A Tiny Python Based High Performance Backend Code Generator

Richard Veras and J. "Ram" Ramanujam

Louisiana State University, USA
rveras@lsu.edu, jxr@ece.lsu.edu

Abstract. Many performance critical domains such as Dense Linear Algebra, Signal Processing, Structured Mesh Computation and Graph Analytics provide efficient libraries by building their operations in terms of high performance computational kernels. It is this performance abstraction that allows these codes to be portable across a variety of hardware, while insulating all but a handful of developers from the low-level details needed for implementing these kernels. However, developing these kernels requires a monumental amount of effort, domain knowledge and hardware knowledge because implementing a single kernel for one hardware target involves the following: First, finding an efficient loop based algorithm for the kernel. Second, realizing that kernel in terms of a mix of Single Instruction Multiple Data (SIMD) instructions that can achieve high throughput within the context of a loop. Third, scheduling those instructions in order to maximize Instruction Level Parallelism. Fourth, emitting the result code in a representation that preserve the instruction selection and schedule. For every target platform this process is repeated. The first step for kernel generation typically is problem specific and relies heavily on the domain expert's knowledge. The last three processes are common across many domains and could be automated. While one could use a conventional compiler for scheduling and code emission, in practice this approach typically falls short of what is achieved by hand implementation in assembly code or produced by an in-house backend code generator. Neither of these approaches are optimal, as programming in assembly involves substantial effort and expertise in order to produce unportable code, and a backend code generator requires maintaining a large infrastructure base for every hardware target. In this work we present the High Performance Backend Code Generator (HPBCG), a framework for generating scheduled portable SIMD code for computational kernels. This framework takes a user's kernel implemented in a Python based representation and emits portable scheduled SIMD code. The scheduler is based on the OASIC model and produces a software pipeline implementation of the kernel code in C. The resulting C code, which we call Wrapped Schedulable (WS) Macro Instructions, maintains the instruction schedule in any ANSI C compiler. Further these WS Macro Instructions can target any Instruction Set Architecture (ISA) through the inclusion of an architecture specific header file without modification of the generated kernel code. In ongoing work, we are investigating extending this concept of architecture specific header files to Cuda, OpenCL and Intel ICPC representations which would allow us to generate kernel code for a wide variety of platforms from the same backend code generator. We will demonstrate how our framework works in the context of kernel code generators. In addition, we will show the effectiveness of HPBCG by evaluating the performance of the generated kernel code for a variety Linear Algebra, Graph Analytic and Machine Learning operations. In particular our performance for Matrix Multiplication kernels approach the performance of hand written assembly kernels.