GPU-Accelerated Parton Cascade in Heavy-Ion Collisions

Qingjun Liu, Weiqin Zhao, Fang Liu, Ningming Nie, and Chunbao Zhou

Abstract—A widely used Monte Carlo event generator is A Multi-Phase Transport model (AMPT) for relativistic heavy-ion collisions. It depends on Zhang’s Parton Cascade (ZPC) package to simulate initial stage parton cascade. Based on ZPC, we have developed a code for the simulation of the parton cascade to exploit the powerful parallel processing capability of GPU. The goal is to accelerate the simulation of the parton cascade in a system of partons that is formed in ultrarelativistic heavy-ion collisions. Named PCG (Parton Cascade on GPU), the code makes real time collision detection among N interacting partons formed in a heavy-ion collision parallelized. The parallelization was implemented by using CUDA C. With simulating Pb-Pb collisions at sqrt(sNN)=2.76 TeV as a use case, we first verified the correctness of PCG through comparison of the output of PCG with those of ZPC, then we estimated the computational efficiency of PCG to be 2x to 3x relative to ZPC, which is a serial code and only runs on CPU. Therefore PCG is viable for being integrated into AMPT for simulating heavy-ion collisions and can save large amount of computing resources for large scale AMPt-based event generation in ultrarelativistic heavy-ion collisions at sqrt(sNN)=2.76 TeV.

Index Terms—GPU, CUDA C, simulation of parton cascade, ultrarelativistic heavy-ion collision.

I. INTRODUCTION

Monte-Carlo event generators [1]-[6] are essential in high energy physics. Through analysis of large number of events generated by event generators, scientists not only get insight into physical mechanisms on the theory side [7]-[9], but also evaluate detector performance on the experiment side [10]-[12]. However in order to generate statistically significant number of events, huge amount of resources, in terms of computing time as well as electrical power, need to be consumed. Therefore it is of great interest to accelerate event generation by utilizing emerging high performance computing technologies, such as those involving graphical processing units (GPUs) and CUDA [13], [14], which have been enabling variety of applications [15]-[18] to gain phenomenal speedups.

Nowadays, in the field of ultrarelativistic heavy-ion collisions, one of the widely used Monte Carlo event generators is A Multi-Phase Transport (AMPT) model [1]. It simulates high energy heavy-ion collisions and is of great help for physicists to study, on the surface of the earth, a new form of matter called quark-gluon-plasma [19], which exists immediately after the birth of the cosmos according to the Big Bang theory [20]. Among the three main modules of AMPT [1], HIJING [2], ZPC [3] and ART [4], which run on CPU serially and are programmed in FORTRAN, ZPC mainly simulates parton cascade process that dominates the initial stage of ultrarelativistic heavy-ion collisions. Because ZPC utilizes most of the time for simulating the initial stage of a heavy-ion collision when QGP is formed through the parton cascade, this work focuses on GPU-based acceleration of the simulation of the cascade process.

The rest of this paper is mainly organized as follows. In Section II, we present brief introduction of the simulation of the parton cascade process in ultrarelativistic heavy-ion collisions. Then an algorithm for developing the parton cascade on GPU (PCG) to accelerate the cascade is introduced in Section III. In Section IV, we present results from PCG compared with those from ZPC. We give summary and conclusion in Section V.

II. OUTLINE OF THE SIMULATION OF PARTON CASCADE

Taking place in a system of interacting N-partons, parton cascade can simply be described as successive two-parton collisions, which follow the laws of the perturbative quantum chromodynamics [21] and satisfy certain geometrical criteria for the collision to happen. According to AMPT, the system of partons may be formed through a melting mechanism due to high temperature in the early stage of heavy-ion collisions [1]. As in ZPC, the first step in the simulation of parton cascade in a heavy-ion collision is to detect the earliest two-parton collision among a large number of possible two-parton collisions that satisfy the collision criteria, then simulates the collision thereby the momentum-energy and space-time information for the involved two partons are updated. According to 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. At this stage the parton cascade terminates. It is based this account of the parton cascade that we designed in this work an algorithm for the GPU-accelerated simulation of the parton cascade in ultrarelativistic heavy-ion collisions. For more details about the simulation of the parton cascade within ZPC and its optimization, interested readers are referred to [3], [22].

