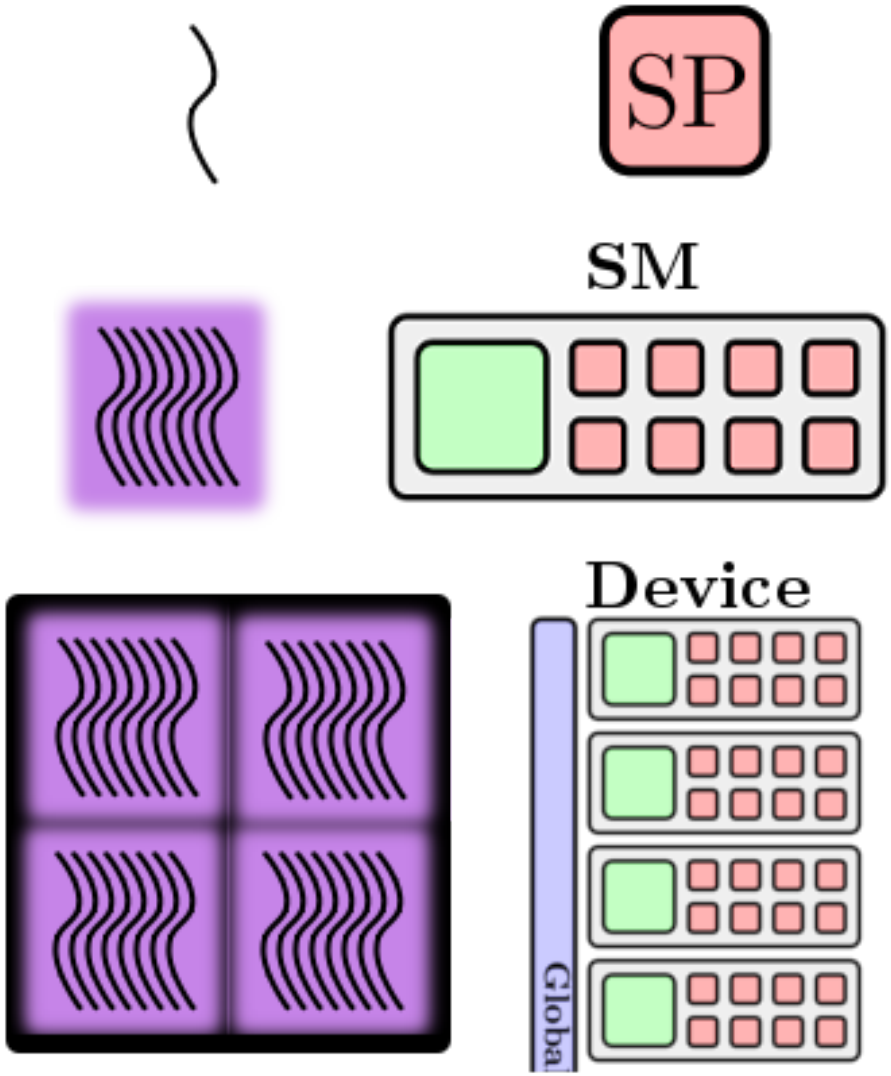
Assembly of Finite Element Methods on Graphics Processors.

Cris Cecka Adrian Lew Eric Darve

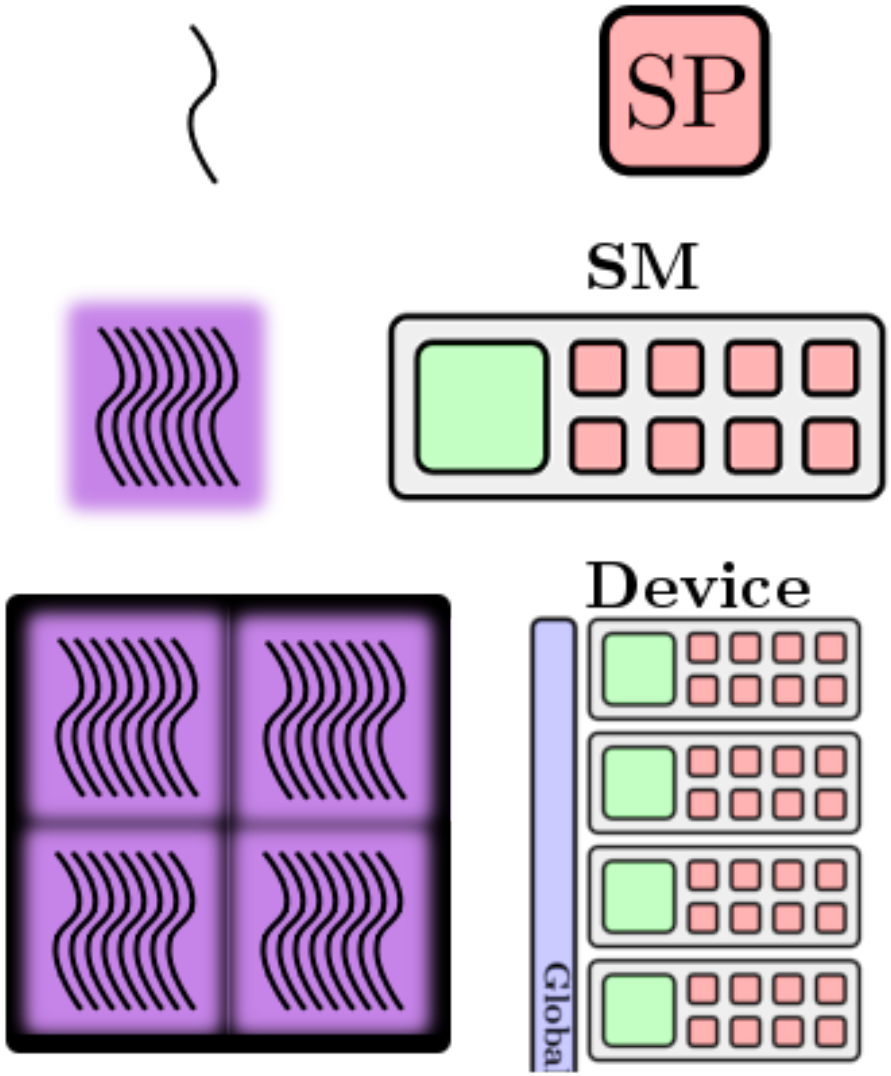
Department of Mechanical Engineering
Institute for Computational and Mathematical Engineering
Stanford University

