PARALLEL OPTIMIZATION OF BEAM-BEAM EFFECTS IN HIGH ENERGY COLLIDERS*

J. Qiang† and R. D. Ryne, LBNL, Berkely, CA 94720, USA

Abstract

Beam-beam effects limit luminosity in high energy colliders. Parallel beam-beam simulation codes were developed to study those beam-beam effects and to help collider design. In this paper, we will present a parallel parameter optimization algorithm integrated together with a parallel beam-beam simulation code to optimize the luminosity of two colliding beams. This algorithm is based on a differential evolutionary global optimization method and takes advantage of the two-level parallelization in both parallel search and parallel objective function evaluation. This significantly increases the scalability of the simulation on petascale supercomputers and reduces the computing time for finding the optimal tune working point.

INTRODUCTION

Collective beam-beam effects play an important role in high energy collider performance. In order to reach a higher luminosity, the beam-beam effects need to be optimized with respect to the collider machine parameters such as tune working points. The brute force method based on parameter scans in multi-dimensional tune space using a self-consistent beam-beam simulation model is extremely time consuming given the fact that a single working point simulation might take more than 10 hours. In this paper, we propose using a parallel differential evolution algorithm together with a parallel beam-beam simulation model for beam-beam studies and collider optimization.

COMPUTATIONAL BEAM-BEAM MODEL

The computer code used in this study is the BeamBeam3D code [1]. The BeamBeam3D is a parallel three-dimensional particle-in-cell code to model beam-beam effect in high-energy ring colliders. This code includes a self-consistent calculation of the electromagnetic forces (beam-beam forces) from two colliding beams (i.e. strong-strong modeling), a linear transfer map model for beam transport between collision points, a stochastic map to treat radiation damping, quantum excitation, an arbitrary orbit separation model, and a single map to account for chromaticity effects. Here, the beam-beam forces can be from head-on collisions, offset collisions, and crossing angle collisions. These forces are calculated by solving the Poisson equation using a shifted integrated Green function method, which can be computed very efficiently using an FFT-based algorithm on a uniform grid. For the crossing angle collisions, the particles are transformed from the laboratory frame into a boosted Lorentz frame, where the beam-beam forces are calculated the same as in the head-on case. After the collision the particles are transformed back into the laboratory frame. The BeamBeam3D code can handle multiple bunches from each beam collision at multiple interaction points (IPs). The parallel implementation is done using a particle-field decomposition method to achieve a good load balance. Recently, three beam-beam compensation models – conducting wire, electron lens, and crab-cavity – were added to the code.

* Work supported by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

† jqiang@lbl.gov

PARALLEL DIFFERENTIAL EVOLUTION ALGORITHM

The differential evolution algorithm is a relatively new method in evolutionary algorithms [2]. It is a simple but powerful population-based, direct-search algorithm with self-adaptive step size to generate next-generation offspring for global optimization. It has been successfully used in a variety of applications and demonstrated its effectiveness [3, 4, 5]. In this algorithm, a group of population with size NP in control parameter space is randomly generated at the beginning. This population is taken as the first generation of the control parameters. A new generation of control parameter population is generated as follows: For each parameter vector $x_{i,G}, i = 0, 1, 2, \cdots, NP-1$ in a population size NP at generation G, a perturbed vector v_i is generated according to

$$v_i = x_{i,G} + F_{CR}\,(x_{b,G} - x_{i,G}) + F_{xc}\,(x_{r1,G} - x_{r2,G}) \quad (1)$$

where the integers $r1$ and $r2$ are chosen randomly from the interval $[1, NP]$ and are different from the running index i, F_{xc} is a real scaling factor that controls the amplification of the differential variation $(x_{r1,G} - x_{r2,G})$, $x_{b,G}$ is the best parameter solution in generation G, and where F_{CR} is a combination weight factor between the original individual parent and the best parent. In most typical simulations, F_{xc} is set to 0.8, and F_{CR} is chosen from a uniform random number between 0 and 1. In order to increase the diversity of the parameter vectors, crossover between the parameter vector $x_{i,G}$ and the perturbed vector v_i is introduced with an externally supplied crossover probability Cr to generate a new trial vector $U_{i,G+1}, i = 0, 1, 2, \cdots, NP-1$. For a D dimensional control parameter space, the new trial parameter vector $U_{i,G+1}, i = 0, 1, 2, \cdots, NP-1$ is generated using the following rule:

