Adaptive Implementation Selection in the SkePU Skeleton Programming Library

Usman Dastgeer, Lu Li, and Christoph Kessler
IDA, Linköping University, 58183 Linköping, Sweden
{usman.dastgeer,lu.li,christoph.kessler}@liu.se

Abstract. In earlier work, we have developed the SkePU skeleton programming library for modern multicore systems equipped with one or more programmable GPUs. The library internally provides four types of implementations (implementation variants) for each skeleton: serial C++, OpenMP, CUDA and OpenCL targeting either CPU or GPU execution respectively. Deciding which implementation would run faster for a given skeleton call depends upon the computation, problem size(s), system architecture and data locality.

In this paper, we present our work on automatic selection between these implementation variants by an offline machine learning method which generates a compact decision tree with low training overhead. The proposed selection mechanism is flexible yet high-level allowing a skeleton programmer to control different training choices at a higher abstraction level. We have evaluated our optimization strategy with 9 applications/kernels ported to our skeleton library and achieve on average more than 94% (90%) accuracy with just 0.53% (0.58%) training space exploration on two systems. Moreover, we discuss one application scenario where local optimization considering a single skeleton call can prove sub-optimal, and propose a heuristic for bulk implementation selection considering more than one skeleton call to address such application scenarios.

Keywords: Skeleton programming, GPU programming, implementation selection, adaptive offline learning, automated performance tuning.

1 Introduction

The need for power efficient computing has lead to heterogeneity and parallelism in today’s computing systems. Heterogeneous systems such as GPU-based systems with disjoint memory address space already became part of mainstream computing. There exist various programming models (CUDA, OpenCL, OpenMP etc.) to program different devices present in these systems and, with GPUs becoming more general purpose every day, more and more computations can be performed on either of the CPU or GPU devices present in these systems.

Known for their performance potential, these systems expose programming difficulty as the programmer often needs to program in different programming models to do the same computation on different devices present in the system which limits code-portability. Furthermore, sustaining performance when porting an application between
different GPU devices (*performance portability*) is a non-trivial task. The skeleton programming approach can provide a viable solution for computations that can be expressed in the form of skeletons, where *skeletons* \(^1, 2\) are pre-defined generic components derived from higher-order functions that can be parameterized with sequential problem-specific code. A skeleton program looks like a sequential program where a skeleton computation can internally exploit parallelism and leverage other architectural features transparently by e.g. keeping different implementations for a single skeleton targeting different architectural features of the system. Map/Zip and Farm are examples of data and task-parallel skeletons respectively.

We have developed the SkePU skeleton programming library for GPU-based systems in our earlier work \[^3\]. The library targets single-node GPU-based systems and provide code portability for skeleton programs by providing sequential C++, OpenMP, CUDA and OpenCL implementations for each of its skeleton. In this paper, we present an adaptive offline machine learning method to tune implementation selection in the SkePU library automatically. The proposed technique is implemented inside the SkePU library allowing automatic implementation selection on a given GPU-based system, for any skeleton program written using the library. To the best of our knowledge, this makes SkePU the first skeleton library for GPU-based systems that provides general-purpose, automatic implementation selection mechanism for calls to its skeleton.

![Diagram](image.png)

**Fig. 1.** Six data-parallel skeletons, here shown for vector operands: Map applies a user-defined function element-wise to input vectors. Reduce accumulates an associative binary user function over all input vector elements to produce a scalar output. MapReduce combines Map and Reduce in one step. MapArray is similar to map but all elements from the 1st operand are accessible. MapOverlap is similar to Map where elements within a (user-defined) neighbourhood are also accessible in the user function. Scan is a generic prefix-sums operation.