July 19th, 2010
9th World Congress on Computational Mechanics

GPU Computing

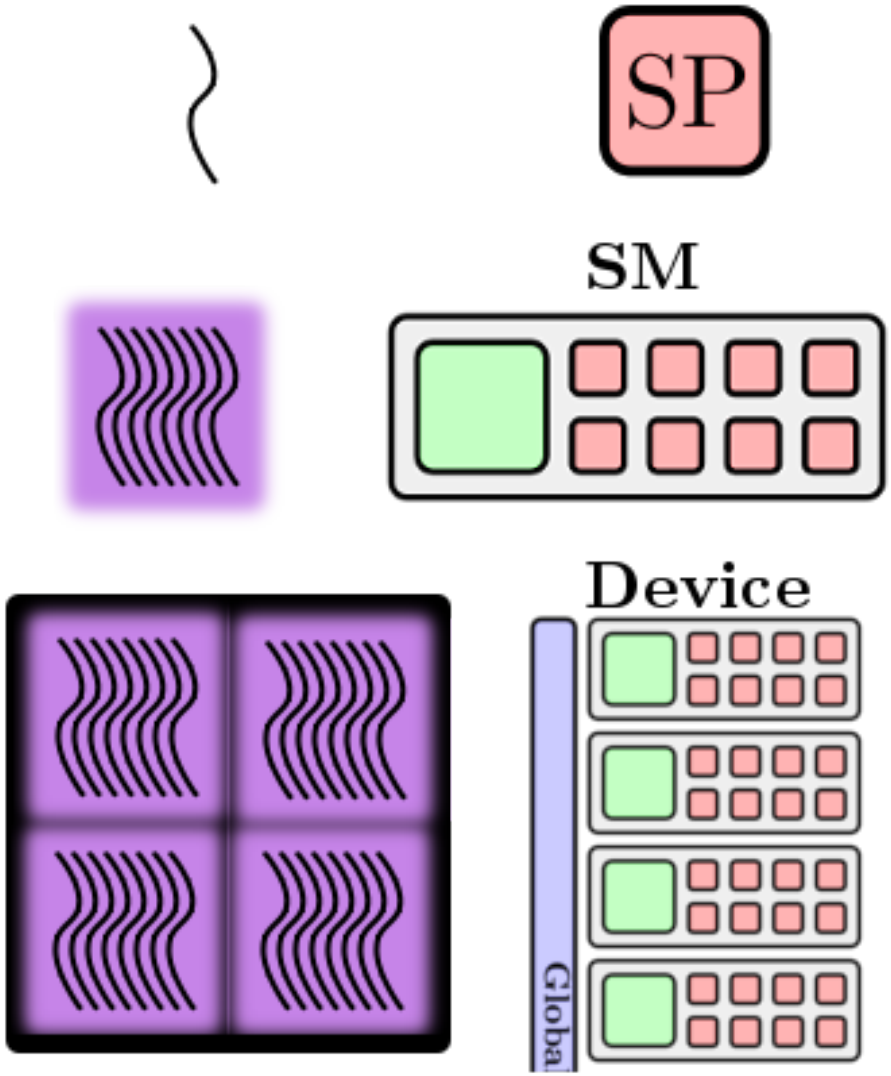


GPU Computing



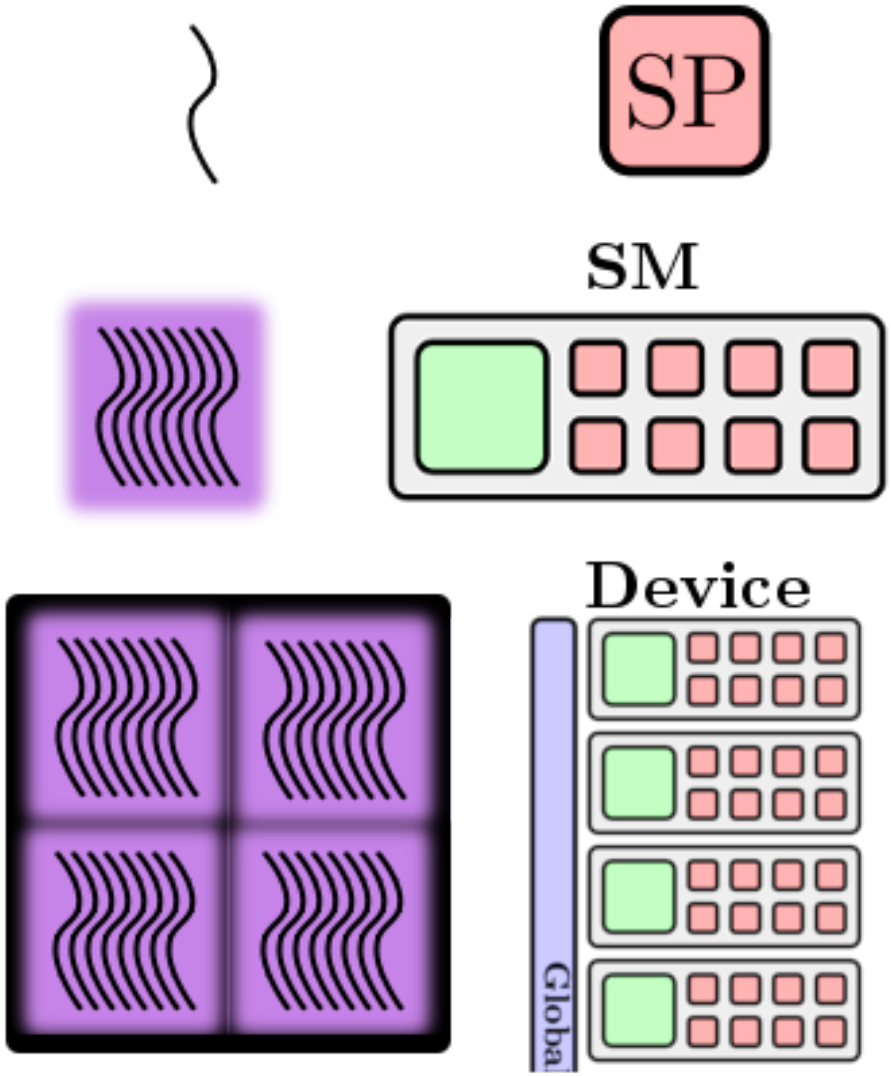
- *Threads* executed by *streaming processors*
  - On-Chip *Registers*
  - Off-chip *Local Memory*

GPU Computing



- *Threads* executed by *streaming processors*
  - On-Chip *Registers*
  - Off-chip *Local Memory*
- *Block* of threads executed on *streaming multiprocessors*
  - On-chip *Shared Memory*

GPU Computing



- *Threads* executed by *streaming processors*
  - On-Chip *Registers*
  - Off-chip *Local Memory*
- *Block* of threads executed on *streaming multiprocessors*
  - On-chip *Shared Memory*
- *Grid* of blocks executed on the *device*
  - Off-chip *Global Memory*

Why FEM Assembly on the GPU?

- Complex, real-time physics common on the GPU.
  - Gaming and graphics community.
  - Simulation and visualization community.
  - More recently, HPC community.

Why FEM Assembly on the GPU?

- Complex, real-time physics common on the GPU.
  - Gaming and graphics community.
  - Simulation and visualization community.
  - More recently, HPC community.
- Sparse Linear Algebra coming of age on GPU.
  - Extensive research on Sparse Solvers on GPU.
  - Extensive research on SpMV.

Why FEM Assembly on the GPU?

- Complex, real-time physics common on the GPU.
  - Gaming and graphics community.
  - Simulation and visualization community.
  - More recently, HPC community.
- Sparse Linear Algebra coming of age on GPU.
  - Extensive research on Sparse Solvers on GPU.
  - Extensive research on SpMV.

- Non-linear and time-dependent problems require many assembly procedures.

Why FEM Assembly on the GPU?

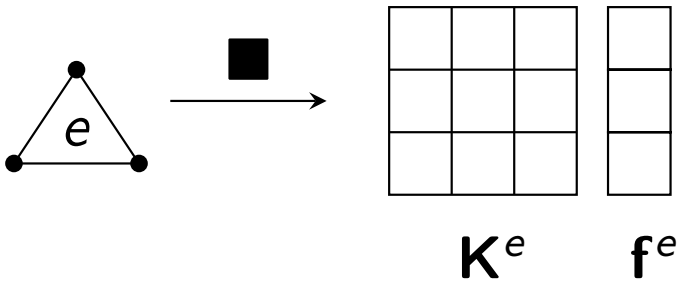
- Complex, real-time physics common on the GPU.
  - Gaming and graphics community.
  - Simulation and visualization community.
  - More recently, HPC community.
- Sparse Linear Algebra coming of age on GPU.
  - Extensive research on Sparse Solvers on GPU.
  - Extensive research on SpMV.

- Non-linear and time-dependent problems require many assembly procedures.
- Can assemble, solve, update, and visualize on the GPU
  - Completely avoid costly transfers with CPU.
  - Fast (real-time) simulations with visualization.

FEM Direct Assembly

Most common FEM assembly procedure:

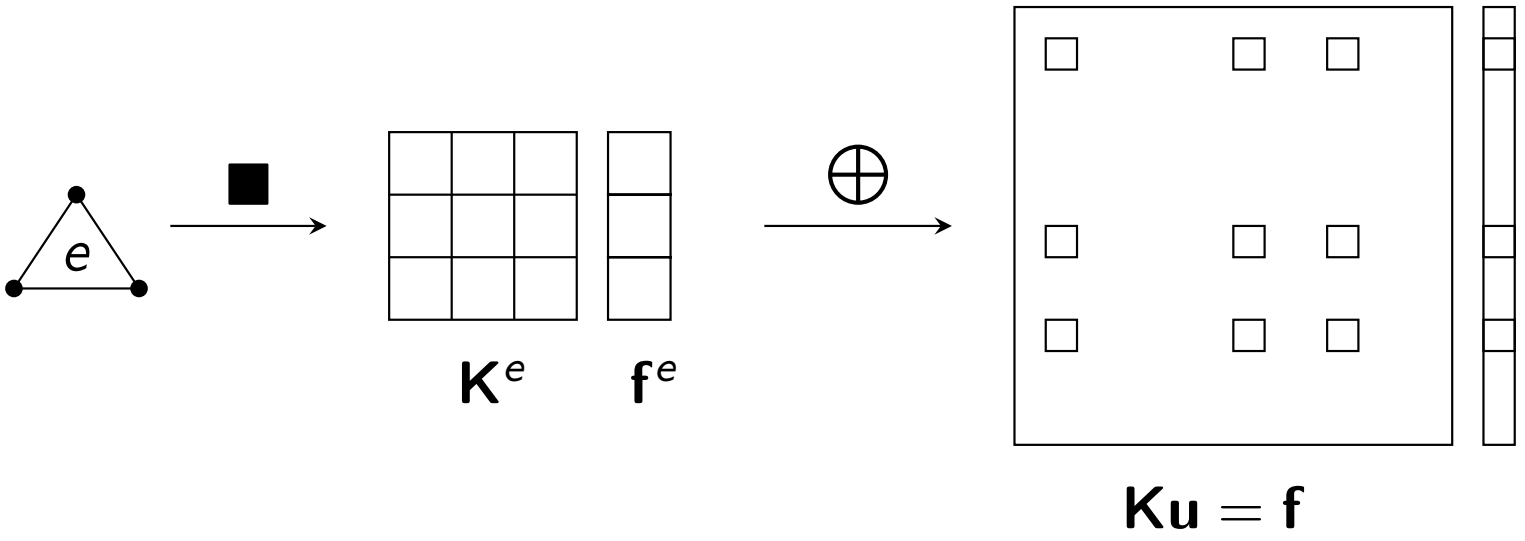
- Compute element data.
  - One by one.



FEM Direct Assembly

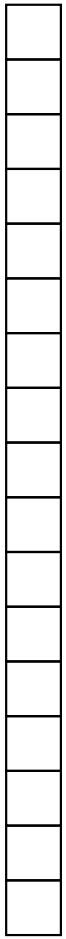
Most common FEM assembly procedure:

- Compute element data.
  - One by one.
- Accumulate into global system.
  - Using a local index to global index mapping.

