LA-UR- 98-3786

Approved for public release; distribution is unlimited.

CONF-981207--

Title: PARTICLE BEAM DYNAMICS SIMULATIONS USING THE POOMA FRAMEWORK

Author(s): William Humphrey, CIC/ACL
Robert Ryne, LANSCE-1
Timothy Cleland, CIC/ACL
Julian Cummings, CIC/ACL
Salman Habib, T-8
Graham Mark, CIC-12
Ji Qiang, LANSCE-1

Submitted to: Iscope Conference
12/8-11/98

Los Alamos
NATIONAL LABORATORY

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Particle Beam Dynamics Simulations Using the POOMA Framework*

William Humphrey, Robert Ryne, Timothy Cleland, Julian Cummings, Salman Habib, Graham Mark, and Ji Qiang

Los Alamos National Laboratory, Los Alamos, NM, USA

Abstract. A program for simulation of the dynamics of high intensity charged particle beams in linear particle accelerators has been developed in C++ using the POOMA Framework, for use on serial and parallel architectures. The code models the trajectories of charged particles through a sequence of different accelerator beamline elements such as drift chambers, quadrupole magnets, or RF cavities. An FFT-based particle-in-cell algorithm is used to solve the Poisson equation that models the Coulomb interactions of the particles. The code employs an object-oriented design with software abstractions for the particle beam, accelerator beamline, and beamline elements, using C++ templates to efficiently support both 2D and 3D capabilities in the same code base. The POOMA Framework, which encapsulates much of the effort required for parallel execution, provides particle and field classes, particle-field interaction capabilities, and parallel FFT algorithms. The performance of this application running serially and in parallel is compared to an existing HPF implementation, with the POOMA version seen to run four times faster than the HPF code.

1 Introduction

Particle accelerators have played a central role in shaping our present understanding of the fundamental nature of matter. At the same time, the application of accelerator theory and technology has contributed to substantial progress in other branches of science and technology. This historical trend is expected to continue with particle accelerators playing an increasingly important role in basic and applied science. As examples of recent applications, many countries are now involved in efforts aimed at developing accelerator-driven technologies for transmutation of radioactive waste, disposal of plutonium, energy production, and production of tritium. Additionally, next-generation spallation neutron sources based on similar technology will play a major role in materials science and biological science research. Finally, other types of accelerators such as the Large Hadron Collider (LHC), the International Linear Collider (ILC), and fourth-generation light sources will have a major impact on basic and applied scientific research.

* This work was performed under the auspices of the U.S. Department of Energy by Los Alamos National Laboratory under Contract No. W-7405-Eng-36.

For all of these projects, high-resolution modeling far beyond that which has ever been performed by the accelerator community is required to reduce cost and technological risk, and to improve accelerator efficiency, performance, and reliability. Indeed, such modeling is essential to the success of many of these efforts. For example, high average power linear accelerators, such as those needed for tritium production, must operate with extremely low beam loss (~ 0.1 nA/m) to prevent unacceptably high levels of radioactivity. To ensure that this requirement will be met, it is necessary to perform very high-resolution simulations using on the order of 100 million particles in which the beam propagates through kilometers of complicated accelerating structures. These simulations can only be performed on the most advanced high performance computing platforms using software and algorithms targeted to parallel and distributed environments. The calculations require performance of hundreds of GFLOPS to TFLOPS, and core memory requirements of hundreds of GBytes.

The beam dynamics modeling effort has concentrated so far on parallel calculations for the design of proton linear accelerators (linacs). Such accelerators are the machines of choice for applications including radioactive waste treatment and tritium production. Two-dimensional and fully three-dimensional beam dynamics codes that take into account both external accelerating and focusing fields, as well as the inter-particle Coulomb forces in the beam are in an advanced stage of development and have already been used for accelerator design studies [1, 2]. This paper describes the design and implementation of a parallel application used to model high-intensity charged particle beams moving through a linear accelerator, using an object-oriented design in C++ based on the POOMA Framework [3, 4]. The performance of this code is compared to an HPF implementation of the application, running serially and in parallel on the SGI Origin2000 parallel computers available at Los Alamos National Laboratory.

