GPU-Accelerated Topology Optimization on Unstructured Meshes

Tomás Zegard, Glaucio H. Paulino

University of Illinois at Urbana-Champaign

July 25, 2011

Goal

Study the feasibility of the topology optimization algorithm to be implemented in a massively parallel fashion using conventional off-the-shelf hardware.
(Hopefully) Speed up the process and prepare for future trends in computer architecture.

Breaking up the title:
GPU-Accelerated Topology Optimization on Unstructured Meshes

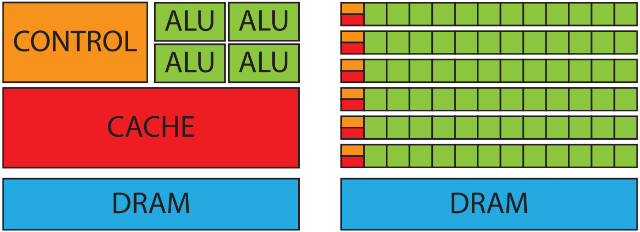
- What are GPUs? Why GPUs? Where can I find these GPUs?
- Why unstructured meshes?
- Challenges in parallelizing TOP...

GPUs: A cheap massively parallel processor

Or your own private and portable supercomputer...

A GPU is a *Graphics Processing Unit* - The video card on your computer.

They have evolved to a tremendous level of parallelism to keep up with the gaming, movie and professional graphics industry.

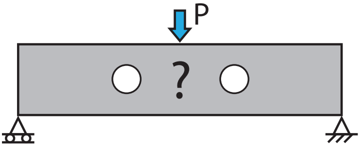


(a) Schematic of a CPU (b) Schematic of a GPU

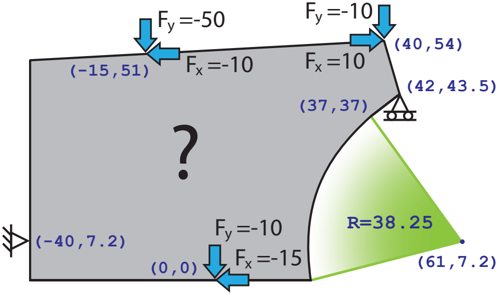
Unstructured Meshes

Is real world structured? Or based on cubes?

Real applications have complex boundaries and/or restrictions.



(c) MBB beam & holes



(d) Curved and diagonal boundaries

- I have never seen a saw-toothed surface in a consumer product.
- Less room for *interpretation* issues.

Race Condition

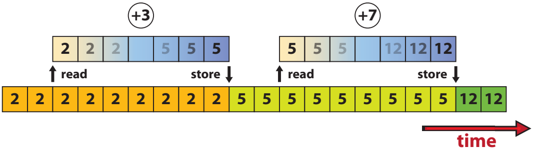
The first big challenge...

Race Condition

A flaw where the output and/or result of a process is wrong because two events that cannot take place at the same time race against each other to influence the result.

It is easier understood with a simple and practical example:

$$2 + 3 + 7 = 12$$

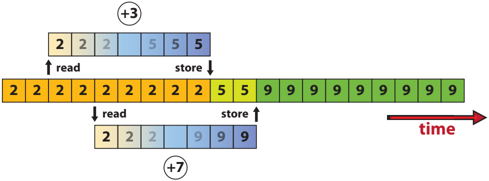


(e) Sequential code with correct result

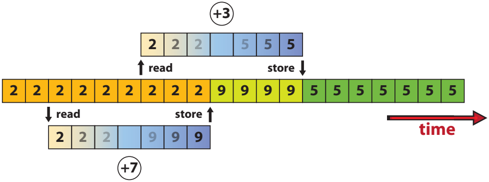
Race Condition

The first big challenge...

What if the +3 operation occurs concurrently with the +7 operation?



