ParCube

W. Randolph Franklin and Salles V. G. de Magalhães, RPI

2017-11

Abstract

- Parallelization of a 3d application (intersection detection).
- Shows good (uniform grid, radix sort) and bad (octree, recursion) data structures and algorithms for parallelization.
- The good parallel algorithm is also a good sequential one.
- Demos that functional programming via Thrust is a useful abstraction level.
- The challenge is expressing the algorithm using those primitives.
- In parallel, 100x faster than CGAL.

Parallel notes

- Almost all processors, even my smart phone, are parallel.
- Algorithms that don't parallelize are obsolete.
- Nvidia GPUs are almost ubiquitous.
- Thousands of cores execute SIMT in warps of 32 threads.
- Hierarchy of memory: small/fast $\rightarrow$ big/slow
- Communication cost $\gg$ computation cost

Thrust

- C++ template library for CUDA based on STL.
- Functional paradigm: can make algorithms easier to express.
- Hides many CUDA details: good and bad.
- Powerful operators all parallelize: scatter/gather, reduction, reduction by key, permutation, transform iterator, zip iterator, sort, prefix sum.
- Surprisingly efficient algorithms like bucket sort.
- Execution cost relative to CUDA: perhaps factor of 3.
- Possible back ends (via setting flag and recompiling).
  - GPU: CUDA,
  - CPU: OpenMP, TBB, sequential.

Implications of 32-thread warp

- 32 threads execute same instruction.
- Biggest cost is data access.
- Ideally access interleaved data.
- Bad: linked lists, trees, recursion.
- Good: arrays, grids.

Functional programming model

- Map / reduce.
- Permute data with scatter / gather.
- Fast radix sort.
- Surprising what can be done efficiently:
  - run-length encode / decode
  - bucket sort

Uniform grid

Summary

- Overlay a uniform 3D grid on the universe.
- For each input primitive — face, edge, vertex — find overlapping cells.
- In each cell, store set of overlapping primitives.

Properties

- Simple, sparse, uses little memory if well programmed.
- Parallelizable.
- Robust against moderate data nonuniformities.
- Bad worst-case performance on extremely nonuniform data.
- As do octree and all hierarchical methods.

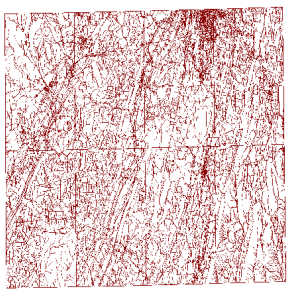
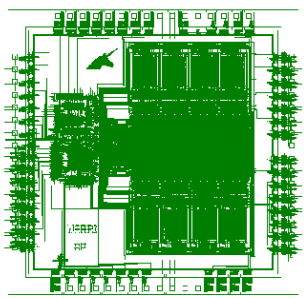
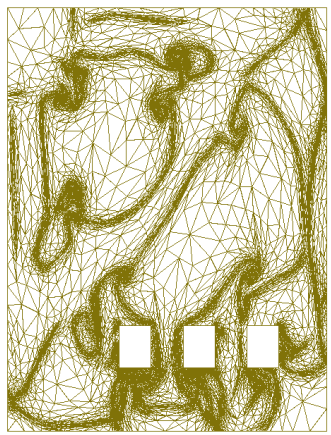
How it works

- Intersecting primitives must occupy the same cell.
- The grid filters the set of possible intersections.

Uniform Grid Qualities

- Major disadvantage: It's so simple that it apparently cannot work, especially for nonuniform data.
- Major advantage: For the operations I want to do (intersection, containment, etc), it works very well for any real data I've ever tried.
- Outside validation: used in our 2nd place finish in November's ACM SIGSPATIAL GIS Cup award.

USGS Digital Line Graph; VLSI Design; Mesh

Uniform Grid Time Analysis

For i.i.d. edges (line segments), show that time to find edge–edge intersections in E^2 is linear in size(input+output) regardless of varying number of edges per cell.

- N edges, length 1/L, $G \times G$ grid.
- Expected # intersections $= \Theta\left(N^2 L^{-2}\right)$.
- Each edge overlaps $\leq 2(G/L+1)$ cells.
- $\eta \stackrel{\Delta}{=}$ # edges per cell, is Poisson; $\overline{\eta} = \Theta(N/G^2(G/L+1))$.
- Expected total # xsect tests: $G^2\overline{\eta^2} = N^2/G^2(G/L+1)^2$.
- Total time: insert edges into cells + test for intersections. $T = \Theta\left(N(G/L+1) + N^2/G^2(G/L+1)^2\right)$.
- Minimized when $G = \Theta(L)$, giving $T = \Theta\left(N + N^2 L^{-2}\right)$.
- $= \Theta$ (size of input + size of output).

■

Sample app: Cube intersection (ParCube)

- useful for
  - collision detection
  - complex boolean operations
- 3D is harder than 2D. (Sweep planes?)
- using N=10M cuts out the toy algorithms,
- output sensitive algorithm required.
- bipartite (red-blue) intersection detection would cause trouble for sweep lines.
- typical prior art: octree.