2 Simulating Linear Accelerators

To simulate the motion of charged particles through a linear accelerator, we have employed an object-oriented (OO) software design in our application. Using an OO design strategy makes it easier to develop modular, maintainable code which can easily be extended to incorporate new algorithms, simulation components, and capabilities. The characteristics of linear accelerators, consisting of sequences of beamline elements through which particles move as they are accelerated, lend themselves quite well to being modeled using an OO design. We can consider this system as being comprised of the following abstractions.

Beamline Elements consist of the distinct portions of the linear accelerator beamline through which the particles move. Particles interact with the elements in various ways as they propagate through them; for example, quadrupole magnet elements focus the beam as the charged particles move through their magnetic fields.

The Beamline comprises the collection of different beamline elements which make

up the linear accelerator, in the order the elements are encountered by the particles.

The Beam is the set of charged particles being accelerated by the system. Particles have characteristics such as phase-space coordinates, charge, and mass, and move through the beamline subject to the equations of motion for a linear accelerator.

The Accelerator is the entire system, comprising the beamline and the beam.

As the particles in the beam move through the beamline, passing through each beamline element, they experience both external forces due to the element they are passing through and internal forces due to the space-charge interaction of the particles with each other. The space-charge forces are calculated using a standard FFT-based particle-in-cell (PIC) algorithm for a collisionless system [5, 6]. In this algorithm, we first solve the Poisson equation

$$\nabla^2 \phi(\boldsymbol{r}) = 4\pi\rho(\boldsymbol{r}) \tag{1}$$

to find the electrostatic potential $\phi(\boldsymbol{r})$ from the charge density field $\rho(\boldsymbol{r})$ of the particles. From $\phi(\boldsymbol{r})$, the space-charge force $\boldsymbol{F}_i(\boldsymbol{r})$ on each particle with charge q_i is computed using

$$\boldsymbol{E}(\boldsymbol{r}) = -\nabla\phi(\boldsymbol{r}) \tag{2}$$

$$\boldsymbol{F}_i(\boldsymbol{r}) = q_i\boldsymbol{E}(\boldsymbol{r}). \tag{3}$$

The standard PIC algorithm, used in codes discussed here, may be summarized as:

1. Scatter charge onto a grid to obtain a discretized charge density $\rho(\boldsymbol{r})$;
2. Solve (1) to determine the electrostatic potential $\phi(\boldsymbol{r})$ on a grid;
3. Compute the electric field vectors $\boldsymbol{E}(\boldsymbol{r})$ from (2) on a grid by finite difference methods;
4. Gather the electric field vectors from the grid to the particle positions, and calculate the force on each particle $\boldsymbol{F}_i(\boldsymbol{r})$ using (3).

The beamline element forces and the space-charge interaction forces result in changes to the momentum and position of the particles, causing them to accelerate through the beamline.

3 Implementation Using the POOMA Framework

Figure 1 presents an overview of the object-oriented design of the particle accelerator simulation code, illustrating the abstractions for the accelerator, beam, and beamline components. Each solid box represents an object; the top half of each box indicates the object name, while the bottom half indicates the important methods or variable for the object. Lines terminating in arrows indicate inheritance ("is a") relationships; lines originating from diamonds indicate "has a" relationships.

The simulation code is implemented in ANSI/ISO C++ using the POOMA Framework [3, 4], and making use of the template facilities of C++. The objects shown in Fig. 1 correspond to C++ classes used in the application. These classes are templated on the number of dimensions and the floating-point type, making it possible to use the same source code base for simulations of different dimensions or data type precision. For the small fraction of the code which cannot be generalized to a dimension-independent formulation, specializations of the relevant functions are provided. At present, this specialization has been done for two and three dimensions.