III. ALGORITHM FOR PCG

The main input to PCG includes the space-time and
momentum-energy information for each of the partons in an 
N-parton system that is formed through string-melting when 
running AMPT to generate an event for an ultrarelativistic 
heavy-ion collision [1], [3]. This information will be updated 
during the parton cascade through two-parton collisions. In 
our previous publication [23], we reported a speedup of 
around 60 for the detection of the first two-parton collision in 
Pb-Pb collisions at sqrt(sNN)=2.76 TeV, where we used 
single precision in our calculations. In this work, all of our 
calculations are carried out by using double precision. The 
algorithm for simulating parton cascade in the N-parton 
system is formulated in the following three sub-sections and is 
implemented in a CUDA C code named cudarun.cu. The 
output of PCG are the space-time and momentum-energy 
information for all of the N partons at their last collisions and 
are stored in the data structures that store the main input 
information. These data structures are eight arrays of double 
precision. Due to space limitations, we intend to describe 
more details of all of the data structures and pseudo-code, in 
a separate paper. However in APPENDIX A, interested readers 
may have a brief sketch about the infrastructure of our 
algorithm for PCG.

A. Detection of the First Two-Parton Collision
First of all, let program loop over all of the 0.5(N-1) N 
parton pairs. In the loop, for each pair of partons, based on 
calculations using the space-time and momentum-energy 
information of the two partons, program can tell if the pair 
collides or not according to the physical [21] and geometrical 
collision criteria as defined in ZPC [3]. If a pair collides then 
the collision time for that pair is saved in global memory, 

together with their indices. What follows is the calculation of 
the smallest collision time among all of the collision time for 
all of the colliding pairs. For the calculation, we used a 
reduction procedure [23] where shared memory on GPU is 
utilized. The first collision is detected to happen between the 
pair of partons that collide at the smallest collision time. 
Using CUDA C, we implemented the aforementioned 
algorithm for GPU in a CUDA kernel named datTime() and it 
is launched with the block size set to be 128, according to the 
calculation using CUDA Occupancy Calculator [13]. The 
algorithm for the detection of the first two-parton collision is 
pretty much as previously reported in [23], however is 
modified in this work to use double precision. Furthermore, in 
this work during the detection of the first two-parton collision, 
we have the collision time and the indices of the colliding two 
partons for all of the colliding pairs saved in global memory. 
These saved data may be updated during the simulation of the 
whole parton cascade process, and are reused for the detection 
of the next earliest two-parton collision, as one may see in the 
following two sub-sections.

B. Simulation of Two-Parton Collision
Once the indices of the colliding two partons and their 
collision time have been determined for the first or the next 
earliest two-parton collision, the immediate task is to simulate 
the two-parton collision. Following the laws of the 
perturbative quantum chromodynamics [21], simulation of 
two-parton collision has been implemented in several 
FORTRAN subroutines in ZPC [3]. In this work, the 
simulation of two-parton collision is realized through 
invoking of these subroutines within the context of our CUDA 
C code named cudarun.cu. These subroutines were not 
rewritten in CUDA C because we found that they are not 
suitable to be parallelized. Executing these subroutines moves 
the involving two partons to the new location where collision 
happens at the collision time. In addition, through calling 
those subroutines, PCG updates their momenta and energies, 
and thus their speeds. Therefore PCG actually simulates 
two-parton collision on CPU, exactly as ZPC does.

C. Collision Detection for the Next Two-Parton Collision
After simulation of the just happened two-parton collision, 
the space-time as well as momentum-energy information for 
the two partons have been renewed. Because it is based on 
calculations using the space-time and momentum-energy 
information that PCG determines if a pair of partons will 
collide or not, and when to collide if the pair collides, it is 
necessary that before detecting the next earliest two-parton 
collision, the collision time for each of the just collided 
partons with other partons need to be updated if they will 
collide. The counting of the number of colliding pairs as well 
as the calculation of the collision time for each of those pairs 
that contain one of the just collided two partons are realized 
through invoking a CUDA kernel named ijUpdatTime(….i,…), where j stands for the index of one of 
those two just collided two partons. Hence for the detection 
of the next earliest two-parton collision, this kernel is invoked 
two times, once for each of the just collided two partons. It 
can be inferred that the update by invoking CUDA kernel 
ijUpdatTime(….i,….) concerns with the parallelized 
calculation of collision time for 2(N-2) pairs. It is 
worth-mentioning that all of the collision time for the rest 
parton pairs are left unchanged in the global memory at this 
stage and can be accessed for the detection of the next earliest 
two-parton collision. Given all the collision time and indices 
for all the colliding pairs of partons, the rest task for detecting 
the next earliest two-parton collision is partly done through a 
reduction procedure [23] that has been used for the detection 
of the first two-parton collision illustrated in sub-section A. 
For this job we implemented a CUDA kernel named 
updateTime(), which also updates the number of colliding 
pairs by ignoring those pairs that will not collide according to 
the result from ijUpdatTime().

