R-Stream: Enabling Efficient Development of Portable, High-Performance, Parallel Applications

Peter Mattson, Allen Leung, Ken Mackenzie,
Eric Schweitz, Peter Szilagi, Richard Lethin
{mattson, leunga, kenmac, schweitz, szilagyi, lethin} @ reservoir.com

Reservoir Labs Inc.
Phone: (212) 780-0527
Fax: (212) 780-0542

Developing portable, high-performance, parallel applications offers substantial benefits in key embedded computing areas such as signal processing, but presents equally substantial challenges. Presently, when a programmer needs to write a parallel application, he or she either writes a serial program and relies on a compiler to parallelize it, or writes an explicitly parallel program. The "parallel" output of a parallelizing compiler given arbitrary serial input is often far from ideal. Writing an explicitly parallel application is time-consuming and difficult. Either the parallelizing compiler or the explicitly parallel program is usually tied to a specific architecture (often implicitly due to assumptions made about the architecture in order to achieve good performance).

This talk will describe the motivation behind the R-Stream compiler being developed by Reservoir Labs in an attempt to overcome these challenges. R-Stream is a *high-level compiler* that maps an application written in a streaming language to a parameterized virtual machine that can represent one of many target architectures. A simple architecture-specific low-level compiler then translates the abstract mapping produced by the high-level compiler to exploit specific features of the target architecture. This flow is shown in Figure 1:

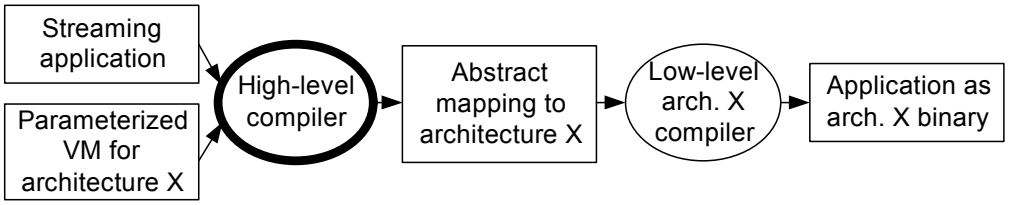

Figure 1: High-level compiler flow

Streaming languages, like Brook [1], StreamC [2], and StreamIt [4] allow the programmer to structure an application so that its parallelism is apparent to the compiler. This approach differs from general purpose languages like C, which obscure parallelism, and explicitly parallel languages or libraries, like MPI, which require the programmer to perform parallelization manually. More concretely, a streaming language separates the control code that describes the high-level structure of the application from the computation code that describes each processing step and explicitly defines the data flow between processing steps. Some streaming languages also restrict processing steps to enforce data parallelism.

R-Stream leverages this clear expression of the application's high-level structure and parallelism to optimize the application as a whole. This optimization involves merging or splitting processing steps, and mapping processing steps across time and resources. This "top-down" approach differs

from most conventional compiler optimizations that are "bottom-up" – they target small, analyzable regions of an application such as single loop nests. Since a streaming language also makes the data flow explicit, R-Stream maps the data within the application and produces an executable that uses explicit DMA transfers or other efficient hardware support instead of relying on a cache.

R-Stream targets a *parameterized* virtual machine, the PCA Virtual Machine (VM), which is being developed to describe the characteristics of different architectures in a uniform, abstract way. The VM uses a set of standard constructs such as processors, memories, and network links to describe an architecture. For each processor, memory, or network link, it specifies key parameters such as IPC, memory size, and bandwidth. Different sets of constructs with different parameters describe different architectures. The VM design is intended to represent such diverse architectures as MIT's RAW processor [5] or UT's TRIPS processor [3]. Using the PCA VM enables R-Stream to produce code optimized for a specific architecture, yet apply the same optimizations to multiple architectures, amortizing the cost of embedded application development and compiler development over multiple generations of different types of architectures.

Using streaming languages and a parameterized VM makes the compilation problem more tractable, but there are significant research questions to be answered in implementing R-Stream:

1. How should a compiler represent the high-level structure of an application?
2. What is a practical algorithm for mapping this structure to a target architecture?
3. How restrictive do streaming languages need to be to enable optimization?
4. How detailed does a parameterized VM need to be to sufficiently describe a target architecture?
5. Is it possible to separate the whole program optimizations performed by a high-level compiler from the architecture-specific optimizations performed by a low-level compiler?

Answering these questions is key to enabling the development of high-level compilers. Tools for efficiently developing parallel applications like high-level compilers are increasingly important as decreasing feature sizes make single chip multiprocessor systems common in embedded systems. This talk will summarize Reservoir's technical approach and progress in this effort.

References:

[1] I. Buck. *Brook Spec v0.2*, unpublished draft, May 2003.

[2] P. Mattson. *A Programming System for the Imagine Media Processor*, Ph.D. Thesis, Dept. of Electrical Engineering, Stanford University, 2002.

[3] R. Nagarajan, K. Sankaralingam, D.C. Burger, and S.W. Keckler. "A Design Space Evaluation of Grid Processor Architectures", *Proceedings of the 34th Annual International Symposium on Microarchitecture*, pp.40-51, December 2001.

[4] W. Thies, M. Karczmarek, and S. Amarasinghe. "StreamIt: A Language for Streaming Applications", *Proceedings of the 2002 International Conference on Compiler Construction*, April 2002.

[5] M. B. Taylor, J. Kim, et al., "The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs", *IEEE Micro*, April/March 2002.