LA-UR- 11-00471

Approved for public release; distribution is unlimited.

Title:	A 3D Front Tracking Method on a CPU/GPU System
Author(s):	Wurigen Bo, John Grove
Intended for:	CCS-2 Staff Activity Seminar, Jan 27, 2011

Los Alamos
NATIONAL LABORATORY
EST. 1943

Form 836 (7/06)

A 3D Front Tracking Method on a CPU/GPU System

Presented by: Wurigen Bo, CCS-2

ABSTRACT: We describe the method to port a sequential 3D interface tracking code to a GPU with CUDA. The interface is represented as a triangular mesh. Interface geometry properties and point propagation are performed on a GPU. Interface mesh adaptation is performed on a CPU. The convergence of the method is assessed from the test problems with given velocity fields. Performance results show overall speedups from 11 to 14 for the test problems under mesh refinement. We also briefly describe our ongoing work to couple the interface tracking method with a hydro solver.

A 3D Front Tracking Method on a CPU/GPU

Wurigen Bo

In collaboration with
John Grove

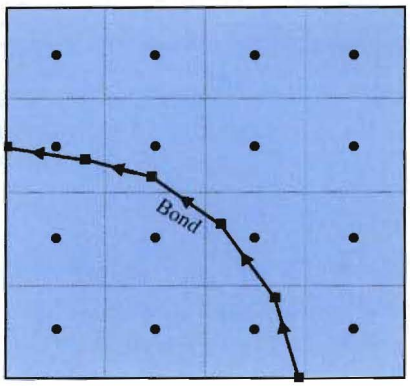
CCS-2
Los Alamos National Lab

Outline

- Introduction
  - Front tracking method
  - CUDA
- Algorithms on CPU/GPU
  - Code structure on CPU/GPU
  - Curvature calculation, point propagation on GPU
  - Redistribution on CPU
- Summary
- Ongoing Work: Coupling with hydro code

Front Tracking Method

- Front tracking method is implemented in code *FronTier.*



- Grid State
- Tracked Points (left,right) States

Courtesy of J.Grove

Major components:

- A moving mesh to represent interface
- Compressible Navier-Stokes equations

Procedure to solve:

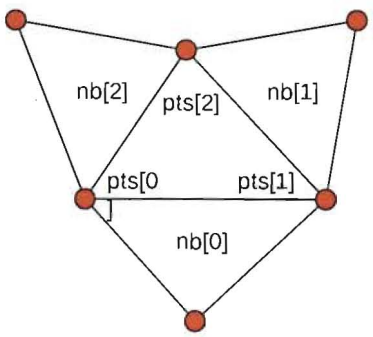
- Compute geometry properties
- Propagate interface points
- Redistribute the interface points
- Resolve the topological change of the interface
- Solve equations to obtain the fluid states at cell centers

Data Structure of a 3D Front

Both points and triangles are stored in linked lists

- The order in the list has no connection to the actual order on the front.
- The addition and removal of points and triangles is simple.
- All triangles in a given front must be oriented in the same way.

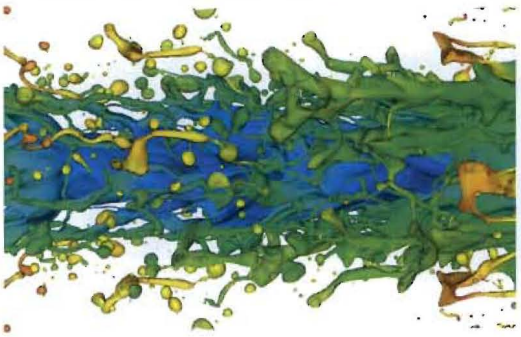
```
struct {
  float  coords[3];
  float  nor[3];
  float  curvature;
} POINT;


struct {
  struct POINT  *pts[3];
  struct TRI    *nb[3];
} TRI;
```

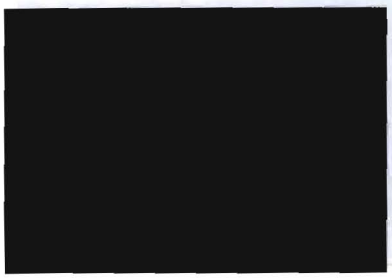


Front Tracking Method: Applications

A 3D simulation of jet breakup

- 4096 bluegene cores
- Total running time ~12 days
- Number of triangles ~12 million





Basic Issues of Front Tracking on GPU

Why GPU:

- Computing geometry properties, propagating points and solving interior states constitute most running time of the code. Moving these operations to GPU will greatly improve the code efficiency.

