Intro. Lecture 08: Practical Considerations*

Prof. Steven M. Lund
Physics and Astronomy Department
Facility for Rare Isotope Beams (FRIB)
Michigan State University (MSU)

US Particle Accelerator School (USPAS) Lectures On
"Self-Consistent Simulations of Beam and Plasma Systems"
Steven M. Lund, Jean-Luc Vay, Remi Lehe, and Daniel Winklehner

US Particle Accelerator School Summer Session
Colorado State U, Ft. Collins, CO, 13-17 June, 2016
(Version 20160616)

* Research supported by:
FRIB/MSU, 2014 On via: U.S. Department of Energy Office of Science Cooperative Agreement DE-SC0000661and National Science Foundation Grant No. PHY-1102511
and
LLNL/LBNL, Pre 2014 via: US Dept. of Energy Contract Nos. DE-AC52-07NA27344 and DE-AC02-05CH11231

Detailed Outline
Introductory Lectures on Self-Consistent Simulations

Practical Considerations:
A: Overview

Intense beam simulations can be highly demanding on computer resources – particularly for higher dimensional models with detailed geometries. The problem size that can be simulated is dictated by computer resources available in fast memory and the run time required to complete the simulation

- Fast Memory (RAM)
- Wall Clock Run Time (Computer Speed)

Both of these can depend strongly on the architecture of computer system that the problem is run on:

- Serial Machine
- Parallel Machine

We will present rough estimates of the computer memory required for simulations and provide some guidance on how the total simulation time can scale on various computer systems. The discussion is limited to PIC and direct Vlasov simulations.

B: Fast Memory

Fast computer memory (RAM) dictates how large a problem can be simulated

- If a problem will not fit into fast memory (RAM), computer performance will be severely compromised
- Writes to hard disks (swap memory) are slow and greatly increase run time

There are 3 main contributions to the problem size for typical PIC or direct Vlasov simulations:

1) Particle Phase Space Coordinates (PIC)
   or Discretized Distribution Function (Direct Vlasov)
2) Gridded Field
3) General Code Overhead

These three contributions to memory required are discussed in turn

Particle and field quantities are typically stored in double precision:

Representation	Digits (Floating Point)	Bytes Memory
Single Precision	8	4
Most problems: Double Precision	16	8

Estimates of Required Fast Memory

1) Particle Phase Space Coordinates (PIC):

B = bytes of floating point number (typically 8 for double precision)
N_p = number macro particles (0 for direct Vlasov)
D = dimension of variables characterizing macro particles: $\mathbf{x}$, $\mathbf{p}$ etc.

Memory = $B * N_p * D$ Bytes

The dimension D depends on the specific type of PIC simulation and methods employed
// Common Examples of D:

3D PIC: $D = 7$	2D Transverse Slice PIC: $D = 5$	
$x,\ y,\ z$	$x,\ y$	
$p_x,\ p_y,\ p_z,\ \gamma^{-1}$	$p_x,\ p_y, \gamma^{-1}$	2 ½ D Case: $+p_z$ (D=6)

- γ^{-1} is often included to optimize the mover
- Sometimes additional particle arrays are stored for various tasks/flags //

Estimates of Required Fast Memory

1) Discretized Distribution Function (Direct Vlasov):

B = bytes of floating point number (typically 8 for double precision)
N_{pm} = number mesh points of grid describing the discretized particle phase space

Memory = $B * N_{pm}$ Bytes

The value of N_{pm} depends critically on the dimensionality of the phase space
// Examples of N_{pm} scaling for a uniform phase-space meshes:

Problem	Phase Space	N_{pm}	Scaling ($n_x = n_{p_x} \equiv n$ etc)
1D $\parallel$	$z - p_z$	$n_z n_{p_z}$	n^2
2D $\perp$ Slice	$x - p_x,\ y - p_y$	$n_x n_y n_{p_x} n_{p_y}$	n^4
$2\frac{1}{2}$D Slice	$x - p_x,\ y - p_y,\ p_z$	$n_x n_y n_{p_x} n_{p_y} n_{p_z}$	n^5
3D	$x - p_x,\ y - p_y,\ z - p_z$	$n_x n_y n_z n_{p_x} n_{p_y} n_{p_z}$	n^6

n_x = number mesh points in x etc.