IV. RESULTS
A. Correctness Check and Speedups for One Event
PCG is developed to take advantages of the high 
performance that a multi-core GPU provides for general 
purpose computing. It has to be correct in order to be 
integrated into AMPT or similar transport code to accelerate 
Monte Carlo event generation for ultrarelativistic heavy-ion 
collisions. We validate the correctness of PCG by checking 
the collision history from PCG with that from ZPC during 
the simulation of the parton cascade process in an N-parton 
system. The information contained in the collision history 
includes space-time and momentum-energy information for 
both of the two partons that are involved in a two-parton 
collision, for all the two-parton collisions in the cascade
process, not only right before the two-parton collision but also right after the two-parton collision. We have examined the collision history for various types of heavy-ion collisions of \( \text{Pb} - \text{Pb} \) at \( \sqrt{s_{\text{NN}}} = 2.76 \) TeV, which is the top LHC (Large Hadron Collider) energy. The result of the examination, together with the computing time for the simulation of the parton cascade process is tabulated in Table I, where \( t_{\text{ZPC}} \) and \( t_{\text{PCG}} \) stand for the computing time used for simulating the parton cascade by ZPC and PCG, respectively. As a result of the examination by using UNIX command diff, we have found that ZPC and PCG record the same two-parton collision history for each of the heavy-ion collisions with the impact parameter \( b \) ranging from 0 to 10 fm. It means PCG passed the correctness check. Therefore PCG can correctly simulate parton cascade as ZPC can, however with the exploitation of the powerful parallel processing capability GPUs have to offer for general purpose computing.

<table>
<thead>
<tr>
<th>( b (\text{fm}) )</th>
<th>check</th>
<th>( N )</th>
<th>( t_{\text{ZPC}} (\text{s}) )</th>
<th>( t_{\text{PCG}} (\text{s}) )</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>passed</td>
<td>43725</td>
<td>419.05</td>
<td>133.80</td>
<td>3.1x</td>
</tr>
<tr>
<td>1</td>
<td>passed</td>
<td>45145</td>
<td>494.31</td>
<td>159.38</td>
<td>3.1x</td>
</tr>
<tr>
<td>2</td>
<td>passed</td>
<td>40858</td>
<td>331.08</td>
<td>107.76</td>
<td>3.1x</td>
</tr>
<tr>
<td>3</td>
<td>passed</td>
<td>37230</td>
<td>259.24</td>
<td>85.37</td>
<td>3.0x</td>
</tr>
<tr>
<td>4</td>
<td>passed</td>
<td>30939</td>
<td>175.47</td>
<td>57.86</td>
<td>3.0x</td>
</tr>
<tr>
<td>5</td>
<td>passed</td>
<td>28808</td>
<td>144.41</td>
<td>48.79</td>
<td>3.0x</td>
</tr>
<tr>
<td>6</td>
<td>passed</td>
<td>23234</td>
<td>75.20</td>
<td>28.08</td>
<td>2.7x</td>
</tr>
<tr>
<td>7</td>
<td>passed</td>
<td>21138</td>
<td>74.26</td>
<td>29.22</td>
<td>2.5x</td>
</tr>
<tr>
<td>8</td>
<td>passed</td>
<td>17222</td>
<td>46.42</td>
<td>19.79</td>
<td>2.3x</td>
</tr>
<tr>
<td>9</td>
<td>passed</td>
<td>13781</td>
<td>28.16</td>
<td>14.08</td>
<td>2.0x</td>
</tr>
<tr>
<td>10</td>
<td>passed</td>
<td>10159</td>
<td>12.88</td>
<td>7.77</td>
<td>1.7x</td>
</tr>
</tbody>
</table>

From Table I, one may also see that PCG has a higher computing efficiency than ZPC for simulating the parton cascade in a heavy-ion collision of \( \text{Pb} - \text{Pb} \) at \( \sqrt{s_{\text{NN}}} = 2.76 \) TeV. This is an indication that parallelized collision detection exploiting the GPU’s multi-core parallel processing capability in simulating the parton cascade are helpful for saving computing resources. Certainly the speedup, which is the ratio of \( t_{\text{ZPC}} \) over \( t_{\text{PCG}} \) in Table I, depends on what hardware and software one uses. The GPU we used is NVIDIA C1060 card [24]. In APPENDIX B we listed in more detail the hardware and software that were used for obtaining the results we presented this paper.