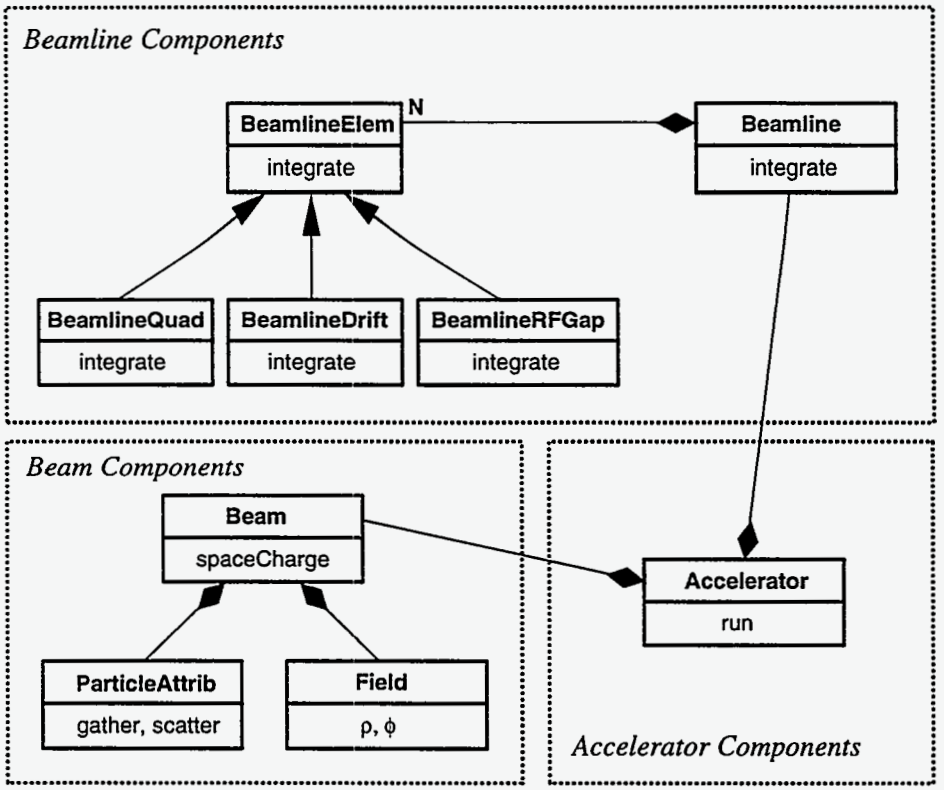


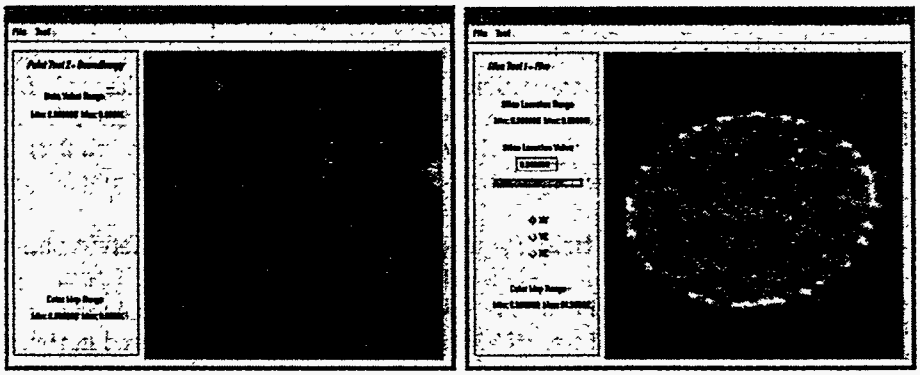
Fig. 1. A summary of the object design for the linear accelerator simulation code. An `Accelerator` consists of a `Beam` (a collection of charged particles) and a `Beamline` (a set of N `BeamlineElems`)

The `Accelerator` class contains the primary components of the simulation, namely a `Beam` instance and a `Beamline` instance. When created, `Accelerator` objects determine simulation parameters and beamline components from an input file, and initialize their `Beam` and `Beamline` accordingly. The `run()` method carries out the steps of the computation, by calling the `integrate()` method of

the `Beamline`. The `Beamline` in turn propagates the particles through each individual `BeamlineElem`, which are polymorphic classes that compute specialized forces used to update the momentum and position of the `Beam` particles. The `BeamlineElem` computations invoke the `spaceCharge()` method of the `Beam` to calculate the space-charge interaction forces for the particles.

