AN IMPROVED PARALLEL POISSON SOLVER FOR SPACE CHARGE CALCULATION IN BEAM DYNAMICS SIMULATION

Dawei Zheng*, Ursula van Rienen, University of Rostock, Rostock, Germany
Ji Qiang, LBNL, Berkeley, CA, USA

Abstract

In numerous beam dynamics simulations of particle accelerators, the space charge calculation is a critical issue. The ways to calculate the space charge force vary in different kinds of methods. A common method based on Green's function in free space is often implemented in various PIC (particle-in-cell) simulation codes for beam tracking. However, the calculation time of the Poisson solver for both, serial and parallel cases can reach a high percentage of the whole simulation time consumption. In this paper, we present an improved parallel Poisson solver. The enhanced solver focuses on three aspects of efficiency improvement: using the reduced integrated Green's function, using the discrete cosine transform for Green's function, using a novel fast implicitly zero-padded convolution routine. The amended, parallel Poisson solver is implemented using MPI and Open MP. We study the strong scaling and the weak scaling for the scalability property of the Poisson solver, too. The novel solver is integrated into the IMPACT-T code in order to compare it with the commonly used Poisson solver.

INTRODUCTION

For the particle accelerator research, simulations are critical tools from the design to the operation. Furthermore, the space charge calculation constitutes a major part of the beam dynamics simulation, which considers the low energy region of the accelerators. Applying the mid-point rule on an equidistant grid for the numerical integration of Poisson's equation, the method based on Green's function in free space is often given as:

$$\varphi(\mathbf{r}_\mathbf{l}) \approx \frac{1}{4\pi\varepsilon_0} \cdot \sum_{\mathbf{l}'=(1,1,1)}^{(N_x,N_y,N_z)} \rho(\mathbf{r}'_{\mathbf{l}'})\tilde{G}(\mathbf{r}_\mathbf{l},\mathbf{r}'_{\mathbf{l}'}), \tag{1}$$

where $\mathbf{l} = (i,j,k)$, $\mathbf{l}' = (i',j',k')$ and $\tilde{G}(\mathbf{r}_\mathbf{l},\mathbf{r}'_{\mathbf{l}'}) = h_x h_y h_z G(\mathbf{r}_\mathbf{l},\mathbf{r}'_{\mathbf{l}'})$. The discrete $\varphi(\mathbf{r}_\mathbf{l})$ is obtained by Fourier convolution theorem. Additionally, Hockney and Eastwood [1] have published an efficient way to extend the charge density, which is required for the Fourier convolution. A further study of an integrated Green's function method which is suitable for the novel Free Electron Lasers (FELs) is described in [2] [3]. For this kind of 3D FFT-typed Poisson solver, the time and memory consumption may easily reach the limit for PC simulation. The parallel implementation of the routine is preferred for a higher level computing performance. In this paper, we present an improved parallel Poisson solver, which differs from the commonly used routine. The efficiency improvement is impressive and valuable referenced for other parallel FFT-typed Poisson solver codes in the community.

* dawei.zheng@uni-rostock.de

THE IMPROVED PARALLEL POISSON SOLVER

The improved parallel Poisson solver is based on the efficient serial Poisson solver outlined in [4]. Three aspects of efficiency improvement are enhanced:

1. Using the reduced integrated Green's function (RIGF):

$$\tilde{G}_{\mathrm{RIGF}}(\mathbf{r}_{(i,j,k)}) = \begin{cases} \tilde{G}_{\mathrm{IGF}}(\mathbf{r}_{(i,j,k)}) & 1 \le w \le R_w; \\ h_x h_y h_z G(\mathbf{r}_{(i,j,k)}) & R_w \le N_w + 1; \end{cases}$$

where $\tilde{G}_{\mathrm{IGF}}$ is described in [2] [3] and R_w is determined by an adaptive method in [4], where w represents the indices x, y, z.

2. Using the discrete cosine transform (DCT) for Green's function: define $g = \{g_1, g_2, \dots g_{n+1}\}$, and the real even symmetric extension (RESE) of g is g_{ex}, i.e. $g_{ex} = \{g_1, g_2, \dots g_{n+1}, g_n, g_{n-1}, \dots g_2\}_{2n}$. Then

$$\mathrm{FFT}_{2n}(\mathrm{RESE}(g)) = 2 \cdot \mathrm{RESE}(\mathrm{DCT}_{n+1}(g)).$$

The same result succeeds by replacing the factor 2 to a factor 8 for 3D extension case. The complexity reduces significantly from $O((\log_2 2n)^3)$ to $O((\log_2 n)^3)$ besides RESE.

