Algorithms and Cuda Concepts Hexahedron for FE Solid Mechanics SAND2013-8675C

John Mitchell

&

Christian Trott

Sandia National Laboratories
Albuquerque, New Mexico

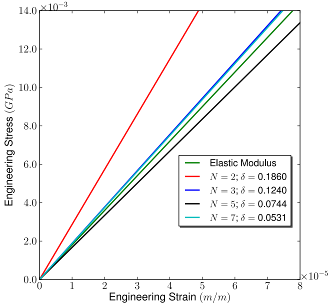
PGI OpenACC Short Course

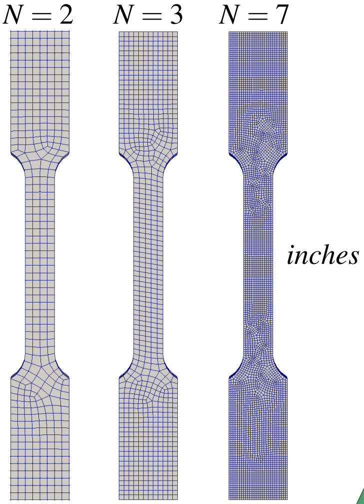
October 9-10, 2013

Sandia National Laboratories is a multi-program laboratory managed and operated, by Sandia Corporation a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL8500.

Mathematical model and discretization of laboratory test

- Momentum equation
- Material model
- Unstructured mesh
- Tensile test



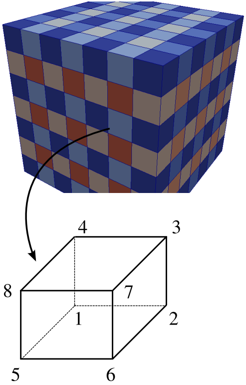


John Mitchell

Uniform Gradient Hex
Key finite element for solid mechanics modeling

Algorithm

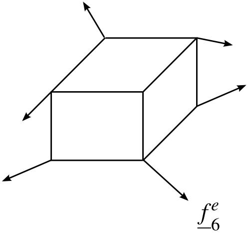
- Deform
- Compute gradient
- Evaluate stress
- Stress divergence
- Assemble



Shape functions ϕ_I

Gradient operator $B_{iI} = \int \frac{\partial \phi_I}{\partial x_i} \, dv$

Stress divergence $f_{iI}^e = \sigma_{ij} B_{jI}$



Colorized assembly $\Sigma_e \underline{f}^e$

Uniform Gradient Hex
Gradient Implementation Concepts

High arithmetic intensity

- Use shared memory
- Size thread blocks to accomodate shared memory
- Maximize use/work of/on shared memory
- Amortize cost of global access across lots of arithmetic

Shape thread blocks: *shape(EPB,dim)*

- Observe column-major ordering of threads
- Align thread layout w/global memory gets/puts
- Select *dim*: accomodate calculation
- Select *dim*: eliminate branching within warp
- *EPB*>=32 && 0==*EPB*%32
  - Prevents warps from crossing *axis* boundary

Gradient calculation (*EPB*: elements per thread block)

Thread hierarchy

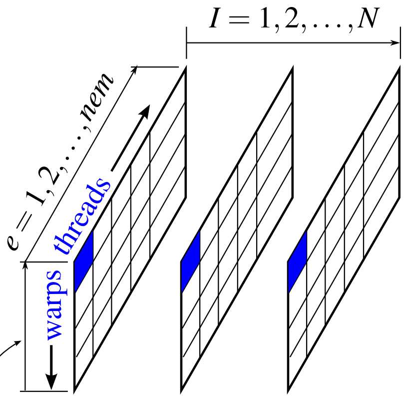
```
dim3 grid((nem+EPB-1)/EPB,1,1)
dim3 block(EPB,3,1)
```

Kernel pseudocode

```
# element id
e=blockIdx.x*EPB+threadIdx.x

# 'early exit'
if(e>=nem) return;
# shared memory
__shared__ real biI[EPB][3][8];
# axis
axis=threadIdx.y
# no branching switch
switch(axis){
    case 0:
        biI[e][0][0:8]=...
        break;
    ...
```

Column-major layout

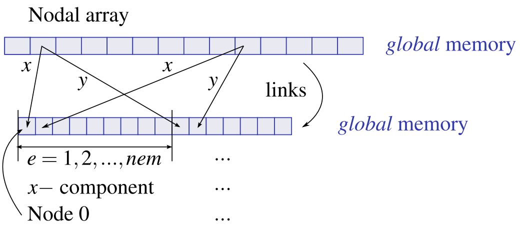


kernel loops over nodes

$N = 8$ on hex

$axis = 1,2,...,d$

Cartoon/Schematic: gather is required at some stage



Coordinates (considerations): time integrator, hourglass implementation, *MPI*

$$
\begin{aligned}
x^e &= X^e + u^e \longleftarrow t_1 \\
y^e &= Y^e + v^e \longleftarrow t_2 \\
z^e &= Z^e + w^e \longleftarrow t_3
\end{aligned}
$$

z^e ← *shared* (3×8); Z^e, w^e ← *global*

Coordinates *shared*: computation of *BiI* and element volume

Displacements *shared*: computation of $\frac{\partial u}{\partial y}$

Calculations: gradient operator, element volume

Gradient operator: *biI*

$$\begin{aligned} bxI &= bxI(x^e, y^e, z^e) \longleftarrow t_1 \\ byI &= byI(x^e, y^e, z^e) \longleftarrow t_2 \quad (3 \times 8)\ \textit{shared} \\ bzI &= bzI(x^e, y^e, z^e) \longleftarrow t_3 \end{aligned}$$

