A 2589 Line Topology Optimization Code Written for the Graphics Card

Stephan Schmidt, Volker Schulz

July 23rd, 2009

Outline

1. The Graphics Card
   - Processing Unit
   - Memory Management
   - Example Applications
2. Linear Elasticity and Topology Optimization
   - Displacements and Compliance
   - Topology Optimization Problem
   - SIMP Method
3. GPU Implementation
4. Results

Literature

- S. Ananiev.
  On equivalence between optimality criteria and projected gradient methods with application to topology optimization problem.
  Multibody System Dynamics, 13(1):25–38, 2003.
- M. P. Bendsøe and O. Sigmund.
  Topology Optimization – Theory, Methods and Applications.
  Springer, Berlin, Heidelberg, New York, 2nd edition, 2004.
- M. Giles.
  Using NVIDIA GPUs for computational finance.
  `http://people.maths.ox.ac.uk/~gilesm/hpc/`.
- O. Sigmund.
  A 99 line topology optimization code written in matlab.
  Structural and Multidisciplinary Optimization, 21(2):120–127, 2001.

The 3D Accelerator Graphics Card

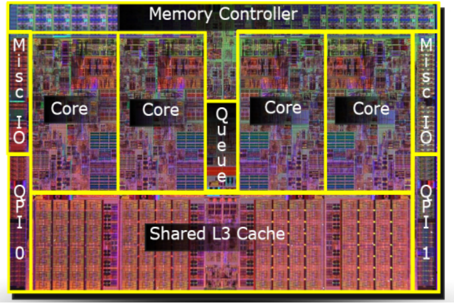
Traditionally:

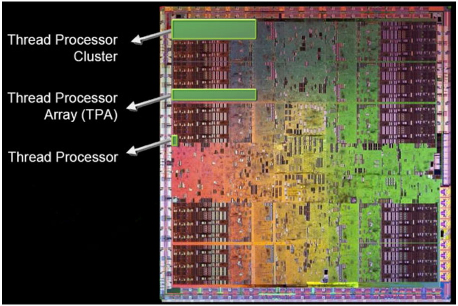
- Highly specialized processor (triangular data types, sub float precision, no integers)

Today:

- Unified shader: Autonomous compute device
- SIMD / Stream architecture
- Highly parallel, up to 512 threads, in order execution
- Ideal for vector processing
- Special programming extensions (CUDA, OpenCL)
- Peta-Flop supercomputers have GPU-like architecture

CPU vs GPU





CPU:

- Cores independent
- Memory accesses hidden by caches automatically
- 10.6 GB/s to RAM (PC2-5300 DDR2), optimized for latency

GPU:

- Cores execute same instructions on different memory address (warp = 32)
- Memory access hidden by coalescence and parallelism by programmer
- 141.7 GB/s to RAM, optimized for bandwidth

GPU Memory Hierarchy

Device Memory (RAM)

- Access time: Up to 600 clock cycles (= 150 float add/mult)
- Remedy: Coalescence: Channel load/store instructions (zero padding, pitch)!!
- Unknowns per thread should be multiple of 4 or 2 but not 3

Shared Memory

- Delivers 32 Bit per clock cycle
- 16 KB in 16 Banks: Bank conflict when too much data needed at the same time or unstructured access (stride)