One may note is that the 2 to 3x speedups shown in the Table I may be improved if ignoring the time used for writing onto disk the data, which defines the parton collision history. To make the point convincible we tabulated in Table II the time ZPC, together with PCG, used for simulating the parton cascade without the overhead of writing the onto disk the data about parton collision history. What we can tell comparing Table I with Table II is that better speedups can be expected for simulating the parton cascade in an \( n \)-parton system formed in a \( \text{Pb-Pb} \) collision during data production when it may not be necessary to record parton collision history.

### B. Speedups for statistically significant number of events

Though we see in both Table I and Table II that PCG can gain 2 to 3-fold speedup relative to ZPC, it has to be noted that the speedup is just out of analysis of simulating the parton cascade in one event. In order to come to a solid conclusion as for what a speedup PCG can gain, we use both ZPC and PCG to simulate the parton cascade in AMPT-based generation of hundreds of events for \( \text{Pb} - \text{Pb} \) collisions at \( \sqrt{s_{\text{NN}}} = 2.76 \) TeV, respectively. The event-averaged computing time for the simulation of the parton cascade in one event was tabulated in Table III, together with the average speedup that we gained by using PCG. The errors are statistical, and are calculated as the standard deviation. From Table III, one may come to the same conclusion as from Table II, i.e. relative to ZPC we may gain 2 to 3 fold speedup by using PCG in the simulation of the parton cascade process varying with the impact parameter \( b \). Because for the time being PCG simulates two-parton collision on CPU the same way as ZPC does, the speedup of PCG relative to ZPC may be attributed to the parallelized simulation of the collision detection on the GPU.

<table>
<thead>
<tr>
<th>( b (\text{fm}) )</th>
<th>( N )</th>
<th>( t_{\text{ZPC}} (\text{s}) )</th>
<th>( t_{\text{PCG}} (\text{s}) )</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>43725</td>
<td>410.04</td>
<td>120.45</td>
<td>3.4x</td>
</tr>
<tr>
<td>1</td>
<td>45145</td>
<td>465.30</td>
<td>144.65</td>
<td>3.2x</td>
</tr>
<tr>
<td>2</td>
<td>40858</td>
<td>321.26</td>
<td>95.40</td>
<td>3.4x</td>
</tr>
<tr>
<td>3</td>
<td>37230</td>
<td>251.25</td>
<td>75.56</td>
<td>3.3x</td>
</tr>
<tr>
<td>4</td>
<td>30939</td>
<td>160.41</td>
<td>50.54</td>
<td>3.2x</td>
</tr>
<tr>
<td>5</td>
<td>28908</td>
<td>138.43</td>
<td>41.89</td>
<td>3.3x</td>
</tr>
<tr>
<td>6</td>
<td>22324</td>
<td>71.34</td>
<td>23.81</td>
<td>3.0x</td>
</tr>
<tr>
<td>7</td>
<td>21138</td>
<td>70.40</td>
<td>24.62</td>
<td>2.9x</td>
</tr>
<tr>
<td>8</td>
<td>17222</td>
<td>43.45</td>
<td>16.83</td>
<td>2.6x</td>
</tr>
<tr>
<td>9</td>
<td>13781</td>
<td>25.97</td>
<td>11.60</td>
<td>2.2x</td>
</tr>
<tr>
<td>10</td>
<td>10159</td>
<td>11.65</td>
<td>6.33</td>
<td>1.8x</td>
</tr>
</tbody>
</table>

### V. SUMMARY AND CONCLUSION

On the basis of ZPC, we have introduced a GPU-based algorithm for accelerating the simulation of the parton cascade process in the early stage of ultrarelativistic heavy-ion collisions. Implementing the algorithm, we have developed a code called PCG (Parton Cascade on GPU) by using CUDA C. Running both ZPC and PCG, we simulated parton cascade in AMPT-generated heavy-ion collision events of \( \text{Pb} - \text{Pb} \) at \( \sqrt{s_{\text{NN}}} = 2.76 \) TeV with impact parameter \( b \) ranging from 0 to 10 fm. We compared results from running ZPC on CPU with those from running PCG. The comparison demonstrates that PCG can give the same parton collision history as ZPC does thus the correctness of PCG is proved. Additionally, the comparison shows that by using PCG speedups in the range of 2x to 3x can be obtained relative...
to ZPC in our computing environment. Therefore PCG is ready for being integrated into AMPT and can help save huge amount of computing resources in the case of AMPT-based large scale Monte Carlo event generation for ultrarelativistic heavy-ion collisions at \( \sqrt{s_{NN}}=2.76 \) TeV.