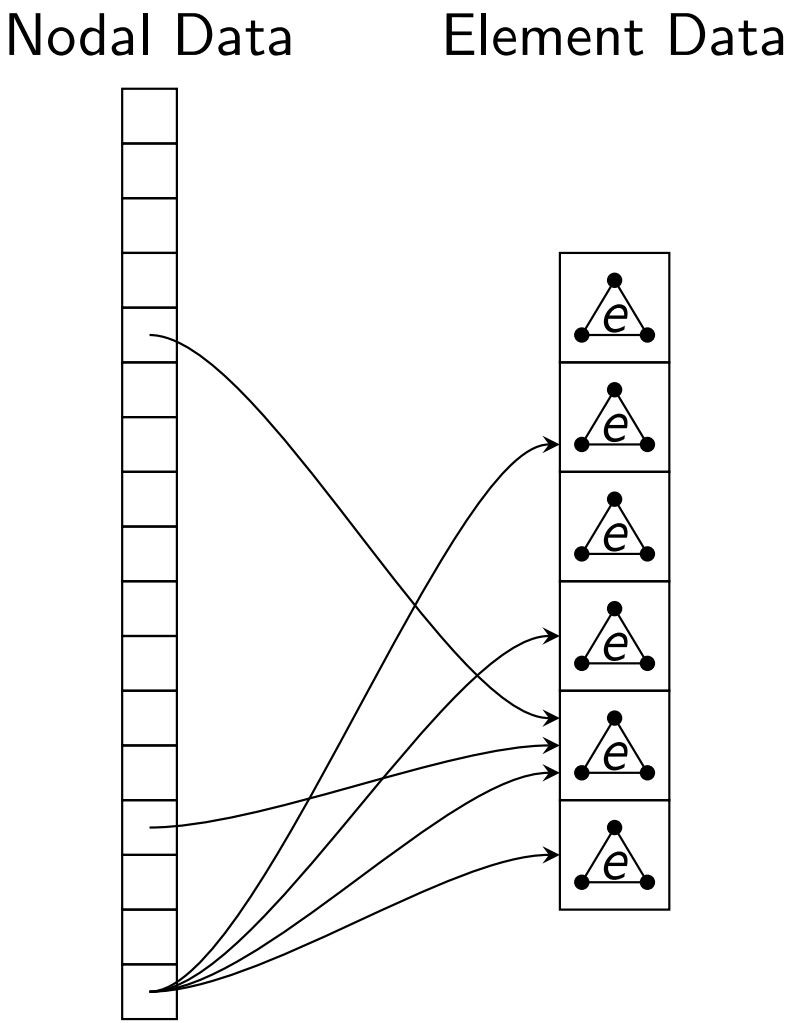


Data Flow

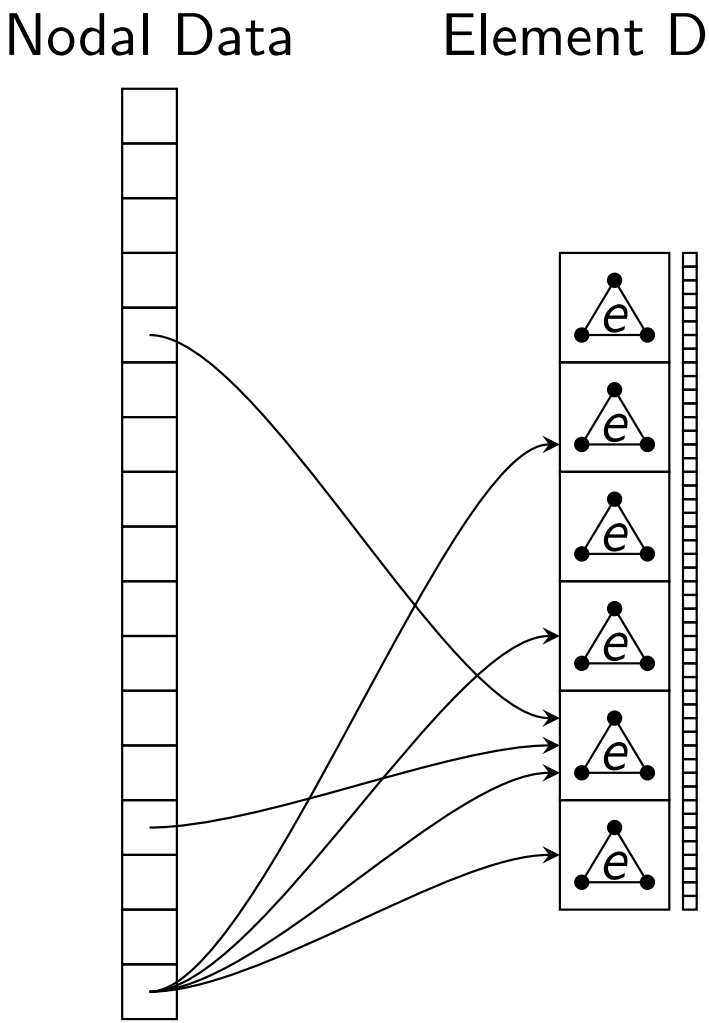
Nodal Data



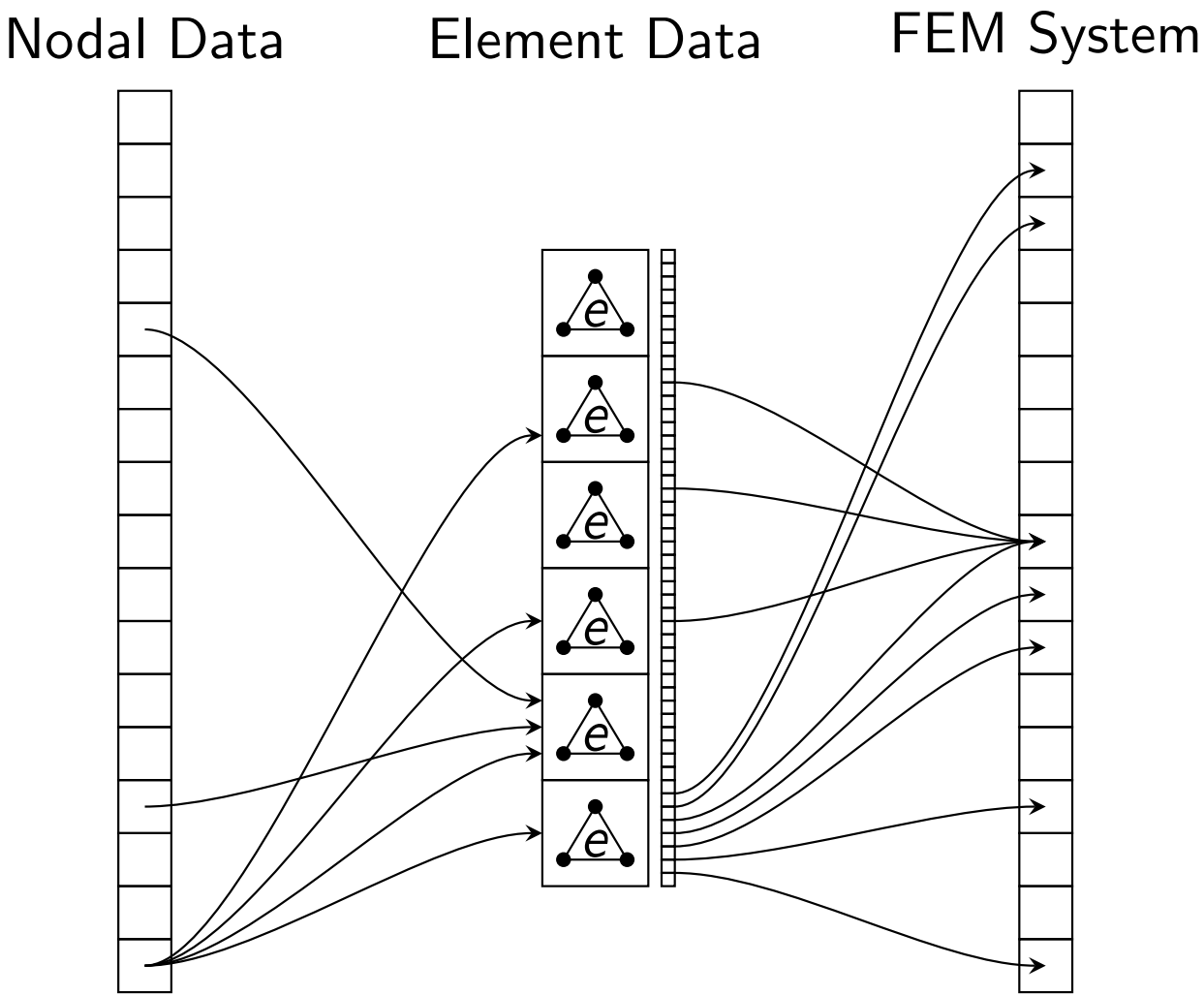
Data Flow



Data Flow



Data Flow



GPU FEM Assembly Strategies

Key concerns for GPU algorithms:

- Distribute the task into independent blocks of work.
  - No inter-block communication.
  - Minimize redundant computations.

GPU FEM Assembly Strategies

Key concerns for GPU algorithms:

- Distribute the task into independent blocks of work.
  - No inter-block communication.
  - Minimize redundant computations.
- Maximize flop::word ratio.
  - Minimize global memory transactions.
  - Use exposed memory hierarchy to maximize data reuse.

GPU FEM Assembly Strategies

Two Key Choices:

Store Element Data In

Threads Assemble By

GPU FEM Assembly Strategies

Two Key Choices:

Store Element Data In

- Global Memory
- Local Memory
- Shared Memory

Threads Assemble By

GPU FEM Assembly Strategies

Two Key Choices:

Store Element Data In

- Global Memory

  Computation – Assembly.
  Min computation.
- Local Memory
- Shared Memory

Threads Assemble By

GPU FEM Assembly Strategies

Two Key Choices:

Store Element Data In

- Global Memory

  Computation – Assembly.
  Min computation.
- Local Memory

  Fast read/write.
  No shared element data.
- Shared Memory

Threads Assemble By

GPU FEM Assembly Strategies

Two Key Choices:

Store Element Data In

- Global Memory
  - Computation – Assembly.
  - Min computation.
- Local Memory
  - Fast read/write.
  - No shared element data.
- Shared Memory
  - Fast read/write.
  - Shared element data.
  - Small size.

