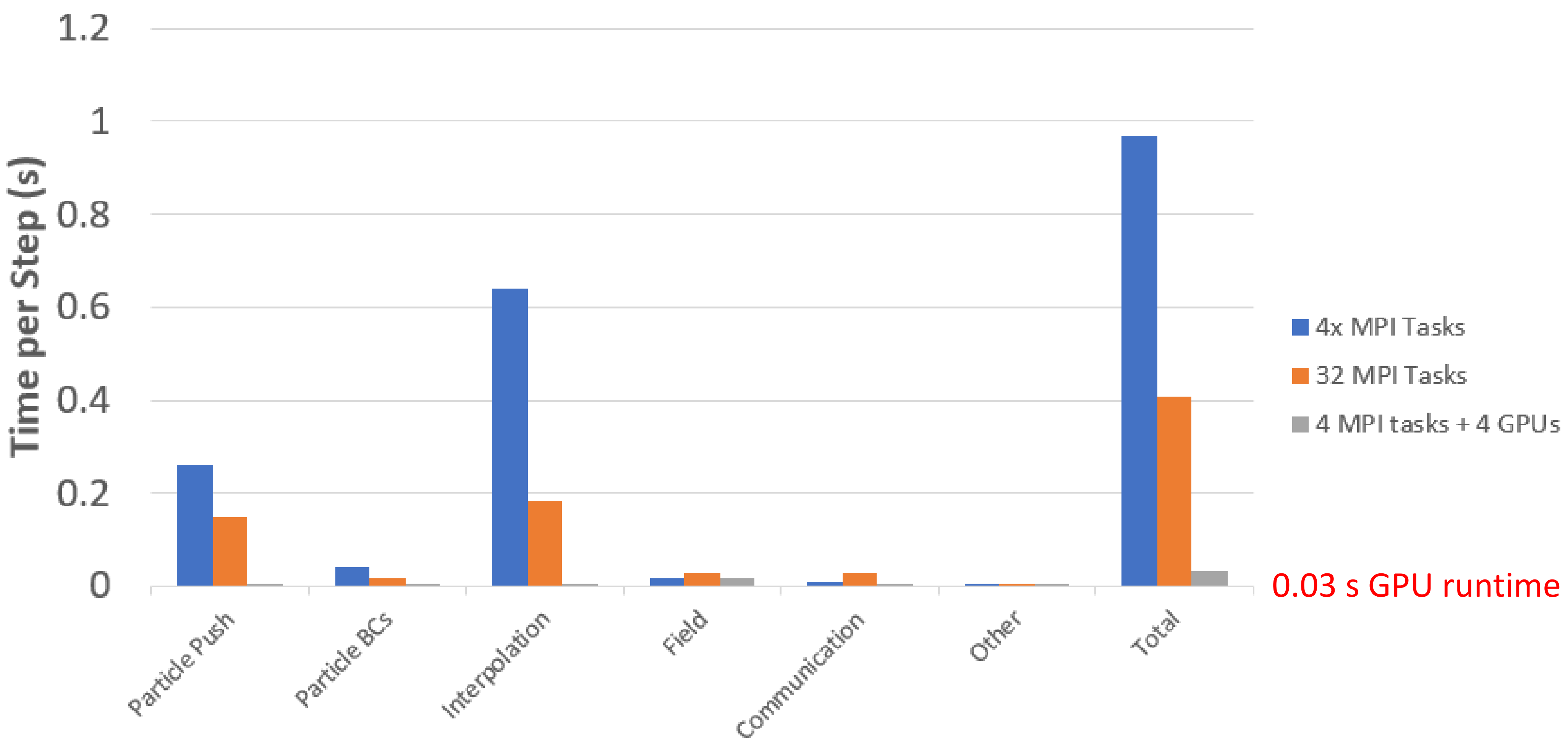
- Most scalable solution is to store a PRNG state with each particle and then generate random numbers locally at the *OpenACC* thread level on the fly
- Block ciphers using the *data encryption standard* can generate random numbers from a 7 byte state!
- These PRNGs have an increased overhead, but we believe that this is tolerable in order to improve scalability
- Passes all SmallCrush (University of Montreal) PRNG quality tests
- In the process of porting this PRNG to the GPU via *OpenACC*