Rapid growth of N_{pm} with dimensionality severely limits mesh size

Memory required for a double precision ($B = 8$) uniform phase-space grid with 100 zone discretization per degree of freedom:

$n_x = n_{p_x} \equiv n = 100$ etc.
D = dimension of phase-space

Memory $= 8 * n^D$ Bytes

Problem	D	Memory (Bytes) (n = 100)
1D $\parallel$	2	$80 \times 10^3 \sim 80$ KB
2D $\perp$ Slice	4	$800 \times 10^6 \sim 800$ MB
$2\frac{1}{2}$D Slice	5	$80 \times 10^9 \sim 80$ GB
3D	6	$8 \times 10^{12} \sim 8000$ GB $= 8$ TB

//

Rapidly increasing problem size with phase-space dimension D practically limits what can be simulated on direct Vlasov simulations with reasonable resolution *even on large parallel computers*:

- Irregular phase-space grids that place resolution where it is needed can partially alleviate scaling problem: provides more help in higher dimensions
- Improved methods also seek to only grid minimal space exterior to the oscillating beam core in alternating gradient lattices

2) Gridded Field:

Required memory for a gridded field solve depends on the class of field solve (electrostatic, electromagnetic), mesh size, and numerical method employed. For a concrete illustration, consider *electrostatic* problems using a simple FFT field solve:

- Discrete Fourier Transform complex, but transform is of real functions. Optimization allows use of transforms using only real ϕ and ρ arrays
- Electric field is typically not stored and is calculated for each particle only where it is needed. Spatial grid location need not be stored.
  - Some methods store gridded E to optimize specific problems

N_{fm} = number mesh points of field spatial grid

Memory = $2 * B * N_{fm}$ Bytes

Factor of 2 for: ρ, ϕ

Number of mesh points N_{fm} depends *strongly* on the dimensionality of the field solve and the structure of the mesh

- Generally more critical to optimize storage and efficiency (see next section) of fieldsolvers in higher dimensions

Examples for uniform meshes:

$N_{fm} = n_z$ 1D (Longitudinal)

$= n_x n_y$ 2D (Transverse Slice)

$= n_r n_z$ 2D (r-z Axisymmetric)

$= n_x n_y n_z$ 3D

n_x = number mesh points in x, etc.

3) General Code Overhead:

System memory is also used for:

- Scratch arrays for various numerical methods (fieldsolvers, movers, etc.)
- History accumulations of diagnostic moments
- Diagnostic routines
- Graphics packages, external libraries, etc.
  - Graphics packages can be large!

Memory = $M_{overhead}$ Bytes

Characteristic of packages used, size of code, and methods employed. But typical numbers can range 1 MB – 20 MBytes

Summary: Total Memory Required:

For illustrative example, add memory contributions for electrostatic PIC

PIC:	Total Memory = $B * (N_p * D + 2 * N_{fm}) + M_{overhead}$	Bytes
Direct Valsov:	Total Memory = $2 * B * (N_{pm} + N_{fm}) + M_{overhead}$	Bytes

Reminder: Machine fast memory (RAM) capacity *should not be exceeded*

- Storing data on disk (swap memory) and cycling to RAM is generally far too slow!
- For optimal performance may want parts of the problem to fix in fast cache memory of the processor which is typically much more limited
- Special considerations may be relevant to parallel machines that use shared or common memory between the processors

C: Run Time

Run time can depend on many factors including:

- Type of problem
- Dimensionality of problem and number of particles and/or mesh points
- Numerical methods employed (particle moving, fieldsolve, ...)
- Moments and diagnostics accumulated
- Architecture/speed of computer system

It is not possible to give fully general guidance on estimating run times. However, to better characterize the time required, it can be useful to benchmark the code on the computer to be employed in terms of:

t_{step} = Time for an "ordinary" run step

Generally, parts of the code that more time is spent in should be more carefully optimized to minimize total run time. Particular care should be taken with:

- Particle mover
- Field solver
- Weighting particles/fields to and from the grid
- Frequent computationally intense diagnostics such as moments

Diagnostics, loaders, problem setup routines, etc. can often be coded with less care for optimization since they are only executed infrequently. However:

- Diagnostics often take a large amount of development time
  - Even at the expense of efficiency, it may be better to code as Simply/clearly as possible to make easier to maintain

Software profiling tools can help understand where "bottlenecks" occur so effort on optimization can be appropriately directed for significant returns.

