Parallel accelerator simulations

past, present and future

James Amundson

Fermilab

November 21, 2011

This Talk

- Accelerator Modeling and High-Performance Computing (HPC)
  - Accelerator Modeling
    - Accelerator Physics
    - Synergia
  - High Performance Computing
    - Supercomputers
    - Clusters with High-Performance Networking
  - Optimizing Synergia Performance

Accelerator Physics

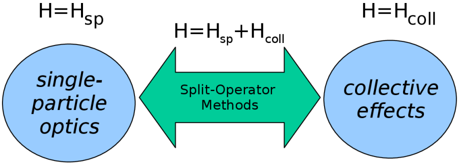
Computational accelerator is a huge topic, crossing several disciplines. The three main areas of current interest are

- Electromagnetic simulations of accelerating structures
- Simulations of advanced accelerator techniques, primarily involving plasmas
- **Beam dynamics simulations**

Independent-Particle Physics and Collective Effects

- Independent particle physics
  - The interaction of individual particles with external fields, *e.g.*, magnets, RF cavities, etc.
  - Usually the dominant effect in an accelerator
    - Otherwise, it wouldn't work...
  - Well-established theory of simulation
  - Easily handled by current desktop computers
- Collective effects
  - Space charge, wake fields, electron cloud, beam-beam interactions, etc.
  - Usually considered a nuisance
  - Topic of current beam dynamics simulation research
  - Calculations typically require massively parallel computing
    - Clusters and supercomputers

Split-Operator and Particle-in-Cell Techniques



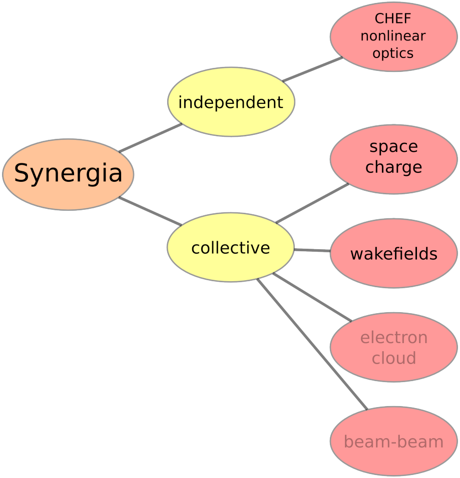
The split operator technique allows us to approximate the evolution operator for a time t by

$$\mathcal{O}(t) = \mathcal{O}_{sp}(t/2)\mathcal{O}_{coll}(t)\mathcal{O}_{sp}(t/2)$$

The Particle-in-Cell (PIC) techique allows us to simulate the large number of particles in a bunch (typically $\mathcal{O}(10^{12})$) by a much smaller number of macroparticles (typically $\mathcal{O}(10^7)$). Collective effects are calculated using fields calculated on discrete meshes with $\mathcal{O}(10^6)$ degrees of freedom.

Synergia

- Beam-dynamics framework developed at Fermilab
- Mixed C++ and Python
- Designed for MPI-based parallel computations
  - Desktops (laptops)
  - Clusters
  - Supercomputers



https://compacc.fnal.gov/projects/wiki/synergia2

Supercomputers and Clusters with High-Performance Networking

Tightly-coupled high-performance computing in the recent era has been dominated by MPI, the Message Passing Interface.

MPI provides

- Point-to-point communications
- Collective communications
  - Reduce
  - Gather
  - Broadcast
  - Many derivatives and combinations

MPI is a relatively low-level interface. Parallelizing a serial program to run efficiently in parallel using MPI is not a trivial undertaking.

Modern supercomputers and HPC clusters differ from large collections of desktop machines in networking.

- High bandwidth
- Low latency
- Exotic topologies

Platforms

In recent times, we have run Synergia on ALCF's Intrepid and NERSC's Hopper. We also run on our (Fermilab's) Wilson cluster.

Intrepid

Intrepid's Blue Gene/P system consists of:

- 40 racks
- 1024 nodes per rack
- 850 MHz quad-core processor and 2GB RAM per node

For a total of 164K cores, 80 terabytes of RAM, and a peak performance of 557 teraflops.