(f) Parallel code with wrong result

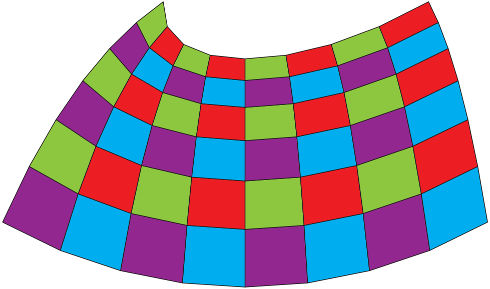


(g) Parallel code with wrong result

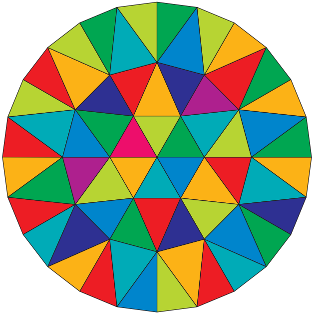
Graph Coloring

Race hazard free FEM assembly...

Only 4 colors are required to assemble a 2D structured mesh of Q4 elements, and 8 for B8.
This does not apply for unstructured meshes.



(h) Structured 2D mesh

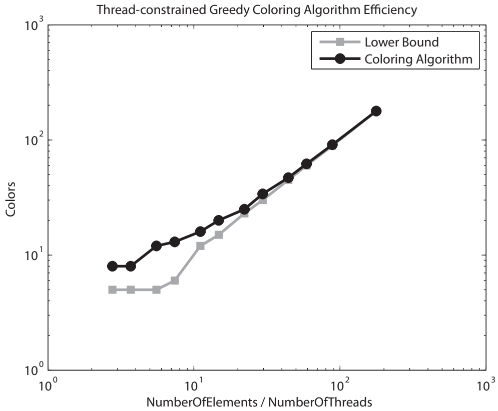


(i) Unstructured 2D mesh

Greedy Coloring

Greedy coloring is not good... most of the time.

Greedy coloring algorithms are among the fastest (and worst) ones.
But! There is a thread restriction: Number of elements per color.

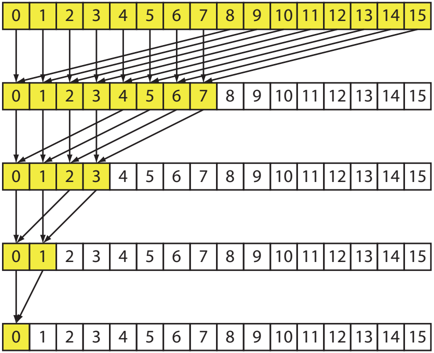


(j) Coloring efficiency for a random mesh

Areas, Volume Fraction and Others

Reduction done the right way, the parallel way...

The calculation of the domain area, volume fraction, change variable and others requires a *reduction*.
A reduction traverses a large list and reduces it to a single value.

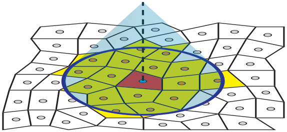


(k) Binary reduction

Filter

Cleaning up undesired things...

The filter for each element is not obvious to compute as in a structured mesh.

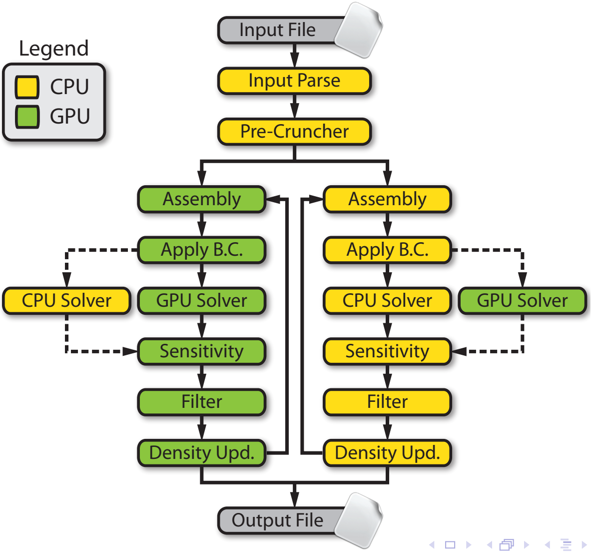


(I) Sample of a filter

The search is done using the communication matrix (moving through neighbors) on a Breadth-First Search fashion (BFS).
Stored in a very compact, simple and easy to read structure.