Threads Assemble By

GPU FEM Assembly Strategies

Two Key Choices:

Store Element Data In

- Global Memory
  - Computation – Assembly.
  - Min computation.
- Local Memory
  - Fast read/write.
  - No shared element data.
- Shared Memory
  - Fast read/write.
  - Shared element data.
  - Small size.

Threads Assemble By

- Non-zero (NZ)
- Row
- Element

GPU FEM Assembly Strategies

Two Key Choices:

Store Element Data In

- Global Memory
  - Computation – Assembly.
  - Min computation.
- Local Memory
  - Fast read/write.
  - No shared element data.
- Shared Memory
  - Fast read/write.
  - Shared element data.
  - Small size.

Threads Assemble By

- Non-zero (NZ)
  - Simple - Indexing.
  - Imbalanced.
- Row
- Element

GPU FEM Assembly Strategies

Two Key Choices:

Store Element Data In

- Global Memory
  - Computation – Assembly.
  - Min computation.
- Local Memory
  - Fast read/write.
  - No shared element data.
- Shared Memory
  - Fast read/write.
  - Shared element data.
  - Small size.

Threads Assemble By

- Non-zero (NZ)
  - Simple - Indexing.
  - Imbalanced.
- Row
  - More balanced.
  - Lookup tables.
- Element

GPU FEM Assembly Strategies

Two Key Choices:

Store Element Data In

- Global Memory
  - Computation – Assembly.
  - Min computation.
- Local Memory
  - Fast read/write.
  - No shared element data.
- Shared Memory
  - Fast read/write.
  - Shared element data.
  - Small size.

Threads Assemble By

- Non-zero (NZ)
  - Simple - Indexing.
  - Imbalanced.
- Row
  - More balanced.
  - Lookup tables.
- Element
  - Min computation.
  - SIMD.
  - Race conditions.

Local-Element

- Assign one thread to one element.
  - Compute the element data.
  - Assemble directly into system.

Local-Element

- Assign one thread to one element.
  - Compute the element data.
  - Assemble directly into system.

Race conditions still possible!

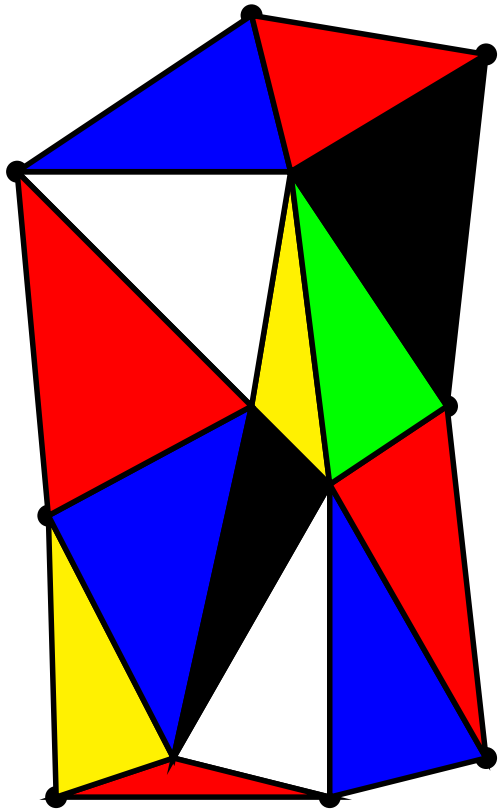
Local-Element - Coloring the Mesh

- Assign one thread to one element.
  - Compute the element data.
  - Assemble directly into system.

Race conditions still possible!

Partition elements to resolve race conditions.

- Transform into a vertex coloring problem.
- In general, k-coloring is NP-complete.
  - But we don't need an optimal coloring.



Local-Element - Coloring the Mesh

- Assign one thread to one element.
  - Compute the element data.
  - Assemble directly into system.

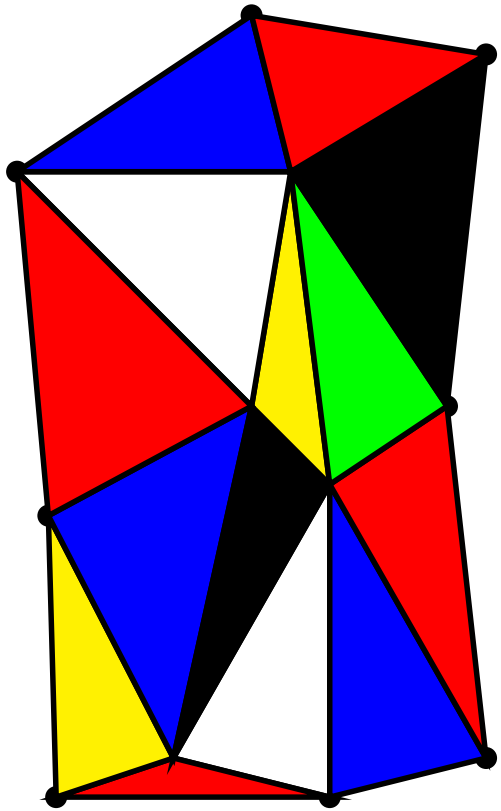
Race conditions still possible!

Partition elements to resolve race conditions.

- Transform into a vertex coloring problem.
- In general, k-coloring is NP-complete.
  - But we don't need an optimal coloring.

Problems.

- No sharing of nodal or element data.
- Little utilization of GPU resources.



Global-NZ

- Kernel1 - Assign one thread to one element.
  - Compute the element data.
  - Store element data in global memory.

Global-NZ

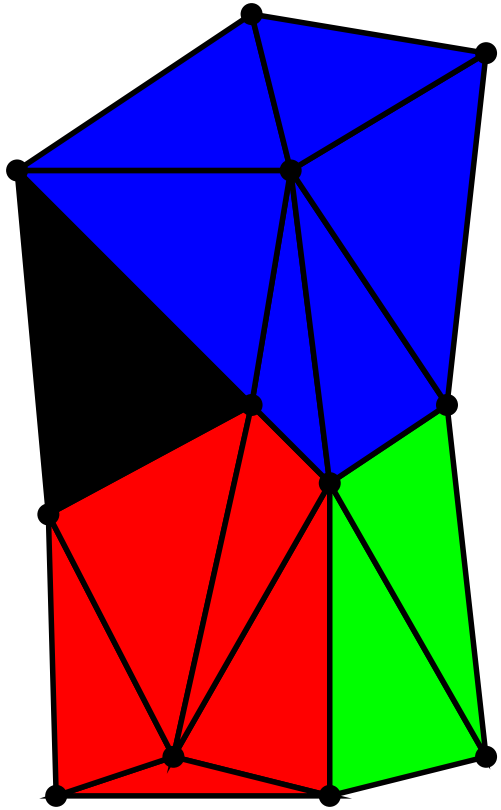
- Kernel1 - Assign one thread to one element.
  - Compute the element data.
  - Store element data in global memory.
- Kernel2 - Assign one thread to one NZ.
  - Assemble from global memory.

Global-NZ

- Kernel1 - Assign one thread to one element.
  - Compute the element data.
  - Store element data in global memory.
- Kernel2 - Assign one thread to one NZ.
  - Assemble from global memory.

Optimizing:

- Cluster the elements so they share nodes.
- Prefetch nodal data into shared memory.
- Up to almost 3x speedup.



Global-NZ

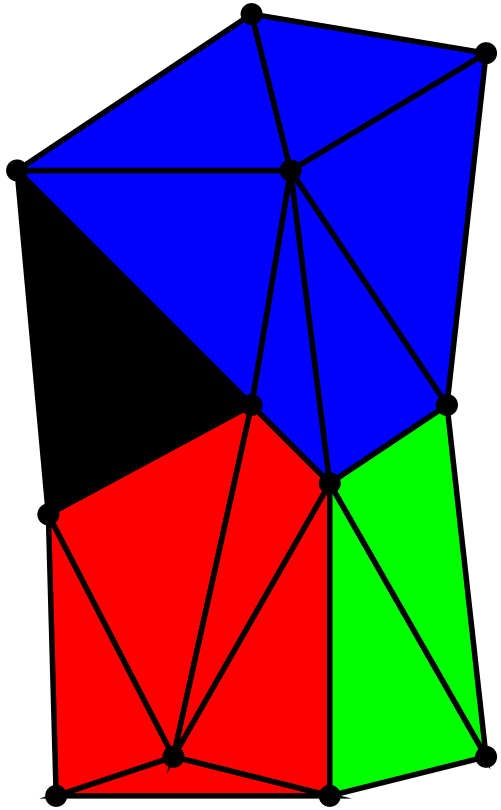
- Kernel1 - Assign one thread to one element.
  - Compute the element data.
  - Store element data in global memory.
- Kernel2 - Assign one thread to one NZ.
  - Assemble from global memory.

Optimizing:

- Cluster the elements so they share nodes.
- Prefetch nodal data into shared memory.
- Up to almost 3x speedup.

