RACECARR: A Heuristic for Automatic Function Specialization on Multi-core Heterogeneous Systems

John R. Wernsing, Dr. Greg Stitt
Department of Electrical & Computer Engineering
University of Florida
Gainesville, FL
wernsing@ufl.edu, gstitt@ece.ufl.edu

Abstract
High-performance computing systems increasingly combine multi-core processors and heterogeneous resources such as graphics-processing units and field-programmable gate arrays. However, significant application design complexity for such systems has often led to untapped performance potential. Application designers targeting such systems currently must determine how to parallelize computation, create device-specialized implementations for each heterogeneous resource, and determine how to partition work for each resource. In this paper, we present the RACECARR heuristic to automate the optimization of applications for multi-core heterogeneous systems by automatically exploring implementation alternatives that include different algorithms, parallelization strategies, and work distributions. Experimental results show RACECARR-specialized implementations achieve speedups up to 117x and average 11x compared to a single CPU thread when parallelizing computation across multiple cores, graphics-processing units, and field-programmable gate arrays.

Categories and Subject Descriptors D.2.2 [Software Engineering]: Design Tools and Techniques – computer-aided software engineering (CASE)

General Terms Algorithms, Design, Measurement, Performance

Keywords elastic computing; heterogeneous; optimization; RACECARR; speedup

1. Introduction
Over the past decade, computing architectures have started on a clear trend towards increased parallelism with multi-core processors. More recently, this trend has focused on increased heterogeneity, with devices such as graphics-processing units (GPUs) and field-programmable gate arrays (FPGAs) commonly being used to improve application performance for various scientific computing domains [1][3][5].

Although multi-core heterogeneous systems have significant high-performance computing potential, the increased application design complexity commonly prevents designers from reaching this potential. In extreme cases, such as for FPGAs, this complexity has prevented designers without low-level device expertise from even using the devices [4]. Such complexity is primarily caused by three challenges: 1) creating parallel implementations to exploit multiple cores or devices, 2) creating device-specialized implementations, often requiring different algorithms, optimizations, and programming languages, and 3) partitioning and load balancing work across numerous resources.

Although in the ideal case, compilers would automatically solve these challenges by optimizing a given function to utilize all of a system’s resources, decades of studies have been unable to achieve this goal. Compilers are effective at optimizing code for an individual resource, but generally do not consider optimizations across multiple, heterogeneous resources. Even for compilers/tools that do consider multiple resources, these tools cannot transform code to use different algorithms to more effectively exploit the features of a heterogeneous device [6].

We claim that the current inability of compilers and operating systems to effectively optimize applications for multi-core heterogeneous systems is caused by a fundamental problem: no single implementation of a function is optimal across all different devices, resource amounts, and input parameters. For example, a sorting function running on a microprocessor would likely use a quick-sort algorithm, whereas on an FPGA, a bitonic-sort algorithm would be more efficient. Furthermore, implementation efficiency extends beyond just algorithmic choices and also requires considering the input parameters, numbers of devices/cores, work partitioning, etc.

To address these issues, we introduce the RACECARR heuristic for automatic function specialization on multi-core, heterogeneous systems. Function specialization is traditionally a compiler optimization that creates more efficient custom (i.e., specialized) implementations of a function for known function invocations. For multi-core, heterogeneous systems, we extend function specialization to explore algorithmic and implementation alternatives, different parallelization strategies, and different work partitionings, which we refer to as multi-core, heterogeneous function specialization (MHFS).

Given a knowledge base of implementation alternatives and parallelization strategies, RACECARR operates by first analysing the performance of each implementation. Based on the implementation performances, RACECARR then determines which implementations to select in different conditions and how to efficiently partition work to utilize parallel resources. By recursively applying this analysis, RACECARR considers nested levels of work partitioning resulting in implementations that utilize up to all of the resources on a system. As the results demonstrate, RACECARR-specialized implementations can utilize multiple microprocessors, GPUs, and FPGAs effectively and achieve speedups up to 117x compared to a single CPU thread.

We envision several usage scenarios for RACECARR. First, compilers could use the heuristic for multi-core heterogeneous
systems, or even for individual devices where specialization using different algorithms for different input parameters would be beneficial. Furthermore, runtime optimizations tools (e.g. [6]) could use the heuristic to make dynamic optimization decisions. The knowledge base required by the heuristic could be realized in function libraries by including multiple implementations provided by device/system vendors, domain experts, or open-source efforts.

2. Experimental Results

To assess RACECAR, we selected eleven functions and created a total of thirty-three alternate heterogeneous implementations of those functions to use as an input for the heuristic. We evaluated RACECAR on four diverse systems, three of which are heterogeneous. The first system, referred to as Gamma, consists of a 2.8 GHz quad-core Intel Xeon W3520 CPU, an Altera Stratix-III L340 FPGA, and two Nvidia GTX-295 GPUs. The second system, referred to as Marvel, consists of eight dual-core 2.4 GHz AMD Opteron 880 CPUs. The third system, referred to as Novo-G, is a node of the Novo-G supercomputer [2] and consists of a quad-core Intel Xeon E5520 CPU and four Altera Stratix-III E260 FPGAs. The fourth system, referred to as Delta, consists of a 3.2 GHz Intel Xeon CPU and a Xilinx Virtex IV LX100 FPGA.

Figure 1 illustrates the speedup achieved by the RACECAR-specialized implementations for the eleven functions. All speedup numbers are relative to the same function executing on a single-thread on the same system. 2DConv is a two-dimensional discrete convolution function. CConv is a discrete circular convolution function. Conv is a discrete convolution function. FW performs the Floyd-Warshall algorithm. Inner is an inner-product function. Mean applies a mean filter to an image. MM is a matrix multiply function. Optical performs an optical flow algorithm on a series of images. Prewitt applies the Prewitt edge filter to an image. SAD performs a sum-of-absolute differences image retrieval algorithm. Sort is a sorting function.

As illustrated in Figure 1, all functions achieved speedup relative to their single-threaded versions. The large amount of variance between different functions is mostly due to the varying efficiency of the implementations provided as an input to RACECAR. For the Conv and SAD functions, we provided FPGA and GPU implementations which were extremely efficient on the supporting systems. While ideally we would have included heterogeneous implementations for all functions, the lengthy process of creating the implementations limited the analysis. The large amount of variance between different systems is due to the different processing resources available on each system. Only the Gamma system includes a GPU and only the Delta, Marvel, and Novo-G systems include FPGAs. However, the Novo-G system includes four FPGAs, while the Gamma system only includes one, and the Delta system’s FPGA is on a relatively slow bus.

3. Conclusions

In this paper, we introduce the RACECAR heuristic to create specialized function implementations for multi-core, heterogeneous systems. From a set of provided implementations, the heuristic evaluates different implementation alternatives and builds data structures that may inform compilers and runtime optimization tools of how to efficiently execute the function for any invocation.

We evaluated the performance of RACECAR-specialized implementations on four diverse systems with various combinations of microprocessors, GPUs, and FPGAs, using a set of eleven functions and thirty-three implementations. For these experiments, RACECAR achieved speedups up to 117x with an average speedup of 11x compared to a single CPU thread.

Figure 1. Speedup of RACECAR-specialized implementations for eleven functions relative to single-threaded implementations.

Acknowledgments

This work was supported by the National Science Foundation, grant CNS-0914474.

References