基于 C86 + DCU 加速模拟高能重离子碰撞早期的部分子级联

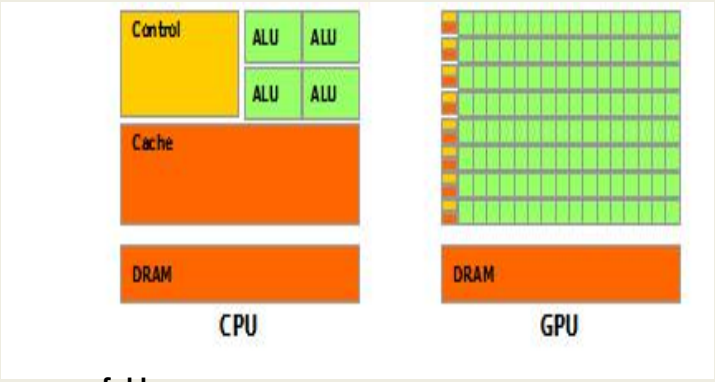
刘庆军
北京石油化工学院, 102617

中国科学院大学雁栖湖东区, 北京, 2023年3月3日

Accelerate Parton Cascade in Relativistic Heavy-ion Collisions on C86 + DCU

- Why to do we use the graphics processing unit (GPU)
- How to build the GPU-accelerated package PCGv2
- What about the recent results of running the package
- Conclusions and future work

•GPUs ---- powerful arithmetic engine, very attractive



It is so powerful because

(1) GPU can do the job in parallel, following single instruction multi-data (SIMD)

(2) It is designed to utilize more transistors for data processing

- **Speedup simulation of heavy ion collisions is attractive, too**

1. Huge amount of measured data by ALICE, STAR too.

2. Comparison with data demands even more of simulated data

3. The simulation is computing-intensive

4. Accelerating the simulation is challenging and thus full of fun!

Naturally, we want to try GPUs for the simulation

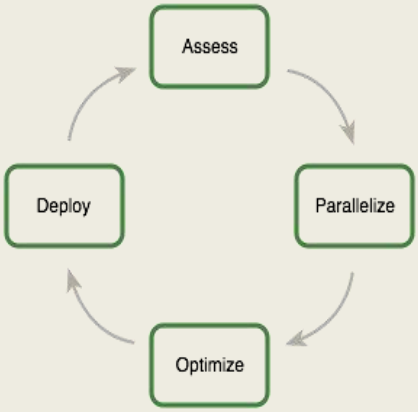
•

•Our aim:

GPU-accelerated event generation in heavy-ion collisions at the Large Hadron Collider(LHC)

•Methodology we apply:

APOD,
which NVIDIA advocates



- **Three ways to get your apps targeting GPUs and the like, DCU**

If from ground zero, follow the first way

1.Program, e.g., in CUDA C (ANSI C/C++ with extensions)

Other two way are, in your original code :

2. Call GPU-optimized libraries, such as cuFFT, cuBLAS
3. Add preprocessor directives, like those defined in Acc

which way(s) to use? It may be out of your assessment

How to build PCGv2 to utilize DCU

•**Assess**

(1) AMPT contains HIJING + ZPC + ART, serially executed

(2) ZPC is the most compute-intensive module among them

(3) In an unique way, ZPC does successive two-parton **collision detection** + collision simulation

(4) During each round of collision detection, **parton pairs can be processed separately, so it can be done in parallel**