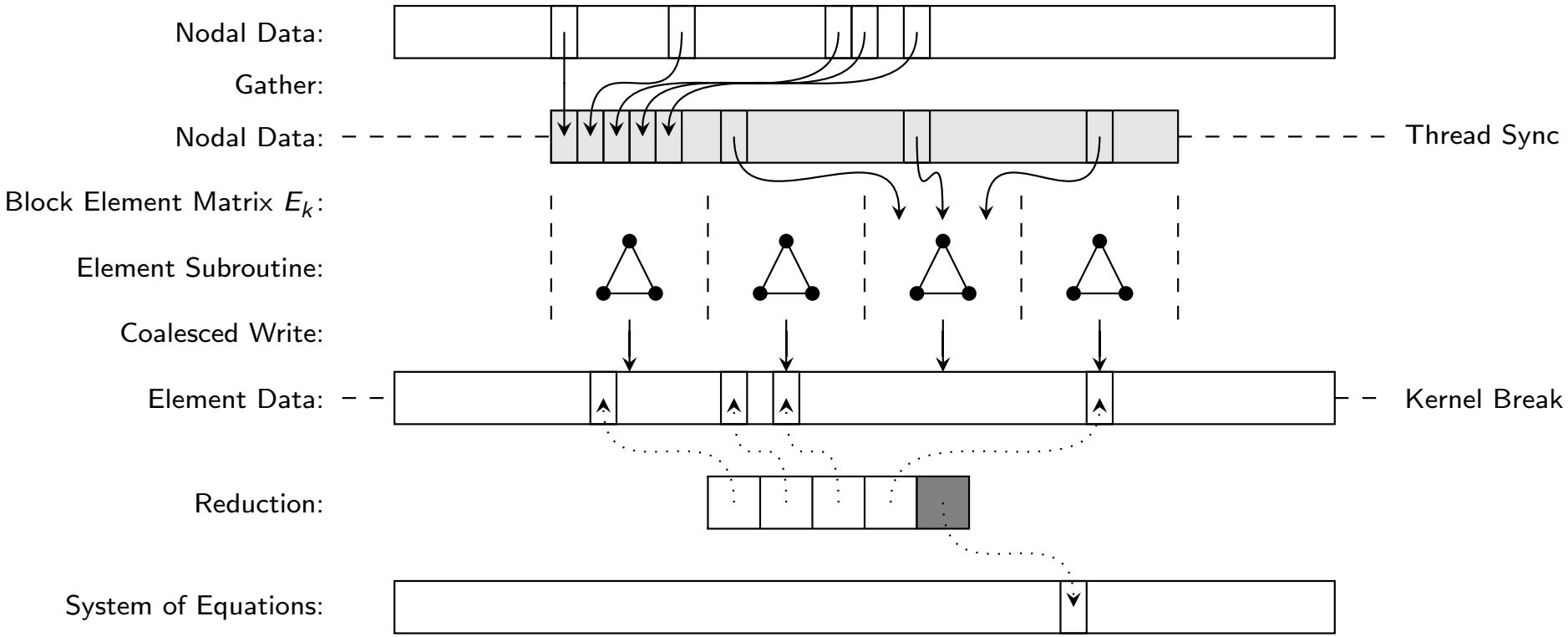
Problems.

- Two passes through global memory.
- Limited by global memory size.



Global-NZ Data Flow

The optimized algorithm looks like:



Shared-NZ

- Assign one thread to one element.
  - Compute the element data.
  - Store element data in shared memory.

Shared-NZ

- Assign one thread to one element.
  - Compute the element data.
  - Store element data in shared memory.
- Reassign threads to NZs.
  - Assemble from shared memory.

Shared-NZ

- Assign one thread to one element.
  - Compute the element data.
  - Store element data in shared memory.
- Reassign threads to NZs.
  - Assemble from shared memory.

A set of NZs requires a set of elements.

- Must compute all "halo" element data.

Shared-NZ

- Assign one thread to one element.
  - Compute the element data.
  - Store element data in shared memory.
- Reassign threads to NZs.
  - Assemble from shared memory.

A set of NZs requires a set of elements.

- Must compute all "halo" element data.

A set of elements requires a set of nodes.

- Must gather all "halo" nodal data.

Shared-NZ

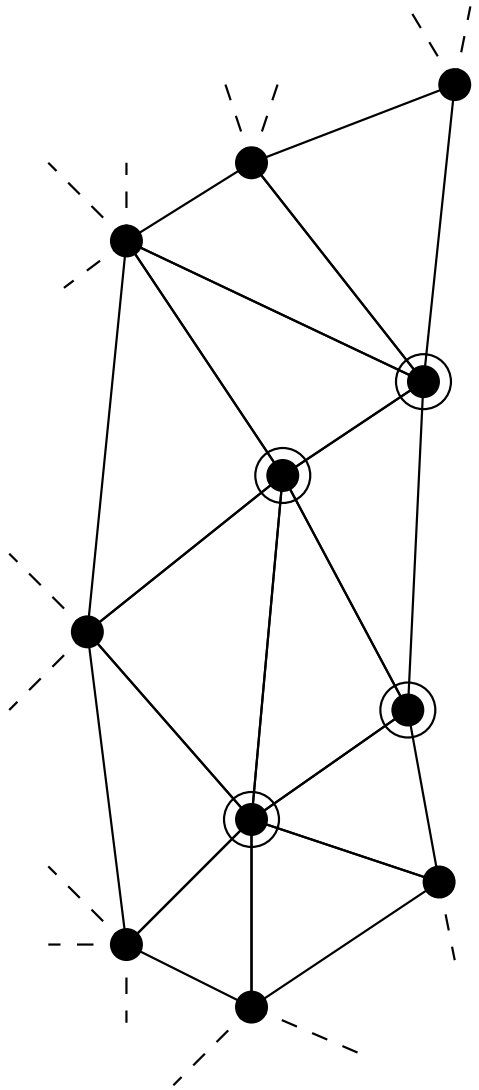
- Assign one thread to one element.
  - Compute the element data.
  - Store element data in shared memory.
- Reassign threads to NZs.
  - Assemble from shared memory.

A set of NZs requires a set of elements.

- Must compute all "halo" element data.

A set of elements requires a set of nodes.

- Must gather all "halo" nodal data.



Shared-NZ

- Assign one thread to one element.
  - Compute the element data.
  - Store element data in shared memory.
- Reassign threads to NZs.
  - Assemble from shared memory.

A set of NZs requires a set of elements.

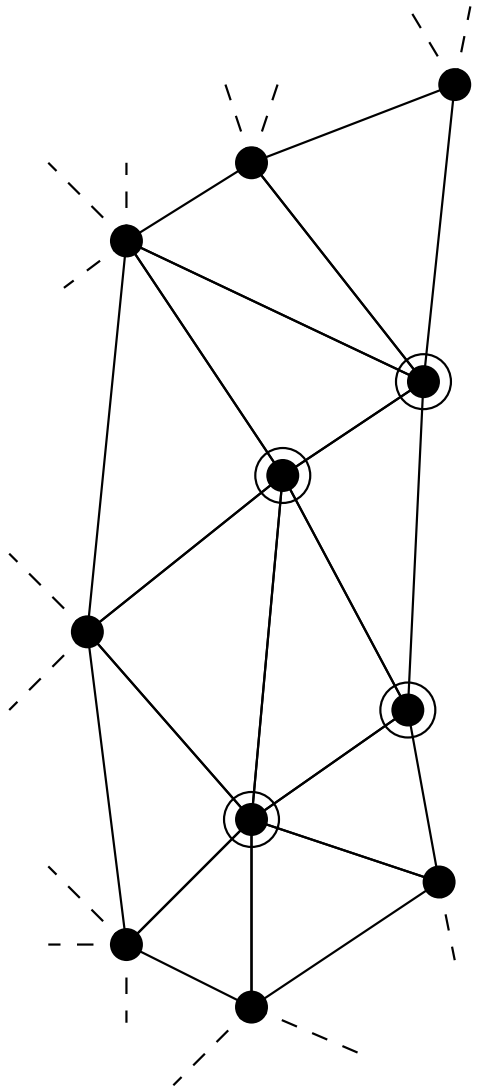
- Must compute all "halo" element data.

A set of elements requires a set of nodes.

- Must gather all "halo" nodal data.

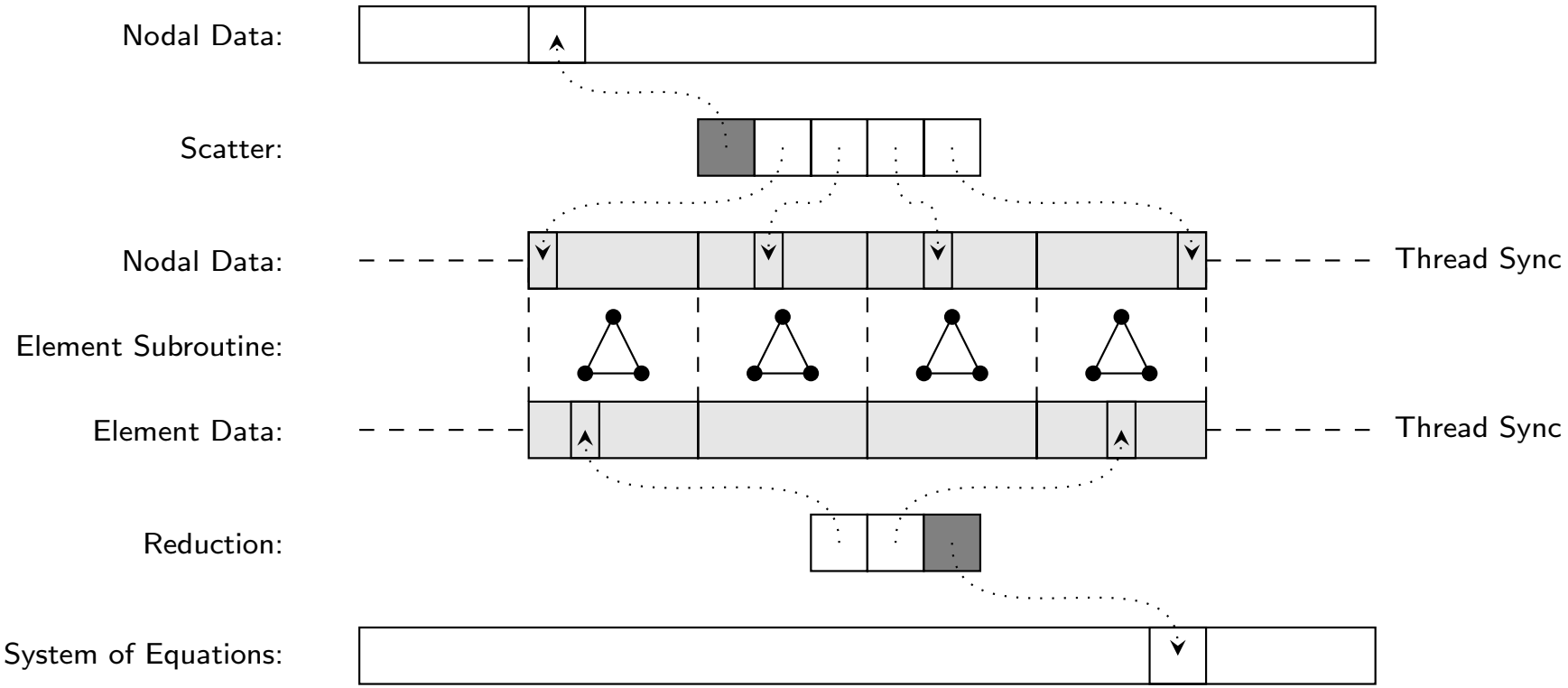
Problems.

