PHAD Documentation

Revised September 29, 2021

Contents

1 Getting Started 1

1.1 Introduction 1

1.2 Data Files 1

1.2.1 Input Data Files 1

1.2.2 Other Input Data Files 2

1.2.3 Output Data Files 4

1.3 Window and Relaunch Capabilities 4

1.4 Running PHAD 4

2 Overview of the PHAD Algorithm 6

2.1 PHAD Code Flowchart 6

2.2 Equations of Motion 7

2.3 Strang Splitting 8

2.4 The Fast Multipole Method 9

2.4.1 The Regular Long Range Integrator 10

2.4.2 The Modified Long Range Integrator 11

2.5 The Simò Integrator 12

3 Memory and Subroutine Details 13

3.1 Global Variables 13
3.2 Function QSORT 14
3.3 Function MERGE 15
3.4 Function COMPARE 16

List of Figures

2.1 Code flowchart 6

3.1 The influence of the number of processors, number of particles, and the q value on the memory required by PHAD code. 14

List of Data Files

1.2.1 positions.ssv 2

1.2.2 momenta.ssv 2

1.2.3 masses.dat/charges.dat 2

1.2.4 phad-mpi.input and phad-mpi.input.relaunch 3

Chapter 1

Getting Started

1.1 Introduction

The Particles' High-order Adaptive Dynamics (PHAD) is a novel collisional N-body method for accurate and efficient simulations of charged particle beam dynamics written in COSYScript for MSU COSY Infinity v9.2 and higher.. Modeling beam dynamics with PHAD achieves an accuracy that is comparable to the direct methods, but with a computational time that is proportional to N (number of particles) instead of N^2 of the direct methods. In addition, PHAD can preserve the symplecticity of the long-time scale simulations up to machine precision. The main components of PHAD algorithm are Strang splitting which separates the forces to near and far forces and ensures symplecticity; the Fast Multipole Method (FMM) that reduces the computational complexity of the pair-wise forces in the far region; and the Simò integrator for an accurate time integration that can resolve all collisions in the near region efficiently. An earlier version of PHAD utilized a Picard-Iteration based integrator for the time stepping as described in this dissertation.

1.2 Data Files

There are three kinds of data files used in PHAD: the initial data files are those containing inputs; the pass through data files are used to communicate between the COSY execution and the C++ execution of the FMM; and the output data files are those containing outputs.

1.2.1 Input Data Files

The initial coordinates, masses, and charges of the particles are included in four data files.

- File positions.ssv contains the initial positions of the particles.

$$\begin{array}{ccc} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{array}$$

Data File 1.2.1: positions.ssv

- File momenta.ssv contains the initial scaled momentum of the particles.

$$\begin{array}{ccc} \hat{p}_{x_1} & \hat{p}_{y_1} & \hat{p}_{z_1} \\ \hat{p}_{x_2} & \hat{p}_{y_2} & \hat{p}_{z_2} \\ \hat{p}_{x_3} & \hat{p}_{y_3} & \hat{p}_{z_3} \\ \vdots & \vdots & \vdots \\ \hat{p}_{x_N} & \hat{p}_{y_N} & \hat{p}_{z_N} \end{array}$$

Data File 1.2.2: momenta.ssv

- Files charges.dat and masses.dat contain the particles scaled charges n_i and masses f_i.

$$\begin{array}{c} f_1 \,/q_1 \\ f_2 \,/q_2 \\ f_3 \,/q_3 \\ \vdots \\ f_N \,/q_N \end{array}$$

Data File 1.2.3: masses.dat/charges.dat

1.2.2 Other Input Data Files

The file phad-mpi.input specifies many input parameters for the code such as the files' names and the locations of the initial data and output data. An additional file, phad-mpi.input.relaunch, is needed for the relaunch functionality (explained in Section1.3). This file is identical to phad-mpi.input, but with the final entry set to '1' instead of '0' (Note, when restarting PHAD, both files will be identical with the last entry set to '1'). An example of these files is shown in 1.2.4.

tmp/	Location to store temporary FMM data files (*outputpath*)
sources.ssv	Path and file name with initial particles' positions (*sourcefile*)
input/charges.dat	Path and file name with initial particles' charges (*chargefile*)
10000	Number of particles (*countsources*)
targets.ssv	Path and file name with targets' locations for FMM (*targetfile*)
10000	Number of FMM target locations (*counttargets*)
60	Largest number of particles allowed in neighborhood (q)
./fmmcpp.intel	Path and file name of FMM executable (*fmmcpp_program*)
6	Order of FMM (*fmm_order*)
0	Format of input files: 0 = asci, 1 = binary, 2 = cosy binary (*binary_input*)
1	Switch for load balancing for FMM C++ data structure: 0 = no, 1 = yes (*load_balance*)
10000	Maximum number of boxes (*max_num_of_boxes*)
input/masses.dat	Path and file name with particles' masses (*massfile*)
input/momenta.ssv	Path and file name with initial particles' momenta (*momentafile*)
10	Maximum allowed order for Simò (*SIMO_ORDER*)
1E−10	Accuracy of Simò (*SIMO_ACCURACY*)
1.75E−15	Minimum timestep size for Simò (*DELTALIMIT*)
10	Number of Simò time bins (*BINS*)
1	Type of binning in the Simo integrator: 0 = equal-width bins, 1 = equal number of particles per bin (*BINNINGTYPE*)
0.5E−2	PHAD timestep size (*timestepsize*)
600	Total number of PHAD timesteps to perform (*num_of_timesteps*)
20	One output per this many timesteps (*OUTPUTERESOLUTION*)
−5E−4	Minimum horizontal window dimension (*XMIN*)
5E−4	Maximum horizontal window dimension (*XMAX*)
−5E−4	Minimum vertical window dimension (*YMIN*)
5E−4	Maximum vertical window dimension (*YMAX*)
0	Number of electrons for cooling applications; 0 = no cooling (*num_of_cool_electrons*)
0	Switch for relaunch: 0 = no relaunch, 1 = relaunching PHAD (*apprelaunch*)

