Journal of Physics: Conference Series **2420** (2023) 012051 doi:10.1088/1742-6596/2420/1/012051

Parallelization of Radia magnetostatics code

A Banerjee[1], O Chubar[2], G Le Bec[3], J Chavanne[3], B Nash[4], C Hall[4] and J Edelen[4]

[1]Stony Brook University, Stony Brook, NY 11794, USA
[2]Brookhaven National Laboratory, Upton, NY 11973, USA
[3]ESRF, 38000 Grenoble, France
[4]RadiaSoft LLC, Boulder, CO 80301, USA

E-mail: anushka.banerjee@stonybrook.edu

Abstract. Radia 3D magnetostatics code has been used for the design of insertion devices for light sources over more than two decades. The code uses the magnetization integral approach that is efficient for solving permanent magnet and hybrid magnet structures. The initial version of the Radia code was sequential, its core written in C++ and interface in the Mathematica language. This paper describes a new Python interfaced parallel version of Radia and its applications. The parallelization of the code was implemented on C++ level, where the semi-analytical calculations of interaction matrix elements and resultant magnetic fields were parallelized using the Message Passing Interface. The parallel performance results are encouraging, particularly for magnetic field calculation post relaxation where a ∼600 speedup with respect to sequential execution was obtained. The new parallel Radia version facilitates designs of insertion devices and lattice magnets for novel particle accelerators.

1. Introduction

The Radia code has been continuously improved for nearly two decades since its inception and development in the Insertion Devices laboratory of the European Synchrotron Radiation Facility (ESRF). From establishing boundary integral method-based calculations as an alternative and better approach than the conventional finite element, for e.g., FLUX3D [1], Radia has been widely used to compute fields for undulators. Over the last few years, Radia has not only provided better time performance and precision results when compared to FEM codes [2], it provides excellent agreement between the calculated and measured field values corresponding to relatively simple structures like quadrupole and sextupole [3], or complicated structures like the iron-dominated electromagnetic structure: SOLEIL Undulator HU256 [4].

Following the concepts of object-oriented programming, the core part of Radia is written in C++ and was initially interfaced to run serially on Wolfram Mathematica and Igor Pro only. Currently, the new, open-source version of Radia is already available in Python interface [5], with ongoing developments to make it executable on web-based graphical interface like Jupyter Notebook [6]. Radia's Python interface has been used for magnetic "cross talk" computations of the recent ESRF-Extremely Brilliant Source upgrade, where a large number of sequential calculations with different input parameters were performed in parallel at the ESRF cluster. The calculated values had good agreement with the measured values (relative errors in the 10^{-4} range) [7].

2. Methods used for parallelization

With an increased development in the field of light sources, the importance of magnetostatics code has increased by manifolds compared to what it was a few decades earlier. Despite Radia's stellar performance when compared to conventional FEM based code, the target computations involving 3D magnetic simulations of the insertion devices and accelerator magnets are quite complicated and CPU-intensive, with further increasing complexities. With an aim to further improve Radia's overall performance, parallelization tasks were undertaken.

MPI is a communication protocol and is used as the industry standard for the message passing model where a certain application comprises of a set of tasks which are assigned their own local memory whose location can be in the same machine or across several machines. Data exchange to conduct the operation by tasks is established by sending and receiving messages [8]. MPI provides programmers the flexibility to use it as a low-level approach with a detailed control on the flow of data, or as a high-level programming approach with parallel libraries designed to provide optimized performance without going into the depths of the MPI algorithm [9].

Solving any 3D magnetostatics problems in Radia comprises of 3 subsequent steps: calculation of elements of a large (often tens GBs memory size) matrix called Interaction Matrix and describing magnetic interaction between 'active' sub-volumes of a magnet geometry, created by segmentation (1), performing a relaxation procedure on it to determine values of the magnetization vector in all the "active" sub-volumes (2), and computing magnetic field and/or field integrals or other characteristics of magnetic fields, created by the sub-volumes with magnetization or current density (3). The two sections– the generation of the interaction matrix (1) and the calculation of magnetic field values after relaxation (3) are 'embarrassingly parallel' algorithms and hence have been parallelized using MPI at the C++ level.