Example: output of Warp timers from a large multi-species x-y slice simulation

```
File  Edit  View  Search  Terminal  Help
>>> printtimers()
                     Total time          Time per step
                        (s)                 (s)
Generate time          1.0856
Step time            807.2523              0.4258
Plot time              0.0003              0.0000
Moments time          37.0052              0.0195
Field Solve time     100.0438              0.0527
Deposition time        0.0000              0.0000
Applied field time   203.9678              0.1075
Dump time              0.0000              0.0000
      aftergenerate      0.0000      0.0000      0.0000      0.0000
      beforeloadrho      6.8198      6.8198      0.0000      0.0036
 callbeforestepfuncs     0.0060      0.0060      0.0000      0.0000
  callafterstepfuncs    25.6206     25.6206      0.0000      0.0135
 callbeforeplotfuncs     0.0014      0.0014      0.0000      0.0000
  callafterplotfuncs     0.0017      0.0017      0.0000      0.0000
   callplseldomfuncs     0.0000      0.0000      0.0000      0.0000
   callplalwaysfuncs     0.0000      0.0000      0.0000      0.0000
      userinjection      0.0248      0.0248      0.0000      0.0000
      beforeloadrho adjustments_before_rho     6.7842      6.7842      0.0000      0.0036
  callafterstepfuncs diag_calls     3.4528      3.4528      0.0000      0.0018
  callafterstepfuncs diag_hist_hl    22.1173     22.1173      0.0000      0.0117
>>>
```


Dimensionality plays a strong role in required run time

Some rough guidance for *electrostatic* PIC Simulations:

1D: (Longitudinal typical)
Fieldsolve generally fast: small fraction of time compared to moving particles

- Green's function methods can be used (Gauss Law) with high efficiency (sum charge to right and left), so no need for gridded solver

2D: (Transverse xy slice and axisymmetric r-z typical)
Fieldsolve typically a small fraction of time relative to moving particles if fast gridded methods are applied (like FFT based methods)

- Special boundary conditions can increase the fraction

Method	Numerical Work
FFT with Periodic BC	Small fraction of particle moving
FFT with Capacity Matrix	.
Multigrid	.
.	.
Green's Function	Dominates particle moving

3D:
Fieldsolve typically comparable in time or dominates time for particle moving even when fast, gridded methods are applied at modest resolution

- Fieldsolve efficiency of *critical* importance in 3D to optimize run time
- Whole classes can be taught just on methods of 3D electrostatic field solves for the Poisson equation $\nabla^2\phi = -\rho/\epsilon_0$ discretized on a mesh

Guidance for Direct Vlasov Simulations:

The rapid growth of the problem size with the phase space-dimension and available fast computer memory can severely limit problem sizes that can be simulated:

- Numerical work can be significant to advance the discretized distribution over characteristics
- Size of gridded field arrays can be very large leading to slow advances
  - Nonuniform mesh can help control size at the expense of code complexity

The type of computer system employed can also strongly influence run time

- Processor Speed
- Memory Speed
  - RAM
  - Fast, optimized cache memory
- System Architecture (see next section)
  - Serial
  - Parallel
- Library Optimization
  - Especially relevant for parallel machines

D: Machine Architectures

Problems may be simulated on:

1) Serial Machines
   - Single processor or an independently run processor on a multi-processor machine (example: most present multi-"core" processors)

2) Parallel Machine
   - Multi-processors coordinated to work as a large single processor
   - Usually employ independent memory for each processor making up the machine but sometimes uses shared memory among processors

Serial machines represent traditional computers (PCs, workstations, etc), whereas parallel machines are generally less familiar.

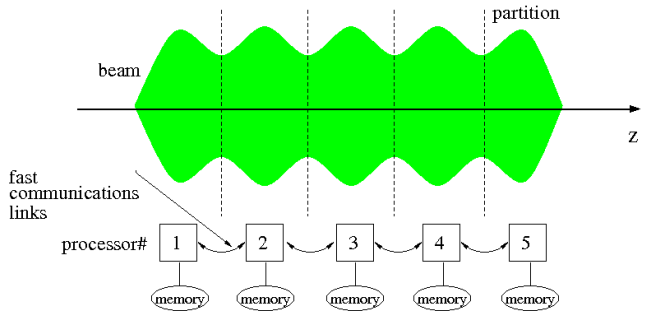
Overview of parallel simulations:

Parallel machines now have developed libraries that allow more "natural" problem formulation with less effort. This is enabling significantly larger simulations to be carried out.