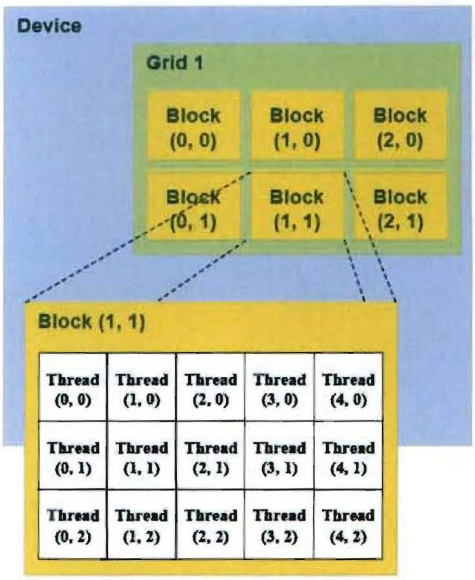
Challenge

- Redesign existing algorithms so that they are suitable on GPU.
- Data dependent memory access in front tracking method has low efficiency on the current GPU.
- Accuracy of single precision floating point calculations on the current GPU.
- Some operations of the front tracking method remain on CPU due to their complexity. We need to study the impact of these operations on the overall code performance.

CUDA Programming Model

CUDA (Compute Unified Device Architecture) is an extension to both C and Fortran on NVIDIA GPUs.

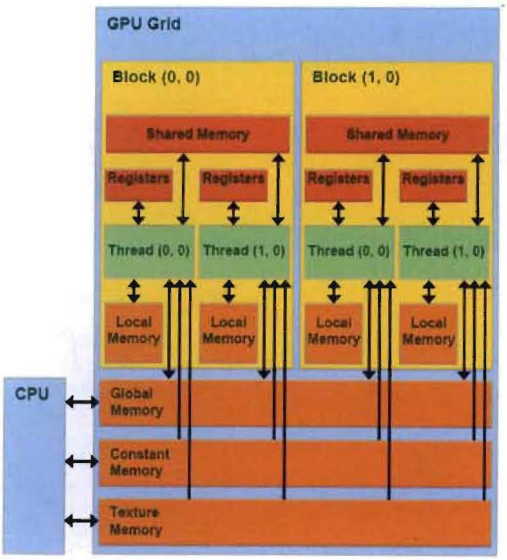
- A kernel is a function that runs on GPU
- A CUDA kernel is executed by an array of threads at a time
- A block is a batch of threads that can cooperate with each other by
  - Sharing data through shared memory
  - Synchronizing their execution
- A grid is a 1D or 2D array of blocks
- Threads from different blocks can not cooperate



CUDA Memory Model

Three main memory types on NVIDA GPUs

- Register memory: accessible to an individual thread only, high bandwidth and low latency.
- Shared memory: accessible to every thread in a single block, high bandwidth and low latency.
- Global memory: accessible to every thread, high latency.
  - Global memory coalescing: The memory throughput can be increased by an order of magnitude over random memory access.



Front Tracking Code Structure on CPU/GPU

- Geometry properties calculation and point propagation are ported from CPU to GPU due to their high computational cost.
- Data transfer happens only when the front connectivity is modified by redistribution.

while(not the final time)	
Transfer point and triangle array from CPU to GPU Determine the number of blocks for kernel launch	CPU

//suppose we redistribute every nredis steps
for i:=1 to nredis step 1 do
Compute normal vectors and curvatures
Propagate points
Determine dt
end
	GPU

Transfer points coordinates from GPU to CPU Redistribute the front Reorder point and triangle array	CPU
end	