Data File 1.2.4: phad-mpi.input and phad-mpi.input.relaunch

1.2.3 Output Data Files

After each *OUTPUTERESOLUTION* number of timesteps, the code will output the particles' information to the data files <variable>_PHAD_<timestep>.ssv. The <variable> denotes the following: x, y, z, $\hat{p}_x$, $\hat{p}_y$, $\hat{p}_z$, f, n, and *beta*.

1.3 Window and Relaunch Capabilities

Charged particle beams travel within a transversely confined region (ie. the beam pipe of the accelerator complex). If particles leave this region, they are no longer relevant to the simulation and should be removed in order to improve the accuracy (and efficiency) of the algorithm. PHAD defines a 'window' in the transverse space in which the simulation will be performed. Any particles that leave the window at a timestep are removed from the simulation, and their information is logged in the output files <variable>_PHAD_<timestep>.ssv of the respective step. The parameters defining the window are set in the file phad-mpi.input.

PHAD is specifically designed to efficiently compute problems of large N accurately and efficiently (such as electron cooling applications), so the assumption is that it will be run in parallel on a high-performance computing (HPC) cluster. Job management systems on some HPCs require the user to specify apriori the length of time the simulation will run. Due to the adaptive nature of the Simò integrator and PHAD timesteps, timing information cannot be known. Therefore, PHAD has a built in relaunch capability which allows the user to restart the code from the last completed PHAD timestep using a flag in the file phad-mpi.input. This flag is *apprelaunch*: 0 = no relaunch, 1 = relaunching PHAD.

1.4 Running PHAD

PHAD is written in COSYscript to be run in parallel using the MPI version of COSY Infinity. COSY Infinity can be obtained after registering at the COSY website. The COSY Infinity Programmer's Manual details how to create the MPI version of COSY through the use of the version utility. Some of these details are also covered in the NIU COSY User Guide.

After extracting phad.zip, the following files need to be included in the directory

1. phad.fox
2. COSY.bin
3. fmmcpp.zip
4. A directory called input containing input data files listed in Section1.2.1

5. A directory called output for resulted output data files described in Section1.2.3
6. phad-mpi.input and phad-mpi.input.relaunch files described in Section1.2.2

The COSY.bin file can be created using MPI COSY executable to execute the cosy.fox file within the same directory. The fmmcpp.zip directory is used to build the FMM, after extracting, by running the `make` command in this directory.

In most beam dynamics applications, the sources.ssv and targets.ssv are equivalent and can be set to input/positions.ssv. Any external electromagnetic fields can be included in the phad.fox file within the Simò integrator procedure.

Assuming the MPI COSY executable is 'mpi_cosy', the following basic command can be used to run PHAD in parallel

```
mpirun -np <number of processors> mpi_cosy phad
```

However, PHAD is best to be run using a pbs script (the file run_phad.pbs is included to run PHAD on Gaea at NIU).

