Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

Achieving Maintainable Cross-Platform Performance in the Particle-in-Cell Accelerator Modeling Code Synergia using Kokkos

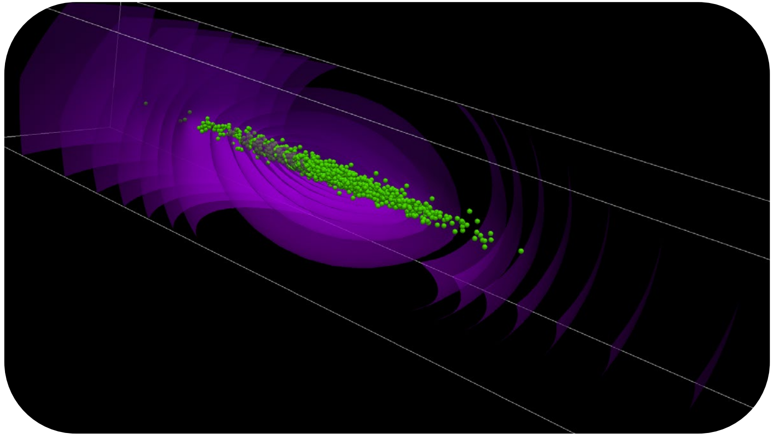
Qiming Lu, Eric G. Stern, Marc Paterno, James Amundson

SIAM CES21, March 5th, 2021

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

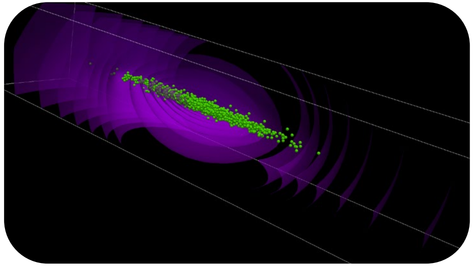
FERMILAB-SLIDES-21-008-SCD

Synergia
Particle Accelerator
Modeling Framework



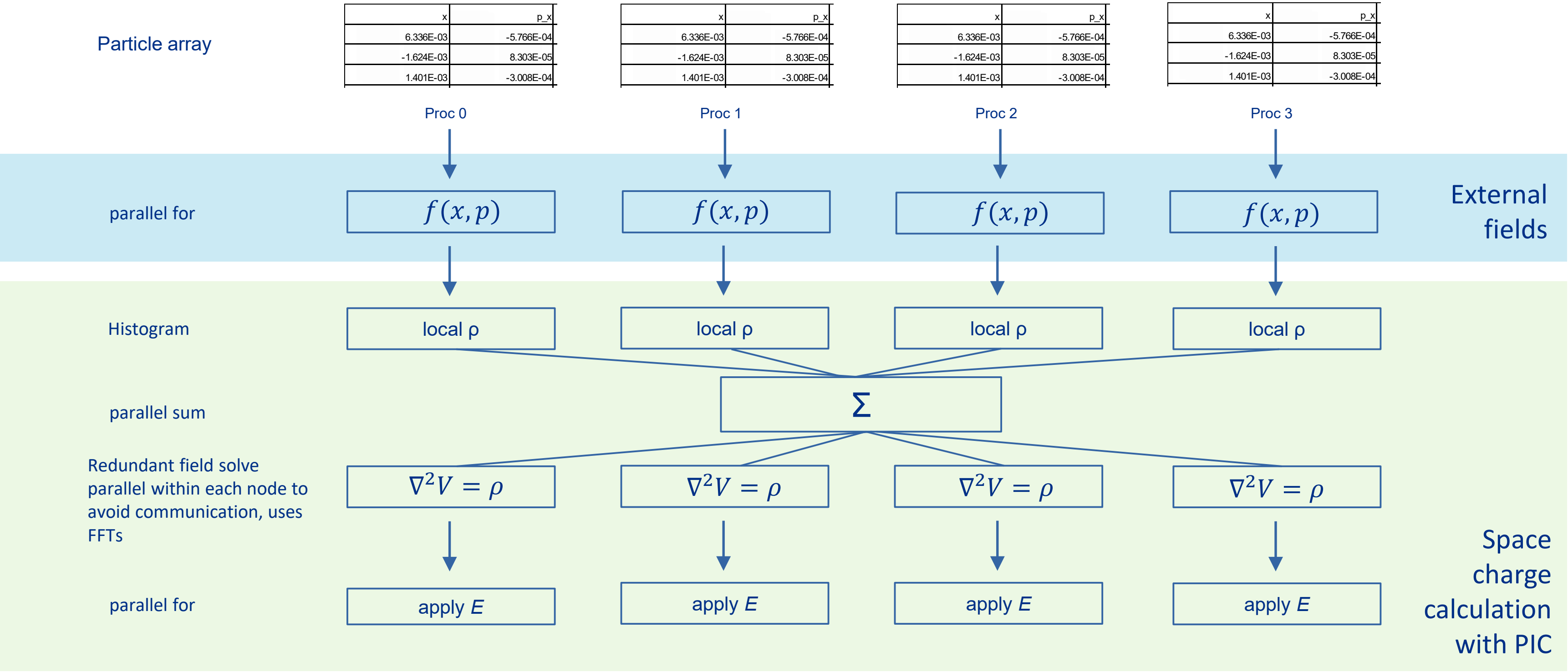
- Beam dynamics simulation and modeling package for particle accelerators
  - Beam optics from external fields
  - Internal fields calculation (space charge with particle-in-cell)
  - Beam-beam interactions, wakefield simulations, etc.