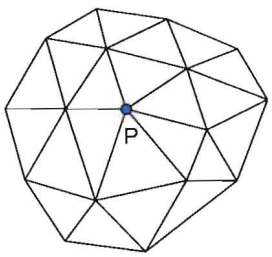
Normal and Curvature Calculation

- Algorithm: Least square fitting for normals and curvatures* at a point P.
  - Construct a local coordinate system at P.
  - Transform the coordinates of the two ring adjacent points of P into the local coordinate system.
  - Fit a local second order polynomial to the local point positions by least square approximations.

$$z = f(x, y) = Ax^2 + Bxy + Cy^2 + Dx + Ey$$

$\min \sum_{i=1}^{N} (z_i - f(x_i, y_i))^2$ Least square fitting for the polynomial coefficients

(x_i, y_i, z_i) The local coordinates of the two ring adjacent points



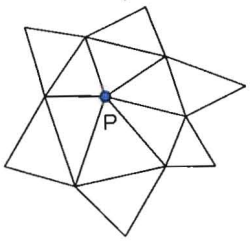
Two ring adjacent points of P

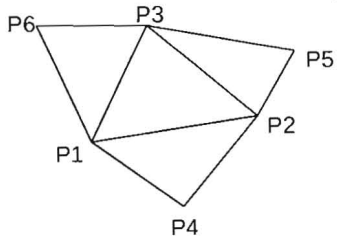
* X. Jiao and H. Zha, Consistent computation of first- and second-order differential quantities for surface meshes, *Proceedings of the ACM Solid and Physical Modeling Symposium*, 2008

Normal and Curvature Calculation on GPU

- Three GPU kernels to compute the normals and curvatures at a point.

One and a half ring adjacent points of P





Kernel 1. Collect the adjacent points of P: one triangle per thread

Add P2 P5 into the one and a half ring of P1
Add P3 P5 into the one and a half ring of P2
Add P1 P4 into the one and a half ring of P3

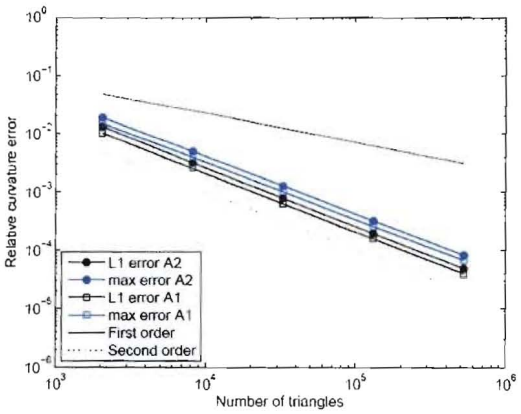
Kernel 2. Compute the local coordinate systems: one point per thread

Kernel 3. Compute normals and curvatures: one point per thread

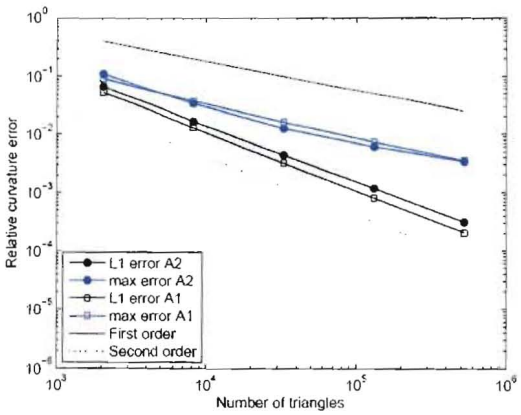
construct and solve normal equations in each point
compute the first and second derivatives of the second order polynomial

Accuracy of Curvature

- Relative curvature error with double precision on GPU.



