

Automated Finite Element Computations in the FEniCS Framework using GPUs

Florian Rathgeber (f.rathgeber10@imperial.ac.uk)

Advanced Modelling and Computation Group (AMCG)
Department of Earth Science & Engineering
Imperial College London

2nd UK GPU Computing Conference
December 13th 2010

Outline

Introduction: The FEniCS Project

Parallel Finite Element Assembly

Avoiding Global Assembly

Benchmarks and Profiling Results

Automating Finite Element Assembly

Conclusions

Outline

Introduction: The FEniCS Project

Parallel Finite Element Assembly

Avoiding Global Assembly

Benchmarks and Profiling Results

Automating Finite Element Assembly

Conclusions

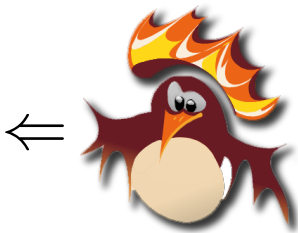
The FEniCS Project

<http://www.fenicsproject.org>

Automation of Computational Mathematical Modeling (ACMM)



DOLFIN



FEniCS



UNICORN

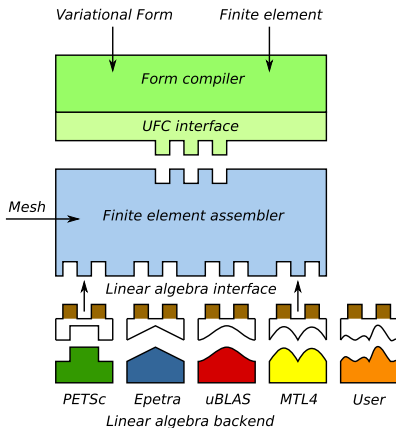
Automating finite element assembly



UFL, FFC, and UFC:

the link between
variational form in **mathematical notation**
and
assembly in DOLFIN

Modular DOLFIN library architecture



source: Logg and Wells - DOLFIN: Automated finite element computing (2010)



The missing link



source: NVIDIA

The missing link



source: NVIDIA

Goals

- ▶ GPU backend for DOLFIN
- ▶ Performance gain
- ▶ No sacrifice in degree of automation

Outline

Introduction: The FEniCS Project

Parallel Finite Element Assembly

The Finite Element Method (FEM)

Finite Element Assembly

Finite Element Assembly on the GPU

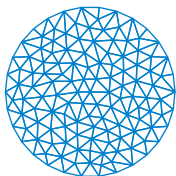
A Question of Data Layout

Avoiding Global Assembly

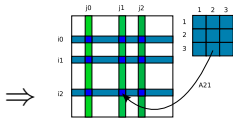
Benchmarks and Profiling Results

Automating Finite Element Assembly

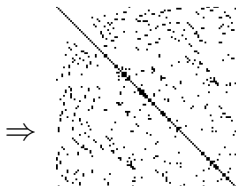
The Finite Element Method (FEM)



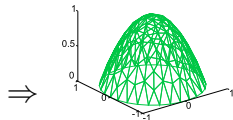
FEM
Triangulation



FEM Assembly



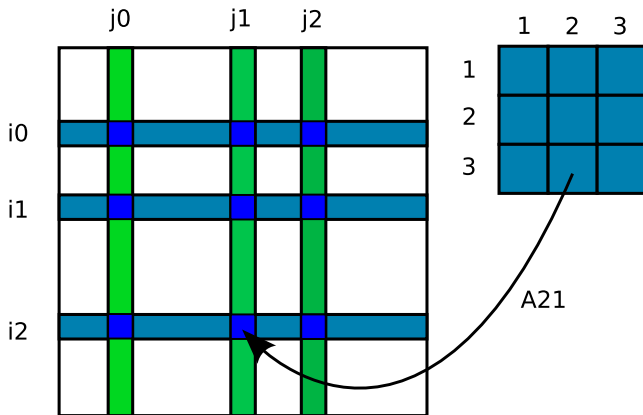
Sparse matrix



FEM Solution

source: [Wikimedia Commons](#)