- Later, use each thread t_i to copy rows *biI* into *global* memory

Element volume: *V*

$$\begin{aligned} V_x &= \Sigma\, x_I^e bxI \longleftarrow t_1 \\ V_y &= \Sigma\, y_I^e byI \longleftarrow t_2 \quad (3 \times 1)\ \textit{shared} \\ V_z &= \Sigma\, z_I^e bzI \longleftarrow t_3 \end{aligned}$$

- *syncthreads*
- Use t_1: $V_x \leftarrow (V_x + V_y + V_z)/3$
- *syncthreads*

Calculations: $\frac{\partial u}{\partial y}$, F

Displacement gradient: $\frac{\partial u}{\partial y}$

$$\begin{pmatrix} u_{x,x} & u_{x,y} & u_{x,z} \\ u_{y,x} & u_{y,y} & u_{y,z} \\ u_{z,x} & u_{z,y} & u_{z,z} \end{pmatrix} = \frac{1}{V_x} \begin{pmatrix} \Sigma\, u_I^e bxI & \Sigma\, u_I^e byI & \Sigma\, u_I^e bzI \\ \Sigma\, v_I^e bxI & \Sigma\, v_I^e byI & \Sigma\, v_I^e bzI \\ \Sigma\, w_I^e bxI & \Sigma\, w_I^e byI & \Sigma\, w_I^e bzI \end{pmatrix}$$

t_1 t_2 t_3

Deformation gradient: $F = \left(I - \frac{\partial u}{\partial y}\right)^{-1}$

- Use analytical expression for 3×3 inverse
- Assign row to each thread
- Redundantly calculate determinant
- Assign *global* memory for F on each element

Gradient calculation on hex *shared* memory requirements

Shared memory per element (64 bit *real*)

field	shape	bytes
displacement u	3×8	192
coordinates y	3×8	192
gradient operator biI	3×8	192
element volume V	3×1	24
displacement gradient $u_{i,j}$	3×3	72
TOTAL		672

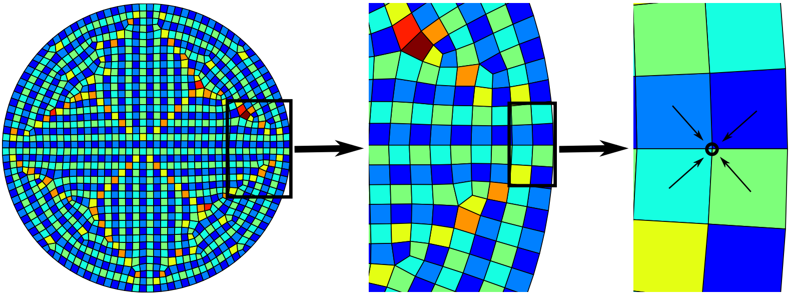
Size *thread* blocks and *grid*

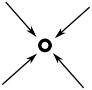
- Respect 48k shared memory limitation per *SM*
- Ensure warp sizes of at least 32
- Since *shared* memory is a limiter, size *grid* with *nem*

Choices for *thread* blocks: EPB =elements per *thread* block

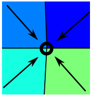
- $EPB = 32 \rightarrow 21k$ *shared* memory; get 2 blocks per *SM*
- $EPB = 64 \rightarrow 43k$ *shared* memory; get 1 blocks per *SM*

Element coloring
What is it? Why do we need it?





Elements sharing a node have a different color



Colorized assembly (efficiently eliminate race condition)

Concurrently process elements of same color

Element/material evaluation for a time step *cuda streams* & *host* calculations (schematic/outline)

Loop material/element blocks (*cuda streams*)

- *cudaMemcpyAsync*: gather nodal field(s) to element on device
- *Asynchronous* gradient calculation
- *cudaMemcpyAsync*: copy F to host

Loop material/element blocks (*host calculations*)

- Compute polar decomposition
- Compute stress

Loop material/element blocks (*cuda streams*)

- *cudaMemcpyAsync*: copy polar decomposition & stress to device
- *Asynchronous* hourglass calculation

Loop *colors*, loop material/element blocks

- *Asynchronous* stress divergence and assembly

Closing
Questions/Discussion for *OpenACC*

Can these concepts/constructs be replicated using *OpenACC*

- Control over shape of thread blocks
- Device *shared memory*
- Barrier __*syncthreads()*
- Early exit on *warps*
- Switch statements on *warps*
- *Streams*