Chapter 2

Overview of the PHAD Algorithm

2.1 PHAD Code Flowchart

The following flowchart gives an outline of the major components of the code

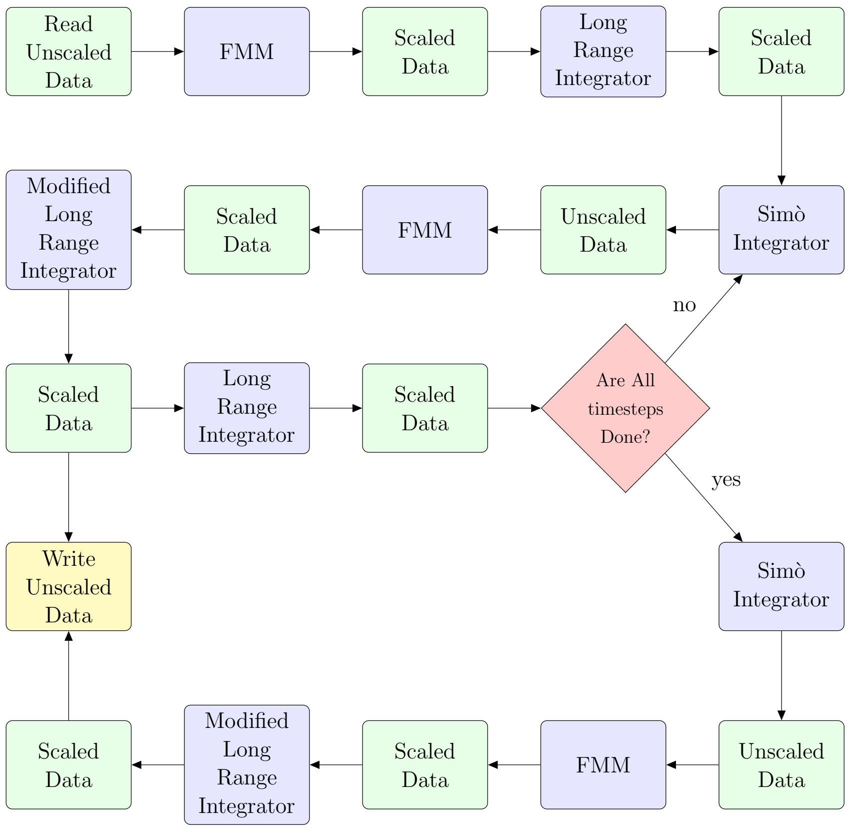


Figure 2.1: Code flowchart

The Scaled Data that is passed between components is that described in Section2.2. The Unscaled Data passed to the FMM has z coordinates multiplied by γ as also described in Section2.2.

2.2 Equations of Motion

The equations of motion for a particle i of charge q_i and mass m_i can be written as

$$\dot{\mathbf{x}}_i = \frac{c\left(p_{x_i}\mathbf{i} + p_{y_i}\mathbf{j} + p_{z_i}\mathbf{k}\right)}{\left(m_i^2c^2 + p_{x_i}^2 + p_{y_i}^2 + p_{z_i}^2\right)^{1/2}} \tag{2.1}$$

and

$$\dot{\mathbf{p}}_i = q_i\left(\frac{1}{4\pi\epsilon_0}\sum_{\substack{j=1\\j\neq i}}^{N}\frac{q_j\left((x_i-x_j)\mathbf{i} + (y_i-y_j)\mathbf{j} + \gamma^2(z_i-z_j)\mathbf{k}\right)}{\gamma\left[(x_i-x_j)^2 + (y_i-y_j)^2 + \gamma^2(z_i-z_j)^2\right]^{3/2}} + \mathbf{E} + \mathbf{v}_i\times\mathbf{B}\right), \tag{2.2}$$

where γ is the Lorentz relativistic factor, c is the speed of light in vacuum, ϵ_0 is the vacuum permittivity, and $\mathbf{E}$ and $\mathbf{B}$ are the external electric and magnetic fields, respectively. The derivation of these equations of motion is detailed in this dissertation.

Each equation of motion will be scaled to avoid numerical errors due to the small quantities included in the simulations. This scaling will be denoted using the $\hat{\cdot}$ notation. After scaling, the equations of motion are given by

$$\dot{\mathbf{x}}_i = \frac{\hat{p}_{x_i}\mathbf{i} + \hat{p}_{y_i}\mathbf{j} + \hat{p}_{z_i}\mathbf{k}}{\left(f_i^2 + \hat{p}_{x_i}^2 + \hat{p}_{y_i}^2 + \hat{p}_{z_i}^2\right)^{1/2}}, \tag{2.3}$$

and

$$\dot{\hat{\mathbf{p}}}_i = \frac{qn_i}{mc^2}\left(\frac{q}{4\pi\epsilon_0}\sum_{\substack{j=1\\j\neq i}}^{N}\frac{n_j\left((x_i-x_j)\,\mathbf{i} + (y_i-y_j)\,\mathbf{j} + \gamma^2\,(z_i-z_j)\,\mathbf{k}\right)}{\gamma\left[(x_i-x_j)^2 + (y_i-y_j)^2 + \gamma^2\,(z_i-z_j)^2\right]^{3/2}} + \mathbf{E} + c\hat{\mathbf{v}}_i\times\mathbf{B}\right). \tag{2.4}$$

Setting m to be the mass of a proton and q to be the elementary charge, the scaled variables are defined as follow

$$\begin{aligned}
\hat{t} &= c\,t & f_i &= \frac{m_i}{m} & n_i &= \frac{q_i}{q}\\
\hat{p}_{x_i} &= \frac{p_{x_i}}{m\ c} & \hat{p}_{y_i} &= \frac{p_{y_i}}{m\ c} & \hat{p}_{z_i} &= \frac{p_{z_i}}{m\ c}\\
\tilde{x} &= \frac{x - x_{\text{min}}}{\text{Dzero}} & \tilde{y} &= \frac{y - y_{\text{min}}}{\text{Dzero}} & \tilde{z} &= \frac{z - z_{\text{min}}}{\text{Dzero}}\\
\hat{\mathbf{E}} &= \frac{\mathbf{E}}{m\ c} & \tilde{\mathbf{E}} &= \text{Dzero}^2\mathbf{E},
\end{aligned}$$

where the $\tilde{\cdot}$ notation denotes a scaled variable to occur in the FMM. The FMM scales the position space $\mathbf{x}$ to the unit cube. The variables $x_{\min}$, $y_{\min}$, and $z_{\min}$ are used to translate the space such that there are no negative components to $\mathbf{x}$, while the variable Dzero is used to scale $\mathbf{x}$ such that no component of $\mathbf{x}$ is greater than 1. This scaling must be reversed when converting the FMM calculated potentials/fields back to real space.

