LLVM runtime compilation for PyOP2 and Firedrake

Florian Rathgeber

April 12, 2014

PyOP2[1] and Firedrake[2] are part of a high-level scientific computing tool chain making use of runtime code generation and compilation to achieve high performance and computational efficiency. This project will make this tool chain robustly scalable to large HPC systems by replacing the existing compilation architecture with an LLVM code generation backend.

PyOP2 is a high-level domain-specific language (DSL) embedded in Python for the parallel execution of computational kernels on unstructured meshes, generating problem-specific code for a range of hardware architectures, targeting multi-core CPUs, GPUs, accelerators and distributed parallel computations with MPI. Backend-specific code tailored to each specific problem is generated, just-in-time compiled and efficiently scheduled for parallel execution at runtime.

Firedrake is a performance-portable framework for the automated solution of partial differential equations using the finite element method (FEM). Scientists can describe problems symbolically at a high level of abstraction very similar to their mathematical model. Assembly operations are transformed into local computations over the mesh and efficiently executed by PyOP2, which is used as the performance-portable parallel execution layer.

LLVM is a modular compiler infrastructure with a source- and target-independent optimizer built around a well specified intermediate representation known as the *LLVM IR*. It includes a Just-In-Time (JIT) code generation system with in-memory compilation for various CPU target architectures. Python bindings are provided by LLVMPY.

Runtime code generation in PyOP2 and Firedrake

PyOP2's code generation technology for CPU targets involves generating a C file which is compiled into a loadable Python module at runtime, which is then `import`ed into the running process. The first step involves invoking a C compiler and requires forking the interpreter process, which presents an issue already for running moderately large problems on a workstation. The second uses the normal Python `import` mechanism, eventually calling `dlopen` to load the shared library into the running executable. This approach is unlikely to be scalable on large HPC systems, since `import`ing Python modules requires all processes to `stat` a shared filesystem, simultaneously searching for the location of the module. Python modules compiled to shared objects are `dlopen`ed by all processes, again requiring a search of the filesystem.

[1] http://op2.github.io/PyOP2
[2] http://firedrakeproject.org

Objectives

The core objective of this project is to replace the existing PyOP2 code generation approach for CPU targets with in-memory compilation using LLVM. The current C code generation and compilation stage is replaced with a backend that builds an LLVM IR directly, which will be compiled to machine code in-memory. As a result, at no point will the runtime code generation ever have to touch the filesystem. This increases both the scalability on current systems and reduces porting efforts to future HPC systems.

Work plan

1. Implement single-threaded code generation backend using LLVM / llvmpy
   - Familiarisation with LLVM and llvmpy software
   - Replacement of existing single-threaded backend with LLVM-based backend
   - Benchmarking of single core performance
   - Benchmarking of scalability
2. Implement shared-memory code generation backend
   - Evaluate suitability of LLVM OpenMP runtime support against the potential use of pthreads
   - Replacement of existing shared-memory parallel backend with LLVM-based backend
   - Benchmarking of single node performance
   - Benchmarking of scalability
3. Profiling and performance tweaking, large scale run
   - Ongoing performance profiling and optimisation
   - Perform large scale simulations to highlight performance gains
4. Dissemination, documentation and reporting
   - Write up and disseminate results
   - Document new code generation strategies

Alignment with PRISM strategy

- Exchange, collaboration and sharing of best practices with other code generation based PRISM projects (e.g. PyFR)
- Regular presentation of progress and findings at the researcher group meetings to keep the wider group informed

Impact

A robust and scalable runtime code generation approach will not only increase the user base of Firedrake on current HPC platforms, it will also contribute significantly to ease the long-term maintenance burden of porting to new platforms and thereby greatly benefit the longer term sustainability of PyOP2 and Firedrake.