3. Using a novel fast implicitly zero-padded convolution routine: define the unit: $\omega_m^m = 1$. We achieve a FFT without the last bit reversal stage named FFTPADBACKWARD and FFTPADFORWARD as in Alg. 1.

Algorithm 1 FFTPADBACKWARD, FFTPADFORWARD

```
Input: $f$
Output: $f, u$
1: function FFTPADBACKWARD
2:     for $k = 1 \to n$ do
3:         $u(k) \leftarrow \omega_{2n}^{k-1} f(k)$;
4:     end for
5:     $f(:) \leftarrow \mathrm{IFFT}[f(:)]$;
6:     $u(:) \leftarrow \mathrm{IFFT}[u(:)]$;
7: end function
```

```
Input: $f, u$
Output: $f$
1: function FFTPADFORWARD
2:     $f(:) \leftarrow \mathrm{FFT}[f(:)]$;
3:     $u(:) \leftarrow \mathrm{FFT}[u(:)]$;
4:     for $k = 1 \to n$ do
5:         $f(k) \leftarrow f(k) + \omega_{2n}^{-k+1} u(k)$;
6:     end for
7: end function
```

In principle, the FFTPADBACKWARD and FFTPADFORWARD transforms are deformed inverse FFT (IFFT) and FFT, respectively. The two outputs of the FFTPADBACKWARD are the same as those obtained by separating the even-odd indices of the $2n$ sized FFT of the explicitly zero padded f, i.e. we define $f_{ex} = \begin{bmatrix} f \\ 0 \end{bmatrix}_{2n}$ and $[f,u]_n = \mathrm{fftpadBackward}_n(f)$, which results in $[\mathrm{IFFT}_{2n}(f_{ex})_{\mathrm{odd}}, \mathrm{IFFT}_{2n}(f_{ex})_{\mathrm{even}}] = [f,u]$. The FFTPADFORWARD transform is obtained by the inverse procedure.

The 2D and 3D FFTPADBACKWARD (FFTPADFORWARD) are the same procedures as 2D and 3D IFFT (FFT), i.e. they

ISBN 978-3-95450-136-6

execute a 1D transform along each direction, serially. The improved 3D parallel Poisson solver is further developed to suit a real to complex (R2C) 1D FFT along the first direction in order to reduce the number of further deformed FFTs in other directions.

THE IMPROVED PARALLEL POISSON SOLVER'S ROUTINE

All processors utilized in the solver are mapped onto a 2D process grid. Each process owns a unique ID (RowID, ColID) as the element of a matrix.

The charge density $\rho(i,j,k)$ is distributed to the process grid by filling (j,k)s into the (RowID,ColID)s as (j_{local}, k_{local})s equally. Only the is are free to be utilized. Since the indices of j,k are distributed to the grid, the elements ordered by the two directions can not be used directly, e.g. for the Fourier transforms in the other two directions. Communications and transports of data are therefore a special challenge in supercomputers. Unlike the shared-memory computer programming (as PCs), the transport of data is a serious problem in parallel programming (as supercomputers). This problem requires intensive concern. We use the *MPI_alltoall* for the high dimension transpose. The specified transpose function rearranges the 3D vector by exchanging the order of direction e.g. ijk to jik or kji for each time. The time consumption of the data transpose can reach a high percentage of the total CPU time comparing to the pure computing time for high dimension Fourier transforms.

The programming routine is organized by four parts:

1. The RIGF calculation and the related DCT (Fig.1) – the Green's function values G, which is distributed in the 2D grid processes by the order (k,j,i) (k(blue), j(red), i(green)) in each process, are calculated by the RIGF method; the 3D DCT of G is implemented. The results are extended in j direction in demand for further multiplication in Part 3.

