GPU Acceleration of Equations Assembly in Finite Elements Method – Preliminary Results

Jiří Filipovič, Igor Peterlík, Jan Fousek

Faculty of Informatics, Masaryk University, Botanická 68a, Brno 602 00, Czech Republic
{fila, peterlik}@ics.muni.cz, izaak@mail.muni.cz

Abstract

The finite element method (FEM) is widely used for numerical solution of partial differential equations. Two computationally expensive tasks have to be performed in FEM – equations assembly and solution of the system of equations.
We present mapping of the equations assembly problem for StVenant-Kirchhoff material to the GPU computation model and show results of its early implementations outperforming our single core CPU implementation by factor of 15. Moreover, presented method used for equations assembly problem solution is more general and can be considered as a technique to manage complex composition of medium-grained operations with different requirements of GPU resources.

Problem mapping to GPU computation model

The continuous domain in FEM is discretized into elements of the mesh. For each element, the small 12×12 matrix is assembled and added to global system of equations. The element's matrix assembly is too complex procedure to be solved per thread because of huge memory resources requirements, but it cannot be parallelized into sufficient number of threads to utilize thread block well.
We address this issue by following steps:

- decomposition of element's matrix assembly into algebraic operations
- implementation of particular algebraic operations
- composition some particular algebraic operations that can efficiently run in one kernel exchanging data via shared memory
- calling of particular and composed algebraic operations to solve equation assembly problem exchanging data via global memory

This method allows us to reuse some code when the FEM is changed (e.g. different material is used) and efficiently compose operations with similar parallelism and sufficiently low memory resources requirements into same kernels.

Implementation

All algebraic operations are implemented to allow both launch from host code using data in global memory and launch from kernel using data in shared memory, allowing their further composition. Many operations used in the equations assembly are medium-grained – they consume too many memory resources to run one operation per thread, but they do not expose sufficient parallelism to run one operation per thread block. The 2-level parallelism within the block is used to overcome this limitation, i.e. one block solves multiple operations in parallel and each operation is solved by several threads.
Basically there are two ways how to compose operations. They can be executed in serial from host code, each operation reading and writing data from/to global memory. The implementation of this model is simpler, but its performance can be limited by global memory bandwidth. On the other hand, we can compose multiple operations into one kernel allowing data exchange through fast shared memory. Especially when operations with poor flop-to-word ratio are used, this model yields significantly better performance, but the problem of composing algebraic operations is more complex. The illustration of operations composition problem is depicted in Figure 1.
Although the implementation of particular algebraic operations require careful programming based on solid CUDA knowledge, the problem of which algebraic operations should be composed into one kernel is more complex and needs more experimenting. The major aspects that has to be carefully balanced are:

- many operations in one block reducing global memory transfers vs. reduced GPU utilization in operations using lower number of threads
- many operations in one block vs. shared memory consumption decreasing GPU utilization
- the execution order of operations allowing better shared memory reusing
- the execution order of operations optimizing threads occupancy within a block by particular operations

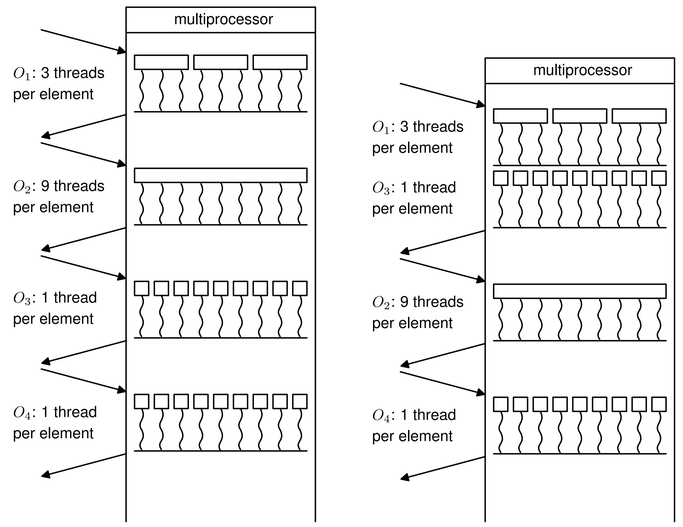

FIGURE 1: Left: all operations are called from host exchanging data in global memory; right: operations are rearranged and 2 of them is composed into one kernel.

Although the number of possibilities how to compose algebraic operations into kernels is large, from the programming point of view, the composition consists only from data reading/storing and calling of algebraic operations functions. Thus, we believe the operations composition is good target for autotuning.

Preliminary results

Two methods of the equations assembly on GPU for St.Venant-Kirchhoff material have been implemented – the first one uses the global memory transfers between all operations, the second one has a fraction of frequently executed operations composed into one kernel performing communication via shared memory.
Our implementations have been benchmarked using machine equipped with Core2 Quad Q9550 (2.83 GHz) and GeForce GTX 280. Only assembling the equations of elements without initial data copy to GPU is benchmarked, because our target is to copy initial data to the GPU once and iteratively assembly the equations and solve them, both in GPU.

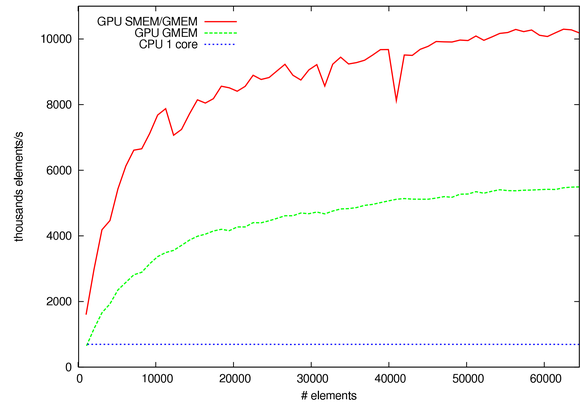

FIGURE 2: Comparison of 3 different implementations of equations assembly for StVenant-Kirchhoff material.

The CUDA implementation using only global memory for intermediate data transfers outperforms our CPU implementation running on one CPU core 8.3×. The part of GPU operations that communicate via shared memory in the second mentioned GPU implementation runs 3.3× faster yielding 1.8× overall speedup and 15× speedup over CPU implementation. In Figure 2, the comparison of CPU implementation using one core, GPU implementation using only global memory and GPU implementation with fraction of operations communicating via shared memory is depicted.

Conclusion and future work

We have been concerned in acceleration of the equations assembly problem in FEM. This problem is special because of two reasons:

- the high parallelism of GPU requires single operation multiple data approach, but the granularity of operation is too coarse to be performed by thread and too fine to be performed by block
- operations should exchange data via shared memory, but its data and parallelism requirements vary significantly yielding underutilization of GPU when all operations are composed into one kernel

We have implemented the equations assembly for St.Venant-Kirchhoff material on GPU addressing these issues obtaining quite promising results. Moreover, the discussed problem is more general and the same approach can be applied to wide class of similar problems.
The future work will be focused on two areas. First, we will try to increase the portion of computations transferring data via shared memory and optimize some algebraic operations code in current St.Venant-Kirchhoff implementation. Moreover, we plan to implement the equations assembly for another material to prove the general usability of our concept. Second, we will target the automatic search for optimal composition of algebraic operations communicating via shared memory.