Electron Cooling for an Electron Ion Collider: Computational Methods and Code Development

Béla Erdélyi
Northern Illinois University

DOE-NP Accelerator R&D PI Meeting
November 14, 2016
Electron Cooling of Heavy Ions

- Electron Collider Ring
- Booster
- Ion Collider Ring
- Interaction Point
- Ion Source
- Electron Source
- 12 GeV CEBAF
- SRF Linac
- Dump
- Injector
- Fast kicker
- Circulator ring
- Cooling section
- Ion bunch
- Electron bunch
Main Goals

• **First particle-based** proof of principle **demonstration** of electron cooling simulations

• **Develop a high-performance code** with capabilities in beam dynamics beyond cooling

• **Applications** to electron-ion colliders
## Expenditures and Milestones

<table>
<thead>
<tr>
<th></th>
<th>FY10+FY11</th>
<th>FY12+FY13</th>
<th>FY14+FY15</th>
<th>FY16</th>
<th>TOTALS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Funds Allocated</strong></td>
<td>0+56</td>
<td>55+52=107</td>
<td>50+54=104</td>
<td>50</td>
<td>$317K</td>
</tr>
<tr>
<td><strong>Actual Costs to Date</strong></td>
<td>56</td>
<td>107</td>
<td>104</td>
<td>50</td>
<td>$317K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quarter 1</th>
<th>FY16</th>
<th>FY17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Shared memory parallelization of FMM data structures and integration with parallel FMM</td>
<td>Electron cooling simulations in the JLEIC pre-booster at injection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quarter 2</th>
<th>FY16</th>
<th>FY17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Variable order Picard integrator with automatic step size control parallelization</td>
<td>Electron cooling simulations in the pre-booster at extraction energy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quarter 3</th>
<th>FY16</th>
<th>FY17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Binned time step implementation in parallel</td>
<td>Setup the re-circulator ring optics and bunched cooling in COSY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quarter 4</th>
<th>FY16</th>
<th>FY17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Parallel PHAD integration, benchmarking and optimizations</td>
<td>Study electron beam dynamics in the re-circulator ring, set maximum useful turn limits</td>
</tr>
</tbody>
</table>
Many **collective beam dynamics** effects can be cast in the form of an N-body problem: space charge, intra-beam scattering, electron cloud, beam-beam, beam-plasma, etc.

**Electron cooling** is one of the most challenging:
- Accurate analytical estimates are difficult to come by
- Large particle numbers, but far from statistical limit
- Both attractive and repelling forces
- Close encounters matter
- Relatively slow process
Algorithms in Scientific Computing

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

Algorithms is the Greek word for action. Algorithm is Latin, to be cold. Neither is the root for algorithm, which stems instead from al-Khwārizmī, the name of the ninth-century Arab scholar whose book al-jabr wa’l-muqābalah developed into today’s high school algebra textbooks. Al-Khwārizmī stressed the importance of methodical procedures for solving problems. He wrote: “He was the grand master, he did not neglect the development of the problems in the eponymous approach.”

Some of the very best algorithms of the computer age are highlighted in the January/February 2000 issue of Computing in Science & Engineering, a joint publication of the American Institute of Physics and the IEEE Computer Society. Guest editors Jack Dongarra of the University of Tennessee and Oak Ridge National Laboratory and Fran cis Sullivan of the Center for Computing Sciences at the Institute for Defense Analyses put together a list they call the “Top Ten Algorithms of the Century.”

“We tried to assemble the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century,” Dongarra and Sullivan write. “As with any top-10 list, their selections—and non-selections—are bound to be controversial, they acknowledge. When it comes to picking the algorithmic best, there seems to be no best algorithm.”

Without further ado, here’s the Top Ten list, in chronological order. (Dates and names associated with the algorithms should be read as first-order approximations. Most algorithms take shape over time, with many contributors.)

1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, co-put up the Metropolis algorithm, also known as the Monte Carlo method.

The Metropolis algorithm aims to approximate solutions to numerical problems with unmanageably many degrees of freedom and combinatorial problems of factorial size, by mimicking a random process. Given the digital computer’s reputation for deterministic calculation, it’s fitting that one of its earliest applications was the generation of random numbers.

1947: George Dantzig, at the RAND Corporation, creates the simplex method for linear programming. In terms of widespread application, Dantzig’s algorithm is one of the most successful of all time: Linear programming dominates the world of industry, economic survival depends on the ability to optimize within budgetary and other constraints. “Of course, the ‘real’ problems of industry are often non-linear, the use of linear programming is sometimes dictated by the computer science and an ingenuity of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm is practical; it’s highly efficient—possibly in such ways as modeling interesting the nature of computation.