Interaction matrix is a dense matrix in Radia, that may occupy a large memory, depending on geometry and its segmentation. Relaxation of this matrix may require a large number of iterations, in particular when solving complicated iron-dominated geometries [2, 4]. At each of these iterations, a relatively small number of multiplications needs to be done to take into account (or update) magnetization vector in each sub-volume (created by segmentation). This makes application of MPI not very efficient for parallelizing this algorithm. Instead shared memory-based parallelization, implemented using native multithreading capabilities introduced in C++11 standard library has been considered for parallelizing the specific section of the Radia code. The structure used for the tests is discussed in detail in the following section titled 'Test Magnetic Structure', followed by the outcomes detailed in 'Results of Benchmarking'.

3. Test magnetic structure

To evaluate the parallel performance of the new version of the Radia 3D magnetostatics code, we used a 3D model of a 14 mm-period, hybrid, in-vacuum, Cryo-cooled Permanent Magnet Undulator (CPMU) at 4 mm magnetic gap (see figure 1). This model comprises of an upper and a lower array with each of these arrays including an alternating sequence of permanent magnets (PrFeB with 1.67 T remnant magnetization) and soft iron poles (Vanadium Permendur with ∼2.29 T magnetization at saturation). The model geometry is generated by mirroring with respect to the three orthogonal symmetry planes passing through the magnetic centre. The geometry (specifically the poles) has been extensively subdivided to achieve solving accuracy better than 0.001 with respect to the peak magnetic field. 22680 independent sub-volumes are produced because of implementing subdivision before applying the symmetries, consequently yielding a dense interaction matrix of size ∼2 GB. The final calculated on-axis vertical field after solving the geometry for the magnetization in the sub-volumes is shown in figure 2.

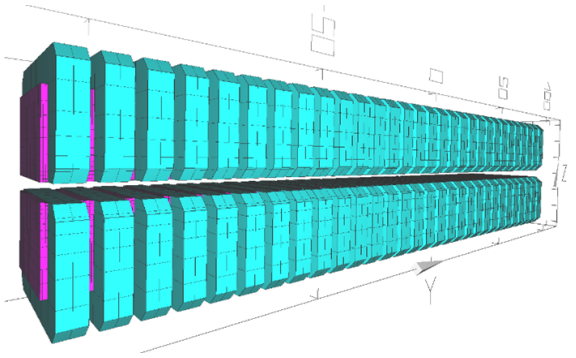

Figure 1. Radia 3D model of the hybrid, in-vacuum, CPMU used for our parallel performance tests.

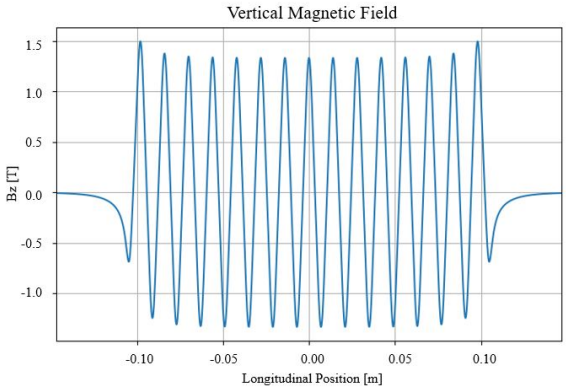

Figure 2. Calculated on-axis vertical magnetic field along the CPMU structure using Radia.

4. Results of benchmarking

Tests to evaluate parallel performance on many server nodes were conducted at the National Energy Research Scientific Computing Centre (NERSC), specifically the NERSC supercomputer Cori. Cori is a Cray XC40 with a theoretical peak performance of 30 petaflops. Execution times for each section were obtained at the end of each test, iteratively performed starting from 1 process (i.e., serial execution) and going up to 1024 processes and then averaged over 3 runs for each process number, thereby estimating the performance characteristics.