GPU Memory Hierarchy

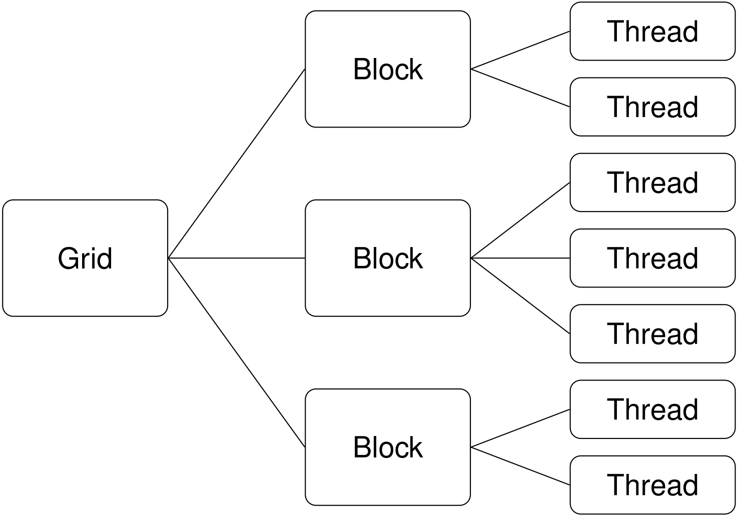
Constant Memory

- Read only
- Cached with broadcast if all threads access same address

Registers

- 8192 per Block, access 0 cycle
- Shared with Shared Memory, potential bank conflicts

Programming Paradigm



Summary GPU Computing

Good Problems:

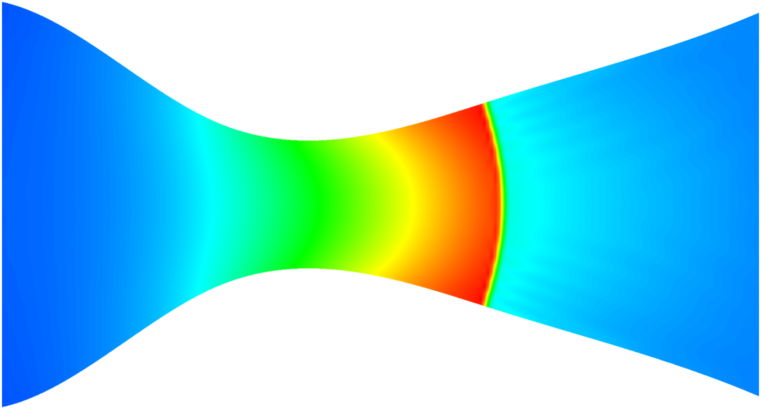
- Dense matvec, Sparse banded matvec
- Fractals
- FFT
- Compute intensive PDE Solvers (High order FVM, Spectral Elements, Lattice Boltzmann)
- Structured meshes (cartesian)

Bad Problems:

- Unstructured sparse matvec
- Unstructured mixed element PDE schemes
- Data intensive tasks

$\Rightarrow$ Numerical schemes should be designed with computer architecture in mind.

GPU Computing in Trier (with M. Siebenborn)



Finite Volume solver for shallow water and Euler equations, JST Scheme, scalar dissipation, structured bodyfitted mesh

Linear Elasticity

- Deformation of a solid body under forces: Displacement vector $u \in \mathbb{R}^3$.
- Linear strain tensor

$$\epsilon_{ij} := \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

- Voight notation for symmetric strain tensor

$$\tilde{\epsilon} := (\epsilon_{11}, \epsilon_{22}, \epsilon_{33}, \epsilon_{12}, \epsilon_{13}, \epsilon_{23})^T =: Bu$$

- Cauchy stress tensor: Young's modulus E, Poisson's ratio ν

$$\sigma = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & & & \\ \nu & 1-\nu & \nu & & & \\ \nu & \nu & 1-\nu & & & \\ & & & 1-2\nu & & \\ & & & & 1-2\nu & \\ & & & & & 1-2\nu \end{bmatrix} \tilde{\epsilon}$$

$$=: CBu$$

Linear Elasticity and Finite Elements

- Weak formulation

$$a(u, v) = \int_{\Omega} (Bv)^T CBu \, dS = L(u) \; \forall v \in V.$$

- Matrix notation

$$K(\Omega)u = f$$

- Forces, loads, supports: f
- Compliance

$$c(u) = u^T f = u^T Ku,$$

Topology Optimization Problem

Mathematical Problem

$$\min_{(u,\Omega)} J(u,\Omega) := u^T K(\Omega) u$$

subject to

$$K(\Omega)u = f$$

$$\mathrm{Vol}(\Omega) = V_0$$

How to deal with the unknown Ω?