The hot spot is successive collision detection

How to build PCGv2 to utilize DCU

- **Parallelize using CUDA C （NVIDIA） or HIP （AMD）**

(1) Use GPU-optimized libraries, or (2) preprocessor directives? **NO**

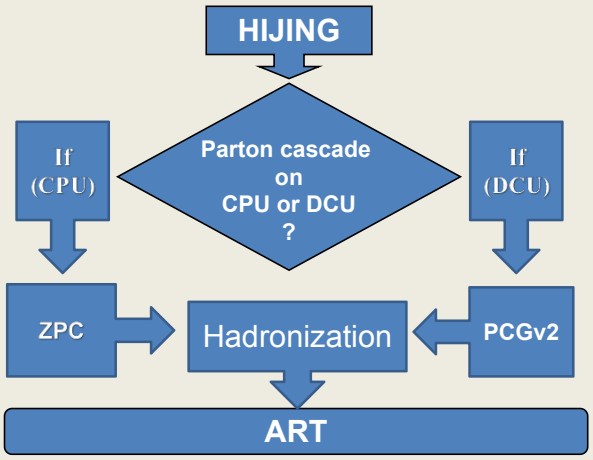
(3) Rewrite parton cascade, in CUDA C (Free)? **YES**

a. Update particle position and momentum info, or simulate two parton collision on GPU or by calling a few FORTRAN subroutines of ZPC

b. Detect two-particle collision, by calling newly built CUDA C code or HIP code
It includes several kernels to facilitate parallel collision detection

PCGv2 may be a mixed language package, in which DCU serves as co-processor, specifically CPU is for collision simulation and DCU for collision detection

How to build PCGv2 to utilize DCU



PCGv2 does what ZPC does: successive collision detection and collision simulation, but involves DCU

How to build PCGv2 to utilize DCU

PCGv2 Algorithm

1. __global__ void firstCollisionTime() // compute and save colliding time for all of the 0.5N(N-1) pairs, get the index of the first colliding pair
2. Subroutines, advancing to the time and let the two collide
3. __global__ UpdateCollisionTime() // update colliding time for the pairs, each of which involve a parton that just committed a collision
4. __global__ void nextCollisionTime() // decide the next collision
5. If time for next collision > thresholdTime, PCG terminates; Otherwise go to step 2 and repeat steps 3—5

PCGv2 includes several kernels written in CUDA C or HIP, and may use a few Fortran subroutines for simulating collisions

How to build PCG to utilize GPU

•**Optimize**

According to "**CUDA or ROCM/HIP programming guide**",

(1) Select block size using cuda occupancy calculator

(2) Use shared memory for reduction

(3) Data re-use by saving colliding time for all colliding pairs in global memory

How to build PCGv2 to utilize DCU

•**Deploy : Compile -> Run ->Check**

(1) hipcc –O2 –c pcgv2.cpp, for example

a. Parton cascade simulated by pcgv2.cpp

b. Compile the code with hipcc , provided free by AMD

(2) Using gfortran to link pcgv2.o with other object files of AMPT: art.o, hijing.o, hipyset.o, main.o, zpc.o …

(3) Run executable to get timing information and physics-related results (e.g., parton collision history)

(4) Check correctness by examining the collision history and/or finale statevariable, look at performance of PCGv2

• **亦可用AMD ROCm 提供的转码工具得到基于C86 + DCU 的PCG**

1. Check for Correctness

Action item: comparing parton collision history with that from ZPC

- **Before each two-parton collision, (r_μ, p_μ) should be the same**
- **After each two-parton collision, (r_μ, p_μ) should be the same**

Findings: we got almost the same parton collision history in the two cases, for collisions of Pb - Pb at $\sqrt{s_{NN}} = 5.02$ TeV in central to ward mid-central collisions.

Notes: collision time ordered in Lab and used the one from ZPC or PCGv2

Therefore, with DCU as a co-processor the parton cascade runs ok

1. Check for Correctness

Action item: comparing rapidity distributions with those from ZPC

Findings: They are consistent nicely for central collisions of Pb - Pb at $\sqrt{s_{NN}} = 5.02$ TeV in central toward mid-central collisions.