2.3 Strang Splitting

Defining the vector

$$\mathbf{Y} = [\; x_1\; y_1\; z_1\; p_{x_1}\; p_{y_1}\; p_{z_1}\; \ldots\; x_N\; y_N\; z_N\; p_{x_N}\; p_{y_N}\; p_{z_N}\;]^{\mathrm{T}}$$

in $\mathbb{R}^{6N}$, the initial value problem (IVP) is described by

$$\begin{cases} \dot{\mathbf{Y}} = \mathbf{F}(\mathbf{Y}, t) \\ \mathbf{Y}(0) = \mathbf{Y}_0. \end{cases} \tag{2.5}$$

The function $\mathbf{F}$ in Eq.(2.5) is given by the right-hand sides of Eq.(2.3) and Eq.(2.4). Using Strang splitting, the function $\mathbf{F}$ is split as $\mathbf{F} = \mathbf{F}^{[1]} + \mathbf{F}^{[2]}$. Denoting the set of indexes of the particles in the neighborhood of particle i by S_i, the splitting of the equations of motion is as follow

$$\hat{\mathbf{F}}_i^{[1]}(\hat{\mathbf{Y}}_i, \hat{t}) = \begin{cases} \dot{\mathbf{x}}_i^{[1]} = \dfrac{\hat{p}_{x_i}\mathbf{i} + \hat{p}_{y_i}\mathbf{j} + \hat{p}_{z_i}\mathbf{k}}{\left(f_i^2 + \hat{p}_{x_i}^2 + \hat{p}_{y_i}^2 + \hat{p}_{z_i}^2\right)^{1/2}} \\[2ex] \dot{\hat{\mathbf{p}}}_i^{[1]} = \dfrac{qn_i}{mc^2}\left(\dfrac{q}{4\pi\epsilon_0}\displaystyle\sum_{\substack{j\in S_i\\ j\neq i}} \dfrac{n_j\left((x_i - x_j)\mathbf{i} + (y_i - y_j)\mathbf{j} + \gamma^2(z_i - z_j)\mathbf{k}\right)}{\gamma\left[(x_i - x_j)^2 + (y_i - y_j)^2 + \gamma^2(z_i - z_j)^2\right]^{3/2}} + \mathbf{E} + c\hat{\mathbf{v}}_i \times \mathbf{B}\right), \end{cases} \tag{2.6}$$

and

$$\hat{\mathbf{F}}_i^{[2]}(\hat{\mathbf{Y}}_i, \hat{t}) = \begin{cases} \dot{\mathbf{x}}_i^{[2]} = 0 \\[2ex] \dot{\hat{\mathbf{p}}}_i^{[2]} = \dfrac{qn_i}{mc^2}\dfrac{q}{4\pi\epsilon_0}\displaystyle\sum_{j\notin S_i} \dfrac{n_j\left((x_i - x_j)\mathbf{i} + (y_i - y_j)\mathbf{j} + \gamma^2(z_i - z_j)\mathbf{k}\right)}{\gamma\left[(x_i - x_j)^2 + (y_i - y_j)^2 + \gamma^2(z_i - z_j)^2\right]^{3/2}}. \end{cases} \tag{2.7}$$

With the initial condition $\hat{\mathbf{Y}}(0) = \hat{\mathbf{Y}}_0$, we will solve three IVP's. First,

$$\dot{\hat{\mathbf{Y}}} = \hat{\mathbf{F}}^{[2]}(\hat{\mathbf{Y}}, \hat{t}) \qquad \hat{\mathbf{Y}}(0) = \hat{\mathbf{Y}}_0$$

at $t = h/2$ to get $\phi^{[2]}_{h/2,1}$. Then,

$$\dot{\hat{\mathbf{Y}}} = \hat{\mathbf{F}}^{[1]}(\hat{\mathbf{Y}}, \hat{t}) \qquad \hat{\mathbf{Y}}(0) = \phi^{[2]}_{h/2,1}(\hat{\mathbf{Y}}_0)$$

at $t = h$ to get $\phi^{[1]}_{h,1}$. Finally,

$$\hat{\dot{\mathbf{Y}}} = \hat{\mathbf{F}}^{[2]}(\hat{\mathbf{Y}}, \hat{t}) \qquad y(0) = \phi^{[1]}_{h,1} \circ \phi^{[2]}_{{}^h/_2,1}(\hat{\mathbf{Y}}_0)$$

to get $\phi^{[2]}_{{}^h/_2,2}$ at $t = h/2$. Composing we get

$$\hat{\mathbf{Y}}_1 = S^1_h(\hat{\mathbf{Y}}_0) = \phi^{[2]}_{{}^h/_2,2} \circ \phi^{[1]}_{h,1} \circ \phi^{[2]}_{{}^h/_2,1}(\hat{\mathbf{Y}}_0)$$

