# GPU-based First Collision Detection in Parton Cascade in Heavy-ion Collisions

Qing-Jun Liu[1], Fang Liu[2], Ning-Ming Nie[2] , Chun-Bao Zhou[2], Wei-Qin Zhao[3]

[1]Department of Mathematics and Physics，

Beijing Institute of Petro-chemical Technology

Beijing 102617, China

liuqingjun@bipt.edu.cn

[2]Supercomputing Center, Computer Network Information Center

Chinese Academy of Sciences

Beijing 100190, China

liufang@sccas.cn, nienm@sccas.cn, zhoucb@sccas.cn

[3]Institute of High Energy Physics，Chinese Academy of Sciences

Beijing 100049, China

zhaowq@mail.ihep.ac.cn

## ABSTRACT

Zhang’s Parton Cascade (ZPC) is an essential module in A Multi-Phase Transport (AMPT) model, which applies Monte-Carlo methods for simulating ultra-relativistic heavy-ion collisions. Our ultimate goal is to accelerate ZPC, and we start with the acceleration of the detection of two-parton collisions in the cascade. In this work, using CUDA C, we parallelized and then tested the code for the detection of the first two-parton collision in the ZPC-based simulation of parton cascade. We compared results from the parallelized code and the serial code programmed in C. The comparison has shown for the first time that a significant speedup can be achieved by utilizing GPU as a co-processor for the detection of the first two-parton collision in the ZPC-based simulation of parton cascade in heavy-ion collisions of Pb - Pb at sqrt(sNN)=2.76 TeV, which is the top colliding energy for heavy-ion collisions at the Large Hadron Collider (LHC). The parallelized code is readily available per request, and will be integrated into GPU-based simulation of parton cascade in the near future. This study clearly indicates that ZPC may be accelerated and so is the event generation of Monte-Carlo events for heavy-ion collisions at the LHC.

## KEYWORDS

GPU, CUDA, Simulation, Parton cascade, Heavy-ion collision.

## 1 INTRODUCTION

Monte-Carlo simulation plays an important role in the study of ultra-relativistic heavy-ion collisions [1-4]. While event generation based on Monte-Carlo model helps to reveal and understand the physical mechanisms of the observed phenomena in heavy-ion collisions, generating the required huge amount of events is usually compute-intensive. In recent years, many compute-intensive applications in areas such as signal processing [5], molecular dynamics [6] and computational fluid dynamics (CFD) [7] have gained significant speedup by using GPU. Naturally, one would expect to explore ways of accelerating event generation by employing GPUs. A highly cited Monte-Carlo event generator is AMPT [1], the version with string-melting. Written in FORTRAN, it consists of three modules: HIJING [2], ZPC [3] and ART [4]. These modules run serially one after the other on CPU. ZPC simulates parton cascade for a system of partons, which are from HIJING after string-melting. The input for ZPC mainly includes the space-time and momentum-energy information for all of the partons. Because the simulation of the parton cascade involves detection of two-body collisions among parton-pairs out of tens of thousands of partons in the case of heavy-ion collisions at the LHC energy, it is non-trivial [8]. Therefore it is important to explore an algorithm of speedup by using GPU for simulating the parton cascade, in order to speed up Monte Carlo event generation in high energy heavy-ion collisions. In this paper, we give in Section 2 an outline of parton cascade and two-parton collision, then in Section 3 the computing environment is briefly introduced; after presenting algorithm, its implementation and results in the 4th Section we give summary and conclusions in section 5.

## 2 TWO-PARTON COLLISION AND PARTON CASCADE

Parton cascade is a successive two-parton collision in a system of partons. The partons, formed in the initial stage of ultra-relativistic heavy-ion collisions through a string-melting mechanism as implemented in AMPT, will experience two-body collisions according to the perturbative quantum chromodynamics [9] and certain geometrical conditions. As in ZPC, in the simulation of parton cascade in a heavy-ion collision event, the first step is to detect the earliest collision among many possible two-parton collisions (collision detection), then simulate the collision and thereby update the momentum-energy and space-time information for the involved two partons. As in ZPC, the rest of the simulation of the parton cascade is to repeat the above two steps until no two-parton collision is detected to happen earlier than a preset physical time threshold. For more details, interested readers are referred to Ref. [3]. One may note that the detection of the first two-parton collision involves the computation of the collision time for all of the possible colliding pairs and the detection of the following successive collision involves only the re-calculation of the collision time for all the pairs including one of the most recently collided two partons. Therefore it is worthwhile to pay special attention to the detection of the first two-parton collision in the simulation of the parton cascade.