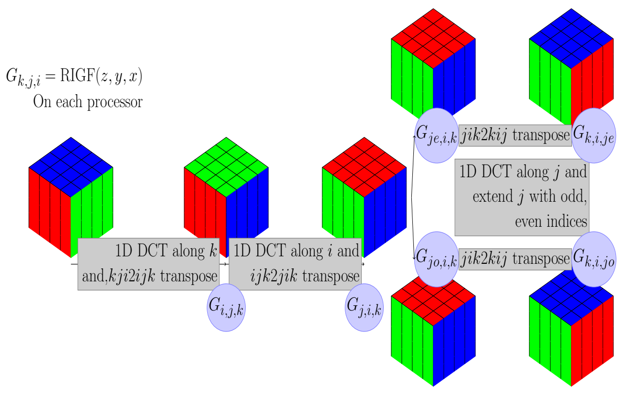
$G_{k,j,i} = \mathrm{RIGF}(z,y,x)$
On each processor
1D DCT along k and, $kji2ijk$ transpose
1D DCT along i and $ijk2jik$ transpose
$G_{i,j,k}$
$G_{j,i,k}$
$G_{je,i,k}$
$jik2kij$ transpose
$G_{k,i,je}$
1D DCT along j and extend j with odd, even indices
$G_{jo,i,k}$
$jik2kij$ transpose
$G_{k,i,jo}$

Figure 1: The RIGF calculation and the related 3D DCT.

2. The 3D backward deformed FFT routine (Fig.2) – the ρ is firstly padded with the same size zeros in i direction as $\mathrm{Ex}\rho_{i,j,k}$. The following deformed 3D FFTs is constructed by the *R2C* FFTs from the real vectors $\mathrm{Ex}\rho_{i,j,k}$ to the complex vectors $C\rho_{i,j,k}$ along i, the FFTPADBACKWARDS along j, and the FFTPADBACKWARDS along k.

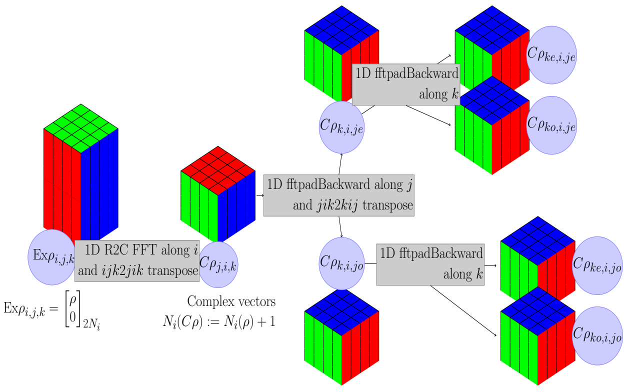
$\mathrm{Ex}\rho_{i,j,k}$
1D R2C FFT along i and $ijk2jik$ transpose
$C\rho_{j,i,k}$
$\mathrm{Ex}\rho_{i,j,k} = \begin{bmatrix}\rho\\0\end{bmatrix}_{2N_i}$
Complex vectors
$N_i(C\rho) := N_i(\rho)+1$
1D fftpadBackward along j and $jik2kij$ transpose
$C\rho_{k,i,je}$
$C\rho_{k,i,jo}$
1D fftpadBackward along k
$C\rho_{ke,i,je}$
$C\rho_{ko,i,je}$
1D fftpadBackward along k
$C\rho_{ke,i,jo}$
$C\rho_{ko,i,jo}$

Figure 2: The 3D backward deformed FFT routine.

3. Multiplication in spectral domain – the results from Part1 and Part2 are multiplied element by element according to the following rule:

$$\begin{aligned} C\rho_{ke,i,jx} &= \mathrm{RESE}(\tilde{G}_{k,i,jx})_{\mathrm{even}} * C\rho_{ke,i,jx}, \text{ and} \\ C\rho_{ko,i,jx} &= \mathrm{RESE}(\tilde{G}_{k,i,jx})_{\mathrm{odd}} * C\rho_{ko,i,jx}, \end{aligned}$$

where x expresses e (even) or o (odd).

