Automated cross element vectorization in Firedrake

Tianjiao Sun, Imperial College London, tianjiao.sun14@imperial.ac.uk,
Lawrence Mitchell, Imperial College London, lawrence.mitchell@imperial.ac.uk,
David A. Ham, Imperial College London, david.ham@imperial.ac.uk,
Paul H. J. Kelly, Imperial College London, p.kelly@imperial.ac.uk,

Keywords: *vectorization, code generation, loop transformation*

Firedrake [1] is a domain-specific language embedded in Python for numerical solution of partial differential equations (PDEs) using finite element method. Firedrake provides the users a high level interface to express the problems in a high level mathematical language while generating efficient low level code. The internal intermediate representations in this code generation pipeline offer performance optimization opportunities at different levels of abstractions. In this work, we present one of the latest developments in Firedrake which enables automated vectorization across elements on unstructured meshes for typical finite element assembly kernels, so as to address the problem of better performance and hardware utilization on SIMD architecture.

Modern CPUs increasingly rely on SIMD instructions to achieve higher throughput and better energy efficiency. It is therefore important to vectorize sequences of computations in order to sufficiently utilize the hardware today and in the future. This requires the instructions to operate on groups of data that are multiples of the width of the vector lane (e.g. 4 doubles, 8 floats on AVX2 instructions). Finite element computations usually require the assembly of vectors and matrices which represents differential forms on the domain. This process consists of applying a local assembly kernel to each element, and increment the global data structure with the local contribution. Typical local assembly kernels suffer from issues that often preclude efficient vectorization. These include complicated loop structure, poor data access patterns, and loop trip counts that are not multiples of the vector width. General purpose compilers often perform poorly in generating efficient, vectorized code for such kernels.

In this work, we present a generic and portable solution in Firedrake based on cross element vectorization. Although vector-expanding the assembly kernel is conceptually clear, it is only enabled by applying a chain of complicated loop transformations. Loo.py [2] is a Python package which defines array-style computations in integer polyhedral model, and supports a rich family of transformations that operate on this model. In Firedrake, We adapt the form compiler, TSFC [3], to generate Loo.py kernels for local assembly operations, and systematically generate data gathering and scattering operations across the mesh in PyOP2 [4]. Firedrake drives loop transformations using Loo.py from this high level interface to generate efficient code vectorized across a group of elements which fully utilizes the vector lane. This toolchain automates the tedious and error-prone process of data layout transformation, loop unrolling and loop interchange, while being transparent to the users.

We will present experimental results performed on multiple kernels and meshes. We achieve speed ups consistent with the vector architecture available compared to baseline which vectorizes inside the local assembly kernels. The global assembly computations reach tens of percent of hardware peak arithmetic performance.

References

[1] F. Rathgeber, D.A. Ham, L. Mitchell, M. Lange, F. Luporini, A.T.T. Mcrae, G. Bercea, G.R. Markall, P.H.J. Kelly: Firedrake: automating the finite element method by composing abstractions, *ACM Trans. Math. Softw.*, 43(3):24:1-24:27, 2016.

[2] A. Klöckner: Loo.py: transformation-based code-generation for GPUs and CPUs, *Proceedings of ARRAY '14: ACM SIGPLAN Workshop*, 2014

[3] M. Homolya, L. Mitchell, F. Luporini, D.A. Ham: TSFC: a structure-preserving form compiler, 2017

[4] F. Rathgeber, G.R. Markall, L. Mitchell, N. Loriant, D.A. Ham, C. Bertolli, P.H.J. Kelly: PyOP2: A high-level framework for performance-portable simulations on unstrctured meshes, *In Proceedings of 2012 SC Companion: High Performance Computing, Networking, Storage, and Analysis*, 1116-1123.