Static sphere

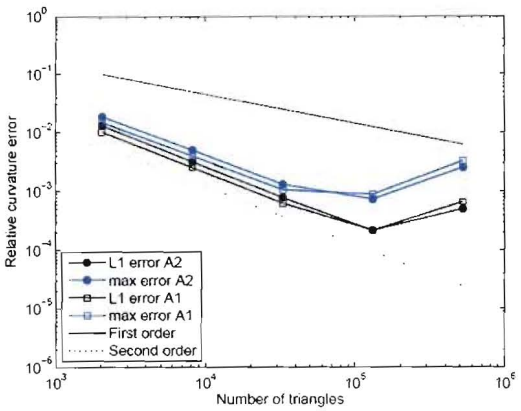


Static torus

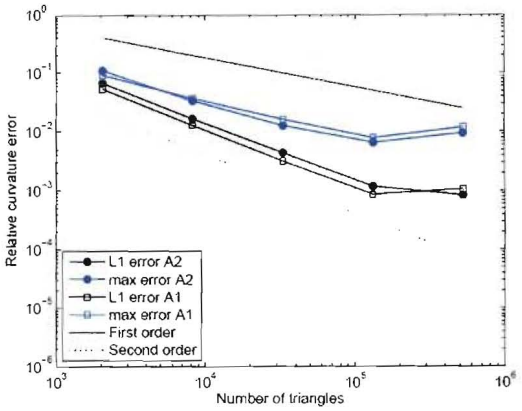
A1: curvature obtained from two ring adjacent points
A2: curvature obtained from one and a half ring adjacent points

Accuracy of Curvature

- Relative curvature error with single precision on GPU.



Static sphere



Static torus

Interface point propagation

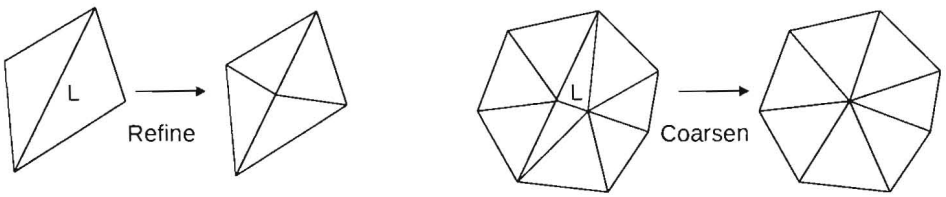
d**x**/dt = **v**(**x**,t)

x: A point on the front
v(**x**,t): point velocity

- In the current test cases, **v**(**x**,t) is a given.
  - It may depends on the normal vectors and/or curvature on **x**.
  - When coupling with hydro code, **v**(**x**,t) can be interpolated from fluid velocity or solved from a Riemann problem on the front.
- Solve the ODE with a fourth order Runge Kutta methods.
- It is a pointwise operation: There is no further need to consider the local connectivity near the points once the front normals and curvatures are obtained. Global memory access is coalesced.

Redistribution

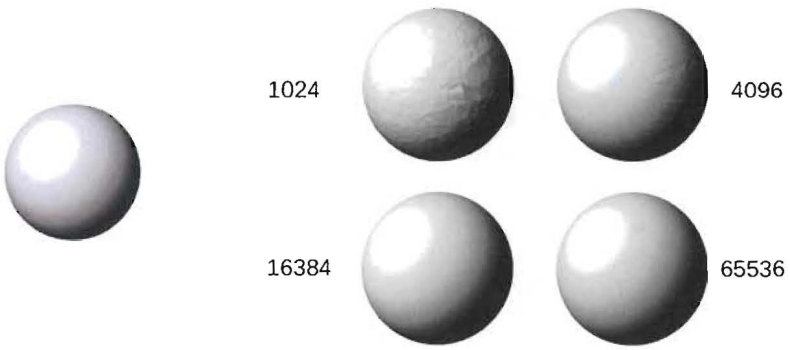
- Redistribution is needed because of the surface deformations as the interface evolves leads to bad geometric properties of the triangles.
- Redistribution is performed on CPU due to its complexity.
  - Complexity: conditional statements, uncoalesced memory access.
  - Redistribution on GPU is a topic under investigation.
- Procedure of redistribution
  Coarsen: delete an edge if L<dx/3 or its corresponding angle<15 degree.
  Refine: divide an edge if L>dx.
  Reorder: sort the triangle and point list.



- Topological validation check is performed before deleting an edge.
- The new points in coarsen and refine are computed with LS fitting.
- Reorder is aimed to relax cache miss on CPU.

Verification: 3D Deformation

- Convergence under mesh refinement on FX 1800.