- Several 100 million particles typically practical to simulate on large machines with fast, gridded fieldsolve

Typical Parallel Machine Architecture

Beam problems often can be conveniently partitioned among processors in terms of axial slices. Schematic example (5 processors):



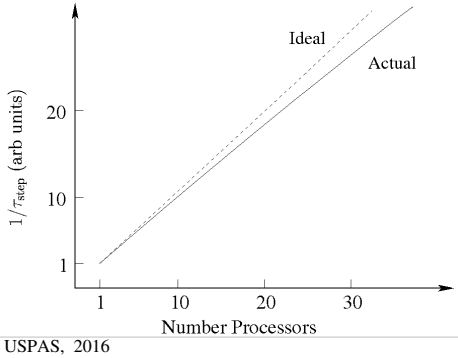
- Sharing of data at boundaries is necessary for fieldsolve in electrostatic case
- Problems with axial velocity spread generally requires sorting of particles to maintain the load balance between processors
  - Processors should ideally all perform an equal amount of work since the slowest will dictate the total time of the advance step

Ideal parallelization results in a linear speedup with processor number

- Actual speedup typically less due to:
  - Overhead in data transfers
  - Lack of ideal load balance causing processors to wait on the slowest one that the problem is partitioned among

$$\tau_{\rm step} = \text{Time "ordinary" step in compuational cycle}$$

$$t_{\rm sim} = \frac{\text{Simulation Time}}{\tau_{\rm step}}$$



Even with the significant advances in problem size and speed promised by parallel computers, the solution of realistic 3D beam problems with direct (not gridded) fields remains far too large a problem to simulate with present computer systems. Also, the continuum limit Vlasov solution to problems is in itself of intrinsic interest as a well posed model of many physical systems. Thus, for detailed simulations, we often push computer resources to the maximum extent possible.

- Better numerical algorithms
- Parallelization
- ....

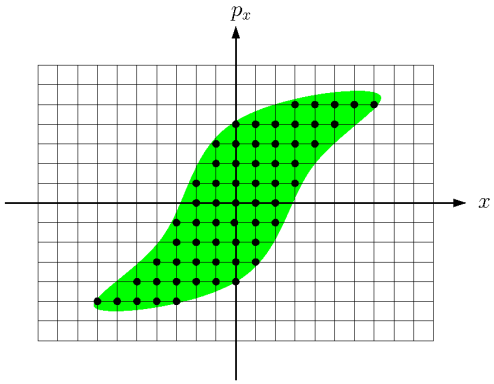
Appendix A: Direct Vlasov Simulations

Here we briefly outline mesh choices for direct Vlasov simulations since very little has been discussed on this topic in the lectures. For direct Vlasov simulations one needs to advance the distribution at discretized locations in phase-space.

Direct Vlasov as an example:

Discretize grid points $\{\boldsymbol{x}_i, \boldsymbol{p}_i\}$

Advance distribution $f(\boldsymbol{x},\boldsymbol{p},t)$ at discrete grid points in time



A1

For (simple) direct Vlasov methods:

- Fields are solved using a discrete spatial mesh as for PIC methods
  - Deposition on mesh is straightforward (f known on mesh already)
- Distribution advance cycle is different than for PIC methods
  - Numerical stability is key
  - Characteristics and "semi-Lagangian" methods can be employed
  - Methods for solving from characteristics are familiar from dynamics/plasma physics

"Pros of Method"

- Low Noise: only discretization effects without statistical noise
- Allows clear analysis of collective effects and tenuous distribution components

"Cons of Method"

- Extreme memory requirements for needed grid resolution in multi-dimensional phase-space
- Numerical stability tends to be more difficult than in particle simulations

Unfortunately, inadequate time to illustrate direct Vlasov distribution methods in this introductory course. However, it is straightforward to develop simple direct Vlasov methods from methods used in PIC simulations.

Corrections and suggestions for improvements welcome!

These notes will be corrected and expanded for reference and for use in future editions of US Particle Accelerator School (USPAS) and Michigan State University (MSU) courses. Contact:

Prof. Steven M. Lund
Facility for Rare Isotope Beams
Michigan State University
640 South Shaw Lane
East Lansing, MI 48824

lund@frib.msu.edu
(517) 908 – 7291 office
(510) 459 - 4045 mobile

Please provide corrections with respect to the present archived version at:

https://people.nscl.msu.edu/~lund/uspas/scs_2016