This process is repeated over n timesteps by solving multiple IVP's, and we get the chain

$$\phi^{[2]}_{{}^h/_2,2n}\circ\phi^{[1]}_{h,n}\circ\phi^{[2]}_{{}^h/_2,2n-1}\circ\phi^{[2]}_{{}^h/_2,2n-2}\circ\phi^{[1]}_{h,n-1}\circ\phi^{[2]}_{{}^h/_2,2n-3}\circ\cdots\circ\phi^{[2]}_{{}^h/_2,4}\circ\phi^{[1]}_{h,2}\circ\phi^{[2]}_{{}^h/_2,3}\circ\phi^{[2]}_{{}^h/_2,2}\circ\phi^{[1]}_{h,1}\circ\phi^{[2]}_{{}^h/_2,1}(\hat{\mathbf{Y}}_0). \quad (2.8)$$

In the code, the FMM is used to generate the solutions $\phi^{[2]}$ and the Simò integrator is used to generate the solutions $\phi^{[1]}$. Because the particle positions are not changed in Eq.(2.7), the FMM can be ran *once* to compute pairs $\phi^{[2]}_{{}^h/_2,2k} \circ \phi^{[2]}_{{}^h/_2,2k+1}$ for $2 \leq k \leq n-1$ if care is taken to add or subtract the contributions of particles between the sets $S_i(t_k)$ and $S_i(t_k+1)$ between timesteps.

2.4 The Fast Multipole Method

The FMM is an efficient hierarchical method to calculate the Coulomb effect of a discrete set of charged particles N on a test charge at a particular location. The scalar electric potential is given by

$$G(x,y,z) = \sum_{j \notin S_i} \frac{n_j}{[(x-x_j)^2 + (y-y_j)^2 + (z-z_j)^2]^{3/2}},$$

where (x_k, y_k, z_k) are the coordinates of particles $k = 1, 2, \ldots, N$. In the FMM, the function $G(x,y,z)$ is approximated by dividing the sum into a set of near and far evaluations. The near evaluations can be computed exactly, and the far evaluations are represented as a sum of multipole expansions. We employed a novel adaptive multilevel FMM in PHAD algorithm which gives a computational complexity that scales asymptotically as $\mathcal{O}(N)$ and can treat different non-uniform spatial distributions.

The FMM takes the coordinates (x_k, y_k, z_k) of the particles from a data file. To handle the relativistic gamma in PHAD, the coordinates $(x_k, y_k, \gamma z_k)$ are stored in a data file and passed to the FMM. Then, PHAD adapts the original FMM code to compute the expansion of

$$\begin{aligned} G(x,y,\gamma z) &= \sum_{j \notin S_i} \frac{n_j}{[(x-x_j)^2 + (y-y_j)^2 + (\gamma z-\gamma z_j)^2]^{3/2}} \\ &= \sum_{j \notin S_i} \frac{n_j}{[(x-x_j)^2 + (y-y_j)^2 + \gamma^2(z-z_j)^2]^{3/2}} \end{aligned}$$

needed for computing the right hand side of Eq.(2.7).

There are then two separate "long range" integrators contained within one procedure in PHAD code. The regular one is for the odd $\phi^{[2]}_{h/2,2k+1}$ and the modified is for even $\phi^{[2]}_{h/2,2k}$. Most of the long range integrations are computed following a sequence starting with the FMM, then the modified long range integrator, and finally the regular long range integrator. The positions remain unchanged for these integrators. However, the neighborhood containing particle i, S_i, is different for the modified and regular integrators. The S_i for the regular long range integrator are defined by the FMM on the positions after to the preceding Simò integrator (the previous FMM). The S_i for the modified long range integrator are defined by the FMM on the positions before the preceding Simò integrator (the FMM before the previous FMM).

The following Eq.(2.9) is a colored version of Eq.(2.8) that shows red regular long range integrations, green Simò integrations, and blue modified long range integrations. Each red/blue pair is computed using one FMM computation.

$$\phi^{[2]}_{h/2,2n}\circ\phi^{[1]}_{h,n}\circ\phi^{[2]}_{h/2,2n-1}\circ\phi^{[2]}_{h/2,2n-2}\circ\phi^{[1]}_{h,n-1}\circ\phi^{[2]}_{h/2,2n-3}\circ\cdots\circ\phi^{[2]}_{h/2,4}\circ\phi^{[1]}_{h,2}\circ\phi^{[2]}_{h/2,3}\circ\phi^{[2]}_{h/2,2}\circ\phi^{[1]}_{h,1}\circ\phi^{[2]}_{h/2,1}(\mathbf{Y}_0) \quad (2.9)$$

2.4.1 The Regular Long Range Integrator

The regular long range integrator uses the local expansion for the field created by the particles outside a particular particle's neighborhood (S_i), and gives a value for the electric field contribution from these particles. It computes the steps in the composition chain involving $\phi^{[2]}_{h/2,2k-1}$ in solving the differential equation Eq.(2.7).