Hopper

Hopper's Cray XE6 system consists of:

- 6,384 nodes
- 2 twelve-core AMD 'MagnyCours' 2.1-GHz processors per node
- 24 cores per node (153,216 total cores)
- 32 GB DDR3 1333-MHz memory per node (6,000 nodes)
- 64 GB DDR3 1333-MHz memory per node (384 nodes)
- 1.28 Peta-flops for the entire machine

Wilson Cluster

2005:

- 20 dual-socket, single-core (2 cores/node) Intel Xeon CPU
  - 0.13 TFlop/s Linpack performance

2010:

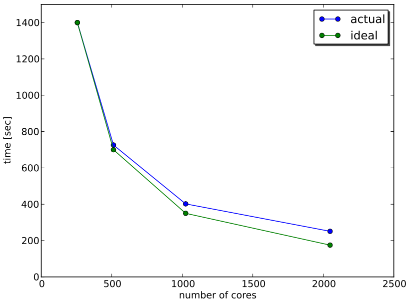
- 25 dual-socket, six-core (12 cores/node) Intel Westmere CPU
  - 2.31 TFlop/s Linpack performance

2011: (last week!)

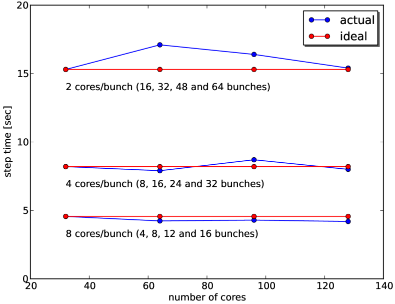
- 34 quad-socket, eight-core (32 cores/node) AMD Opteron CPU

Strong and Weak Scaling

Strong scaling: *fixed problem size*



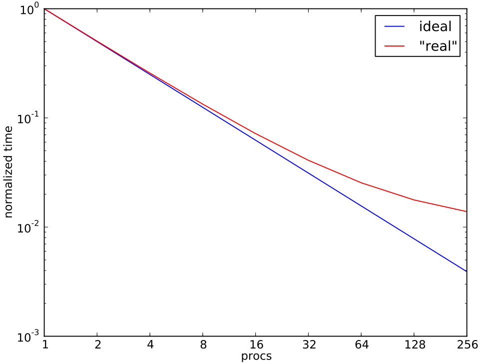
Weak scaling: *fixed ratio of problem size to number of processes*



Strong Scaling is Hard

Take a serial program. Profile it.

- Parallelize routines taking up 99% of runtime.
  - Assume scaling is *perfect*.
- Restrict the remaining 1% to non-scaling.
  - Could be worse!



Optimizing Synergia Performance

In Synergia, particles are distributed among processors randomly. Each processor calculates a spatial subsection of the field in field solves. (Other schemes have been tried.)

Major portions a Synergia space charge calculation step:

- Track individual particles (twice)
    - Easily parallelizable.
- Deposit charge on grid locally.
    - Easily parallelizable.
- Add up total charge distribution (semi-) globally.
    - A communication step.
- Solve the Poisson Equation.
    - Uses parallel FFTW.
        - Internal communications.
- Calculate electric field from scalar field locally.
    - Easily parallelizable.
- Broadcast electric field to each processor.
    - A communication step.
- Apply electric field to particles.
    - Easily parallelizable.

The Benchmark

A space charge problem using

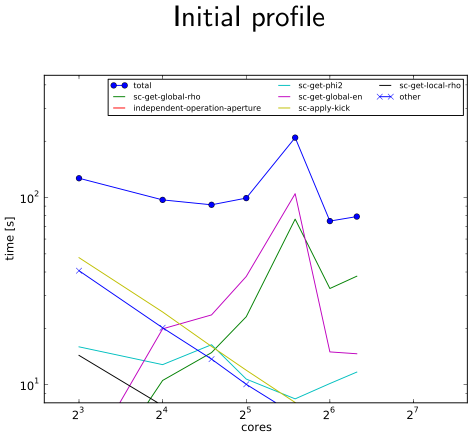
- a $64 \times 64 \times 512$ space charge grid
- with 10 particles per cell
  - for a total of $20,971,520$ particles.
- There are 32 evenly-spaced space charge kicks.
- The single-particle dynamics use second-order maps.

Real simulalations are similar, but thousands of times longer.

Performed all profiling and optimization on Wilson Cluster. Hopper has similar performance characteristics, but networking is a few times faster.

Initial Profile

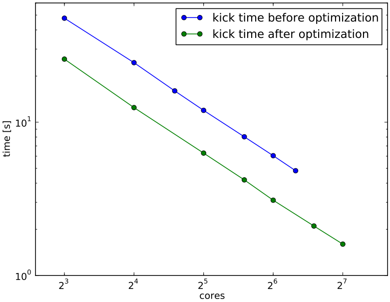
In May 2011, we embarked on an optimization of the newest version of Synergia, v2.1.



Decided to look at field applications and communication steps.