4. The 3D forward deformed FFT routine part (Fig.3) – the inverse procedure of Part 2 is performed.

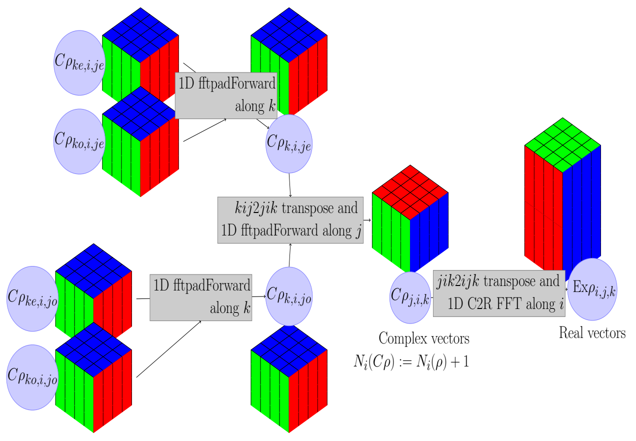
$C\rho_{ke,i,je}$
$C\rho_{ko,i,je}$
1D fftpadForward along k
$C\rho_{k,i,je}$
$kij2jik$ transpose and 1D fftpadForward along j
$C\rho_{ke,i,jo}$
$C\rho_{ko,i,jo}$
1D fftpadForward along k
$C\rho_{k,i,jo}$
$C\rho_{j,i,k}$
$jik2ijk$ transpose and 1D C2R FFT along i
$\mathrm{Ex}\rho_{i,j,k}$
Complex vectors
$N_i(C\rho) := N_i(\rho)+1$
Real vectors

Figure 3: The 3D forward deformed FFT routine.

STUDY OF THE IMPROVED PARALLEL POISSON SOLVER

The parallel Poisson solver is written in Fortran 90 with the Open MPI, compiled by the Intel compiler, and executed on the Edison supercomputer at the NERSC (National Energy Research Scientific Computing Center).

1. We carry out the strong scaling and weak scaling study of the solver. The results are shown in Fig.4. The weak scaling figures (above) are plotted by fixing the ratio between the size of the problem and the cores in order to keep it constant. The starting points are 32^3/core (left) and 64^3/core (right), respectively. The strong scaling figures (below) are

ISBN 978-3-95450-136-6

plotted by fixing the size of the problem, 128^3 (left) and 256^3 (right), as well as increasing the number of cores in processing. Therefore, the problem is more like a cpu-bound rather than a memory-bound problem.

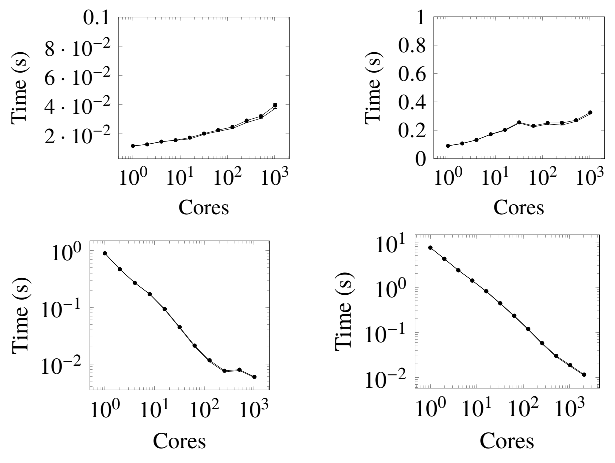

Figure 4: The weak scaling (top) and strong scaling (bottom) study.

2. The OpenMP is a shared memory multiprocessing API, which cooperates with the OpenMPI process together in the same computational node. Probably, the best choice may be 1-4 MPI processes per NUMA node (each processor (12 cores) constitutes a NUMA node, and each node has two processors) with 12-3 OpenMP threads each node on Edison. Additionally, there is a time cost in the preparation of OpenMP when we link the OpenMP library. The OpenMP parallel parts are mainly developed for "do-loops". The SIMD, up to the date, is not further optimized. In Fig. 5, the process numbers N_p in use are fixed by 96, and the thread number per node varies from 1 to 6. For $N_{x,y,z} = 512$ (left) and $N_{x,y,z} = 256$ (right), we plot all the CPU time for different OpenMP threads per process in use (for two memory affinity situation: numa_node and depth).