1950s: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis as the National Bureau of Standards, extend the development of Krylov subspace iteration algorithms. These algorithms address the seemingly simple task of solving equations of the form $Ax = b$. The catch, of course, is that $A$ is a huge $n \times n$ matrix, so that the algebraic answer $x = A^{-1} b$ is not easy to compute. Indeed, matrix “division” is not a particularly useful concept; iterative methods—such as solving equations of the form $x_{k+1} = x_k + b/A$ with a simple matrix $K$ that’s “ideally” close to $A$—lead to the study of Krylov subspaces. Named for the Russian mathematician Nikolai Krylov, Krylov subspaces are spanned by powers of a matrix applied to an initial “remainder” vector $x_0 = b/A$. Lanczos found a way to generate an orthogonal basis for such a subspace when the matrix is symmetric. Hestenes and Stiefel proposed an even faster method, known as the conjugate gradient method, for systems that are both symmetric and positive definite. Over the last 50 years, numerous researchers have improved and extended these algorithms. The current suite includes techniques for non-symmetric systems, with algorithms like GMRES and Bi-CGSTAB. (GMRES and Bi-CGSTAB premiered in SIAM Journal on Scientific and Statistical Computing, in 1986, and 1992, respectively.)

1951: Alan Houser (Householder) of Oak Ridge National Laboratory formulates the decomposition approach to matrix computations. The ability to factor matrices into triangular, diagonal, orthogonal, and other special forms has turned out to be extremely useful. The decompositional approach has enabled software developers to produce flexible and efficient matrix packages. It also facilitates the analysis of rounding errors, one of the big bugs of numerical linear algebra. (In 1961, James Wilkinson of the National Physical Laboratory in London published a seminal paper in the Journal of the ACM, titled “Error Analysis of Direct Methods of Matrix Inversion,” based on the LU decomposition of a matrix as a product of lower and upper triangular factors.)

1957: John Backus leads a team at IBM in developing the Fortran optimizing compiler.

The creation of Fortran may rank as the single most important event in the history of computer programming. Finally, scientists (and others) could tell the computer what they wanted it to do, without having to descend into the netherworld of machine code. Almost immediately, modern compilers began the more than 23,800 assembly language instructions and a standards-compliant compiler was nonetheless capable of surprising sophistication calculations. As Backus himself recalls in a recent history of Fortran I, II, and III, published in 1998 in the IEEE Annals of the History of Computing, the compiler “produced code of such efficiency that it made short work for many engineers.”

1959-61: J.G.F. Francis of Ferranti Ltd., London, finds a stable method for computing eigenvalues, known as the QR algorithm. Eigenvalues are arguably the most important numbers associated with matrices—and they can be the trickiest to compute. It’s relatively easy to transform a square matrix into a matrix that’s “almost” upper triangular, meaning one with a single extra set of nonzero entries just below the main diagonal. But chipping away those final nonzeros, without launching an avalanche of error, is non-trivial. The QR algorithm is just the ticket. Based on the QR decomposition, which writes $A$ as the product of an orthogonal matrix $Q$ and an upper triangular matrix $R$, this approach iteratively changes $A = QR$ into $A = QR^2$, with a few bells and whistles for accelerating convergence to upper triangular form. By the mid-1960s, the QR algorithm had turned once-formidable eigenvalue problems into routine calculations.


Putting $N$ things in numerical or alphabetical order is mind-numbingly mundane. The intellectual challenge lies in devising ways of doing it so quickly. Hoare’s algorithm uses the age-old divide-and-conquer strategy to divide and conquer to solve the problem. Pick one element as a “pivot,” separate the rest into piles of “big” and “small” elements (as compared with the pivot), and then repeat this procedure on each pile. Although it’s possible to get stuck doing all $N(N-1)/2$ comparisons (especially if you use as your pivot the first element or last element you’ve processed), Quicksort runs on average with $O(N \log N)$ efficiency. In elegant simplicity has made Quicksort the poster-child of computational complexity.

1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University and AT&T Bell Laboratories unveil the fast Fourier transform.

Almost the most far-reaching algorithm in applied mathematics, the FFT revolutionized signal processing. The underlying idea goes back to Gauss (who needed to calculate orbits of asteroids), but it was the Cooley–Tukey paper that made it clear how easily Fourier transforms can be computed. Like Quicksort, the FFT relies on a divide-and-conquer strategy to reduce an ostensibly $O(N^2)$ one to $O(N \log N)$ by recursion. But unlike Quicksort, the implementation is at first-intuitive and less than straightforward. This in itself is a very important computational lesson.

1970: Helaman Ferguson and Rodney Fordacre of Brigham Young University advance an integer relation detection algorithm.