- Level-Set method: Ω is zero level of function θ
  - Extract zero-level curve of θ $\Rightarrow$ unstructured curve $\Rightarrow$ unstructured discretization of Ω
- X-FEM
  - Special treatment of bisected elements
- Solid Isotropic Material with Penalization (SIMP), aka homogenization approach

Solid Isotropic Material with Penalization (SIMP)

- Overlay Ω with cartesian grid
- Pseudo-Density in each Finite Element $\rho = (\rho_1, \cdots, \rho_N)^T$:

$$\min_{(u,\rho)} J(u,\rho) := u^T K(\rho) u$$

subject to

$$K(\rho)u = f$$

$$\sum_{e=1}^{N} \rho_e = V_0$$

$$\rho_e \in \{0,1\}$$

- Replace $\rho_e \in \{0,1\}$ by $\rho_e \in [0,1]$

$$a(u,v) = \sum_{e=1}^{N} \int_{\Omega_e} (Bv)^T \rho_e^p C B u \, dS = L(u) \; \forall v \in V$$

- Penalty parameter p

Optimality Criteria

- Lagrangian:

$$
\begin{aligned}
\mathcal{L} =& u^T K(\rho) u + \lambda \left(\sum_{e=1}^{N} \rho_e - V_0 \right) + \mu^T \left(K(\rho) u - f \right) \\
&+ \sum_{e=1}^{N} \alpha_e (-\rho_e) + \sum_{e=1}^{N} \beta_e (\rho_e - 1)
\end{aligned}
$$

- Optimality condition (self-adjoint in μ):

$$
\begin{aligned}
-u_e^T \frac{\partial K_e}{\partial \rho_e} u_e + \lambda - \alpha_e + \beta_e &= 0 \\
\sum_{e=1}^{N} \rho_e - V_0 &= 0 \\
-\rho_e = 0 \quad \text{or} \quad \rho_e - 1 &= 0
\end{aligned}
$$

Optimality Criteria Method

- Gradient of Lagrangian without constraint $\rho_e \in \{0, 1\}$:

$$B_e := \frac{1}{\lambda} u_e^T \frac{\partial K_e}{\partial \rho_e} u_e = 1$$

- Update for ρ_e:

$$\rho_e \leftarrow \begin{cases} \max(\rho_0, \rho_e - m) & \text{if } \rho_e B_e^\eta \leq \max(\rho_0, \rho_e - m) \\ \rho_e B_e^\eta & \text{if } \max(\rho_0, \rho_e - m) < \rho_e B_e^\eta < \min(1, \rho_e + m) \\ \min(1, \rho_e + m) & \text{if } \min(1, \rho_e + m) \leq \rho_e B_e^\eta \end{cases}$$

- Move-limit $m > 0$, damping $\eta = 0.5$
- Bisection for λ
- OC-Update can be interpreted as special projected gradient method for $\rho_e \in [0, 1]$ constraint
- Implemented in One-Shot, i.e. inexact gradient

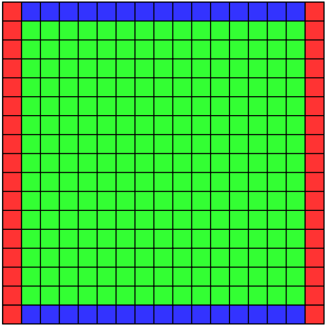
Finite Elements: CPU Code

Compute $\mathrm{RefStiff}[i][j] \in \mathbb{R}^{24\times 24}$
$u^{k+1} = 0$
for all Finite Elements T **do**
    for all vertices i of T **do**
        $t = 0$
        for all vertices j of T **do**
            i_g = Global-Index i
            j_g = Global-Index j
            $t = t + \rho_T \mathrm{RefStiff}[i][j] u^k[j_g]$
        end for
        $u^{k+1}[i_g] = u^{k+1}[i_g] + t$
    end for
end for

Cons:

- Requires 32 global load operations per element
- Requires 8 global store operations per element
- Final store must be atomic! Prohibitive for GPU!