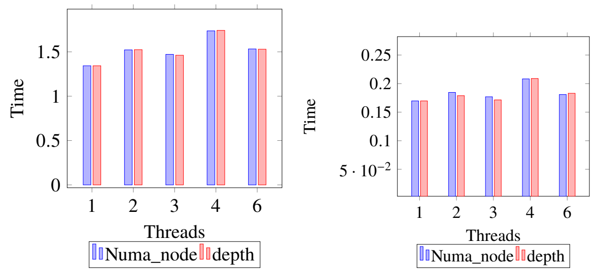

Figure 5: CPU time for different OpenMP threads per process in use.

As shown in Fig.5, the Open MPI obtains better results than the hybrid Open MPI+OpenMP case with the same number of cores in use.

3. Comparing the factors of time consumption and accuracy, the improved Poisson solver computes faster than the commonly used Poisson solver, whereas their calculations are equally accurate.

Example 1: We track an extreme long bunch with the transverse aperture size $R_{x,y} = 0.15$m, longitudinal size $L_z = 1.0e5$m, at the initial sections of a virtual accelerator. Some simulation parameters are $dt = 1.0e - 12$
, s, $t_{\text{step}}20,000,000$, $N_{x,y,z} = 64$, $N_{\text{col}} = 8$, $N_{\text{row}} = 8$. In Fig. 6, we have shown a perfect match in RMS bunch size and RMS emittance in x direction between the improved parallel Poisson solver and the commonly used Poisson solver routine. For the other directions, y and z, the results match perfectly.

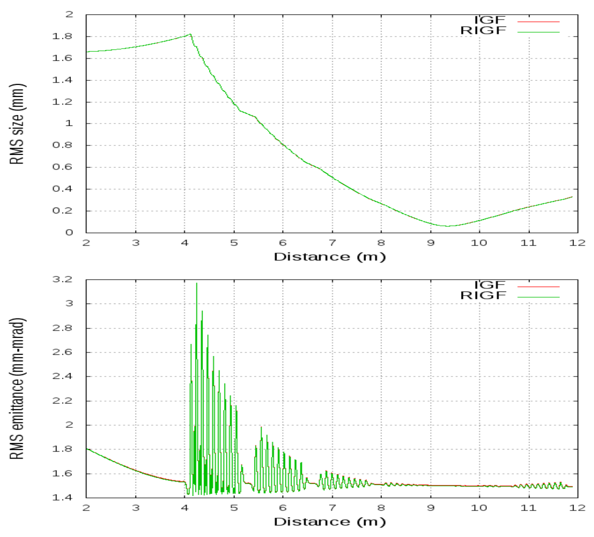

Figure 6: A schematic plot of comparison of the RMS size and RMS emittance in x direction.

Example 2: We compare the efficiency improvement of the new solver. As shown in Tab.1, the speed-up is as high as 2, which is significant. For small size problems with a large number of processes, the speed-up is not obvious. This may be because the data transports occupy a high percentage of the whole time consumption in both solvers.

Table 1: Comparison of the Common IGF and Novel RIGF Solvers

N_w	N_p	t_{RIGF}	t_{IGF}	N_w	N_p	t_{RIGF}	t_{IGF}
64	4	93.57 s	221.9 s	64	256	7.136 s	8.311 s
64	16	26.82 s	64.73 s	128	256	25.95 s	40.45 s
64	64	9.871 s	20.28 s	256	256	198.5 s	298.8 s

In conclusion, the new improved parallel Poisson solver proves to be more efficient than the commonly used routine in the calculation of space charge for beam dynamics simulations.

ACKNOWLEDGMENT

Dawei Zheng is supported by the China Scholarship Council. This research was partially supported by the U.S. Department of Energy under Contract no.DE-AC02-05CH11231.

REFERENCES

[1] R.W. Hockney and J.W. Eastwood, Computer Simulation Using Particles, Institut of Physics Publishing, Bristol, (1992).

[2] J. Qiang, S. Lidia, R.D. Ryne, and C. Limborg-Deprey, Phys. Rev. ST Accel. Beams, vol 9, 044204 (2006).

[3] J.Qiang, S. Lidia, R.D. Ryne, and C. Limborg-Deprey, Phys. Rev. ST Accel. Beams, vol 10, 129901 (2007).

[4] D. Zheng, G. Pöplau and U. van Rienen, “Efficiency optimization of fast Poisson solver in beam dynamics simulation”, Computer Physics Communications, In press (2015).