Optimizing Field Applications

- Minimized data extraction from classes
- Minimized function calls
- Inlined functions in inner loop
- Added a periodic sort of particles in z-coordinate
  - Minimize cache misses when accessing field data
  - std::sort is really fast
- Added a faster version of floor

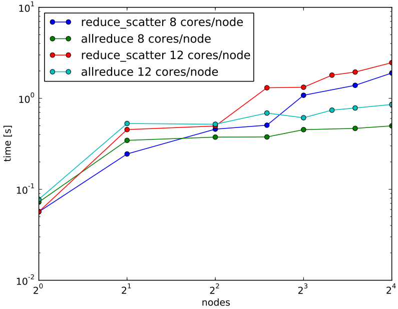


Overall gain was $\sim 1.9\times$

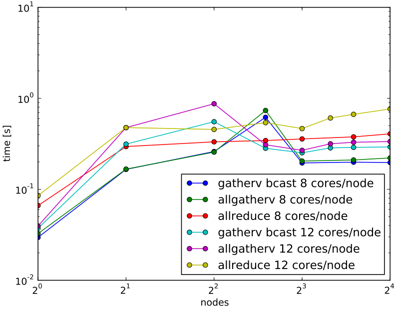
Optimizing Communication Steps

Tried different combinations of MPI collectives.

Charge communication

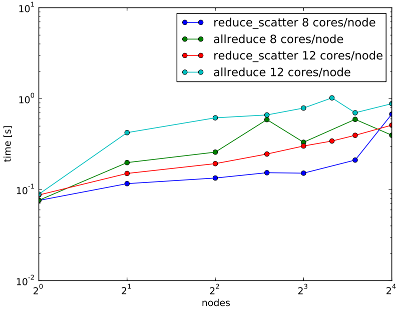


Field communication

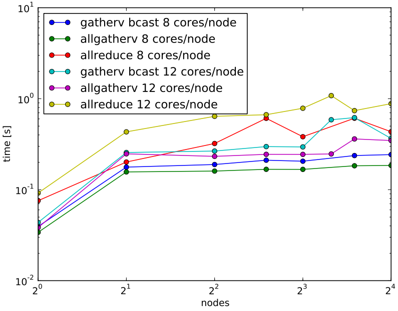


The previous results used OpenMPI 1.4.3rc2. Try MVAPICH2 1.6:

Charge communication



Field communication

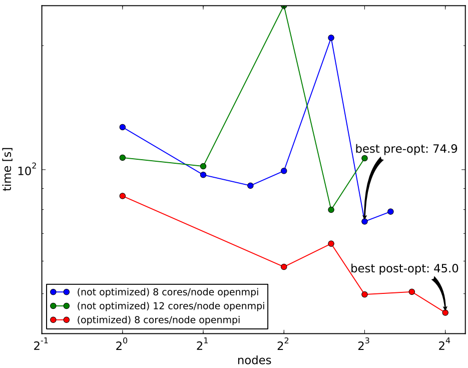


Communication Optimization

- No single solution won.
- Keep all options.
- Add a function to try all communications types (once) and keep the fastest one.
    - User can choose his/herself if desired.

Final Results

We gained a factor of ~ 1.7 in peak performance.

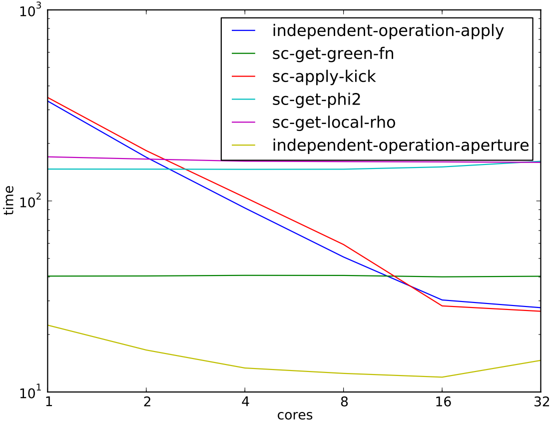


The Future

Where to go next?