Synergia Modeling Framework



- C++ library with Python wrappers
  - Most simulations are written in Python and import modules to perform the heavy calculation. Main processing loop is in C++.
- Uses MPI parallel processing to scale to large problems.
- Runs on desktop/laptop, small/medium clusters, supercomputers.
  - Small problems can be run on small systems (number of particles, size of accelerator, etc.)
  - Code scales well for large problems on large systems.

Synergia computational ingredients



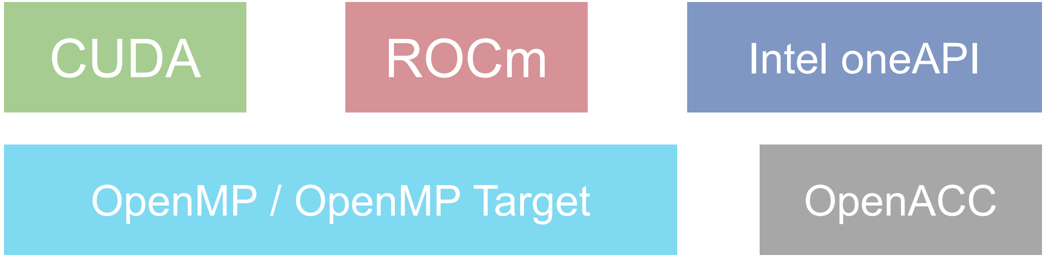
Fermilab

New Challenges in the Era of Exascale Computing

The bulk of computing power at the new large facilities is heavily shifting towards accelerators such as GPU or other co-processors

- Summit at OLCF: Power9 + Nvidia V100 GPU
- Frontier at OLCF: AMD EPYC CPU + AMD Radeon GPU
- Aurora at ALCF: Intel Xeon CPU + Intel Xe GPU
- Perlmutter at NERSC: AMD EPYC CPU + NVidia A100 GPU

Along with emerging parallel programming models and tools, oftentimes locked-in to specific hardware or platform



New Challenges in the Era of Exascale Computing

The application needs to adapt and make use of the accelerators

- Shifting the paradigm from CPU centric parallelization to a hybrid of CPU and accelerator parallelization

… and be portable

- Keep broad accessibility across computing platforms.
- Use “standard” languages and programming techniques as much as possible.
- Avoid architecture lock-in for code maintainability and execute-everywhere capability.
- Minimize architecture specific code and algorithms.
  - a previous CUDA specific Synergia version was unmaintainable and rotted into uselessness

… with high performance!

- Portability is not an excuse for poor performance

kokkos

- https://kokkos.org
- Part of the Exascale Computing Project
- C++ library maintained as an open-source repository on Github.

- Shared memory programming model that supports architecture specific backends, e.g., OpenMP or CUDA.
- Hardware agnostic: supports NVIDIA (now), AMD and Intel GPUs (promised)

- Provides abstractions for both parallel execution of code and data management
- Allows significant expressibility, particularly on GPUs

Kokkos Data Storage

- `Kokkos::View<T>` is a generic multi-dimensional data container
  - Allows the user to control “where” (memory spaces) the data resides,
  - and “how” (memory layout) the data are stored
    - E.g., `Kokkos::View<double**, CudaSpace, LayoutLeft>` is a 2d double array stored in the CUDA device memory with column major (left) layout.

