FERMILAB-CONF-07-281-CD

James F. Amundson†, Panagiotis Spentzouris, Fermilab, Batavia, IL 60510 USA
Ji Qiang, Robert D. Ryne, LBNL, Berkeley, CA 94720 USA
Douglas R. Dechow, Tech-X, Boulder, Colorado, 80303 USA

Abstract

The need for realistic accelerator simulations is greater than ever before due to the needs of design projects such as the ILC and optimization for existing machines. Sophisticated codes utilizing large-scale parallel computing have been developed to study collective beam effects such as space charge, electron cloud, beam-beam, etc. We will describe recent advances in the solvers for these effects and plans for enhancing them in the future. To date the codes have typically applied to a single collective effect and included just enough of the single-particle dynamics to support the collective effect at hand. We describe how we are developing a framework for realistic multi-physics simulations, i.e., simulations including the state-of-the-art calculations of all relevant physical processes.

INTRODUCTION

As the demands for precision in the design and operation of particle accelerators increases, so must the precision of accelerator simulations increase. High-precision simulations must eventually include collective effects such as space-charge, electron-cloud, beam-beam, wakefields, etc. Precision calculations of these effects require computationally expensive field calculations and application, which in turn require efficient calculations on parallel machines. The SciDAC Accelerator Science and Technology project made significant progress on this task. Much work remains to be done, however, and a new SciDAC-funded computational accelerator physics project, the COMPASS collaboration, is starting to work on the next level of these challenges. Throughout both projects, there have been two very compatible themes: First, do not re-invent the wheel. Second, take advantage of the state-of-the-art wherever possible.

PARALLEL COMPUTERS, LARGE AND SMALL

The scope of computational resources available to accelerator physicists is very broad. Our solver development strives to take into account the fact that rough calculations performed on desktop computers are frequently desirable, while detailed calculations are usually performed on Linux clusters and supercomputers. Even within this three categories, there are wide variations. Desktop computers are increasingly utilizing multiple-core processors. Linux clusters exist with a variety of interconnect options and supercomputers vary widely in size and architecture.

Parallel computations require communications between processors and therein lies the difficulty in writing efficient parallel code. The community has largely settled on the Message-Passing Interface (MPI) as the mechanism of choice for performing parallel communication. MPI allows many different communication patterns; the most efficient pattern may very well depend on the detail of the network beneath it.

Aspects of Networks

The most important aspects of networking for parallel calculations are latency, bandwidth and topology. Latency represents the shortest time in which a message can be passed between two processors. Bandwidth is usually expressed as the maximum speed at which *large* amounts of data can be moved between processors. For smaller messages, latency will dominate speed. The network topology is important in that it determines how the communication speed is affected by the pattern of communication. The simplest and least expensive network fabric seen in Linux clusters, is gigabit networking, which has a high bandwidth but also high latency. More exotic networking options are Myrinet and Infiniband, both of which supply bandwidth of the same order of magnitude as gigabit networking, but with much smaller latencies. The cost to add Myrinet and Infiniband networking to a Linux cluster is usually of the same order of magnitude as the CPUs themselves. Supercomputers typically have proprietary networking, often with exotic topologies. The most extreme example is IBM's BlueGene [1].

FRAMEWORKS AND COMPONENTS

Under the auspices of the SciDAC Accelerator Science and Technology, two frameworks were developed to combine state-of-the-art single-particle optics with state-of-the-art collective effects. Synergia2 [6] utilizes the single-particle optics from the CHEF libraries [5] and the parallel space-charge calculation from IMPACT [2], while MaryLie/IMPACT (ML/I) utilizes the single-particle optics from MaryLie [3] and space-charge from IMPACT. (See Figures 1 and 2, respectively.)

Our next-generation development will take the concept of code re-utilization much further by introducing true software components. The tools provided by the Common

* Work supported by the U.S. Department of Energy SciDAC program.
† amundson@fnal.gov