Benchmarking & Performance

Benchmarking & Testing Performance

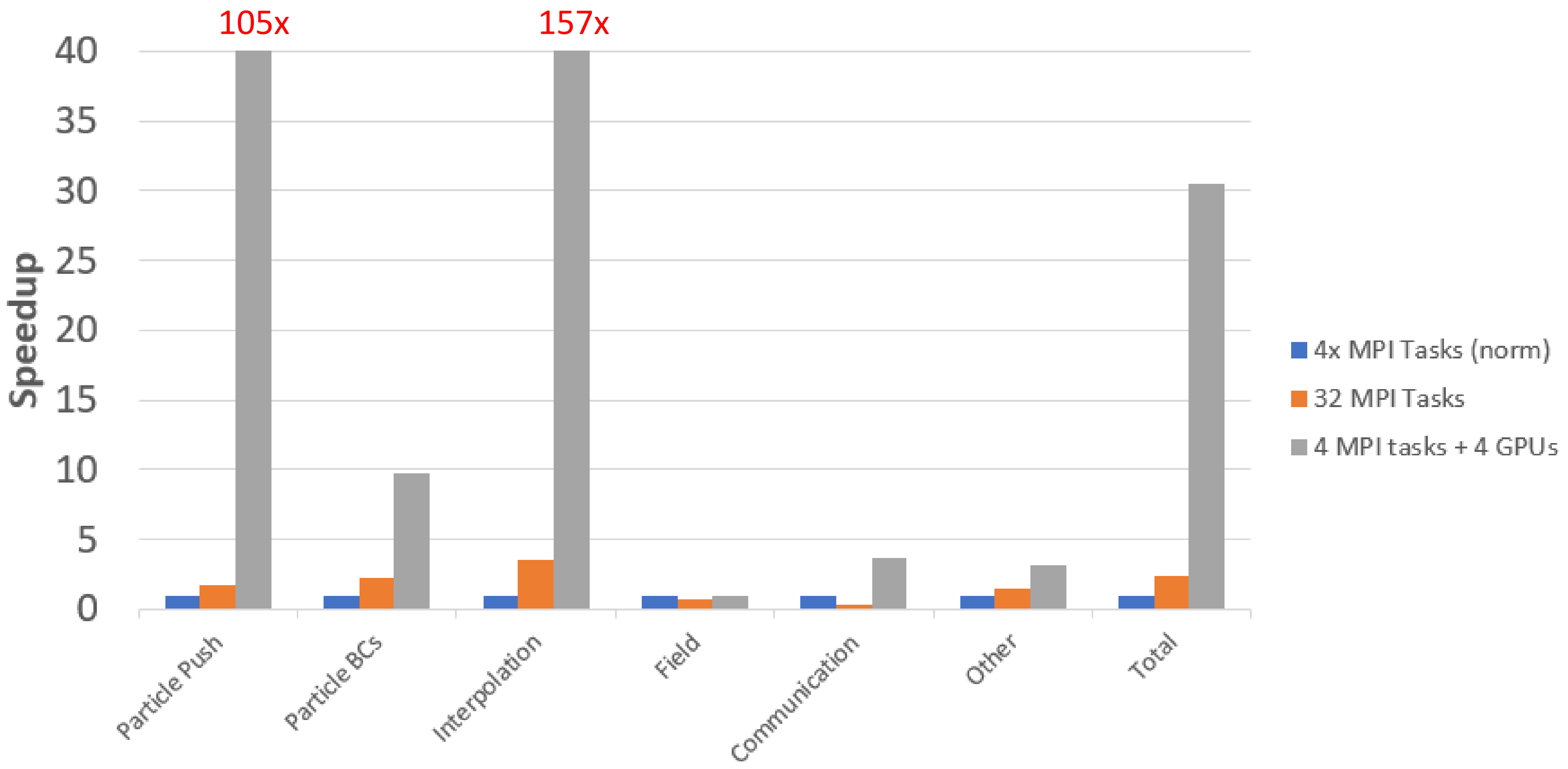
- The code was benchmarked successfully against published results from 6 different (independently developed) codes [Charoy et al 2019]
- We ran performance tests on PPPL/Princeton's *Traverse* computer:
  - 46 IBM POWER9 nodes with four NVIDIA V100 GPUs per node
- Performance comparisons are made on 1 node with a typical *per node* simulation setup
  - 128x250 cells, 80 million particles, I/O every 1,000 time steps