## 3 BRIEF INTRODUCTIONS ABOUT THE COMPUTING ENVIRONMENT

The computing environment for this study is provided by the facilities Deepcomp 7000, which is hosted at the Supercomputing Center of the Chinese Academy of Sciences. More specifically, we used a CPU that is Intel Xeon E5410 @ 2.33GHz, and a GPU that is NVIDIA C1060 Tesla T10; CUDA3.2 [10] and g++/gcc/gfortran-4.1.2 [11] are the main development tools; Red Hat 4.1.2-44 Linux 2.6.18-128.el5 for x86_64 is the operating system we used.

## 4 ALGORITHM AND RESULTS

The algorithm is formulated as follows. First of all, let program loop over all the 0.5(N-1)N parton pairs for the N-parton system obtained by running AMPT after string-melting. If a pair collides according to the physical and geometrical collision criteria defined in ZPC, then the collision time is calculated and recorded for this pair. What follows is the calculation of the smallest collision time among all of the colliding pair. The first collision is detected to happen between the pair of partons that collide at the smallest collision time.

Here we briefly introduce the implementation of the aforementioned algorithm for GPU by using CUDA C [10,12]. We launch the CUDA kernel with the block size set to be 128, according to the calculation using CUDA Occupancy Calculator [9]. Each thread in a block first computes the possible collision time of one parton with its partners and then saves the smallest collision time for the parton in shared memory. After synchronization by invoking, the smallest collision time for each of the 128 partons within the same block is read out from the shared memory and the smallest collision time among the 128 smallest, is calculated, then together with the indices of this colliding pair of partons are stored into the corresponding positions of three arrays allocated on the global memory. Now it is the instant for the CUDA kernel to exit. Next, when all the blocks of threads have finished their jobs, we copy the above three arrays back to host to do further reduction on CPU for better performance since the block number is a few hundreds and relatively small. Finally the task of detecting the first two-parton collision is terminated through this stage of reduction, which reads the three arrays and gets us the wanted information for the first collision: indices of the two partons and the time at which the first collision happens.

Additionally, we have also implemented the algorithm for collision detection in C for CPU.

Running the two implementations, one can get the same collision time and the same indices for the colliding two partons for the first two-parton collision in the parton cascade, the same as the original ZPC outputs. Therefore the two implementations are both correct.

In Table.1, we tabulated the time both CPU and GPU used for detecting the first two-parton collision in parton cascade for parton system consisting of N partons. With a specific value of N, each of the parton systems is formed in a collision of Pb + Pb at sqrt(sNN) =2.76 TeV with a specific impact parameter b, which is in the range of 0 to 10 fm. It can be seen that the speedup using GPU relative to using CPU is over 60. When obtaining the N-parton systems, we ran AMPT with the random number for HIJING set as 53153511. Additionally, both t_cpu and t_gpu are from analysis of one parton system formed in one Monte Carlo event of heavy-ion collision, and are the average of 20 runs, respectively. The significant speedups listed in Table.1 shows that it is promising to accelerate real-time simulation of the parton cascade in heavy-ion collisions at top LHC energy by using GPU.