The problem is an old one: Given a bunch of real numbers $x_1, x_2, \ldots, x_n$, are there integers $a_1, a_2, \ldots, a_n$ (not all 0) for which $a_1 x_1 + a_2 x_2 + \ldots + a_n x_n = 0$? (For $n = 4$, the venerable algorithm does the job, computing terms in the broad fraction expansion of $x_1/x_2$. If $x_1/x_2$ is rational, the expansion terminates and, with proper unraveling, gives the “smallest” integers $a_i$ and $a_n$. If the Euclidean algorithm doesn’t terminate—or if you simply get tired of computing $b_i$—then the unraveling procedure at least provides a sequence of integers $a_i$ of size $O(N \log N)$. But it’s often possible to use this algorithm to prove that $x_1, x_2, \ldots, x_n$ is linearly independent, and, in some cases, to prove that they are independent. This is itself a very important computational lesson.

1977: Leslie Greengard and Vladimir Rokhlin of Yale University invent the fast multipole method.

This algorithm overcomes one of the biggest headaches of N-body simulations: the fact that accurate calculations of the motions of N particles interact via gravitational or electric or electromagnetic forces (think stars in a galaxy, or atoms in a protein) would seem to require $O(N^2)$ computations—one for each pair of particles. The fast multipole method gets by with $O(N \log N)$ computations. It does so by using multipole expansions (not charge or mass, dipole moment, quadrupole, and so forth) to approximate the effects of a distant group of particles on a local group. A hierarchical decomposition of space is used to define ever larger groups as distances increase. One of the distinct advantages of the fast multipole algorithm is that it comes equipped with rigorous error estimates, a feature that many methods lack.

What new insights and algorithms will the 21st century bring? The complete answer obviously won’t be known for another hundred years. One thing seems certain, however. As squirrel writes in the introduction to the top-10 list, “The new century is not going to be very useful for us, but it is not going to be dull either!”

Barry A. Cipra is a mathematician and writer based in Northfield, Minnesota.
... the dynamics of particle accelerators actually motivated the construction of the first symplectic integrators (Ruth 1983)

Combine: Numerical integrators with automatic differentiation

Combine: fast multipole methods with automatic differentiation
Main Challenges of an N-Body Solver

- **Efficient Force Computation**
  - Adaptive hierarchical space decomposition

- **Accurate Time Stepper**
  - Variable high order, adaptive integrators with automatic steps size and order selection, and dense output

- **Ability to deal with very large N**
  - Distributed, high performance computing on hybrid architecture supercomputers

- **Ability to deal with long time-scale dynamics**
  - Time does not parallelize
“Exact” (scales as $O(N^2)$):

- Particle-particle

“Fast” methods (scale as $< O(N^2)$):

- Basis-function method
  - Orthogonal polynomials

- Grid-based methods:
  - PIC, particle-mesh, FFT-based
  - Multi-grid

- Hierarchical space decomposition:
  - Tree: cell-particle
  - Cluster: particle-cell
  - Fast multipole: cell-cell

\[
\vec{f}_i = \nabla \sum_{j=1}^{N} K(\vec{r}_i, \vec{r}_j) q_j = \nabla \sum_{i=1}^{N} K(\vec{r}_i - \vec{r}_j) q_j = \nabla \sum_{i=1}^{N} K_{ij} q_j \quad i = 1, N.
\]
The Fast Multipole Method

Rokhlin and Greengard arguably provided the first numerically defensible method for reducing the N-body problem’s computational complexity to $\mathcal{O}(N)$.

Multipole methods and their descendants will be ubiquitous. – L.N. Trefethen (Oxford) Predictions for Scientific Computing Fifty Years From Now (Mathematics Today, 2000)
A Picture Is Worth a Thousand Words
Multiple Gaussians in Linear Time and Memory
Time-Stepping Algorithms

- Low vs. high order
- Constant step size vs. adaptive
- Single step vs. multi-step
- Explicit vs. implicit
- Symplectic vs. non-symplectic
- Reversible vs. non-reversible
- Single level vs. hierarchical
- Serial vs. parallel
3.3 Block-step procedure

Let us examine the sequential procedure for one block-step.

(i) Obtain the next time for integration,
\[ t_{\text{next}} = \min_{1 \leq i \leq N} (t_{1,i} + \Delta t_{1,i}), \]
where \( t_{1,i} \) and \( \Delta t_{1,i} \) are the time of the last irregular force calculation and irregular time-step of particle \( i \).

(ii) Make the active particle list for regular and irregular force calculation,
\[ \mathcal{L}_{\text{act},R} = \{ i | t_{R,i} + \Delta t_{R,i} = t_{\text{next}} \}, \]
\[ \mathcal{L}_{\text{act},I} = \{ i | t_{I,i} + \Delta t_{I,i} = t_{\text{next}} \}, \]
where the subscripts \( R \) and \( I \) denote regular and irregular terms. Here, \{ \{ i | \text{cond.} \} \} defines a set of \( i \) such that it satisfies \( \text{cond.} \).