The accelerator simulation code is built upon the POOMA Framework, a templated C++ class library which provides C++ abstractions for physical quantities such as particles and fields. POOMA provides N-dimensional parallel data structures for the beam particles and for the space-charge field quantities such as the charge density $\rho(\boldsymbol{r})$, electrostatic potential $\phi(\boldsymbol{r})$, and electric field $\boldsymbol{E}(\boldsymbol{r})$. POOMA encapsulates the complexity of providing a parallel run-time system, maintaining parallel data structures, and efficiently performing data-parallel computations. C++ template techniques such as expression templates [7] are used to implement a data-parallel syntax for expressions involving field and particle quantities; such expressions are evaluated at the same speed as hand-coded evaluation loops [3]. POOMA allows the user to write scientific simulation codes that can be run serially or in parallel with no change to the source code. The `Beam` class in Fig. 1 uses POOMA `ParticleAttrib` objects for the particle position and momentum data, and POOMA `Field` objects for $\rho(\boldsymbol{r})$, $\phi(\boldsymbol{r})$, etc.

The solution of the Poisson equation from (1) is computed with an FFT-based algorithm that uses multi-dimensional FFT routines from the POOMA Framework. POOMA also provides a number of particle-field interaction capabilities such as gather/scatter algorithms with different interpolation schemes. At present, both cloud-in-cell [8] and nearest-grid-point interpolation mechanisms are supported; additional algorithms are straightforward to implement and use with the POOMA gather/scatter routines.



(a) (b)

Fig. 2. Visualization of a sample 2D accelerator simulation. (a) Particle positions colored by kinetic energy. (b) Charge density field $\rho(\boldsymbol{r})$

The POOMA Framework also provides a run-time visualization option that can be used to visualize particle and field data structures either at run-time or post-processed from data files. Figure 2 shows a sample visualization from a 2D linear accelerator simulation, using the POOMA run-time visualization facilities. Figure 2a displays the positions of particles within the accelerator colored by their kinetic energy, and Fig. 2b displays the charge-density field $\rho(r)$ that results from scattering the electric charge of the particles onto a grid.

The use of a toolkit such as the POOMA Framework for development of high-performance simulation codes has proven to be an important tool in the implementation of the linear accelerator simulation code. The strong support in C++ for object-oriented programming features such as polymorphism, inheritance, and data abstraction, coupled with C++'s template facilities, makes it a useful language with which to implement a scientific application such as this. Also, templates provide a mechanism to avoid unnecessary run-time costs normally associated with the use of languages that support OO design, while still maintaining a high degree of flexibility and extensibility in a program.

Software development frameworks such as POOMA have proven to be a powerful tool for high-performance parallel scientific applications. The POOMA Framework has been used for several other codes in fields such as neutron transport [9], and as a basis for other frameworks such as Tecolote [10]. Several other libraries such as PETSc [11], which includes several linear and nonlinear system solvers, and Overture [12], which provides explicit support for overlapping grids in complex geometries, are used as a basis for parallel simulation codes in a wide range of applications. The advantage of using these different systems is clear: building your simulation code on top of an existing parallel application framework simplifies application design, shortens development time, and improves portability to different parallel platforms and communication mechanisms.

4 Performance

Table 1 and Fig. 3 compare the performance of the POOMA-based linear accelerator simulation code with a similar application written in High-Performance Fortran. This comparison was carried out on the Silicon Graphics Origin2000 parallel supercomputers at Los Alamos National Laborary, using the SGI C++ compiler (version 7.2) and the Portland Group HPF compiler (version 2.2). The calculations were all 2D simulations with a beamline comprising ten beamline elements.