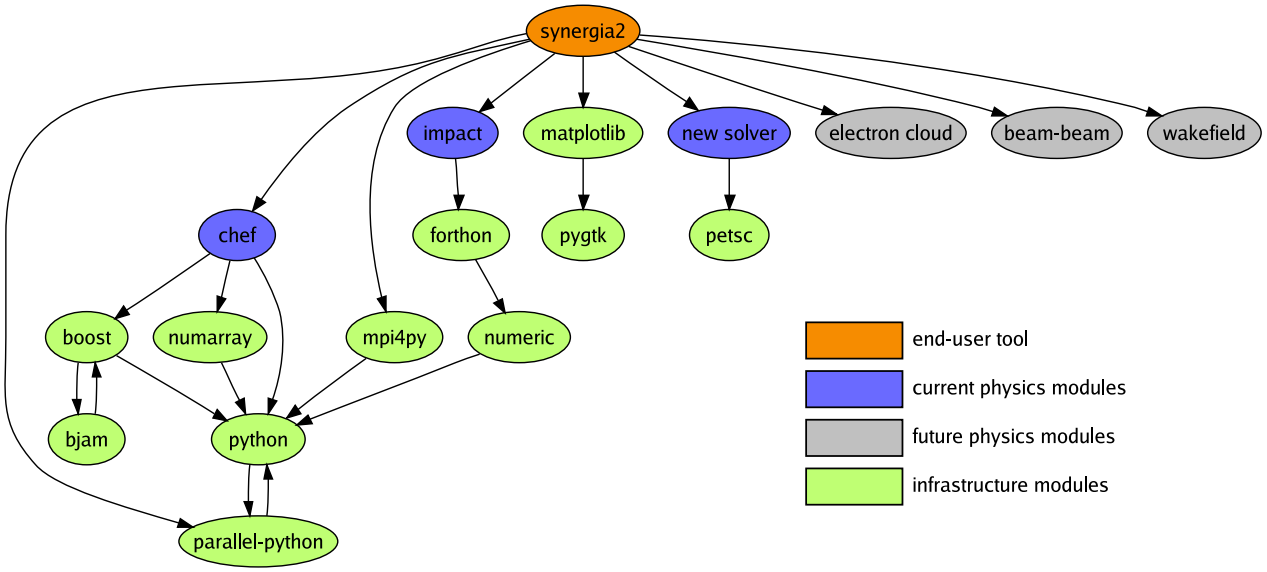


Figure 1: Synergia2, showing the relationship of the many software packages utilized within it.

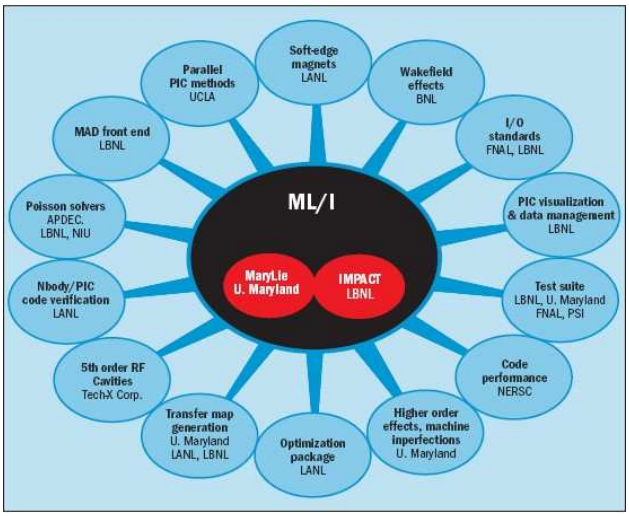


Figure 2: The multiple components of ML/I.

Component Architecture (CCA) Forum [9], will provide the necessary infrastructure for building self-contained, interchangeable components. Of particular interest is the Babel [10] tool, which handles language interoperability, allowing components to be written in Fortran 77, Fortran 90, C, C++, Python, and Java. A first report on using the CCA tools for accelerator applications can be found in Reference [11].

POISSON SOLVERS

Poisson solvers are at the heart of space-charge, beam-beam and electron cloud simulations. While other collective simulations may not require Poisson solvers specifically, the general features of a field calculated on a finite grid interacting with discrete particles are at the core of the particle-in-cell (PIC) technique which underlies virtually every collective simulation. The development of accurate and fast parallel Poisson solvers is therefore crucial for advanced simulations.

FFT-based Solvers

The fastest available 3D Poisson solvers utilize the fast Fourier transform (FFT) to solve the field equation. The most advanced such solvers currently available are in IMPACT, which contains its own parallel FFT implementation. These solvers have been incorporated into Synergia2 and ML/I.

Our first step toward the next generation of Poisson solvers has been to develop a solver utilizing the FFTW package [8]. The advantages of using FFTW are many. The first, and most obvious, advantage is speed. FFTW has been shown to be much faster than naive FFT implementations and as fast as hand-tuned, proprietary, platform specific implementations. Since the FFTW package is in wide usage its accuracy has been tested and verified on a wide variety of platforms and problems. Another, perhaps lesser-known advantage, is the flexibility in the size of the grid used for the FFT. While the best-known FFT algorithm is restricted to powers of two, FFTW can efficiently perform transforms on arbitrary length arrays. Since a factor of two change in grid size on a 3D grid corresponds to a factor of eight in memory and computation, the ability to use arbitrary grids is very important. For a long time, the current version of FFTW3 did not have parallelism via MPI. However, the recently-released version 3.2alpha does have MPI; we have used this version for our solver. It should be emphasized that all of these features, including parallelism, come with the FFTW package. Utilizing furthers our original objectives of re-use and obtaining the state-of-the-art.