The parallel version of the Radia code performed well for both the cases – computation of elements of the interaction matrix and computation of magnetic field post relaxation of this matrix. For both the instances, a systematic decline in the required execution times was observed with increasing number of processes, see figure 3, that can be interpreted as the corresponding computational speedup achieved because of parallelizing the Radia code, leading to a maximum speedup value of ∼200 for the setting up of interaction matrices, and approximately ∼600 for computation of field values post relaxation, see figure 4.

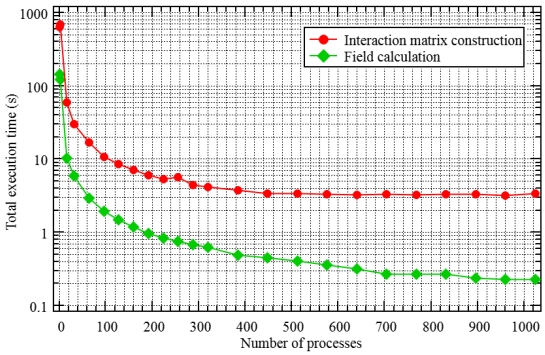

Figure 3. Mean time to set up the interaction matrix and calculate magnetic field, both as a function of the number of MPI processes.

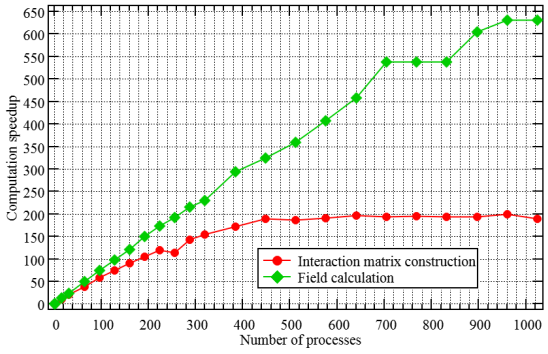

Figure 4. Total speedup in time values to set up interaction matrix and calculate field vs. number of MPI processes.

The performance corresponding to computation of elements of the interaction matrix demonstrates significant speed gain only until number of parallel processes is lesser than 400,

whereas no such parallel performance saturation is observed for the function of field calculation after relaxation. Thus, profiling of the parallel performance was implemented at the C++ level, where time stamps were introduced at the beginning and end of specific functions in the code, to evaluate their corresponding execution times. Functions specifically related to communication and exchange of data between processes were profiled, namely 'MPI_Send' and 'MPI_Recv'. Primarily, 'MPI_Send' is executed by workers after they have computed the data packets to be sent to the master, and 'MPI_Recv' is performed by the master to receive the data packets sent by each worker. The plots demonstrating the total time required for the execution of 'MPI_Send' by each worker and 'MPI_Recv' by the master are given in figure 5 for setting up of interaction matrix and figure 6 for calculation of field post relaxation.

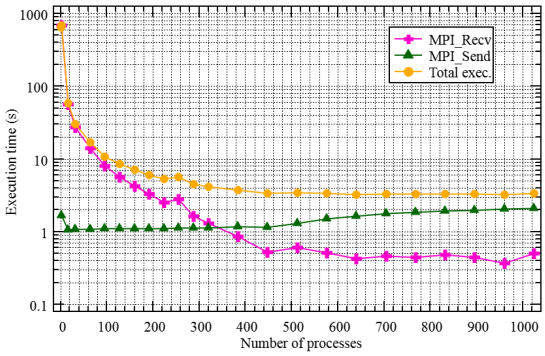

Figure 5. Total time to execute 'MPI_Send' by each worker, 'MPI_Recv' by master, and complete execution of interaction matrix generation vs. number of MPI processes.