**Table 1.** The time used for the detection of the first two-parton collision in simulating parton cascade in N-parton systems formed in heavy-ion collisions of Pb + Pb at sqrt(sNN)=2.76 TeV with impact parameter $0 \leq b \leq 10$ fm, and the speedups of using GPU against CPU.

| N | b/fm | t_cpu /s | t_gpu/s | Speedup |
|---|---|---|---|---|
| 45189 | 0 | *36.133* | *0.538* | 67 |
| 44837 | 1 | *35.392* | *0.544* | 65 |
| 43150 | 2 | *32.799* | *0.518* | 63 |
| 37082 | 3 | *24.079* | *0.353* | 68 |
| 30201 | 4 | *15.880* | *0.228* | 70 |
| 22248 | 7 | *8.619* | *0.116* | 74 |
| 10355 | 10 | *1.85* | *0.026* | 71 |

## 5 SUMMARY AND CONCLUSIONS

We have formulated an algorithm for the detection of two-parton collision in the simulation of parton cascade in ultra-relativistic heavy-ion collisions, and then implemented the algorithm in C and CUDA C, respectively. Running each of the two implementations, C code on CPU and CUDA C code on GPU, the time used for detecting the first two-parton collision was calculated for each of the parton systems formed in heavy-ion collisions of Pb + Pb at sqrt(sNN)=2.76 TeV with impact parameter ranging from 0 to 10 fm. Then we compared the results from running the C code on CPU with those from running the CUDA C code using GPU as a co-processor. The comparison demonstrated that speedups over 60 can be achieved for detecting the first two-parton collision by employing GPU. The significant speedups imply that ZPC[2] can be accelerated if modified to make good use of GPU, therefore one may expect that the event generation for ultra-relativistic heavy-ion collisions at the top LHC energy can be accelerated if using AMPT[1] with the ZPC replaced by a GPU-based parton cascade.

## ACKNOWLEDGEMENT

Computing support is provided by Supercomputing Center，Computer Network Information Center, Chinese Academy of Sciences. This work is financially supported by Beijing Natural Science Foundation, under Project No.1132017 and National Natural Science Foundation of China.

## 6 REFERENCES

[1] Z.W. Liu, C.M. Ko,B.A. Li, and S. Pal, “A Multi-phase transport model for relativistic heavy ion collisions,” Phys. Rev. C, Vol. 72, pp. 064901, December 2005

[2] X.N. Wang and M. Gyulassy, “HIJING: A Monte Carlo model for multiple jet production in pp, pA and AA collisions,” Phys. Rev. D, Vol. 44, pp 3501-3516, December 1991; “HIJING 1.0: A Monte Carlo program for parton and particle production in high-energy hadronic and nuclear collisions,” Comput. Phys. Commun.,vol. 83, pp. 307–331, December 1994.

[3] B. Zhang, “ZPC 1.0.1: A Parton cascade for ultrarelativistic heavy ion collisions, ” Comput. Phys. Commun., vol. 109, pp. 193–206, April 1998.

[4] B.A. Li, and C. M. Ko, “Formation of superdense hadronic matter in high-energy heavy ion collisions,” Phys. Rev. C., Vol. 52, pp. 2037-2063, October 1995.

[5] C. Harris, K. Haines, L.S. Smith, “GPU accelerated radio astronomy signal convolution,” Exp. Astron. Vol. 22, pp. 129-141, October 2008.

[6] M.S. Friedrichs, P. Eastman, V. Vaidyanathan, M. Houston, S. Legrand, A.L. Beberg, D.L. Ensign, C.M. Bruns, V.S. Pande, “Accelerating molecular dynamic simulation on graphics processing units,” J. Comput. Chem., Vol 30, pp. 864–872 , April 2009.

[7] C.M. Bard and J.C. Dorelli “A simple GPU-accelerated two-dimensional MUSCL-hancock solver for ideal magnetohydrodynamics,” J. Comput. Phys., Vol. 259, pp. 444-460, February 2014.

[8] R. Li, H. Jiang, H.C. Su, J. Jenness, B. Zhang, “Speculative parallelization of many-particle collision simulations,” Proceedings of the 2007 International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 128–136, June 2007.

[9] B. L. Combridge, J. Kripfgang, and J. Ranft, “Hadron production at large transverse momentum and QCD,” Phys. Lett. B, Vol. 70, pp. 234-238, September 1977.

[10] NVIDIA Corp., CUDA Toolkit, version 3.2, November 2010.

[11] GCC Team, https://gcc.gnu.org/gcc-4.1.2, February, 2007.

[12] J. Sanders and E. Kandrot, CUDA by Example: An Introduction to General-Purpose GPU Programming. Beijing: Addison Wesley and Tsinghua University Press, 2010.