$$U_{i,G+1} = (u_{i1,G+1}, u_{i2,G+1}, \cdots, u_{iD,G+1}) \quad (2)$$

$$u_{ij,G+1} = \begin{cases} v_{ij}, & if\ rand_j \le CR\ \ or\ j = mbr_i \\ x_{ij}, & otherwise \end{cases} \quad (3)$$

where $rand_j$ is a randomly chosen real number in the interval $[0, 1]$, and the index mbr_i is a randomly chosen integer in the range $[1, D]$ to ensure that the new trial vector contains at least one parameter from the perturbed vector. In most simulations, the Cr is chosen as 0.5. Next, the new trial solution $U_{i,G+1}$ is checked against the original parent $x_{i,G}$. If the new trial solution produces a better objective function value, it will be put into the next generation $(G + 1)$ population. Otherwise, the original parent is kept in the next generation population. The above procedure is repeated for all NP parents to generate a new generation of population. This completes one iteration. Many iterations or generations are used to attain the final global optimal solution.

The above population based differential evolutionary optimization algorithm naturally leads to a multi-processor parallel implementation. Our method contains two levels of parallelization. First, the whole population is distributed among a number of groups of parallel processors. Each group of processors contains a subset of the whole population. Different sets of the sub-population can be tracked simultaneously. Second, each objective function evaluation corresponds to an accelerator simulation, for which parallel codes are available. Here, the objective function is the peak luminosity computed using the parallel BeamBeam3D code run on the number of processors inside a group. Such a two-level parallelization scheme has good parallel scalability and allows a large number of processors (beyond many tens of thousands) to be used during the process of the optimization. This will significantly reduce the computational time-to-solution required to find an optimal working point in high energy colliders. Table 1 shows a weak scaling test of the parallel optimization code using the BeamBeam3D code on a Cray XT-5 supercomputer at the Oak Ridge National Laboratory. It is seen that the simulation scales very well even up to 100, 000 processors.

Table 1: Weak Scaling Test on Cray XT-5

processors	time (sec)	problem size	efficiency
6400	2522	100	1
12800	2611	200	0.97
25600	2700	400	0.93
51200	2890	800	0.87
102400	2710	1600	0.93

OPTIMIZATION OF LUMINOSITIES AT THE LHC AND THE ELIC

In the following, we applied the above parallel differential evolution algorithm together with the parallel beam-beam simulation to optimizing luminosities at the Large Hadron Collider (LHC) and the proposed Electron Light Ion Collider (ELIC). The major parameters used in the beam-beam simulation for the LHC are summarized in Table 2 [6]. Fig. 1 shows the peak luminosity from single bunch collisions as a function of the population generation for the LHC optimization. Here, we have used a population size of 100 in each generation. It is seen that after 600 generations, the luminosity has been improved by about 100%. The above simulation took 3 hours on 12800 Cray XT4 processors at the National Energy Scientific Computing Center (NERSC). The same simulation would have taken more than one year if a single processor workstation were used. Fig. 2 shows the peak luminosity evolution with the nominal tune working point (0.31, 0.32) and the optimized tune working point (0.4752, 0.4998), (0.4983, 0.4997). The optimized working point produces significantly larger luminosity in comparison with the nominal tune working point. This is due to the large dynamic beta effects with the new working point. However, this working point might not be practically usable due to the presence of quadrupole errors that could drive a half integer resonance.

Table 2: Major Parameters for LHC

Quantity	LHC p
energy (GeV)	7000
particles per bunch (10^{11})	1.15
horizontal β^* (m)	0.5
vertical β^* (m)	0.5
horiztonal emittance (unnorm.) (um)	0.000512
vertical emittance (unnorm.) (um)	0.000512
rms bunch length (m)	0.077
rms relative energy spread (10^{-3})	0.111
damping time	10^9
synchrotron tune	0.00212
nominal horizontal tune	0.31
nominal vertical tune	0.32

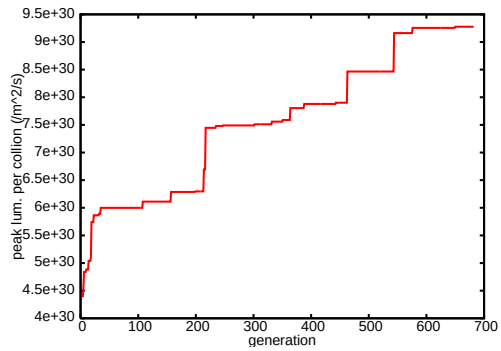

Figure 1: Peak luminosity as a function of generation for the LHC optimization.