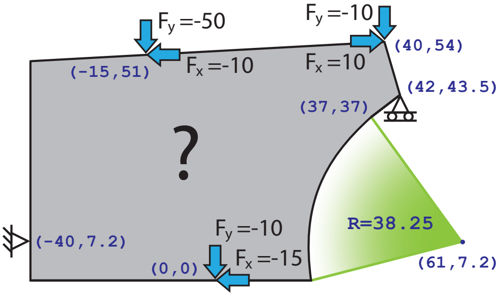
Code Organization

How it works...



Example 1: Bike

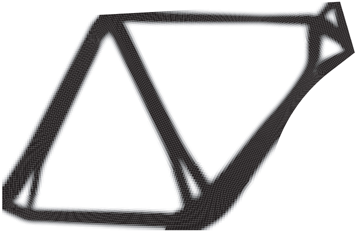
A domain is generated considering the geometrical restrictions from an actual bike.
Loading is expected to resemble a plausible loading scenario on a bike.



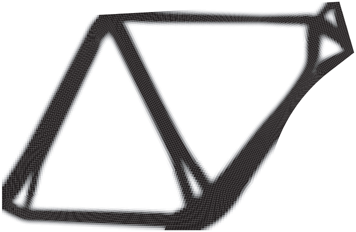
(n) Bike domain and boundary conditions

$E = 700000$, $\nu = 0.3$, $f = 0.3$, $p = 3$ and $R = 1.2$.
30 iterations, 20378 elements and 20635 nodes.

Example 1: Bike



(o) CPU chain



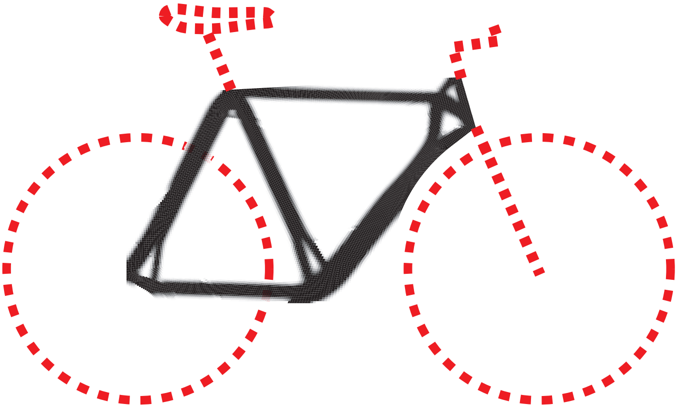
(p) CPU chain (GPU solver)

(q) GPU chain

(r) GPU chain (CPU solver)

Examples 1: Bike

What have we done?!

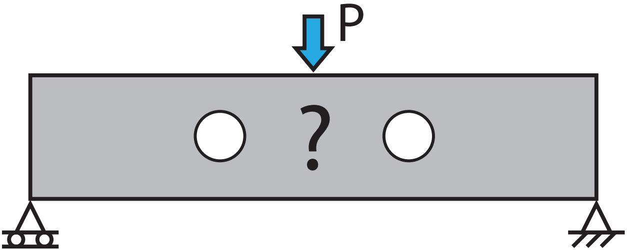


(s) Final bike design with components traced

Examples 2 & 3: Messerschmitt-Bölkow-Blohm Beam

Example 2: Structured, 1 : 6 aspect ratio. 30 iterations, 43200 elements and 46381 nodes.

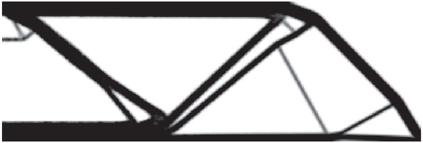
Example 3: Unstructured, 0.4 : 1 : 6 - $D : H : L$ ratio. 30 iterations, 55200 elements and 55900 nodes.