- Managing the bulk of particle data with Kokkos::View<>
  - Resides in the device memory during its lifetime for fast accessing from computing kernels
  - has a host mirror and gets synced manually when necessary
    - For OpenMP threads backend, syncing between host and "device" has virtually no costs
  - Uses column major for both CPU and GPU backends, optimal for
    - CPU vectorization
    - GPU memory coalescing

Kokkos Parallel Dispatch

An example with drift propagation in Synergia with Kokkos

```
// simplified for demonstration ...
KOKKOS_INLINE_FUNCTION
double drift_unit(double px, double t)
{ return px * t; }

const size_t N = ...;
View<double*[6]> p("particles", N);
double t = ...;

// fill p with some numbers ...

parallel_for(N, [=](int i) {
     p(i,0) += drift_unit(p(i,1), t);
     p(i,2) += drift_unit(p(i,3), t);
});
```

- The same code can be compiled and run on both CPU (OpenMP) and GPU (CUDA, or other backends supported by Kokkos)
- 3 types of parallel dispatchers serve as the building blocks for more complicated algorithms
  - parallel_for()
  - parallel_reduce()
  - parallel_scan()

SIMD Vectorization with Kokkos

- Very limited vectorization support from Kokkos
  - Auto-vectorization with compiler directives, available with only Intel compilers in the OpenMP backend

- Yet being able to use vectorization on CPU is crucial to the performance

- Synergia has implemented a portable SIMD primitive with explicit vector types to work with Kokkos kernels
  - C++ templated class for a range of SIMD vector types
  - Uses Agner Fog's vectorclass (https://github.com/vectorclass) for x86/64 SSE/AVX/AVX512 intrinsic/types
  - Supports Quad Processing eXtension(QPX) for IBM Power CPUs
  - Compatible with GPU kernels (by falling back to single width data types)

SIMD Vectorization with Kokkos

```
template<class T>
struct Vec : public VecExpr<Vec<T>, T>
{
    T data;

    KOKKOS_INLINE_FUNCTION
    static constexpr int size();

    KOKKOS_INLINE_FUNCTION
    Vec(const double *p);

    KOKKOS_INLINE_FUNCTION
    void store(double *p);

    KOKKOS_INLINE_FUNCTION
    T & cal()       { return data; }

    KOKKOS_INLINE_FUNCTION
    T   cal() const { return data; }

    template <typename E>
    KOKKOS_INLINE_FUNCTION
    Vec(VecExpr<E, T> const& vec)
    : data(static_cast<E const&>(vec).cal()) { }
};
```

- The template class Vec<T> can be instantiated with vector types that has basic operators (+-*/, etc) overloaded
  - SSE: Vec<Vec2d>
  - AVX: Vec<Vec4d>
  - AVX512: Vec<Vec8d>
  - On GPU it is just Vec<double>

- VecExpr<E, T> is an expression template where expressions are evaluated only as needed. It …
  - avoids the need for creating temporaries
  - avoids the need for multiple loops in evaluating vectors

SIMD Vectorization with Kokkos

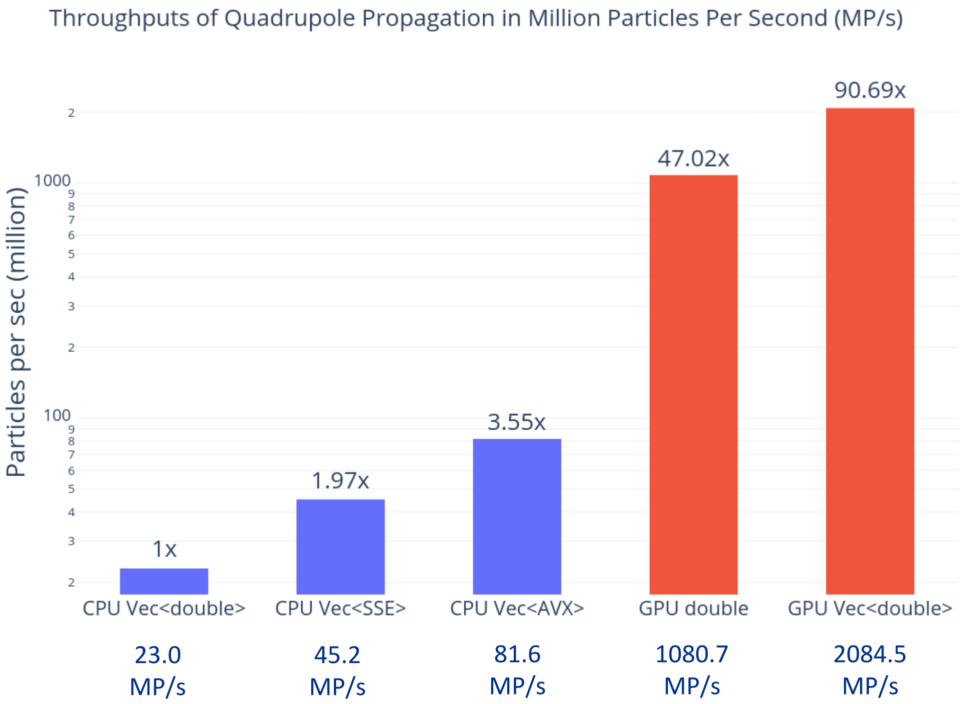
```
template<class T>
KOKKOS_INLINE_FUNCTION
T drift_unit(T px, double t)
{ return px * t; }

// Vec4d is the avx type from vector class
using gsv = Vec<Vec4d>;

parallel_for(N, [=](int i) {
      int idx = i * gsv::size();

      gsv p0(&p(idx,0));
      gsv p1(&p(idx,1));

      p0 += drift_unit(p1, t);

      p0.store(&p(idx,0));
});
```