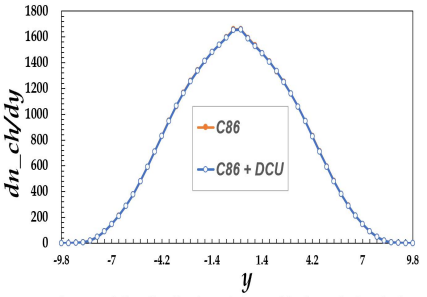


Fig. 3 Rapidity distribution of charged hadrons before hadron cascade begins for b = 2 fm.

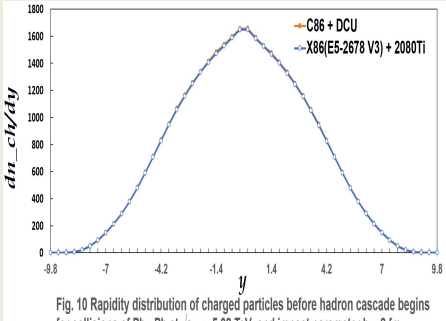


Fig. 10 Rapidity distribution of charged particles before hadron cascade begins for collisions of Pb - Pb at $\sqrt{s_{NN}}$ = 5.02 TeV and impact parameter b = 2 fm.

Therefore, with DCU as a co-processor PCGv2 runs ok

2. Check for speed-up

Definition: Speed-up (加速比) = time_ZPC/time_PCG

Collision centrality: central toward mid-central collisions

Collision type: Pb + Pb at $\sqrt{s_{NN}}$ = 2.76 TeV , PCGv1

Number of events	time_ZPC(s) E5-2680V2 @ 2.80GHz	time_PCG(s) Tesla K20m	Speed-up for 2.76 TeV with 0<b<3fm
402	92977	23104	4x

Collision type: Pb + Pb at $\sqrt{s_{NN}}$ = 5.02 TeV , PCGv2

碰撞参数 b（fm）		2	4	6	8
模拟每事例所需平均时间（秒）	C86	2273±24	1256±13	517.4±0.48	158.5±0.1
	C86+DCU	163±1	99.8±0.2	57.32±0.06	27.33±0.02
模拟每事例平均加速比		13.9±0.2	12.6±0.2	9.03±0.02	5.80±0.01
模拟的事例数目（个）		960	900	2850	1280

- **Conclusions**

We've updated a parton cascade code PCGv1 and arrived at PCGv2. With a GPU-like accelerator called DCU as a co-processor, parton cascade as simulated by PCGv2, complete much more rapidly than as simulated by ZPC. PCGv2 mainly differs from ZPC in algorithm for collision detection, and speedup reaches 14 for central collisions of Pb - Pb at $\sqrt{s_{NN}}$ = 5.02 TeV. Rapidity distributions of final state charged particles agree in the two cases where ZPC and PCGv2 are used for simulating parton cascade, respectively. PCGv2 helps accelerate AMPT-based event generation once it is integrated into AMPT.

- **Future work**

Aiming at higher performance, and thorough test on A100.

---------- 谢谢! ----------

刘庆军,中国科学院大学雁栖湖东区,北京 2023年3月3日

Acknowledgement

This work was partly supported by Beijing Natural Science Foundation grant no. 1132017, Beijing Municipal Education Commission's major scientific project grant no. KZ201910017019, the GHfund B(20210702) and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry. We thank the Computer Network Information Center and the Institute of Process Engineering, the Chinese Academy of Sciences, for providing computing support.


```
cpu family      : 6              model          : 62
model name  : Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz
cpu MHz         : 1200.000   cache size      : 25600 KB
MemTotal:       66006708 kB
For device #1 named Tesla K20m
clock rate is 705500
Device major and minor is 3.5
Total Global mem:5032706048
Total const mem:65536
#of Multiprocessor: 13
Shared mem per block: 49152     Registers per block: 65536
MaxThreadsPerBlock: 1024        concurrent kernels:1
MaxThreadsDim: ( 1024, 1024, 64)
MaxGridDim: ( 2147483647, 65535, 65535)
go ahead with streams here!
the device supports executing multiple kernels within the same
context simultaneously!
```

Why graphics processing unit (GPU)