APPENDIX A

Cudarun \((x, y, z, w, px, py, pz, pw, \ldots, N)\) may also be invoked in AMPT thus space-time and momentum energy information for \( N \) partons are ready for parton cascade. The information are contained in the following data structures: \( double x[N], y[N], z[N], w[N], px[N], py[N], pz[N], pw[N]\).

The sketch for algorithm of PCG:
cudarun \((x, y, z, w, px, py, pz, pw, \ldots, N)\) {
  \text{step1: after memcpyp(...)}, on device datTime(...) calculates for each parton the number of collisions, and for each collision the collision time and indices of the two colliding partons; uses a shared-memory-based reduction procedure to get and then save in global memory the smallest collision time in each block of threads, together with the indices of the two colliding partons;

  \text{step2: after using memcyp(...), on host calculate t\_collision\_time for the first two-parton collision and the indices of the two colliding partons \( i \) and \( j \);}

  \text{step3: while (t\_collision\_time < t\_preset\_threshold) {}

  \text{step3\_1: on host, simulate two\_parton collision between parton \( i \) and \( j \) and update \( x[i], y[i], z[i], w[i], \ldots, N \), \( px[i], py[i], pz[i], pw[i], \ldots, N \).}

  \text{step3\_2: after memcyp(...), on device ijUpdatTime(..., \( i, j \)) updates collision time and indices for all of the colliding pairs that include \( i \);}

  \text{step3\_3: after memcyp(...), on device ijUpdatTime(..., \( i, j \)) updates collision time and indices for all of the colliding pairs that include \( j \);}

  \text{step3\_4: on device updatTime(...) updates for each parton the number of collisions, and for each collision the collision time and indices of the two colliding partons, one of which is either \( i \) or \( j \); uses a shared-memory-based reduction procedure to get and then save in global memory the smallest collision time in each block of threads, together with the indices of the two colliding partons;}

  \text{step3\_5: after memcyp(...), on host update t\_collision\_time for the next earliest two-parton collision and the indices of the two colliding partons \( i \) and \( j \);}

  \}}

APPENDIX B

The version of AMPT, which generates the parton system for both PCG and ZPC to simulate parton cascade, is v1.26t4–v2.26t4. The compiler [25] we used to compile code written in FORTRAN is GNU FORTRAN (GCC) 4.1.2 20080704 (Red Hat 4.1.2-44). The compiler options for gfortran are -fdefault-real-8 -O2. For compiling our CUDA C code cudarun.cu, which contains three kernels datTime(), ijUpdatTime() and updatTime(), we in command line issued nvcc --arch=sm_13 -O2 -c cudarun.cu. The computing environment for this work mainly consists of one core of CPU that is Intel Xeon E5410 @ 2.33GHz, and one GPU that is NVIDIA C1060 Tesla T10 [24], in addition to CUDA3.2 [13] and Red Hat 4.1.2-44 Linux 2.6.18-128.el5 for x86_64. The CUDA C codes as well as the FORTRAN subroutines PCG uses are available per request.

ACKNOWLEDGMENT

Qing-Jun Liu would like to thank Professors Zi-Wei Lin and Bin Zhang for helpful discussions and allowing us to use AMPT and ZPC code, on which this work is based. Computing support is provided by the Supercomputing Center, Computer Network Information Center, Chinese Academy of Sciences.

REFERENCES


Chunbao Zhou received the Ph.D. degree in 2012 in the College of Computer Science and Technology in Ji-Lin University, China. Since 2012, he has been working in Supercomputing Center of CAS, Computer Network Information Center, Chinese Academy of Sciences, China.


Fang Liu received the BS degree in 2005 in Nanjing University, the Ph.D. degree in 2010 in the Institute of Software, Chinese Academy of Sciences, where she received The Excellence Award at the Presidential Academy Awards. Since 2010, she has been working in Supercomputing Center, Computer Network Information Center, Chinese Academy of Sciences, China. She was promoted to associate professor at the beginning of this year. Her major research field is high performance computing, general purpose computing on gpu. Her major publications include: 1) F. Liu, M. Huang, X. Liu, and E. Wu, “efficient depth peeling via bucket sort,” in Proc. ACM High Performance Graphics, 2009, New York: ACM Press, pp. 51-57. 2) F. Liu, M. Huang, X. Liu, and E. Wu, “Free2Pipe: A programmable, parallel rendering architecture for efficient multi-fragment effects,” in Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 2010, New York: ACM Press, pp. 75-82.