The three components of the position in Eq.(2.7) are zeroes, so there is no change to the particles positions. This implies that the last three components can be integrated directly. Viewing these equations as

$$\dot{\hat{\mathbf{Y}}} = C \qquad \hat{\mathbf{Y}}(0) = \hat{\mathbf{Y}}_0,$$

they have the exact solution $\hat{\mathbf{Y}}_0 + C\hat{t}$. The solution to Eq.(2.7) with the initial condition $\hat{Y}_{i0} = [x_i^0\ y_i^0\ z_i^0\ \hat{p}^0_{x_i}\ \hat{p}^0_{y_i}\ \hat{p}^0_{z_i}]^{\mathrm{T}}$ and with the FMM scaling is

$$\hat{Y}_i(h) = \begin{bmatrix} \tilde{x}_i^0 \\ \tilde{y}_i^0 \\ \tilde{z}_i^0 \\ \hat{p}_{x_i}^0 + h\,\dfrac{q^2\ n_i}{4\pi\epsilon_0\ m\ c^2\ \text{Dzero}^2}\left[\displaystyle\sum_{j\notin S_i}\frac{n_j(\tilde{x}_i-\tilde{x}_j)}{\gamma[(\tilde{x}_i-\tilde{x}_j)^2+(\tilde{y}_i-\tilde{y}_j)^2+\gamma^2(\tilde{z}_i-\tilde{z}_j)^2]^{3/2}}\right] \\ \hat{p}_{y_i}^0 + h\,\dfrac{q^2\ n_i}{4\pi\epsilon_0\ m\ c^2\ \text{Dzero}^2}\left[\displaystyle\sum_{j\notin S_i}\frac{n_j(\tilde{y}_i-\tilde{y}_j)}{\gamma[(\tilde{x}_i-\tilde{x}_j)^2+(\tilde{y}_i-\tilde{y}_j)^2+\gamma^2(\tilde{z}_i-\tilde{z}_j)^2]^{3/2}}\right] \\ \hat{p}_{z_i}^0 + h\,\dfrac{q^2\ n_i}{4\pi\epsilon_0\ m\ c^2\ \text{Dzero}^2}\left[\displaystyle\sum_{j\notin S_i}\frac{n_j\gamma^2(\tilde{z}_i-\tilde{z}_j)}{\gamma[(\tilde{x}_i-\tilde{x}_j)^2+(\tilde{y}_i-\tilde{y}_j)^2+\gamma^2(\tilde{z}_i-\tilde{z}_j)^2]^{3/2}}\right] \end{bmatrix}, \qquad (2.10)$$

The sums in the square brackets of the last three components of Eq.(2.10) are computed in the FMM as an electric field. We will denote this as $\tilde{\mathbf{E}}_i(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i) = [\tilde{E}_{x_i}\ \tilde{E}_{y_i}\ \tilde{E}_{z_i}]$ to get

$$\hat{Y}_i(h) = \begin{bmatrix} \tilde{x}_i^0 \\ \tilde{y}_i^0 \\ \tilde{z}_i^0 \\ \hat{p}_{x_i}^0 + h\,\dfrac{q^2\ n_i}{4\pi\epsilon_0\ m\ c^2\ \text{Dzero}^2}\tilde{E}_{x_i} \\ \hat{p}_{y_i}^0 + h\,\dfrac{q^2\ n_i}{4\pi\epsilon_0\ m\ c^2\ \text{Dzero}^2}\tilde{E}_{y_i} \\ \hat{p}_{z_i}^0 + h\,\dfrac{q^2\ n_i}{4\pi\epsilon_0\ m\ c^2\ \text{Dzero}^2}\tilde{E}_{z_i} \end{bmatrix}. \qquad (2.11)$$

2.4.2 The Modified Long Range Integrator

The modified long range integrator uses the neighborhoods from the FMM based on the positions before the Simò integrator. Thus, the sets S_i are based on the previous positions. In the overall code, each modified long range integration is represented by $\phi^{[2]}_{h/2,2k}$ as a solution to Eq.(2.7) and the computations are made in terms of the current positions S_i^*. We then need the two sets $S_i\backslash S_i^*$ and $S_i^*\backslash S_i$ which are found using the COMPARE function (see Section3.4). We will use that

$$S_i^{\mathrm{c}} = \left(S_i^*\backslash S_i\right) \cup \left(S_i^{*\mathrm{c}}\backslash S_i\right) = \left(S_i^*\backslash S_i\right) \cup \left(S_i^{*\mathrm{c}}\backslash (S_i\backslash S_i^*)\right),$$