- Shared memory size is very limiting.



Shared-NZ Data Flow

The optimized algorithm looks like:



Scatter and Reduction Arrays

General procedure:

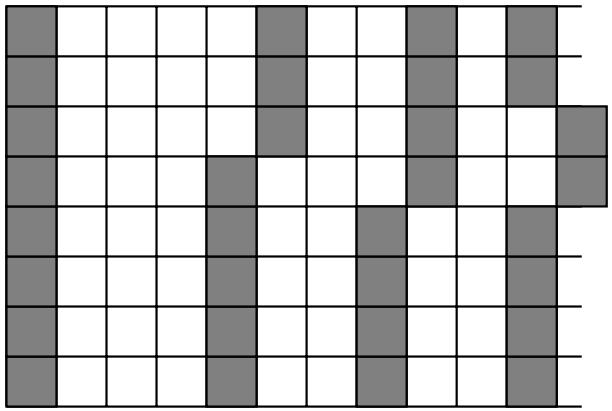
- Make a set of operations to be done for each partition.
- Pack these into an array such that reading is coalesced.

Scatter and Reduction Arrays

General procedure:

- Make a set of operations to be done for each partition.
- Pack these into an array such that reading is coalesced.

Scatter Array:

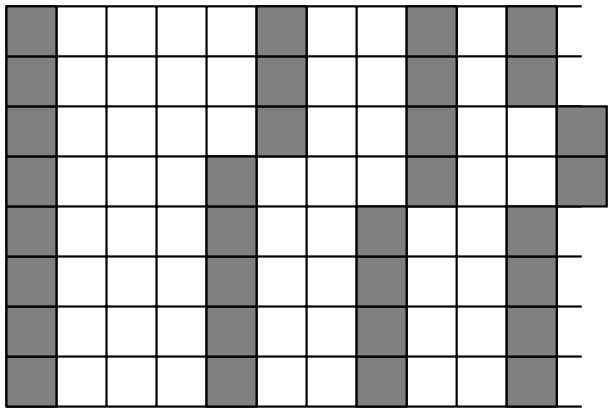


Scatter and Reduction Arrays

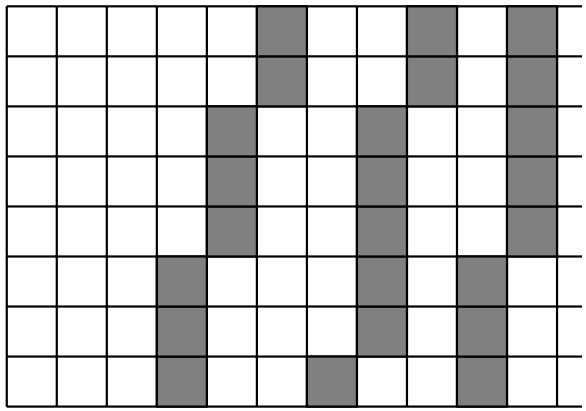
General procedure:

- Make a set of operations to be done for each partition.
- Pack these into an array such that reading is coalesced.

Scatter Array:



Reduction Array:

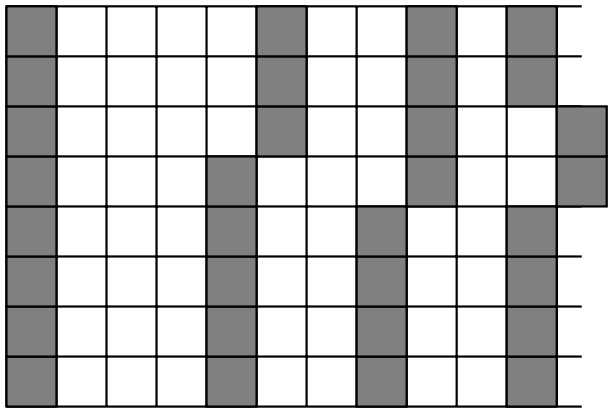


Scatter and Reduction Arrays

General procedure:

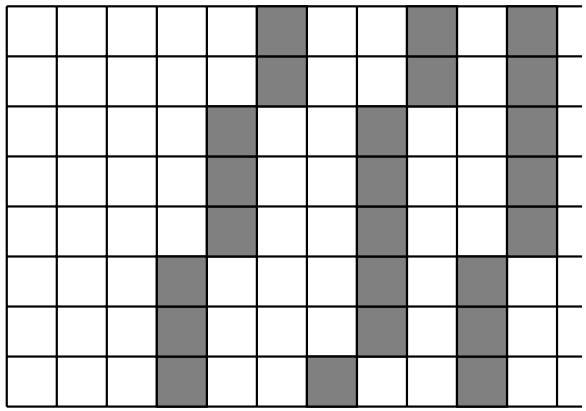
- Make a set of operations to be done for each partition.
- Pack these into an array such that reading is coalesced.

Scatter Array:



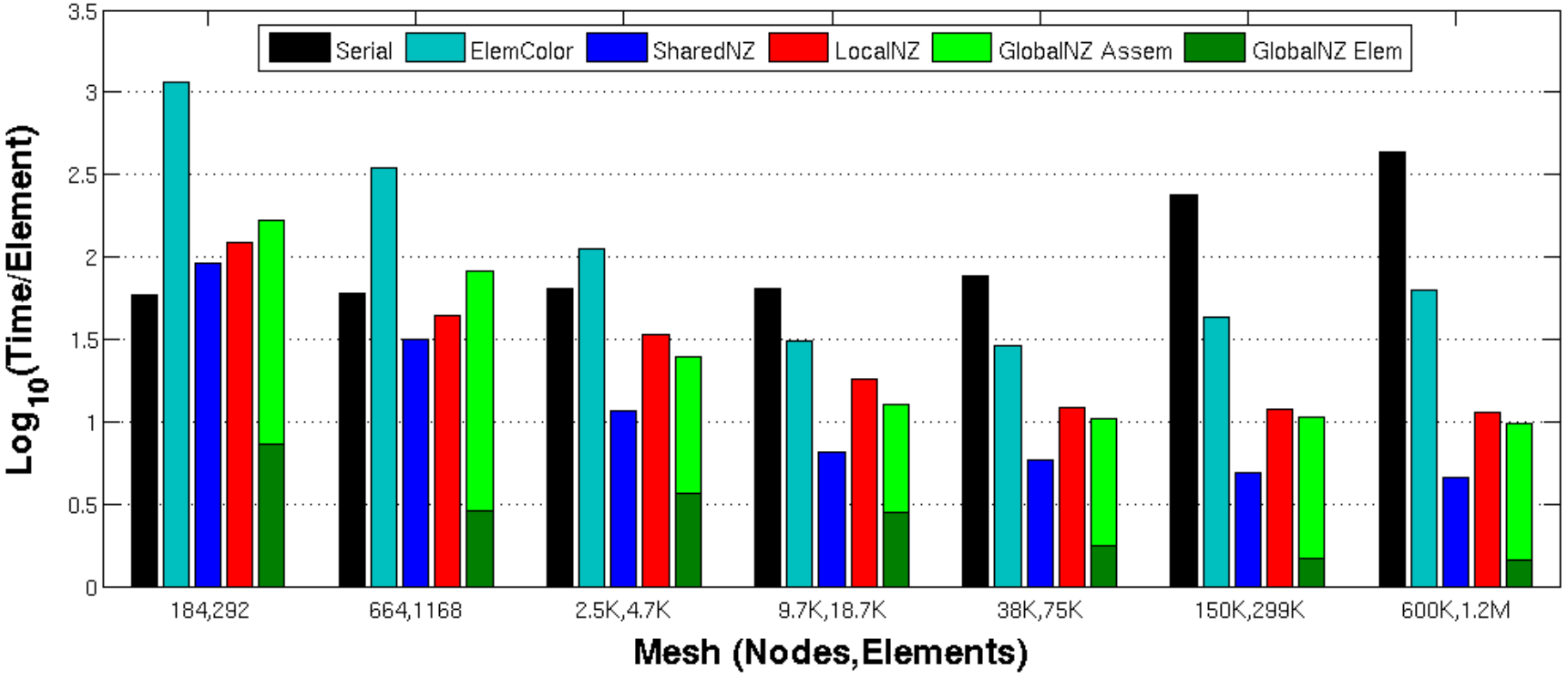
- Very fast.
- Highly adaptable.