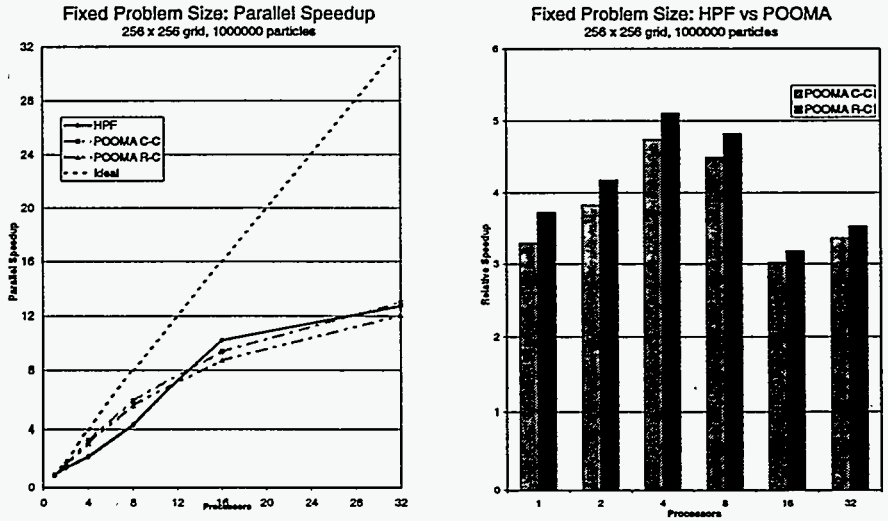
Table 1 shows running times for a 2D fixed-size problem on different numbers of processors. The problem modeled 10^6 particles moving through 10 beamline elements, using a 256^2 grid for the space-charge computation. The codes used were an HPF program and two POOMA-based versions that differed in their use of FFT routines. The POOMA code labeled "C-C" in the table used a complex-to-complex FFT algorithm, and the POOMA code labeled "R-C" used real-to-complex FFT routines. All three codes produced equivalent diagnostic results. The table gives the total simulation time (averaged across the processors) and

the amount of time spent in the gather/scatter and FFT portions of the space-charge computation, which is the single largest part of the simulation time.

Table 1. Run times (seconds) for a fixed problem size (10^6 particles, 256^2 grid)

	Total			Gather/Scatter			FFT		
Nodes	R-C	C-C	HPF	R-C	C-C	HPF	R-C	C-C	HPF
1	537.2	608.6	1998.8	385.6	392.5	1500.2	31.5	83.5	120.1
2	312.0	340.2	1300.8	197.6	198.3	1037.3	23.5	44.7	70.8
4	171.2	184.2	873.2	99.1	99.4	714.9	13.6	24.2	51.3
8	96.9	104.0	467.2	49.3	49.7	384.3	7.4	13.0	24.5
16	61.7	65.0	195.8	24.6	24.7	157.5	4.7	7.4	11.2
32	44.6	46.8	157.1	12.2	12.2	120.8	3.9	5.4	14.4

From the first three columns of Table 1, which list the total simulation time, we see that the POOMA codes outperformed the HPF codes by a factor between



(a) (b)

Fig. 3. Performance comparison between POOMA and HPF implementations of the accelerator simulation code, in two dimensions. (a) Parallel speedup (single-processor simulation time divided by multi-processor simulation times) for POOMA and HPF codes for a simulation of 10^6 particles on a 256^2 grid. (b) Relative speedup of POOMA over HPF codes for the same problem (HPF simulation time divided by POOMA simulation time)

3 and 5. Figure 3a, which shows the parallel speedup of the three codes, and Fig. 3b, which shows the speedup of the two POOMA codes relative to the HPF code, both demonstrate that the improvement is consistent from one to 32 nodes. The largest improvement between the timings for the HPF code and the POOMA codes is in the time to perform the gather and scatter operations, shown in the middle three columns of Table 1. The times to perform the FFT operations for the POOMA codes were also shorter than for the HPF code, particularly for the real-to-complex version of the POOMA code, but for this problem size the gather/scatter time represents the majority of the computation.