such that the solution becomes

$$\hat{Y}_i(h) = \begin{bmatrix} x_i^0 \\ y_i^0 \\ z_i^0 \\ \hat{p}_{x_i}^0 + h\,\frac{q^2}{4\pi\epsilon_0\; m\; c^2}\; n_i \left[\tilde{E}_x^* + \sum_{j\in S_i^*\backslash S_i} \frac{n_j(x_i-x_j)}{\gamma[(x_i-x_j)^2+(y_i-y_j)^2+\gamma^2(z_i-z_j)^2]^{3/2}} - \sum_{j\in S_i\backslash S_i^*} \frac{n_j(x_i-x_j)}{\gamma[(x_i-x_j)^2+(y_i-y_j)^2+\gamma^2(z_i-z_j)^2]^{3/2}}\right] \\ \hat{p}_{y_i}^0 + h\,\frac{q^2}{4\pi\epsilon_0\; m\; c^2}\; n_i \left[\tilde{E}_y^* + \sum_{j\in S_i^*\backslash S_i} \frac{n_j(y_i-y_j)}{\gamma[(x_i-x_j)^2+(y_i-y_j)^2+\gamma^2(z_i-z_j)^2]^{3/2}} - \sum_{j\in S_i\backslash S_i^*} \frac{n_j(y_i-y_j)}{\gamma[(x_i-x_j)^2+(y_i-y_j)^2+\gamma^2(z_i-z_j)^2]^{3/2}}\right] \\ \hat{p}_{z_i}^0 + h\,\frac{q^2}{4\pi\epsilon_0\; m\; c^2}\; n_i \left[\tilde{E}_z^* + \sum_{j\in S_i^*\backslash S_i} \frac{n_j\gamma(z_i-z_j)}{[(x_i-x_j)^2+(y_i-y_j)^2+\gamma^2(z_i-z_j)^2]^{3/2}} - \sum_{j\in S_i\backslash S_i^*} \frac{n_j\gamma(z_i-z_j)}{[(x_i-x_j)^2+(y_i-y_j)^2+\gamma^2(z_i-z_j)^2]^{3/2}}\right] \end{bmatrix} \quad (2.12)$$

2.5 The Simò Integrator

The Simò integrator is used in PHAD algorithm for an accurate and efficient time stepping by numerically solving Eq.(2.6). All collisions and close encounters are captured by the Simò integrator with accuracy up to machine precision, where the external electromagnetic fields and relativistic fields are also considered. In the overall PHAD code description, we denoted the Simò integrator solution at k steps as $\phi_{h,k}^{[1]}$.

Chapter 3

Memory and Subroutine Details

3.1 Global Variables

The main global variables used to set storage in the code and can influence the memory required for the run are the following

NP number of processors

NUM_OF_PARTICLES number of sources

NUM_OF_TARGETS number of targets

FMM_PROC_BALANCE_EST =2, estimate of the factor of the largest number of neighbors, sources per processor to what would be balanced

MAX_NUM_OF_BOXES max. number of boxes

PROC_MAX_NUM_OF_BOXES max. number of boxes per processor

q largest q value for FMM

DLeafNodesPrev

NM1 number of monomials for DA vectors

NBDT

FMM_ORDER order of the FMM

NBCF

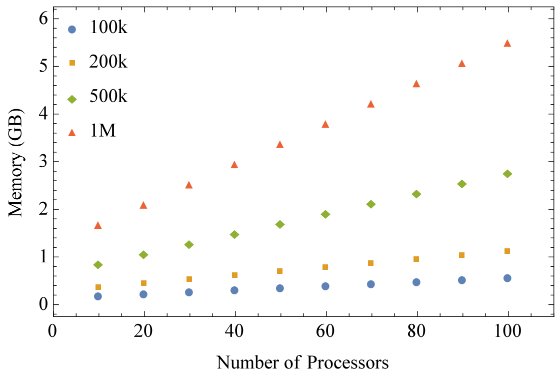
MAX_FMM_UP

MAX_FMM_DOWN

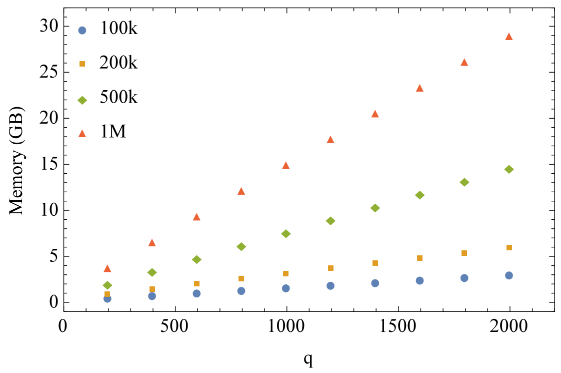
SIMO_ORDER max. allowed order of the Simò integrator

BINS number of bins used in the Simò integrator

As an example, the effect of the NP, NUM_OF_PARTICLES, and q on the memory is demonstrated in Fig.3.1. The memory of the MPI COSY executable can be adjusted by modifying



(a) Memory vs. NP



(b) Memory vs. q

Figure 3.1: The influence of the number of processors, number of particles, and the q value on the memory required by PHAD code.

the value of LMEM in cosy source files. This can be performed manually at each instant of each source file, or can be changed using the *cosyresize* utility as explained in the NIU COSY User Guide. Note that the LMEM value cannot exceed 2 GB.

3.2 Function QSORT

The function QSORT orders the components of a vector from smallest to largest. The vector itself is the only input. The sorting is done by choosing a pivot from the vector components and putting the components less than the pivot the left of the pivot. It repeats this process on the components to the left of the vector until there is only one component to the left of the current pivot. Then, it changes the vector components under consideration to those between the previous RIGHT values. This continues until there are no more such vectors with lengths more than one and the vector is sorted.