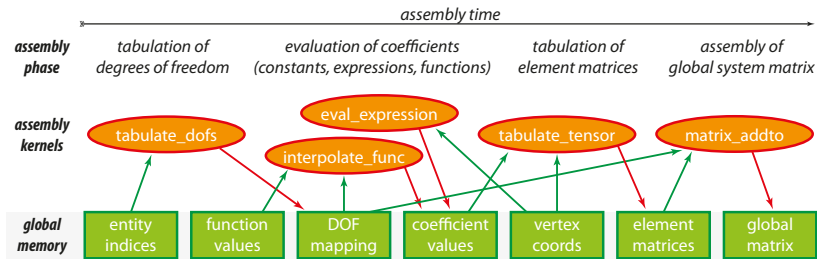
Finite Element Assembly



Assembling a 3×3 element matrix into the global system matrix

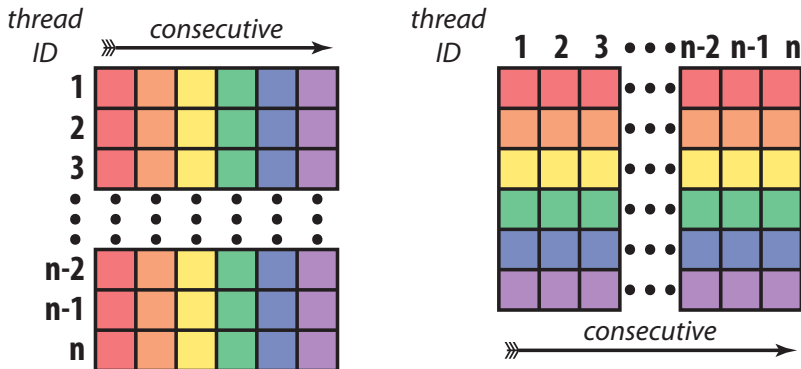
source: Alnaes et al. - Unified Framework for Finite Element Assembly (2009)

Finite element assembly on the GPU



A data flow diagram showing input and output of kernels for the different assembly stages and how data is streamed between them

A question of data layout



GPU data layout to achieve coalesced transfers from and to global device memory (right) compared to a corresponding layout in CPU memory (left)

Outline

Introduction: The FEniCS Project

Parallel Finite Element Assembly

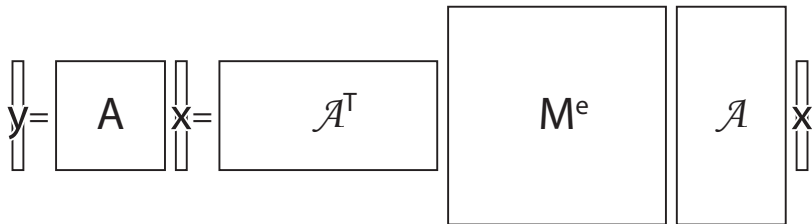
Avoiding Global Assembly

Benchmarks and Profiling Results

Automating Finite Element Assembly

Conclusions

Sparse matrix-vector product without assembling the matrix



Linear algebra representation of the sparse matrix-vector multiplication:

- ▶ M^e block-diagonal matrix of element matrices
- ▶ A sparse matrix corresponding to local-to-global mapping
rows: all local degrees of freedom
columns: global degrees of freedom

Outline

Introduction: The FEniCS Project

Parallel Finite Element Assembly

Avoiding Global Assembly

Benchmarks and Profiling Results

Profiling GPU Assembly

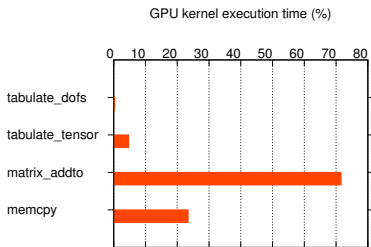
Assembly Benchmarks

Assembly and Solve Benchmarks

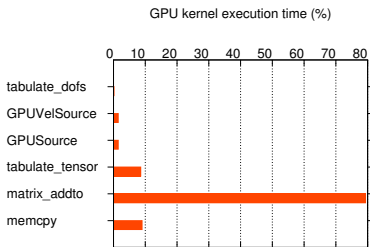
Interpreting Speedup Figures

Automating Finite Element Assembly

Profiling GPU assembly

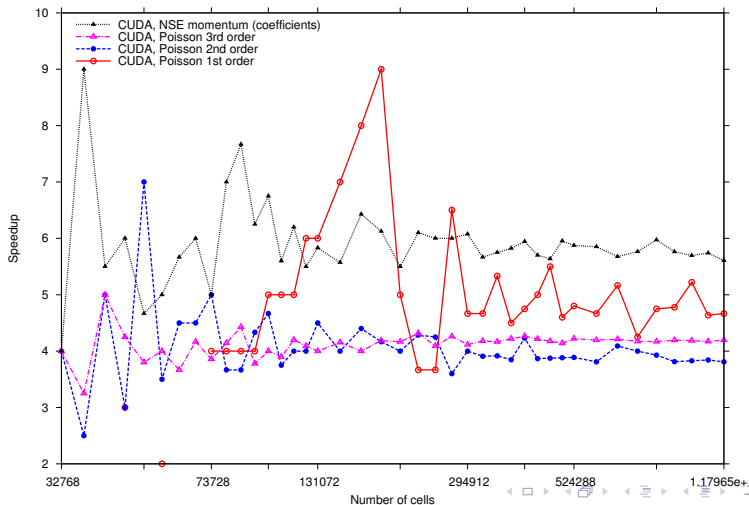


Poisson's equation (P1)