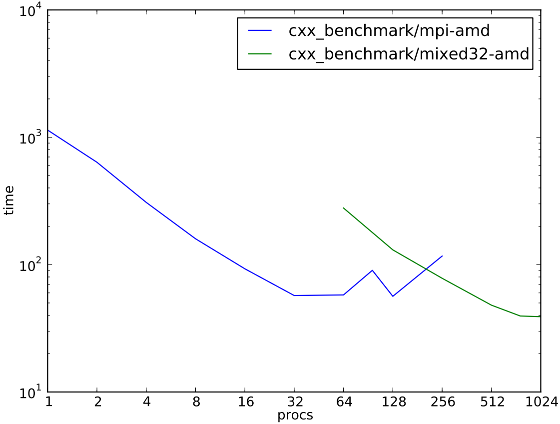
- Optimize for multiple threads
  - OpenMP
    - Not very hard
    - Cannot be a final solution – not enough threads
  - Hybrid OpenMP-MPI
    - Promising
- Utilize GPUs
  - CUDA
    - Not very easy
    - Cannot be a final solution – single GPUs not fast enough
  - Hybrid CUDA-MPI
    - Promising
- Hybrid CUDA-OpenMP-MPI
  - Sounds complicated
  - Where we will probably have to end up

First Steps with OpenMP



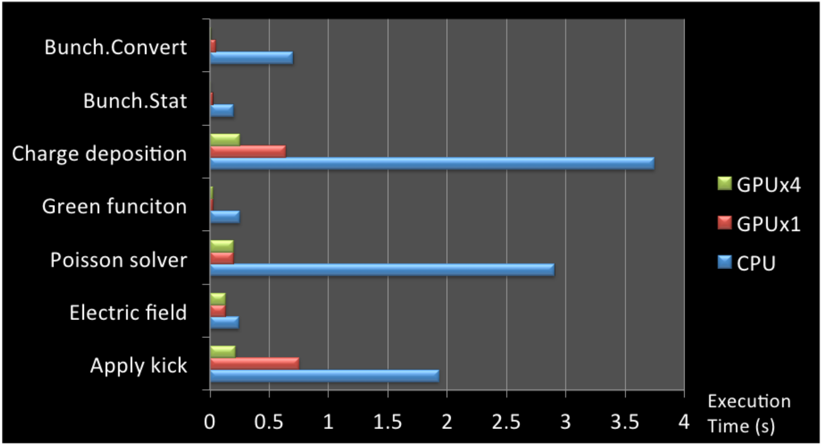
Charge deposition is actually harder than the MPI case.

First Steps with Hybrid OpenMP-MPI

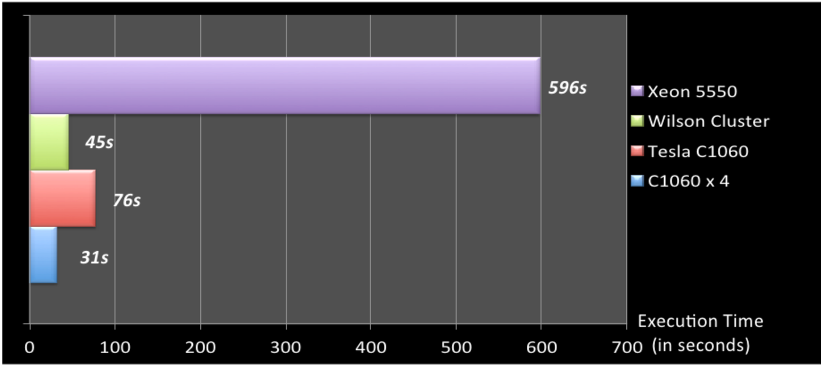


Peak performance improved, but we still have a long way to go.

First Steps with CUDA (profiling)



First Steps with CUDA (comparison)



Next Step

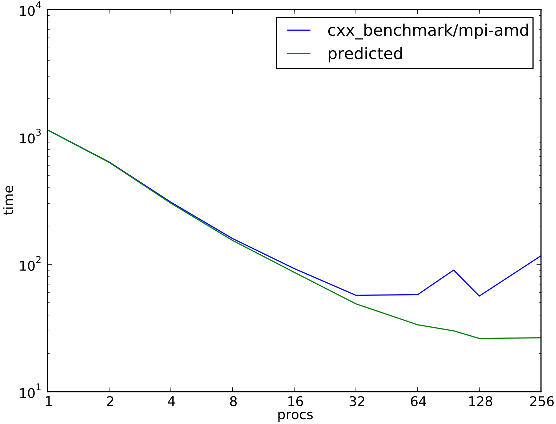
Next Step: Communication avoidance

In principle: *Do more computation in order to avoid computation*

In practice: *Solve Poisson on each* processor*. Avoids broadcasting field.*

Already did this in multi-GPU calculation.

Predicted Behavior



Peak performance expected to improve by $\sim \times 2$

Conclusions

- High-performance computing passed the $100k$ core mark quite a while ago.
- Evolution is toward more cores per cpu.
- Future promises more cores, GPUs.
  - Exascale computing discussions have considered millions to $\sim$billion cores.
- Our techniques are evolving to include hybrid approaches to parallelism.