The performance gain with a real-to-complex FFT, which requires less storage and fewer elements in the FFT calculation, is particularly noticeable for problems with a small number of particles per cell. A set of simulations of 10^5 particles on a 256^2 grid using increasing numbers of nodes is summarized in Table 2, using the C-C and R-C versions of the POOMA application. Here, the relative improvement in performance using the real-to-complex version is much more noticeable than in the 10^6-particle simulation. While the parallel speedup for the C-C version is greater than that of the R-C version, the R-C version has much better single-node performance and reaches the point of diminishing parallel returns earlier than the C-C code.

Table 1 and Table 2 demonstrate that the gather and scatter portions of the POOMA codes scale reasonably well with the number of processors. The POOMA version performs an initial particle load-balancing that equally partitions the particles among processors and contributes to the nearly linear scaling behavior of the gather/scatter routines. This highlights a major difference between the POOMA and HPF simulation codes: the parallelization strategy for the particle data. POOMA employs a spatial decomposition strategy, which keeps particles local to the processor containing their charge density field and electric field data by reassigning particles to processors when the particle positions are changed. With a spatial decomposition, gather/scatter operations between the particles and fields require a minimum of communication. The HPF code employs a static partitioning of particles across the processors, requiring extra communication for the gather/scatter phase. In both cases, a roughly equal portion of the particles is kept on each processor. The extra time spent by POOMA to maintain particle locality and to perform the initial load balancing is more than made up for by reduction in the times for gather/scatter operations.

For large problem sizes, the majority of the computation time is spent in particle gather/scatter operations. In addition to the use of a spatial decomposition strategy to minimize the communication during gather and scatter calculations, POOMA provides an option to cache the particle-field interpolation generated in one gather or scatter operation for later gather/scatter calls. Interpolation between particle and field positions involves determination of nearest grid positions and interpolation weights, which do not change from one gather/scatter call to the next unless the particle positions change. For these linac simulation codes, the particles do not move between the time when charge is scattered onto the charge-density field and when the electric field vectors are gathered back to

Table 2. Run times (seconds) for a fixed problem size (10^5 particles, 256^2 grid)

	Total		Gather/Scatter		FFT		Speedup	
Nodes	R-C	C-C	R-C	C-C	R-C	C-C	R-C	C-C
1	93.4	158.6	38.8	39.4	31.2	83.4	-	-
2	58.9	84.6	20.0	19.9	23.4	25.3	1.59	1.87
4	32.5	45.9	9.7	9.9	13.3	24.3	2.87	3.46
8	17.8	24.9	4.7	4.8	7.1	12.8	5.25	6.40
16	10.9	14.3	2.3	2.4	4.1	6.8	8.57	11.09
32	8.9	10.2	1.3	1.3	3.5	4.7	10.49	15.55

determine the electrostatic force. By caching the interpolation information from the scatter and reusing it during the gather, the gather operations in the 2D POOMA codes are seen to run up to three times faster than the corresponding scatter operation.

Table 3 compares the execution times for the POOMA and HPF versions of the linac simulation code on two different parallel architectures. In addition to the Origin2000 machines at Los Alamos National Laboratory, the codes were run on the Cray T3E at the National Energy Reseach Scientific Computing Center. On the T3E, the POOMA code was compiled with the Kuck and Associates KCC 3.2b2 compiler (version 3.2d), and the HPF code was compiled with the Portland Group HPF compiler (version 2.4). The results in Table 3 are for a 2D simulation of 500,000 particles on a 256^2 grid, and the real-to-complex FFT version of the POOMA code was used. On the T3E, the POOMA version executes from just about the same speed to 50 percent faster than the HPF code. This scaling is not as dramatic as what is observed on the Origin2000 machines, but is consistent with the previous results in that the difference in times is due primarily to faster gather/scatter operations in the POOMA implementation.