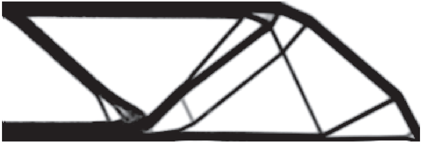
(t) MBB beam with holes

$E = 100$, $\nu = 0.3$, $f = 0.3$, $p = 3$ and $R = 0.02$.

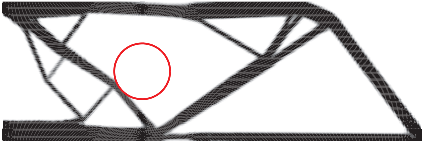
Examples 2 & 3: Messerschmitt-Bölkow-Blohm Beam



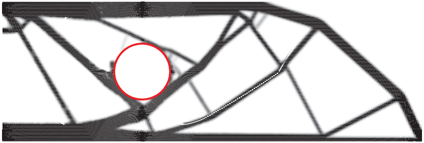
(u) MBB: CPU chain



(v) MBB: GPU chain (CPU solver)



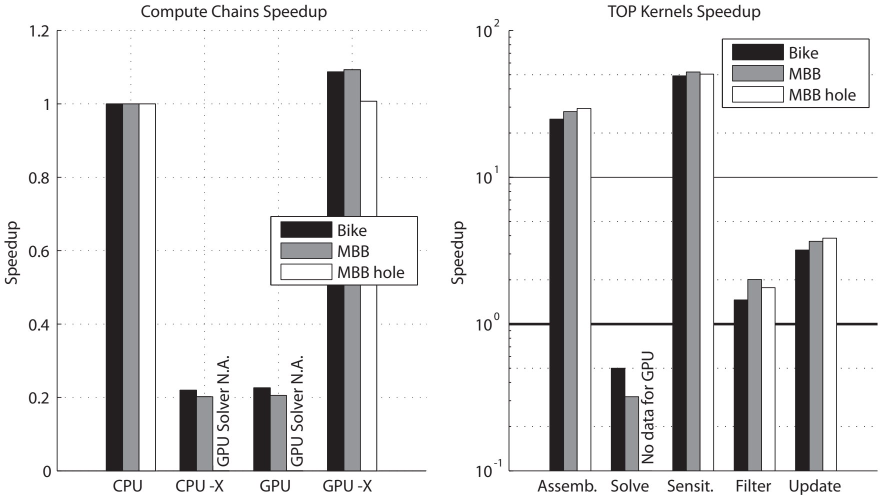
(w) MBB hole: CPU chain



(x) MBB hole: GPU chain (CPU solver)

Benchmarks

Did we do well?



(y) Runtime speedup

(z) Broken down speedup

Conclusions

What have we learned so far?

- TOP in GPUs for unstructured meshes is indeed plausible and the current naïve implementation is slightly faster.
- The solver dominates the overall speedup taking approximately 90% of the runtime.
- Differences in the floating point precision and error handling does make a difference.
- A Greedy coloring algorithm is used with excellent results for large and practical problems.
- Current $1.0\times$ to $1.1\times$ speedup may not seem appealing now. But from this point onwards, improvements would raise the speedup to a clearly better-than-the-CPU code.
- Tremendous need for an efficient GPU solver.

Future Work

Where do we go from here?

- Full support for 3D problems.
- Support for Continuous Approximation of Material Distribution (CAMD) approach.

  Matsui K, Terada K, (2004)
- Faster and better coloring algorithms or race-condition-free assembly procedures.

  Cecka C, Lew AJ, Darve E, (2010)
- Add a flag to the code that indicates if the mesh is structured, and act accordingly.

  Schmidt S, Schulz V, (2009)
- Replace the GPU solver with a **GOOD** one.

  Tomov S, Nath R, Ltaief H, Dongarra J, (2010)

Questions & Comments