Memory Coalescence

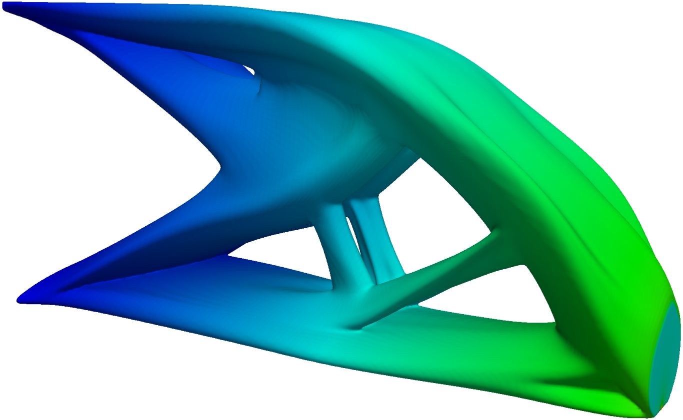


- Memory access is expensive!
- Strategy: Matrix-free FEM with CG
- Cartesian Mesh: $n_x \times n_y \times n_z$ tensor mesh
- Parallelism: Process matvec per 2D slice and stream in k-plane
- Partition 2D slice in *warpsize* $\times$ n blocks, where n is determined from avaliable shared memory

Finite Elements: GPU Code

Compute $\mathrm{RefStiff}[i][j] \in \mathbb{R}^{24\times 24}$ and copy to constant memory
Partition x-y-plane in $\mathit{warpsize} \times n$ patches, launch GPU blocks
Init shared memory, synchronize threads
for all k-planes **do**
    $(i, j) = \mathit{Thread}\text{-}\mathit{ID}$, $\mathit{Res} = 0$ in thread register
    Discard slice, load new one, synchronize
    for all Elements T that have (i, j, k) as a vertex **do**
        (i_2, j_2, k_2) = local index (i, j, k) has in T
        $u_{thread} = u(i, j, k)$ from shared memory
        for all (i_1, j_1, k_1) vertex of T **do**
            $\mathit{Res} = \mathit{Res} + \rho_T \mathrm{RefStiff}[(i_1, j_1, k_1)][(i_2, j_2, k_2)]u_{thread}$
        end for
    end for
    Synch threads
    Upload *Res* from shared to global memory
end for

3D Cantilever



- $180 \times 180 \times 360$ mesh
- $46.5 \cdot 10^6$ unknowns

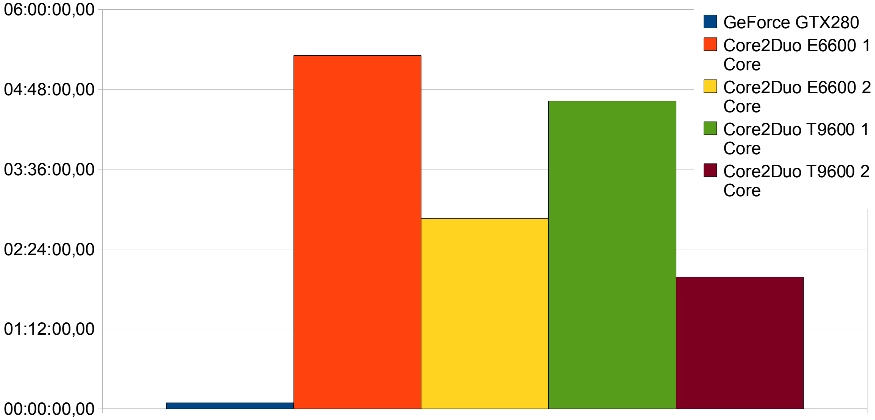
3D Cantilever

(cantsmooth.u3d)

- $80 \times 80 \times 160$ mesh
- Full load in k-direction

Speed-Up

Time for 1000 CG iterations on $180 \times 180 \times 360$ mesh



Conclusions and Future Work

Conclusions

- GPU very fast for problems with specific structure
- Programming: Easy to pick up, hard to master

Future Work

- Multigrid
- Fluid / structure interaction
- Multi-GPU
- Heterogenous CPU / GPU parallelism
- Adaptive load balancing

Code avaliable

`http://www.mathematik.uni-trier.de/~schmidt/gputop`