Table 3. Run times (seconds) for different architectures (500000 particles, 256^2 grid)

	SGI Origin2000		Cray T3E	
Nodes	R-C	HPF	R-C	HPF
1	291.0	1064.7	473.2	586.6
2	170.1	708.0	263.5	370.6
4	113.5	397.5	143.0	198.9
8	65.9	247.2	80.3	110.8
16	38.7	107.3	50.9	63.6
32	31.8	107.6	36.7	35.8

5 Conclusions

Using the POOMA Framework, a C++ application which models the motion of high-intensity charged particle beams through a linear accelerator has been

developed that runs substantially faster than an equivalent HPF application on a number of different platforms. This performance increase can be attributed in part to the use of a spatial decomposition strategy for the parallel computation in the POOMA version of the code that reduces the parallel communication required during parallel gather and scatter operations, and in part to the use of a real-to-complex FFT algorithm in the POOMA version. The linac simulation code employs an object-oriented design strategy; by using the POOMA Framework as a basis for the development, the design is able to focus on the specific physics abstractions of the accelerator in a modular, extensible manner. POOMA automatically provides the parallel data structures and algorithms, efficient evaluation of data-parallel expressions, and abstractions of the hardware-specific parallel communication issues for the accelerator code.

References

[1] Robert D. Ryne and Salman Habib. Beam dynamics calculations and particle tracking using massively parallel procesesors. *Part. Accl.*, 55:365, 1996.

[2] Graham A. Mark, William F. Humphrey, Julian C. Cummings, Timothy J. Cleland, Robert D. Ryne, and Salman Habib. Modeling particle accelerators using C++ and the POOMA framework. In *ICNSP '98*, February 1998. Santa Barbara, CA.

[3] William Humphrey, Steve Karmesin, Federico Bassetti, and John Reynders. Optimization of data-parallel field expressions in the POOMA framework. In *ISCOPE '97*, December 1997. Marina del Rey, CA.

[4] John Reynders et al. POOMA: A framework for scientific simulations on parallel architectures. In Gregory V. Wilson and Paul Lu, editors, *Parallel Programming using C++*, pages 553–594. MIT Press, 1996.

[5] C. K. Birdsall, A. B. Langdon, and H. Okuda. Finite-size particle physics applied to plasma simulation. *Methods Comput. Phys.*, 9:241–258, 1970.

[6] A. B. Langdon. Energy conserving plasma simulation algorithms. *J. Comput. Phys.*, 12:247–268, 1973.

[7] Todd Veldhuizen. Expression templates. Technical Report 5, C++ Report 7, June 1995.

[8] C. K. Birdsall and D. Fuss. Clouds-in-clouds, clouds-in-cells physics for many-body plasma simulations. *J. Comput. Phys.*, 3:494–511, 1969.

[9] Stephen Lee, Julian Cummings, and Steven Nolen. MC++: Parallel, portable, monte carlo neutron transport in C++. Technical report, Los Alamos National Laboratory, 1996. Document LA-UR 96-4808.

[10] Jean Marshall, John Hall, Lee Ankeny, Sean Clancy, Jodi Heiken, Kathy Holian, Stephen Lee, Guy McNamara, James Painter, and Mark Zander. Tecolote: An object-oriented framework for physics development. Technical report, Los Alamos National Laboratory, April 1998. Document LA-UR 98-1319.

[11] Lois McInnes and Barry Smith. PETSc 2.0: A case study of using MPI to develop numerical software libraries. In *The MPI Developers Conference*, June 1995. Notre Dame, IN.

[12] David Brown, William Henshaw, and Daniel Quinlan. Overture: An object-oriented framework for solving partial differential equations. In *ISCOPE '97*, December 1997. Marina del Rey, CA.