Reduction Array:

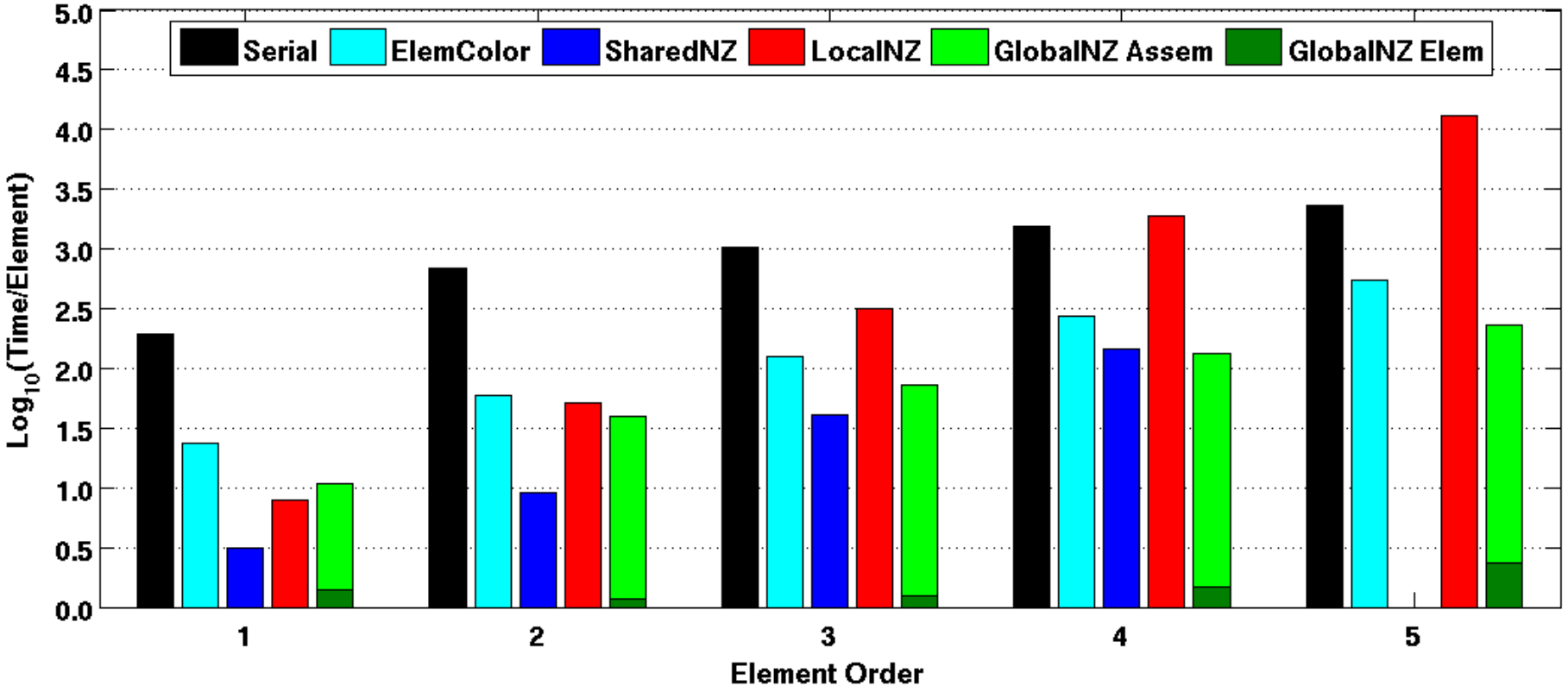


- Significant setup cost.
- Significant memory cost.

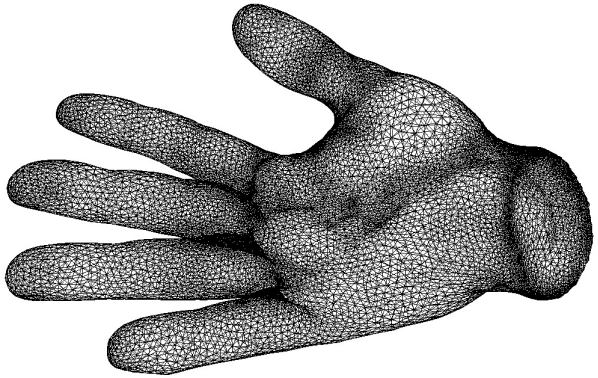
Scaling with Element Number



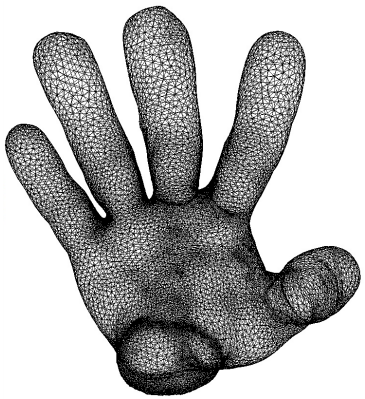
Scaling with Element Order



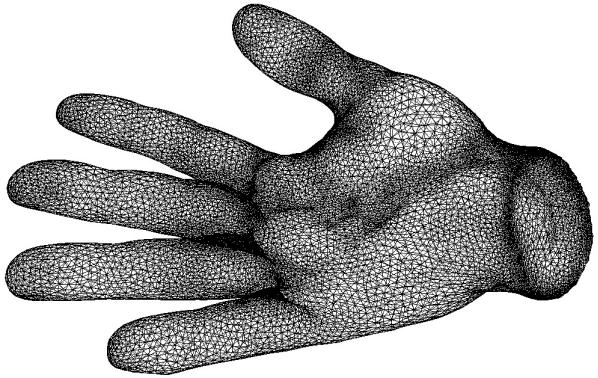
Application



GPU non-linear neoHookean model.

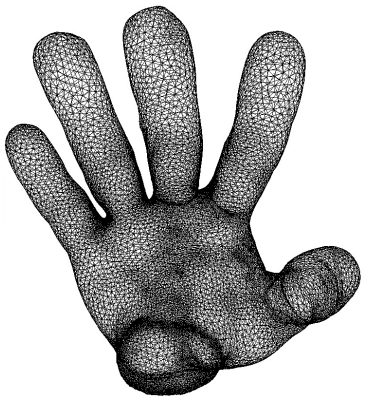


Application

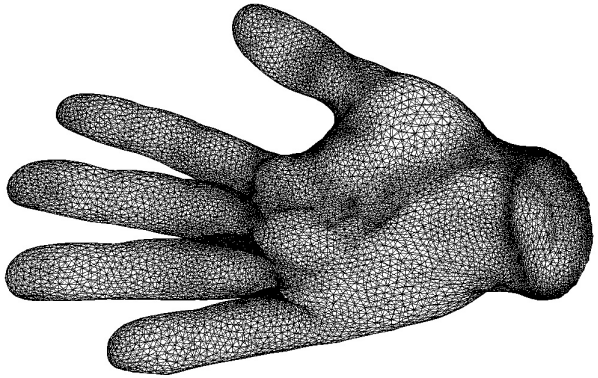


GPU non-linear neoHookean model.

- Newton-Raphson update at each step.
- Assemble, solve, update, and render at each step.

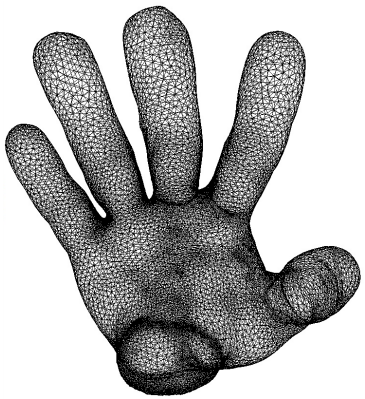


Application

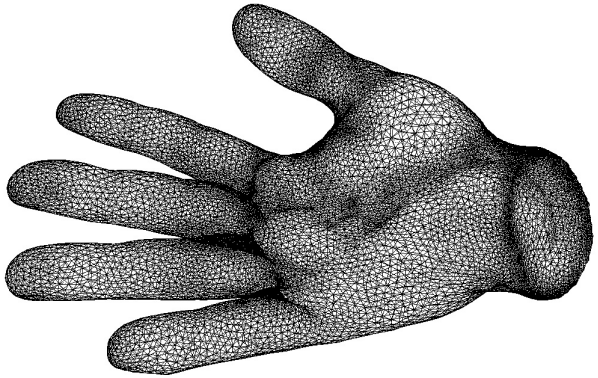


GPU non-linear neoHookean model.

- Newton-Raphson update at each step.
- Assemble, solve, update, and render at each step.
- 28,796 Nodes. 125,127 Elements.



Application

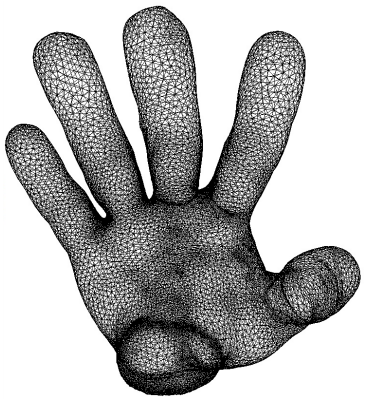


GPU non-linear neoHookean model.