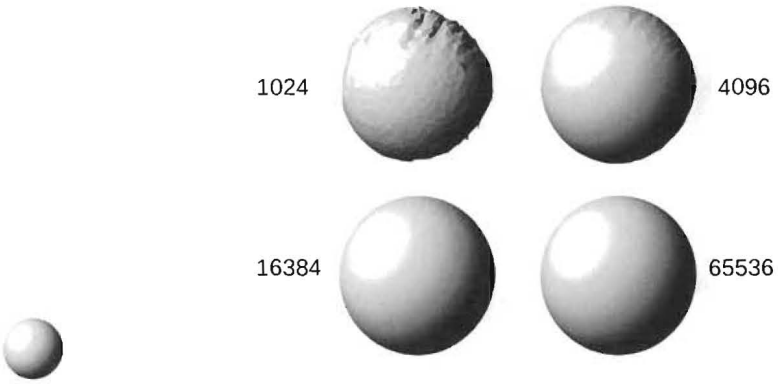
Interface at the final time

- Relative errors at the final time

# of Triangles	L_∞ error	Order	L_1 error	Order	Volume error	Order
1024	6.361×10^{-2}	1.53	8.938×10^{-3}	2.56	2.985×10^{-2}	2.49
4096	2.188×10^{-2}	1.58	1.509×10^{-3}	2.77	5.288×10^{-3}	2.99
16384	7.312×10^{-3}	1.38	2.197×10^{-4}	2.63	6.637×10^{-4}	2.76
65536	2.790×10^{-3}		3.545×10^{-5}		9.732×10^{-5}	

Verification: 3D Shear

- Convergence under mesh refinement on FX 1800.



Interface at the final time

- Relative errors at the final time

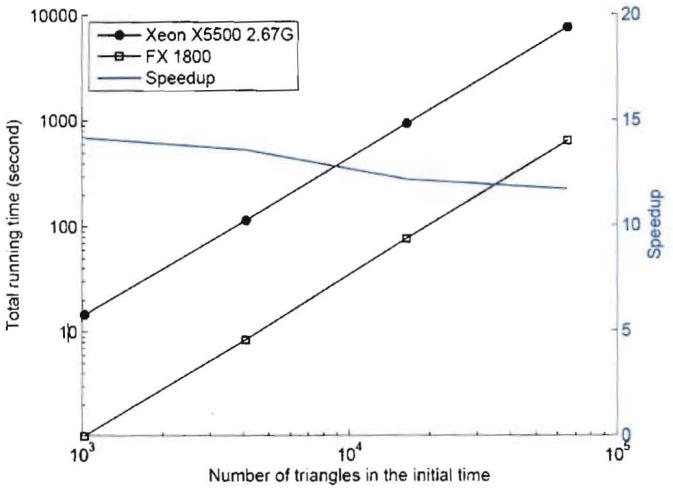
# of Triangles	L_∞ error	Order	L_1 error	Order	Volume error	Order
1024	1.721×10^{-1}	1.83	2.748×10^{-2}	2.21	6.402×10^{-2}	1.76
4096	4.816×10^{-2}	1.63	5.906×10^{-3}	2.60	1.884×10^{-2}	2.62
16384	1.548×10^{-2}	1.72	9.741×10^{-4}	2.86	3.065×10^{-3}	3.45
65536	4.684×10^{-3}		1.340×10^{-4}		2.801×10^{-4}	

Verification: 3D Curvature dependent velocity

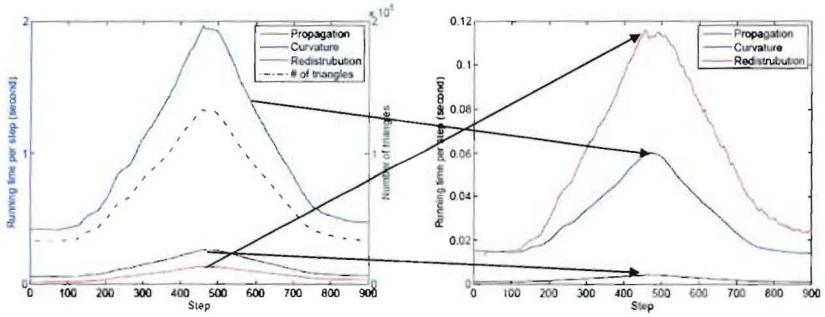
- Velocity field: **v**(**x**) = **n**(**x**)(0.1- 0.0001□(**x**))
  n(**x**): normal vectors at **x**
  □(**x**): curvature at **x**

Performance Comparison

- Performance is compared between CPU and GPU for 3D deformation test.