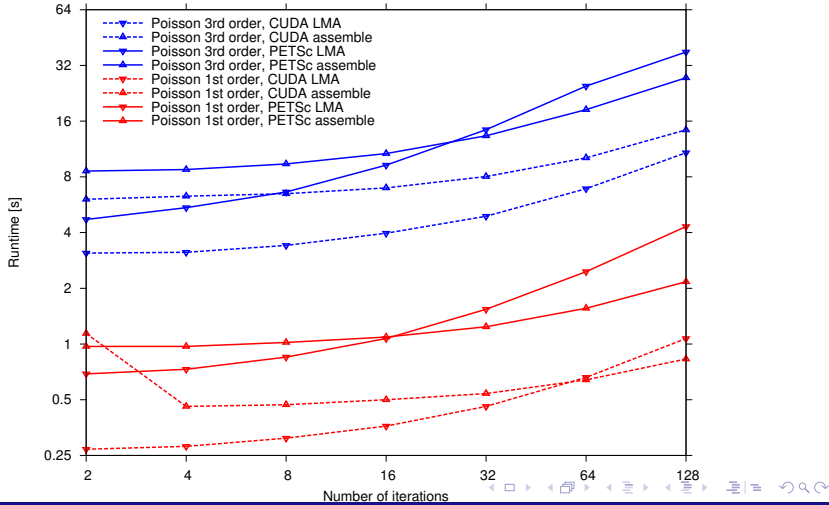


Stabilized Navier-Stokes momentum term (P1)

Speedup reassembly



Total runtime assembly and solve



Interpreting speedup figures

- ▶ Computations in **double precision**
 - factor 8 penalty for floating point computation, factor 2 for memory transfer
 - **78 GFlop/s** peak performance, on par with Intel Nehalem 8-core CPU
 - **Fermi** architecture improves double precision penalty to factor 2
- ▶ High speedup figures
 - often: mediocre CPU against highly optimized GPU implementations
 - here: highly optimized linear algebra (PETSc) and tensor contraction (generated by FFC) against generated GPU kernels without any hand-optimization
- ▶ Performance comparisons **include** data transfer between host and device
 - true one-to-one comparisons including all transfer and setup times

Outline

Introduction: The FEniCS Project

Parallel Finite Element Assembly

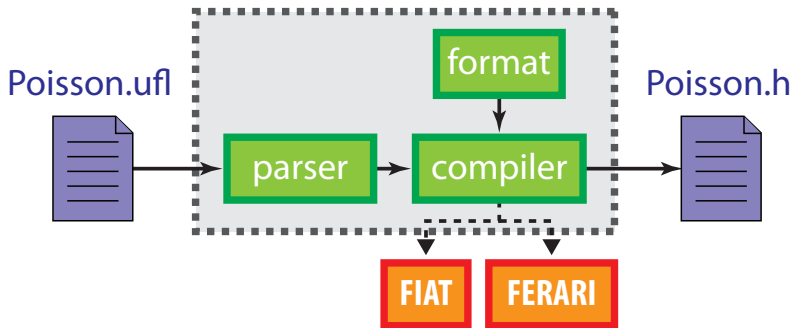
Avoiding Global Assembly

Benchmarks and Profiling Results

Automating Finite Element Assembly

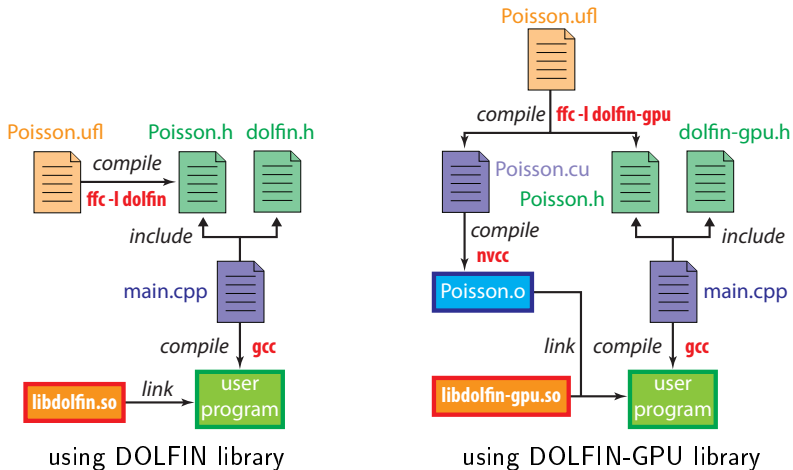
Conclusions

FFC code generation



Compilation of a variational form given as a UFL file to a UFC compliant header using FFC