(iii) Predict all particles needed for force evaluation.
(iv) Calculate the irregular force and its time derivative for particle \( i \in \mathcal{L}_{\text{act},I} \),
\[ F_{i,i} = \sum_{j \in \mathcal{L}_i} m_j \frac{R_{ij}}{|R_{ij}|^3}, \]
\[ \dot{F}_{i,i} = \sum_{j \in \mathcal{L}_i} m_j \left[ \frac{\dot{R}_{ij}}{|R_{ij}|^3} - 3 \frac{(R_{ij} \cdot \dot{R}_{ij})R_{ij}}{|R_{ij}|^5} \right]. \]

(v) Apply the corrector for the active irregular particles and decide the next time-step \( \Delta t_{1,i} \).

(vi) Accumulate the regular force and its time derivative for each active regular particle \( i \in \mathcal{L}_{\text{act},R} \) and construct the neighbour list,
\[ F_{R,i} = \sum_{j \neq i}^{N} \begin{cases} m_j \frac{R_{ij}}{|R_{ij}|^3} & (R_{ij,\text{min}} > h_i) \\ 0 & \text{(otherwise)} \end{cases}, \]
\[ \dot{F}_{R,i} = \sum_{j \neq i}^{N} \begin{cases} m_j \left[ \frac{\dot{R}_{ij}}{|R_{ij}|^3} - 3 \frac{(R_{ij} \cdot \dot{R}_{ij})R_{ij}}{|R_{ij}|^5} \right] & (R_{ij,\text{min}} > h_i) \\ 0 & \text{(otherwise)} \end{cases}, \]
\[ \mathcal{L}_i = \{ j | j \neq i, R_{ij,\text{min}} < h_i \}. \]

(vii) Execute the regular corrector. Since the neighbour list \( \mathcal{L}_i \) has been updated, the force polynomials should be corrected to reflect the difference between the old and new list.

---

Accelerating NBODY6 with Graphics Processing Units

Keigo Nitadori\(^1\)* and Sverre J. Aarseth\(^2\)*

\(^1\)Center for Computational Science, University of Tsukuba, 1-1-1, Tennodai, Tsukuba, Ibaraki 305-8577, Japan
\(^2\)Institute of Astronomy, University of Cambridge, Madingley Road, Cambridge, CB3 0HA, UK
Proposition. Assume that the function $h \mapsto x(t_m + h)$ is analytic on a disk of radius $\rho_m$, and that there exists a positive constant $M_m$ such that

$$|x_m^{[j]}| \approx \frac{M_m}{\rho_m^j}, \quad \forall j \in \mathbb{N}.$$ 

Then, if the required accuracy $\varepsilon$ tends to 0, the optimal value of $h$ that minimizes the number of operations tends to

$$h_m = \frac{\rho_m}{\varepsilon^2},$$

and the optimal order $p_m$ behaves like

$$p_m = \frac{1}{2} \ln \left( \frac{\varepsilon}{M_m} \right) - 1.$$
**Examples**

<table>
<thead>
<tr>
<th>Energy</th>
<th>Time bins of equal time widths</th>
<th>Time bins of equal number of particles</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 TeV</td>
<td><img src="#" alt="Graph 1" /></td>
<td><img src="#" alt="Graph 2" /></td>
</tr>
<tr>
<td>1 MeV</td>
<td><img src="#" alt="Graph 3" /></td>
<td><img src="#" alt="Graph 4" /></td>
</tr>
</tbody>
</table>

For accuracies of $10^{-7}$ and $10^{-16}$, time step sizes used in each step are the same, while orders vary between 2 and 7.
Simo Integrator
Hardware Trends
Our findings suggest that, depending on system size and desired accuracy, the FMM- and FFT-based methods are most efficient in performance and stability.
FMM World Record

3,011,561,968,121 particles

Number of Cores

<table>
<thead>
<tr>
<th>Method</th>
<th>1</th>
<th>294,912</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMM</td>
<td>220 days</td>
<td>11 minutes</td>
</tr>
<tr>
<td>Direct</td>
<td>1 billion years</td>
<td>32,000 years</td>
</tr>
<tr>
<td>PC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUGENE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Credit: Jülich Supercomputing Centre (JSC)
We are developing a parallel code (PHAD) based on these new methods that will be the first one capable of particle-based simulations of electron cooling and other difficult beam dynamics phenomena with high fidelity, efficiently.

http://niu.edu/beamphysicscode/
First Particle-Based Cooling Simulation
Computational beam physics plays an important part in modeling and simulating electron cooling; designing, operating, and improving current and future particle accelerators and their performance.

Algorithmic and hardware improvements multiply, making high fidelity large-scale problems feasible.

Fundamental algorithms and methods are general enough to be adaptable/applicable to many other beam dynamics problems and different scientific fields:

Current and next generation high-performance computing systems are well matched to these algorithms.

Entering a new phase of high fidelity electron cooling simulations.