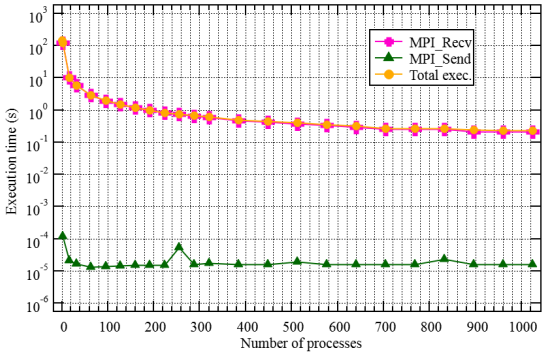

Figure 6. Total time to execute 'MPI_Send' by each worker, 'MPI_Recv' by master, and complete execution of field calculation vs. number of MPI processes.

For interaction matrix construction (figure 5), it can be observed that at some point near the saturation, the time spent by a worker in 'MPI_Send' exceeds the total time spent by the master in 'MPI_Recv', such a crossover is absent in case of the field calculation post relaxation, where the time spent by a worker in 'MPI_Send' is always much smaller than the time spent by the master in 'MPI_Recv', as observed in figure 6. The role of the master is to receive data after the calculation by workers. Thus, initially (for lesser number of workers), for interaction matrix generation, master time is mostly spent waiting for workers to finish the computation. But with increasing number of workers, the computation time decreases (i.e., each worker executes lesser number of calculations) and eventually master is unable to receive quickly enough all the computed data, thus delaying the workers who have already finished computing and instead get queued to send the data. This saturation is absent in the case of calculation of the field after the relaxation, when each worker performs large number of summing up operations to account for contributions from all sub-volumes of the 3D geometry to the field value at a given point in space. This is confirmed by the fact that in our tests, the time spent by a worker on calculation of one element of the interaction matrix was only 6.6 µs, while the calculation of the magnetic field after the relaxation by a worker at one observation point took a much longer time, 22.5 ms. These observations explain the earlier saturation of the overall speedup curve in the case of the interaction matrix generation, as compared to the field calculation after the relaxation.

Scalability of shared memory parallelization when compared to MPI, is weaker, but the resource sharing is better optimized. Preliminary tests to estimate the parallel performance of the relaxation of interaction matrix yielded a speedup of approximately 10 times compared to sequential execution of the code. The estimated overall speedup for the test geometry is ∼41 when using 400 processes, the extent of parallelization dependent on the number of field

calculations performed. Further algorithm-specific implementations are required to reach higher levels of parallelism.

5. Conclusion

The upgraded version of parallel Radia is now available for the Python interface, and it offers considerable speedups in different types of 3D magnetostatics computations [5]. Efforts are ongoing to improve its parallel performance in order to efficiently meet the computation requirements of the next generation light sources.

Acknowledgements

This work was supported by the US DOE BES SBIR grant no. DE-SC0018556.

References

[1] Chubar O, Elleaume P and Chavanne J 1998 *J. Synchrotron Radiat.* **5** 481-84
[2] Elleaume P, Chubar O and Chavanne J 1997 *Proc. 17th Particle Accelerator Conf. (PAC'97)* (Vancouver, B.C., Canada) pp 3509-11
[3] Andersson A, Chubar O and Lindgren L J 1998 *Proc. 6th European Particle Accelerator Conf. (EPAC'98)* (Stockholm, Sweden) p 1207
[4] Chubar O, Benabderrahmane C, Chavanne J, Elleaume P, Marcouillé O and Marteau F 2004 *Proc. 9th European Particle Accelerator Conf. (EPAC'04)* (Lucerne, Switzerland) pp 1675-7
[5] Radia `https://github.com/ochubar/Radia`
[6] Hall C *et al.* 2022 *The 14th Int. Conf. on Synchrotron Radiation Instrumentation (SRI2021)* (Humburg, Germany) unpublished
[7] Le Bec G, Chavanne J, Liuzzo S and White S 2021 *Phys. Rev. Accel. Beams* **24** 072401
[8] Kang S J, Lee S Y and Lee K M 2015 *Adv. Multimedia* **2015**
[9] Gropp W D and Lusk E 2003 *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting* 27