ParCube

- use specific example here for clarity.
- input: 10M cubes, length 0.003.
- Every following step paralllelizes.
- overlay 300x300x300 grid.
- compute 80M (cell,cube) pairs.
- sort to form ragged array of cubes in each cell.
- compute number of (cube,cube) pairs in each cell (total: 100M pairs).
- compute function mapping each pair to a unique location in pair array, and insert pairs.
- compute which 6M pairs actually intersect and filter array.
- time from when array of input cubes is in computer to when have list of intersecting pairs.
- total time on good Nvidia GPU: 0.33 elapsed seconds.
- 130x faster than CGAL.
- asymptotic time is output sensitive: linear in output size.

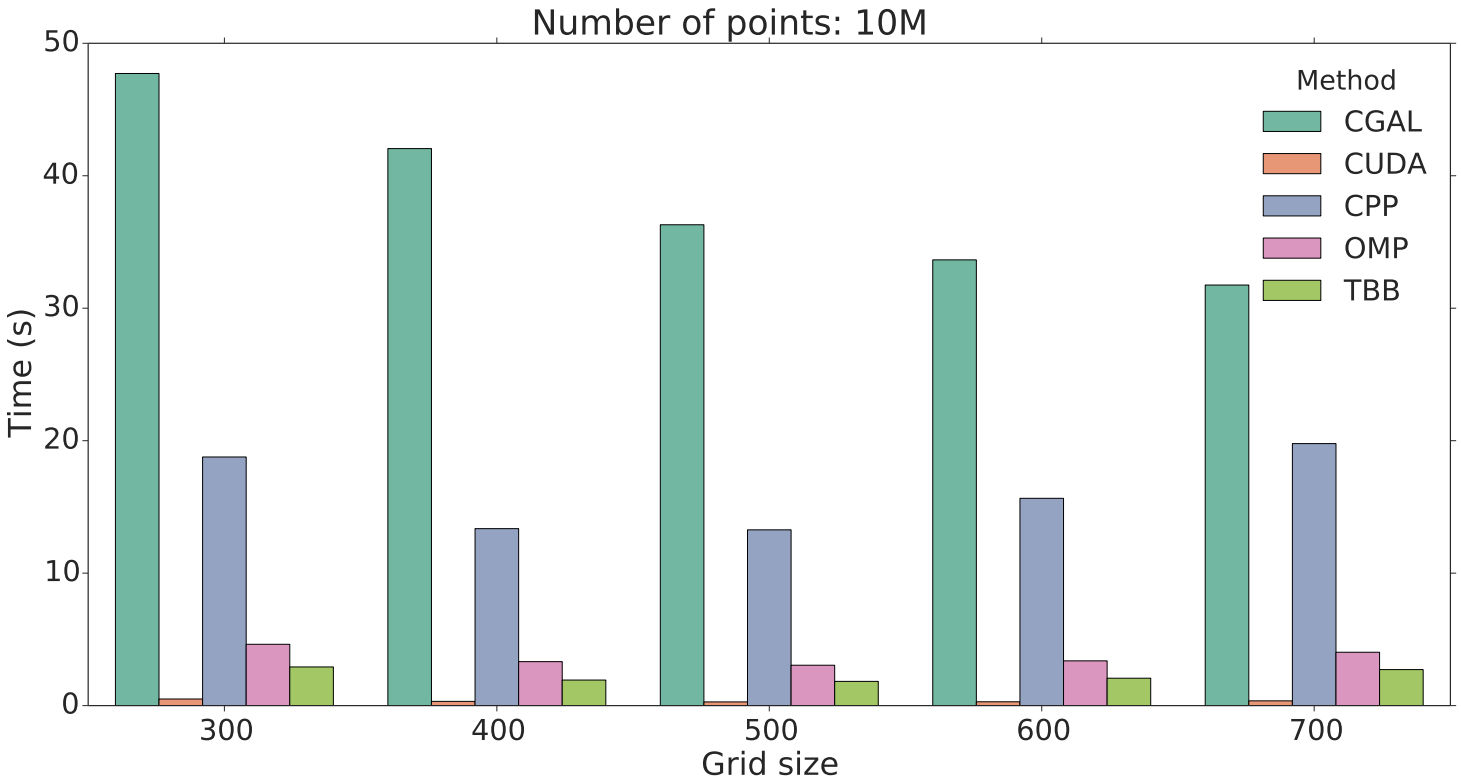
Commentary

- possible backends: sequential, OpenMP, TBB, CUDA.
- hardest part: expressing algorithm within restrictions of Thrust.
- result: very compact straight-line program.
- even sequential is sometimes 3x faster than CGAL.
- more sophisticated algorithms are slower.
- adversary can create bad input, but same with octrees.
- sweep lines not so good in 3D.
- ParCube would extend to higher dimensions.

Validation

- separate implementation using CGAL.
- hardest part was ensuring intersection test did floating roundoff compatibly.
- compared list of intersecting pairs for sample parameters.
- perfect match.

Performance



General lesson, and Future

- simple regular algorithms work very well and parallelize.
- applicable to 7D for robot configuration space collisions.
- Try to compute intersecting graded material properties in additive manufacturing.