Performance - Runtime

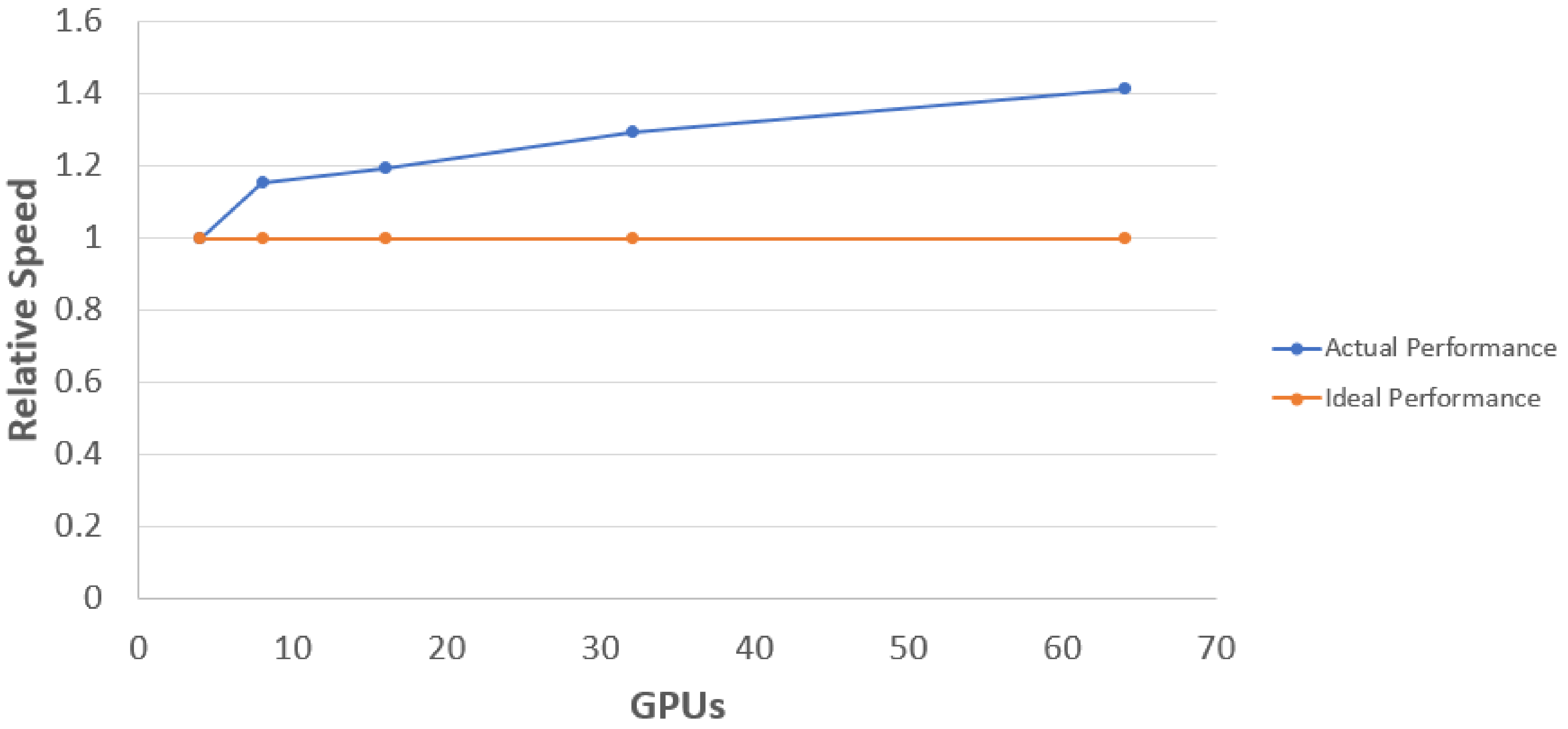


0.03 s GPU runtime

Performance - Speedup

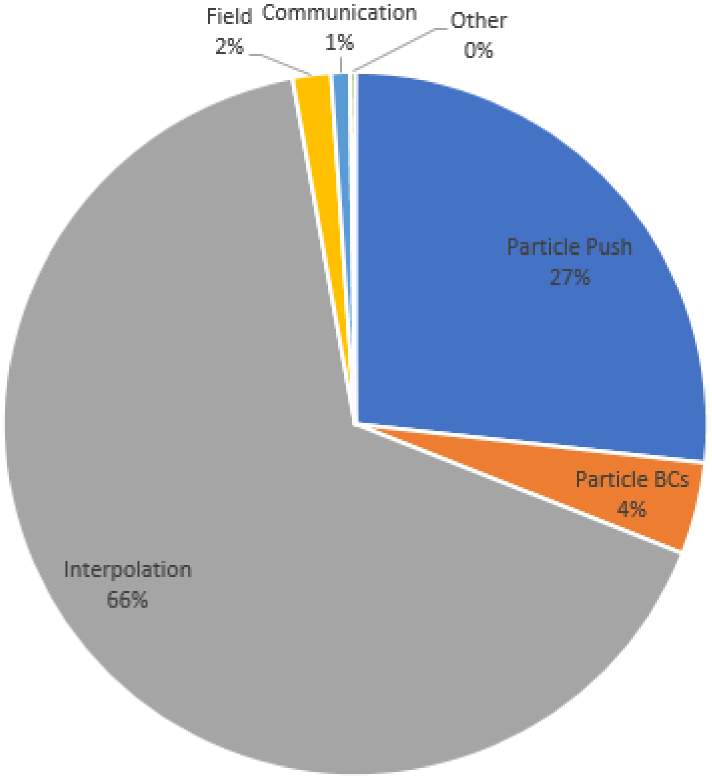


Performance – Weak Scaling



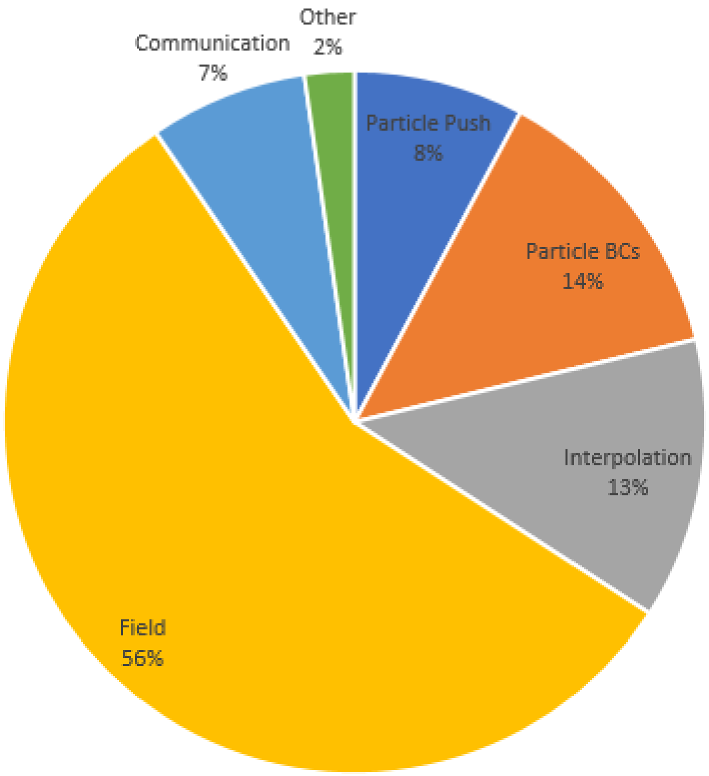
Performance – Field Solver Bottleneck

4x MPI Tasks



0.97 s per time step

4x MPI Tasks + 4x GPUs



0.03 s per time step

Portability

- We have LTP-PIC running on numerous high performance and local systems:
  - *Traverse* – Power9 IBM and V100 architecture
  - *Ascent* – Power9 IBM and V100 architecture (near identical to *Summit*)
  - *Perseus* – Intel Broadwell chip architecture
  - PPPL Clusters – Intel and AMD chip architectures
  - Local machines – Linux and Mac OS

- One caveat is that we still maintain *OpenMP* operability so that we can compile with Intel. This adds some verbosity to the code
- Otherwise *OpenACC* has allowed us to maintain a single code base, and interoperability which we believe could not be maintained by any other approach in such an straightforward way

Conclusions

Lesson’s Learnt

- In general, *OpenACC* is **easy** to implement, and in most cases can just directly replace or be put inline with *OpenMP* flags
- There are some inevitable learning curves associated with memory management (which we chose to do explicitly)
- GPUs are powerful and *OpenACC* allowed us to easily access this performance. Memory latency is better than expected as shown by a speedup in interpolation algorithms!
- Portable random numbers on the GPU are not straightforward with *OpenACC*
- Scalable linear algebra solvers for elliptical PDEs seems to be an open problem on GPUs

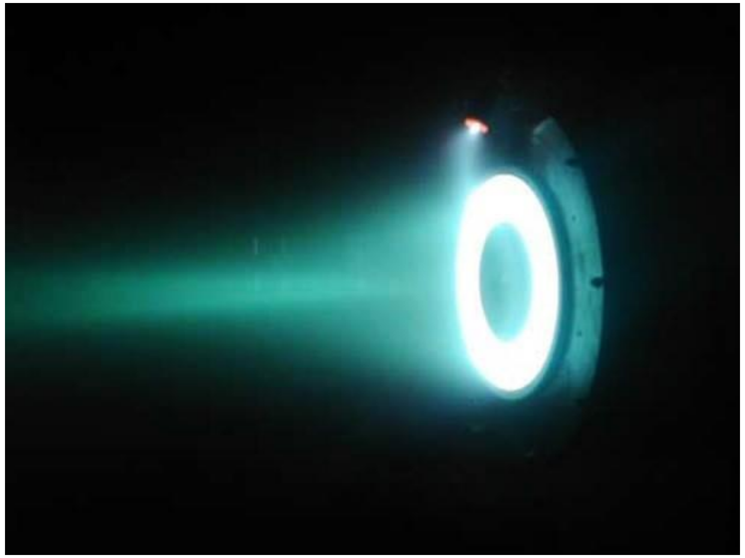
OpenACC and GPU Wish List

OpenACC Wish List

- Further interoperability with *OpenMP* and other compilers
- A good quality and portable pseudo-random-number-generator

GPU Wish List

- Good elliptic/global solvers



Questions?

References

- Falgout, R. D., & Yang, U. M. (2002, April). hypre: A library of high performance preconditioners. In *International Conference on Computational Science* (pp. 632-641). Springer, Berlin, Heidelberg.
- L'Ecuyer, P., & Simard, R. (2007). TestU01: AC library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*, *33*(4), 1-40.
- Charoy, T., Boeuf, J. P., Bourdon, A., Carlsson, J. A., Chabert, P., Cuenot, B., ... & Powis, A. T. (2019). 2D axial-azimuthal particle-in-cell benchmark for low-temperature partially magnetized plasmas. *Plasma Sources Science and Technology*, *28*(10), 105010.
- Introduction pictures were sourced (and credited) from the 2019 *Gaseous Electronics Conference* website Picture Gallery: http://apsgec.org/gec2019/gallery.php