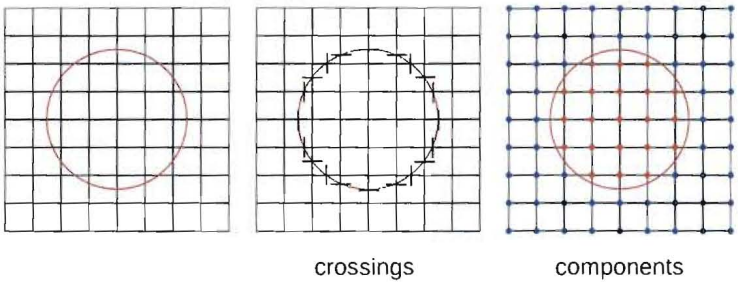
Performance Comparison

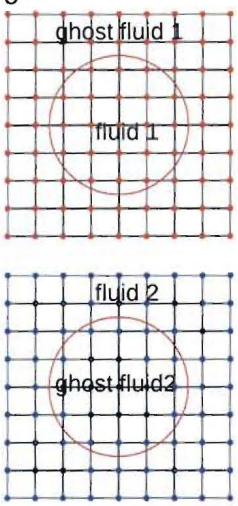


	number of triangles	Interface operations (propagation, curvature)	Redistribution	Total (s)
CPU	1024	13.87 (95%)	0.6259 (5%)	14.5
	4096	108.1 (95%)	5.141 (5%)	113.3
	16384	873.9 (95%)	49.6 (5%)	923.5
	65535	7092 (94%)	419.9 (6%)	7512
GPU	1024	0.4002 (38%)	0.6301 (62%)	1.030
	4096	3.268 (38%)	5.148 (62%)	8.417
	16384	27.12 (35%)	49.60 (65%)	76.73
	65536	219.0 (34%)	422.3 (66%)	641.3

Ongoing Work: Coupling

- Coupling with a hydro solver: Solve Euler equations in each side of the interface with a Ghost Fluid Method* on GPU.
  - Compute interface-grid crossings.
  - Determine the components of grid points from crossings.
  - Solve Euler equations.

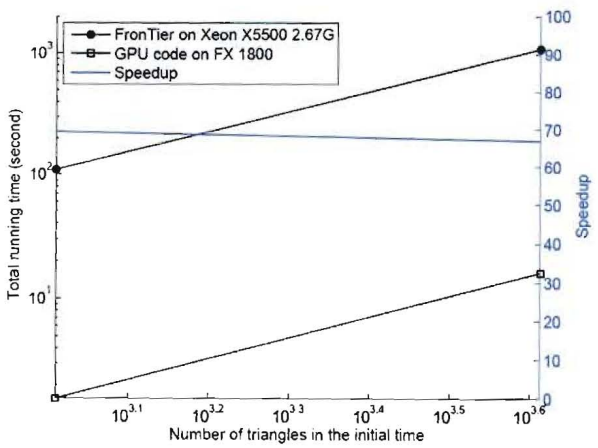




*W. Bo, X. Liu, J. Glimm and X. Li, A robust front tracking method: verification and application to simulation of the primary breakup of liquid jet, *SISC*, to appear.

Comparison of speedup with *FronTier*

- 3D deformation test
- Calculation on GPU: Point propagation, normals, curvatures, grid crossings, components
- Calculation on CPU: redistribution



Summary

- Interface operations in the front tracking method is ported from CPU to GPU, 11-14 times speedup is obtained.
- Finding the one and half ring adjacent points in curvature evaluation is the most time-consuming part in GPU because of the random memory access. It may benefit from the newer GPUs which have caches.
- For CPU code, geometry properties calculations (normals and curvatures) dominate the running time. For GPU code, Redistribution dominates the running time.
- Curvature calculations are very sensitive. On very fine grid, double precision float is necessary for convergent curvature. For curvature independent velocity field, single precision float is enough for convergence.