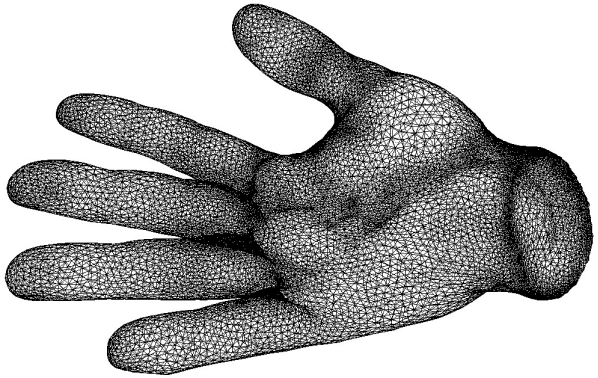
- Newton-Raphson update at each step.
- Assemble, solve, update, and render at each step.
- 28,796 Nodes. 125,127 Elements.

Preliminary Results:

- Assembly on CPU and transferring requires $\sim$0.5s (optimized).



Application

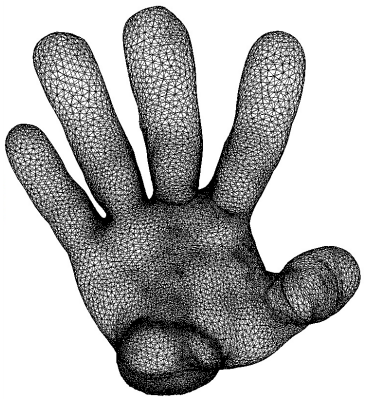


GPU non-linear neoHookean model.

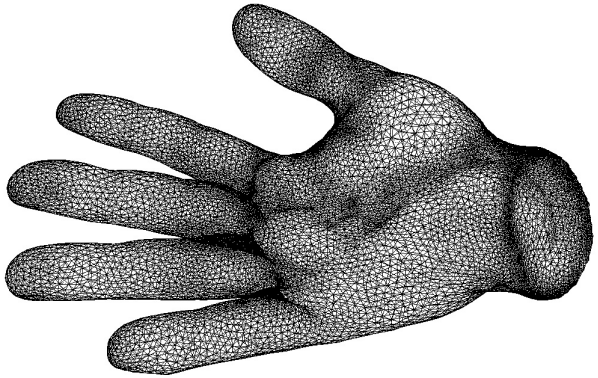
- Newton-Raphson update at each step.
- Assemble, solve, update, and render at each step.
- 28,796 Nodes. 125,127 Elements.

Preliminary Results:

- Assembly on CPU and transferring requires $\sim$0.5s (optimized).
- Assembly on GPU requires $\sim$0.01s (optimized).



Application

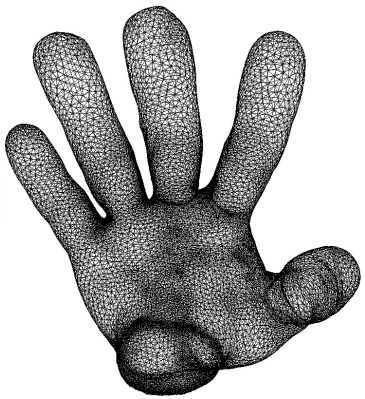


GPU non-linear neoHookean model.

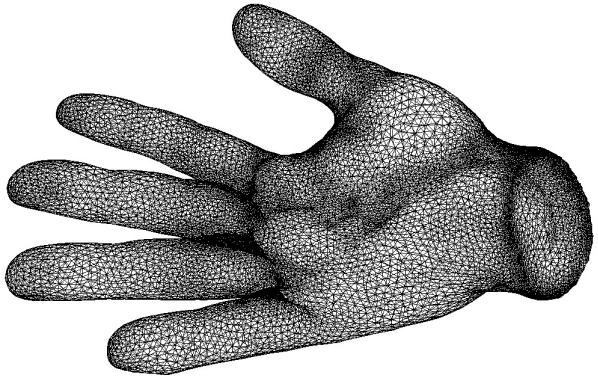
- Newton-Raphson update at each step.
- Assemble, solve, update, and render at each step.
- 28,796 Nodes. 125,127 Elements.

Preliminary Results:

- Assembly on CPU and transferring requires $\sim$0.5s (optimized).
- Assembly on GPU requires $\sim$0.01s (optimized).
- With a GPU sparse solver, currently 5fps on GTX480.



Application

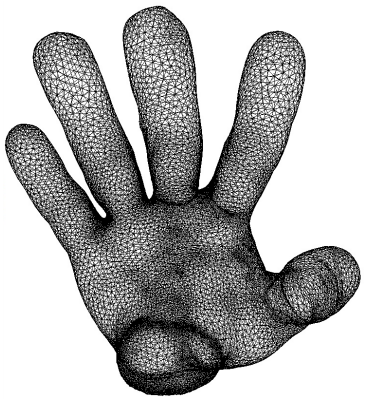


GPU non-linear neoHookean model.

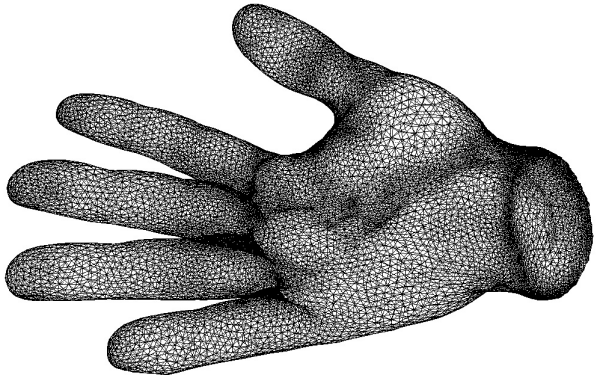
- Newton-Raphson update at each step.
- Assemble, solve, update, and render at each step.
- 28,796 Nodes. 125,127 Elements.

Preliminary Results:

- Assembly on CPU and transferring requires $\sim$0.5s (optimized).
- Assembly on GPU requires $\sim$0.01s (optimized).
- With a GPU sparse solver, currently 5fps on GTX480.
- Expect 15+fps after a few more solver optimizations.



Application

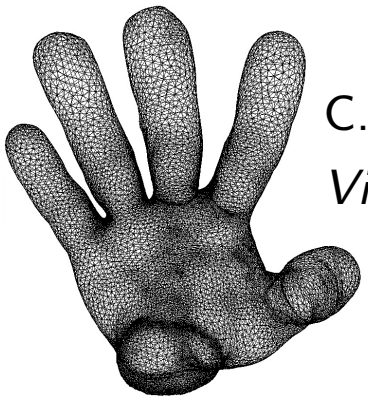


GPU non-linear neoHookean model.

- Newton-Raphson update at each step.
- Assemble, solve, update, and render at each step.
- 28,796 Nodes. 125,127 Elements.

Preliminary Results:

- Assembly on CPU and transferring requires $\sim$0.5s (optimized).
- Assembly on GPU requires $\sim$0.01s (optimized).
- With a GPU sparse solver, currently 5fps on GTX480.
- Expect 15+fps after a few more solver optimizations.



C. Cecka, A. Lew, E. Darve. *Application of Assembly, Solution, and Visualization of FEM on GPUs.* GPU Gems. Preprint.

Conclusion

C. Cecka, A. Lew, E. Darve, *Assembly of Finite Element Methods on Graphics Processors*. IJNME, 2009.

C. Cecka, A. Lew, E. Darve. *Application of Assembly, Solution, and Visualization of Finite Elements on Graphics Processors.* GPU Gems. Preprint.

Conclusion

C. Cecka, A. Lew, E. Darve, *Assembly of Finite Element Methods on Graphics Processors*. IJNME, 2009.

- Create and classify several GPU FEM assembly algorithms.

C. Cecka, A. Lew, E. Darve. *Application of Assembly, Solution, and Visualization of Finite Elements on Graphics Processors.* GPU Gems. Preprint.

Conclusion

C. Cecka, A. Lew, E. Darve, *Assembly of Finite Element Methods on Graphics Processors*. IJNME, 2009.

- Create and classify several GPU FEM assembly algorithms.
- Identification of optimizations and limitations of each algorithm.

C. Cecka, A. Lew, E. Darve. *Application of Assembly, Solution, and Visualization of Finite Elements on Graphics Processors.* GPU Gems. Preprint.

Conclusion

C. Cecka, A. Lew, E. Darve, *Assembly of Finite Element Methods on Graphics Processors*. IJNME, 2009.

- Create and classify several GPU FEM assembly algorithms.
- Identification of optimizations and limitations of each algorithm.
- Optimal method depends on the element:
  - Memory requirements of element kernels.
  - Computational requirements of element kernels.

C. Cecka, A. Lew, E. Darve. *Application of Assembly, Solution, and Visualization of Finite Elements on Graphics Processors.* GPU Gems. Preprint.

Conclusion

C. Cecka, A. Lew, E. Darve, *Assembly of Finite Element Methods on Graphics Processors*. IJNME, 2009.

- Create and classify several GPU FEM assembly algorithms.
- Identification of optimizations and limitations of each algorithm.
- Optimal method depends on the element:
  - Memory requirements of element kernels.
  - Computational requirements of element kernels.
- Precomputation algorithms and support data structures.

C. Cecka, A. Lew, E. Darve. *Application of Assembly, Solution, and Visualization of Finite Elements on Graphics Processors.* GPU Gems. Preprint.

Conclusion

C. Cecka, A. Lew, E. Darve, *Assembly of Finite Element Methods on Graphics Processors*. IJNME, 2009.

- Create and classify several GPU FEM assembly algorithms.
- Identification of optimizations and limitations of each algorithm.
- Optimal method depends on the element:
  - Memory requirements of element kernels.
  - Computational requirements of element kernels.
- Precomputation algorithms and support data structures.

C. Cecka, A. Lew, E. Darve. *Application of Assembly, Solution, and Visualization of Finite Elements on Graphics Processors.* GPU Gems. Preprint.

- Applying the methods to a high-performance FEM application.