Integration of generated code with a user program



Outline

Introduction: The FEniCS Project

Parallel Finite Element Assembly

Avoiding Global Assembly

Benchmarks and Profiling Results

Automating Finite Element Assembly

Conclusions

Conclusions

Future work

Conclusions

Code generation for Finite Element Computations on GPUs

1. Finite element computations in the FEniCS framework on the GPU, showing a speedup of up to 9 over PETSc on single CPU
2. Automated assembly from a variational form in mathematical notation using the FEniCS Form Compiler FFC

Conclusions

Code generation for Finite Element Computations on GPUs

1. Finite element computations in the FEniCS framework on the GPU, showing a speedup of up to 9 over PETSc on single CPU
2. Automated assembly from a variational form in mathematical notation using the FEniCS Form Compiler FFC

Current limitations:

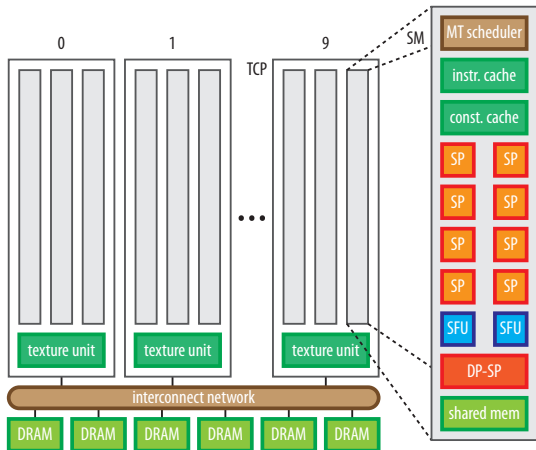
1. Only cell integral assembly
2. No support for **strongly enforced** boundary conditions
3. Only **conjugate gradient** solver → **symmetric positive definite** matrices
4. Bugs in **nvcc** compiler prohibit complex forms

Future work

- ▶ Integrate with **OP2** and **Fluidity**
- ▶ Optimise generation of CUDA kernels
- ▶ **MPI**-parallelisation (distributed memory)
- ▶ Fully implement UFC
- ▶ Port to **OpenCL**

Thank You!

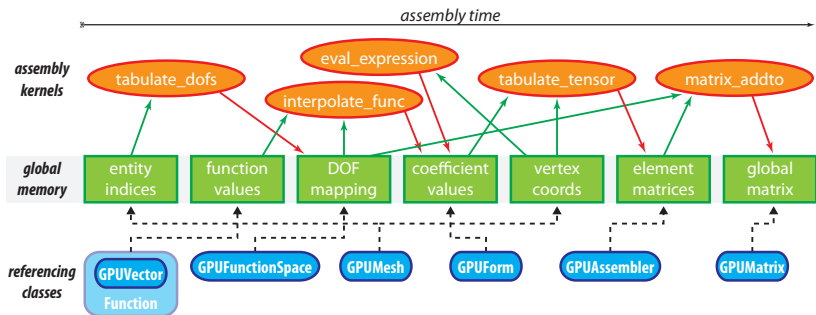
The NVIDIA Tesla GPU architecture



NVIDIA Tesla C1060

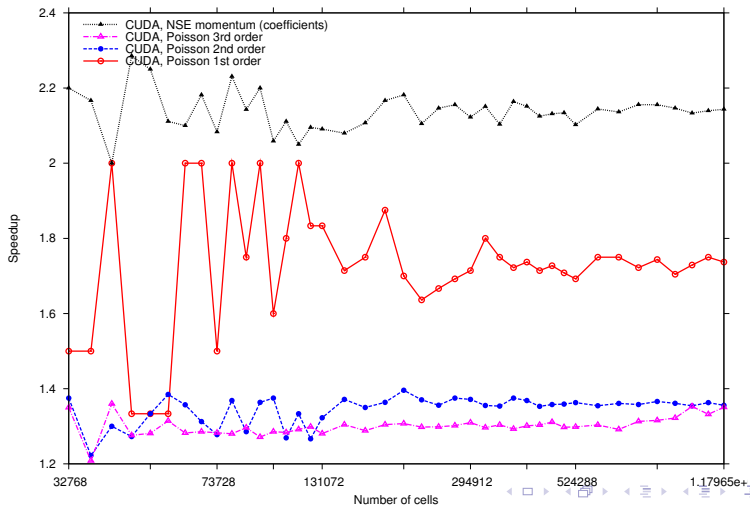
- ▶ 30 multiprocessors (MP)
 - ▶ 8 stream processors (SPs)
 - ▶ 16384 registers
 - ▶ 16 KB shared memory
- ▶ 1.30 GHz shader clock
- ▶ 4096 MB global memory (frame buffer)
- ▶ 933.0 GFlop/s arithmetic peak
- ▶ 102.4 GB/s memory bandwidth

Finite element assembly on the GPU



A data flow diagram showing input and output of the assembly kernels, how data is streamed between them, and where it is stored

Speedup assembly



Speedup assembly and solve