Figure 3 shows the result of the steps in a Poisson solve,

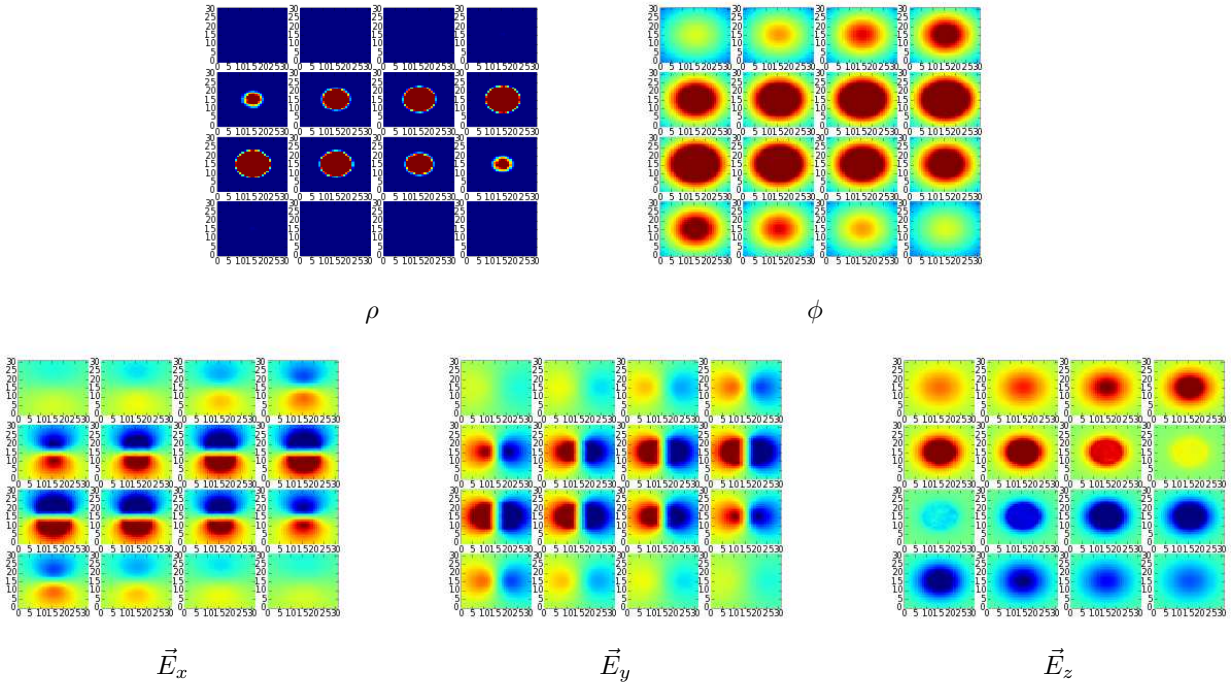


Figure 3: Result of steps in a Poisson solve in 3D. The charge density ρ is used to determine the scalar potential ϕ, which is then differentiated to obtain the components of the electric field $\vec{E}$. This example uses a $32 \times 32 \times 32$ grid. The slices are along the z-axis; each slice is the average of two adjacent z values.

