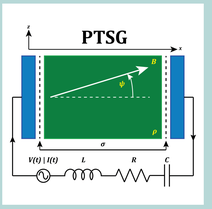
# Increasing Efficiency of Monte Carlo Particle-Fluid Collision Calculations on GPU

Charles Bardel, John Verboncoeur
Electrical and Computer Engineering, Michigan State University




## Abstract

Monte Carlo particle collision calculations can be very computationally expensive for particle-in-cell codes. In the case of background fluid collisional calculations, where each particle calculation is totally independent of other collisions, the calculations can be setup as highly parallel. Porting to GPU platforms has shown two orders of magnitude decrease compared to single processor performance.

One approach is to simply apply a function to every particle which involves computing the particle energy, a square root to obtain the speed, and either interpolation of tabled cross sections or computation of a curve fit for each process for every particle [1]. Then, based on this probability of collision the collisional dynamics code might be executed. For collisional probabilities « 1 this is inefficient for finding particles to collide and load imbalanced for the collisional dynamics on the vector architecture (Single-Instruction Multiple-Data SIMD) like capabilities available on the GPU[2].

The alternative approach is to use the null collision method[1] where particles selected for collision are selected at random using the total collision probability, which is independent of particle energy and position. However, this sparse random access of particles in the particle array, as needed for the null collision method[1], has drawbacks on the GPU due to SIMD architecture. GPU threads that are grouped in hardware are called warps. Each warp can only issue one computational or memory instruction. However, when two memory instructions are located with 128 bytes[2] of each other they can be 'coalesced' into one instruction.

Using the data structure and algorithm presented in [3] for efficient particle to grid charge accumulation on the GPU, which ensures that all particles contained within a cell are contiguous in memory, this poster examines the effect of selecting particles for colliding that are contiguous in that same list. This setup would capitalize on the null collision method's not needing to calculate the energy of each particle and optimize the memory bandwidth through the GPU.

## Overall Scheme

This work represents an effort to maintain an efficient data structure for two components of the PIC scheme

**Cell charge accumulation** Memory contention for cell edge charge

**Null Collision** Random access pattern causing high memory latency

A method to increase effective memory bandwidth to these algorithm components is to ensure that particles used are contiguous in memory

## Charge Accumulation

Charge accumulation on mesh nodes from particles is one of the most memory-bandwidth limited procedures of PIC.

The procedure described in the paper [3] lays out a scheme to keep the the read/write operations low by keeping a running sum of charge from each cell called particle pull. Here $k$ is the number of mesh vertices and $k \ll N$ and $d$ is dimensions of the domain.

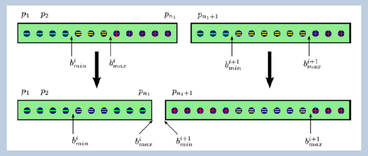
**Particle Push** Particles 'push' on to the grid location requiring $\mathrm{O}\big((2^d+1)N\big)$ read/writes.

**Particle Pull** Particles are 'pulled' by the grid location requiring $\mathrm{O}\big(2^d N + k\big)$ read/writes.

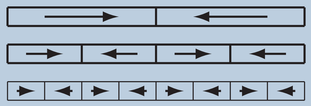
This data structure provides particles in a cell to be contiguous in memory allowing for a running sum for each mesh node with reduced write conflict

## Bucket-Sort Defragmentation

- 'Particle Pull' is memory bandwidth efficient since it requires fewer writes and exclusive access to mesh nodes
- Since this optimization only requires cell particles to be contiguous a form of 'bucket' sort is used
- In the following figure, each cell is sorted into 3 groups of particles: going left(-), staying (=), going right (+).
- 2N reads and 4p writes are needed to maintain this data structure, where $p$ is particles leaving the cell




This method can be adapted to multi-cell crossing scenarios such as particle insert, 2D and 3D space-filling curves. The following diagram shows a hierarchy of bucket sorts that can handle multiple cell crossings following an arbitrary space-filling curve.



## Null Collision Method

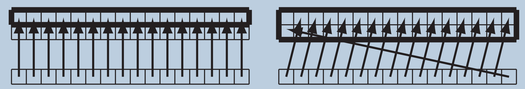
The Null Collision method significantly reduces the memory-bandwidth requirements at each step since only a fraction of the particles are selected to have collisions *independent* of the actual particle energy. This is accomplished by observing the max collisional frequency:

$$v_{max} = \max_x(n_g(\mathbf{x}))\max_{\mathscr{E}}(\sigma_T(\mathscr{E})v)$$

where $n_g(\mathbf{x})$ is a spacial varying target density and $\sigma_T(\mathscr{E})$ is the total cross section, $v$ is speed.

## Coalesced Memory Access

Memory-bandwidth efficiency is determined by how well the random memory requests are aligned to memory locations. First shown is memory being requested with 100% utilization, in one memory transaction. The second shows a memory access misalignment causing 50% utilization of memory requested, requiring two memory transactions. Since the target access pattern is random, cache access will not help, so the most efficient access pattern is required.



## Simulation

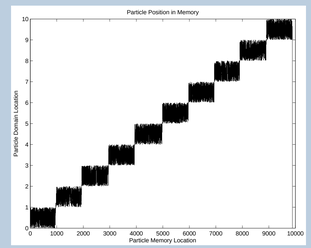
To explore the usage of this technique the following experimental simulation was performed: three drift tubes are setup where 1,600, 16,000, and 160,000 electrons are bounded in the $X$ dimension by elastic boundary conditions and an electric field is applied in the $Z$ dimension. An electrostatic periodic field solve is applied for the simulation but it has no significant contribution compared to the applied field. Ionization and particle insertion occurs when the particles have a collision at a high enough energy.

## References

[1] J. P. Verboncoeur, PLASMA PHYSICS AND CONTROLLED FUSION **47**, A231 (2005).
[2] Nvidia, Cuda c best practices guide, 2012.
[3] N. G. George Stantchev, William Dorland, Parallel Distrib. Comput. **68**, 1339 (2008).

## Particles in Memory

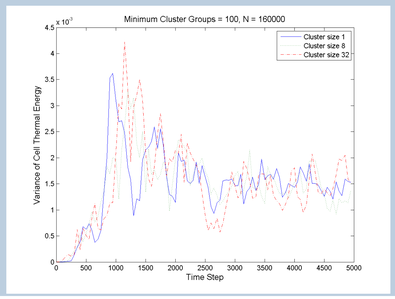
The layout of particles in memory is key to achieve high memory throughput on either CPU or GPU. The algorithm described achieves this balance with the following memory layout

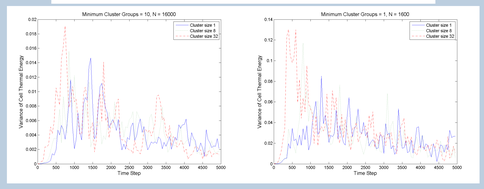



As shown this method keeps particles belonging to a cell contiguous and randomly represented in a cell.

## Simulation Effects

- Does not exhibit non-uniform particle growth in simulation
- Variance in cell mean can be large unless number of cells are less than the number of cluster groups




## Conclusion

This method achieves the stated goal of both keeping particles within a cell contiguous and uniformly representing the cell. Presented are 3 scenarios with different number of cluster groups. The scenario that has more cluster groups than cells shows agreement with non-grouped method.