Example

In this example, we will show how the function sorts the vector

$$[\,8 \quad 1 \quad 5 \quad 4 \quad 9 \quad 7 \quad 3 \quad 6 \quad 2\,].$$

Initially, `LEFT` is set to 8 and `RIGHT` is set to 2. The pivot is the center (odd length vector) or "right of center" (even length vector), here it is the center 9. Parenthesis are employed to show the current values of `LEFT` and `RIGHT`. A vertical line, | to the left of a component, denotes a previous `RIGHT` values. Once a component is in its final place it is put in blue.

$$\left[\left(8\ 1\ 5\ 4\ 9\ 7\ 3\ 6\ 2\right)\right]$$

$$\left[\left(8\ 1\ 5\ 4\ 7\ 3\ 6\ 2\right)\ |9\right]$$

$$\left[\left(1\ 5\ 4\ 3\ 6\ 2\right)\ |7\ 8\ |9\right]$$

$$\left[\left(1\ 2\right)\ |3\ 5\ 4\ 6\ |7\ 8\ |9\right]$$

$$\left[1\ 2\ |3\ \left(5\ 4\ 6\right)\ |7\ 8\ |9\right]$$

$$\left[1\ 2\ |3\ |4\ |\left(5\ 6\right)\ |7\ 8\ |9\right]$$

3.3 Function MERGE

The function merge joins two ordered vectors into a single ordered vector. It takes the two ordered vectors as input and outputs a single ordered vector.

Example

In this example, we will join the two vectors [1 4 5 7] and [2 3 6 8 9].

Sorted vector 1	Sorted vector 2		Merged vector
[1 4 5 7]	[2 3 6 8 9]	$\longrightarrow$	[1]
[4 5 7]	[2 3 6 8 9]	$\longrightarrow$	[1 2]
[4 5 7]	[3 6 8 9]	$\longrightarrow$	[1 2 3]
[4 5 7]	[6 8 9]	$\longrightarrow$	[1 2 3 4]
[5 7]	[6 8 9]	$\longrightarrow$	[1 2 3 4 5]
[7]	[6 8 9]	$\longrightarrow$	[1 2 3 4 5 6]
[7]	[8 9]	$\longrightarrow$	[1 2 3 4 5 6 7]
[ ]	[8 9]	$\longrightarrow$	[1 2 3 4 5 6 7 8 9]

Once one of the vectors is empty, as is the case on the last line, the elements in the nonempty vector are appended to the merged vector.

3.4 Function COMPARE

The `COMPARE` function compares two ordered lists [A] and [B] and finds [$A\backslash B$] and [$B\backslash A$]. Its purpose within the code is to compare the sets S_{i-1} and S_i within the modified long range integrator. It takes two vectors containing the sorted elements of A and B and returns two vectors with sorted vectors containing $A\backslash B$ and $B\backslash A$. Within the procedure, a variable `CHECK` stores the smallest remaining value in A. If this value is smaller than the smallest remaining value in B, then `CHECK` is placed in $A\backslash B$. If the smallest value of B is smaller than `CHECK`, the smallest value of B is placed in $B\backslash A$. When the `CHECK` is equal to the smallest value in B, `CHECK` is reassigned to the next element of A and nothing is added to $A\backslash B$ nor $B\backslash A$. Once all elements of either A or B have been exhausted, what remains remains of B is placed in $B\backslash A$ or the remainder of A is placed in $A\backslash B$.

Example

For this example, we will consider the vectors [1 2 6 8] and [2 4 8 9]. At each step the smallest elements of the sets are compared. When both elements are equal they are in red and discarded. When the smallest element of A is less than the smallest element of B it is colored blue and put into $A\backslash B$. When the smallest element of B is less than the smallest element of A, it is colored brown and placed into $B\backslash A$.

$$\begin{array}{c} \text{A} \; [\; 1 \;\; 2 \;\; 6 \;\; 8 \;] \\ \text{B} \; [\; 2 \;\; 4 \;\; 8 \;\; 9 \;] \\ A\backslash B = [\,] \\ B\backslash A = [\,] \end{array} \quad \longrightarrow \quad \begin{array}{c} [\; 2 \;\; 6 \;\; 8 \;] \\ [\; 2 \;\; 4 \;\; 8 \;\; 9 \;] \\ A\backslash B = [\, 1 \,] \\ B\backslash A = [\,] \end{array}$$

$$\begin{array}{c} [\; 6 \;\; 8 \;] \\ [\; 4 \;\; 8 \;\; 9 \;] \\ A\backslash B = [\, 1 \,] \\ B\backslash A = [\,] \end{array} \quad \longrightarrow \quad \begin{array}{c} [\; 6 \;\; 8 \;] \\ [\; 8 \;\; 9 \;] \\ A\backslash B = [\, 1 \,] \\ B\backslash A = [\, 4 \,] \end{array}$$

$$\begin{array}{c} [\; 8 \;] \\ [\; 8 \;\; 9 \;] \\ A\backslash B = [\, 1 \; 6 \,] \\ B\backslash A = [\, 4 \,] \end{array} \quad \longrightarrow \quad \begin{array}{c} [\;] \\ [\; 9 \;] \\ A\backslash B = [\, 1 \; 6 \,] \\ B\backslash A = [\, 4 \,] \end{array}$$

$$A\backslash B = [\, 1 \; 6 \,] \qquad B\backslash A = [\, 4 \; 9 \,]$$