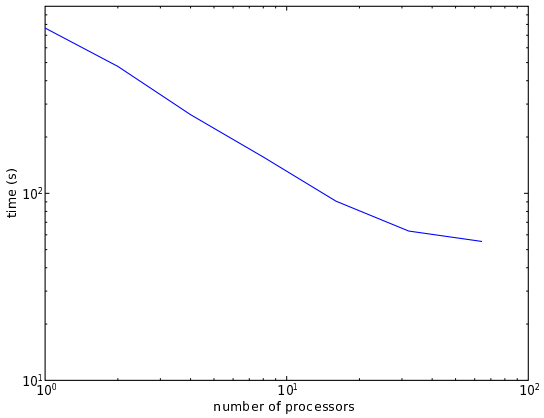


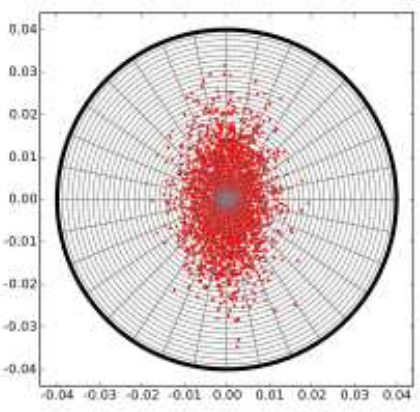
Figure 4: Performance of our FFTW-based solver. The simulation is of a revolution of the Fermilab booster utilizing 1048576 particles and a $32 \times 32 \times 256$ space-charge grid. The simulation consisted of 96 space-charge kicks with accompanying second-order map steps. The Linux cluster used consists of 48 dual-processor 2.4 GHz Pentium IV CPUs connected via Myrinet.

in this case using our FFTW-based solver. The parallel performance of our solver appears in Figure 4.

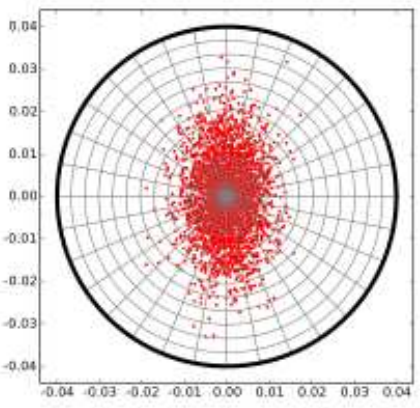
Finite-difference solvers

While FFT-based solvers have pr oven to be very fast, they also have limitations. They are limited to uniform grids and open boundary conditions or finite boundary conditions with very limited geometry. More flexibility can be used in the finite-difference approach, which reduces partial differential equations to linear algebra problems. Incorporation of non-uniform grids and arbitrary boundary conditions are straightforward exercises when utilizing finite-differences. Furthermore, solving the linear algebra problem via the multigrid method has the best scaling properties of any known method, including FFTs. Unfortunately, it is not clear that the asymptotic limit where multigrid beats FFT has yet been seen in Poisson solves.

Fortunately, a great deal of work has been done on parallel linear algebra. We are developing solvers based on the PETSc libraries [7], which represent the current state-of-the art in the area. Our first application has been to create a solver with a dual-density grid – the grid is fine where the beam is densest, coarse where the beam fades away. This approach allows accurate modeling of a high-aspect beam in a conducting pipe, such as is found in the ILC damping rings. Figure 5 displays the difference between uniform and dual-density density grids for a flat beam.



(a)



(b)

Figure 5: Transverse view of a uniform grid (a) vs. a dual-density grid (b) for a high-aspect-ratio beam in a conducting pipe. Both grids lead to similar accuracy in the field calculation, but the second grid requires far less computing time.

PARALLELIZATION SCHEMES

Parallel performance is limited by communication times. There are two major contributions to communication times in a parallel PIC code. The first is the communication required to solve the field equations in parallel. This problem has been well studied for both FFT and linear algebra solvers in the general case. The second contribution is specific to PIC codes; it is the scheme used to communicate between the particles and the fields. Our codes utilize two different such schemes: particle/field decomposition and field decomposition. The two schemes have very different advantages and disadvantages.

Particle/Field Decomposition

In the particle/field decomposition scheme, the fields are spatially distributed across processors, i.e., each processor only knows about a limited spatial portion of the field. The particles are distributed similarly by their spatial coordinates. In order to maintain this decomposition, the particles may have to be moved to the appropriate processor after steps that change their position. IMPACT uses a scheme of this type utilizing a two-dimensional processor distribution. One advantage of this scheme is that the particle movement can be done using nearest-neighbor communication utilizing simple MPI send/receive commands. A disadvantage of this scheme is that the grid must be periodically redistributed if the distribution of the particles changes, otherwise the number of particles on each processor will become unbalanced, leading to poor performance. An unfortunate consequence of this scheme is that performance can be highly process dependent. A simulation of a beam whose envelope is oscillating unexpectedly can be much slower than a stationary beam. The large number of small messages being passed leads this scheme to be most sensitive to the networks latency.

Field Decomposition

In the field decomposition scheme, the fields are once again spatially distributed. The particles, however, are distributed without regard to their spatial. This method avoids the complex movement of particles across processors at the price of reducing charge density information from all processors at the beginning of a solve and gathering the resulting fields to all processors at the end of a solve. The performance of this scheme does not depend on the physics being modelled; oscillating beams and stationary beams will perform equally well.