- The same drift method written in SIMD primitive
- drift_unit() is now a function template to work with various vector types
- Particle data is still a double array (as opposed to a vector typed array)
  - Extra load() and store() to construct and writeback the vectors around the calculation
  - Allows flexible control over whether to use vector calculation (not all algorithms are suitable for vectorization)

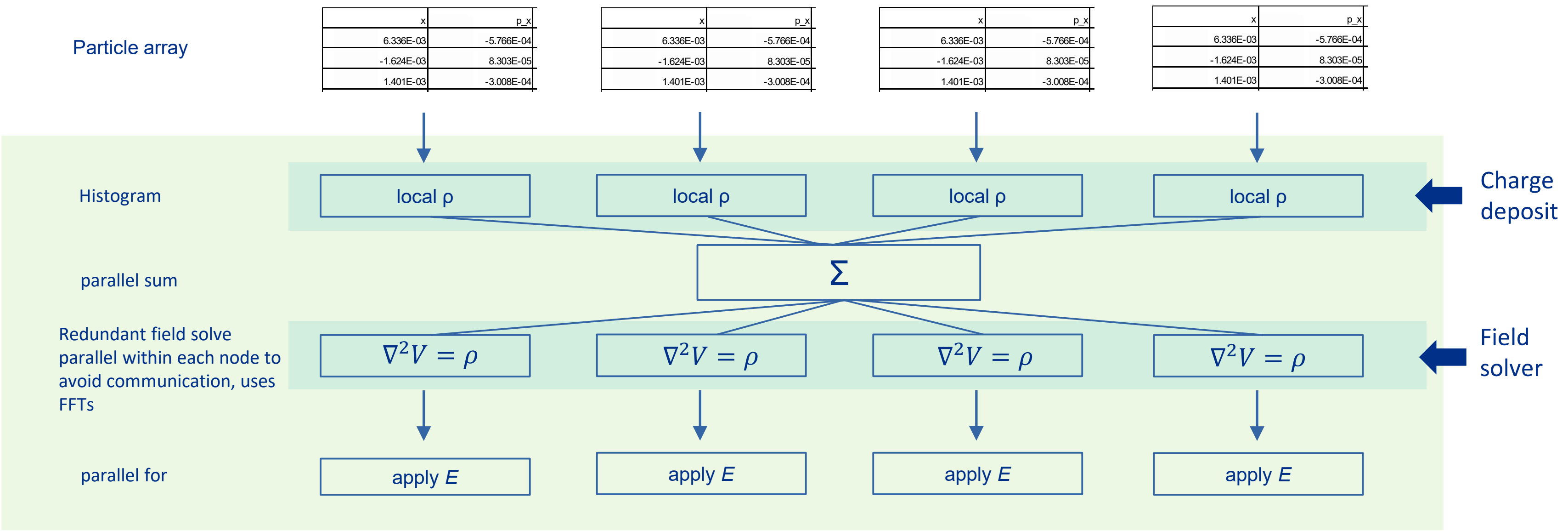
Fermilab

Performance Comparison of Unified Computing Kernels

- Intel Xeon 6248
  - 20 cores / 40 threads @ 3.90GHz turbo
  - SSE4.2 / AVX / AVX2 / AVX512
  - Max throughput @ 81.6 MP/s
  - 81.8 MP/s for pure OpenMP implementation
  - 2x and 3.5x for SSE and AVX vectorization
- Nvidia Volta V100 GPU
  - 84 SM / 5120 CUDA cores
  - Max throughput @ 2084.5 MP/s
  - Using expression templates has nearly doubled the throughputs on GPUs!
- Nvidia Ampere A100 GPU
  - Max throughputs @ 2876.8 MP/s
  - ~40% increases vs V100