We also applied the parallel beam-beam optimization to ELIC luminosity optimization. The major parameters used in the beam-beam simulation for ELIC are summarized in Table 3 [7]. Fig. 3 shows the peak luminosity from single

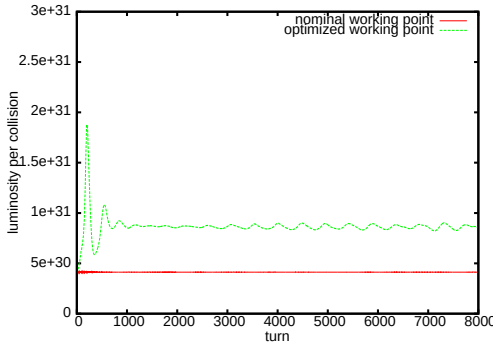

Figure 2: Peak luminosity evolution with the nominal tune working point and the optimized tune working point at LHC.

Table 3: Major parameters for ELIC

Quantity	e^-	**p**
energy (GeV)	5	60
particles per bunch (10^{10})	1.25	0.416
horiztonal β^* (m)	0.1	0.1
vertical β^* (m)	0.02	0.02
horiztonal emittance (unnorm.) (um)	0.00548	0.00548
vertical emittance (unnorm.) (um)	0.0011	0.0011
rms bunch length (m)	0.0075	0.01
rms relative energy spread (10^{-3})	0.71	0.71
damping time	1516	10^9
synchrotron tune	0.045	0.045
nominal horizontal tune	0.1	0.083
nominal vertical tune	0.1618	0.1343

bunch collisions as a function of the evolution generation for the ELIC optimization. The peak luminosity has been

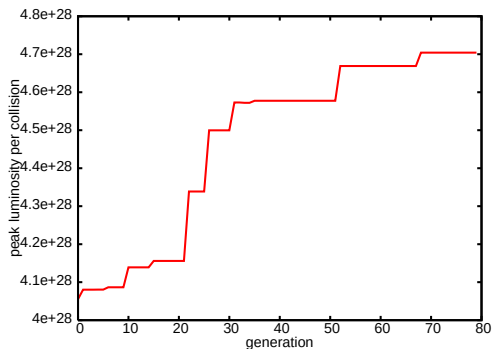

Figure 3: Peak luminosity as a function of generation for ELIC optimization.

improved by about 20% after 80 generations. Fig. 4 shows the peak luminosity evolution with the nominal tune working point $(0.1, 0.1618)$, $(0.083, 0.1343)$ and the optimized tune working point $(0.02498, 0.08563)$, $(0.5011, 0.5862)$. The optimized working point produces about 30% larger luminosity in comparison with the nominal tune working point.

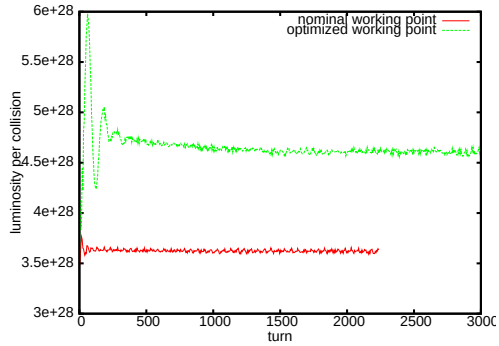

Figure 4: Peak luminosities evolution with the nominal tune working point and the optimized tune working point at ELIC.

ACKNOWLEDGEMENTS

This research used computer resources at the National Energy Research Scientific Computing Center and at the National Center for Computational Sciences.

REFERENCES

[1] J. Qiang, M. A. Furman, R. D. Ryne, J. Comp. Phys., vol. 198, p. 278 (2004).

[2] R. Storn and K. Price, Journal of Global Optimization 1a1:341-359, (1997).

[3] F. Xue, A. C. Sanderson and R. J. Graves. in Proceedings of the 2003 Congress on Evolutionary Computation (CEC'2003), Volume 2, pp. 862–869, IEEE Press, Canberra, Australia, December 2003.

[4] B. V. Babu, P. G. Chakole, J. H. Syed Mubeen, Chemical Engineering Science, vol. 60, p. 4822 (2005).

[5] X. Wang, M. Hao, Y. Cheng, R. Lei, Journal of Universal Computer Science, vol. 15, no. 4 (2009), 722-741.

[6] Y. Papaphilippou, F. Zimmermann, PRST-AB 2, 104001 (1991).

[7] B. Terzic, Y. Zhang, in Proceedings of IPAC'10, p. 1910, Kyoto, Japan, 2010.