While the field decomposition scheme is less sensitive to the process being modelled, it is more sensitive to the underlying MPI infrastructure. Where the particle/field decomposition scheme utilizes primarily MPI sends and receives, the field decomposition scheme utilizes the MPI collective *reduce* and *scatter* methods. The performance of these methods depends on the quality of the algorithms in the particular MPI implementation. The differences among currently available implementations vary to a surprising degree. (See Figures 6 and 7.) The reliance on the MPI collectives can be a positive feature. Supercomputers are typically optimized for MPI collective performance by taking into machine characteristics such as network topology. This is particularly true for BlueGene; we expect to see good scaling of the field decomposition when our code is ported to that platform. At the opposite end of the spectrum the field decomposition scheme has a different advantage. Since the messages passed around are the fields, which are relatively large, the network performance should be limited by bandwidth, not latency. This means that inexpensive gigabit networking should compare favorably with the more expensive Infiniband and Myrinet fabrics.

REFERENCES

[1] http://www.research.ibm.com/bluegene/

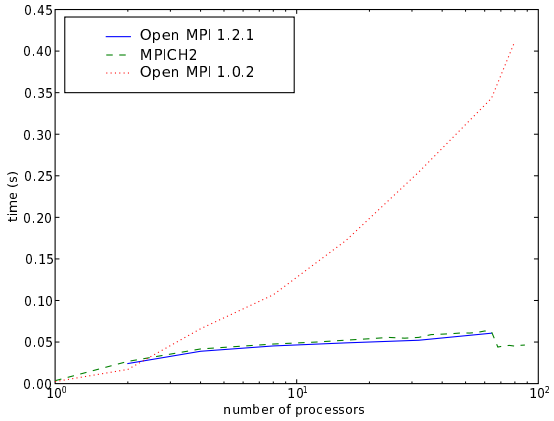


Figure 6: Time required to perform the *allreduce* operation for the charge density in our FFTW-based solver for a $32 \times 32 \times 256$ grid on the cluster described in the performance figure above. Note how an algorithmic change between different versions of OpenMPI leads to dramatically different performance.

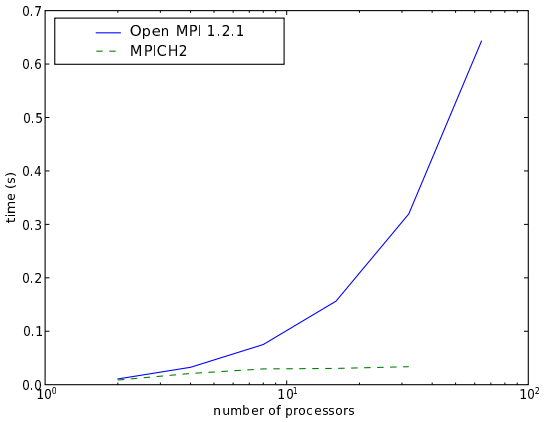


Figure 7: Time required to perform the *allgather* operation for the charge density in our FFTW-based solver for a $32 \times 32 \times 256$ grid on the cluster described in the performance figure above. While the later version of OpenMPI compared favorably to MPICH2 fort the reduce operation, the same is not true for the gather operation.

[2] J. Qiang, R. D. Ryne, S. Habib and V. Decyk, J. Comp. Phys. **163**, 434 (2000).

[3] Dragt, A. J., D. R. Douglas, E. Forest, F. Neri, L. M. Healy, P. Schtt, and J. van Zeijts, 1999, MARYLie 3.0 User's Manual (University of Maryland, Physics Department Report).

[4] `http://scidac.nersc.gov/accelerator/mli/manual.pdf`

[5] L. Michelotti, FERMILAB-CONF-91-159 *Presented at 14th IEEE Particle Accelerator Conf., San Francisco, CA, May 6-9, 1991*.

[6] J. Amundson, P. Spentzouris, R. Ryne and J. Qiang, Journal of Computational Physics, Volume 211, Issue 1, 1 January 2006, Pages 229-248.

[7] S. Balay, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley and L. McInnes and B. Smith, H. Zhang, ANL-95/11, 2004.

[8] M. Frigo and S.G. Johnson, "The Design and Implementation of FFTW3," Proceedings of the IEEE 93 (2), 216231 (2005).

[9] http://www.cca-forum.org/

[10] http://www.llnl.gov/CASC/components/babel.html

[11] D. Dechow, D. Abell, P. Stoltz, L. Curfman McInnes, B. Norris, J. Amundson, "A Beam Dynamics Application Based on the Common Component Architecture," *these proceedings.*