Space Charge



Parallel Charge Deposit in Shared-memory

- Two approaches:
  - Data duplication is often faster on the host, but too memory expensive on GPUs
  - Atomics are faster on GPUs, but slow on the host

```
View<double**, LayoutLeft> p;
View<double***> grid;
ScatterView<double***> sv(grid);

parallel_for(N, [=](int i) {
     auto access = sv.access();

     auto [ix, iy, iz] = get_indices(
          p(i,0), p(i,2), p(i,4));

     access(ix, iy, iz) += charge;
     access(ix+1, iy, iz) += charge;
     ...
});

contribute(grid, sv);
```

- Kokkos::ScatterView<> does duplication on the host backend, and atomics on GPUs
- As of the latest Kokkos version (3.3.1), ScatterView still has some performance issues on OpenMP backend
- Synergia has manually implemented the data duplication histogram on the OpenMP code path

Fermilab

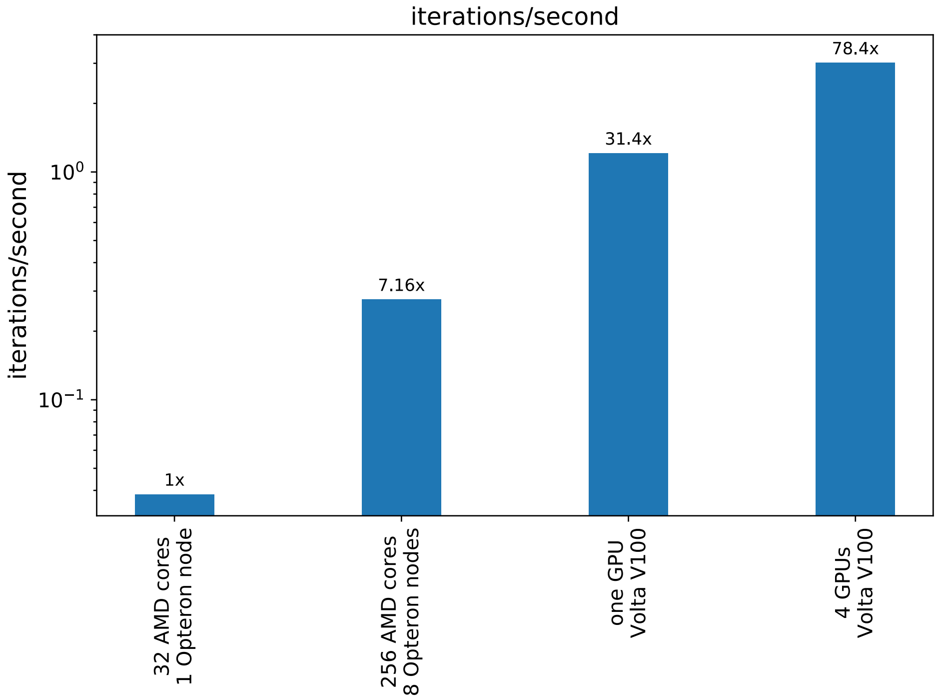
Field Solver

- The 2D and 3D open boundary conditions space charge solver in Synergia uses the convolution method to solve the field

- Needs a portable FFT method for the solver to be truly portable
  - Provides unified FFT interfaces for 2d/3d R2C/C2R DFTs, 3d DST/DCT, etc.
  - Handles device/host data movement, memory padding, and data alignments automatically
  - Calls FFTW on host
  - Calls CUFFT on CUDA backend
  - Needs to be extended for AMD GPUs and Intel GPUs

Benchmark accelerator simulation results

- Overall performance comparison
  - Real world particle accelerator simulations
  - 4M particles, 3D space charge @ 64x64x128 grid size
- 1 or 8 AMD 32 core Opteron nodes
- Power9 + Nvidia V100 GPUs
  - 1 - 4 GPUs per node
  - Similar to Summit nodes



Conclusion

- It is possible to achieve portable performance with a unified codebase
  - Shifts the burden of hardware specific implementations/optimizations to the third-party libraries and people with expertise, so we can focus on the algorithms of our specific problems
  - A portable and unified codebase is much more maintainable than multiple hardware specific code branches

- Caveats
  - Took a year of work to migrate the code from mostly OpenMP parallelization to Kokkos
  - Even though the code can be hardware agnostic, doesn’t mean the developers should also ignore the differences in underlying hardware – some algorithms and data structures are not suitable for GPUs and the memory model, therefore needs to be redesigned
  - Still some device-specific code was necessary

Acknowledgment

Synergia development was developed through the SciDAC-4 ComPASS project funded by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research and Office of High Energy Physics, Scientific Discovery through Advanced Computing (SciDAC) program.

Work supported by the Fermi National Accelerator Laboratory, managed and operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy.