ScalaPipe: A Streaming Application Generator

Joseph G. Wingbermuehle, Roger D. Chamberlain, and Ron K. Cytron
Dept. of Computer Science and Engineering, Washington University in St. Louis
{wingbej,roger,cytron}@wustl.edu

The streaming approach to parallel programming is a popular programming paradigm. Examples of systems that use the streaming approach include Auto-Pipe [2], Streams-C [3], and StreamIt [6]. In the streaming paradigm, an application is decomposed into processing kernels, or blocks, which are then connected using explicit communication channels.

Extant toolsets that support the development of streaming applications therefore provide tools for developing the blocks of an application along with some mechanism for configuring and allocating resources to those blocks on a streaming platform. We observe that these two activities currently take place in completely separate languages and tools. We point out the deficiencies of current approaches and we examine the benefits of using a single language to realize both activities.

Our primary basis for comparison is the Auto-Pipe toolset [2]. In Auto-Pipe, blocks are authored in a language suitable for the deployment target (CPUs or FPGAs). Auto-Pipe’s X coordination language is then used to specify how the blocks are to be connected and to what resources they should be assigned.

The Auto-Pipe coordination language approach works well for many problems [1]. However, Auto-Pipe suffers from the following deficiencies:

1) Creating applications can require a great deal of X code.
2) Conceptually simple changes to the application or resource mapping can require many changes to the corresponding X code.
3) Blocks cannot be polymorphic.
4) Blocks must be implemented separately for each platform.
5) Finally, communication overhead can be substantial even for blocks that are mapped to the same resource.

To address the aforementioned problems with X, we developed a new system, ScalaPipe, to generate applications that use Auto-Pipe blocks as well as the Auto-Pipe blocks themselves. ScalaPipe is a set of domain-specific languages (DSLs) embedded in the Scala programming language [5]. Using Scala as the host language allows for much more expressive power when creating applications. Further, blocks implemented in the Scala DSL can be compiled to different targets (for example, CPUs, GPUs, and FPGAs) to generate families of blocks.

A ScalaPipe program is a Scala program that uses the ScalaPipe domain-specific languages to generate an Auto-Pipe application. ScalaPipe defines two main DSLs: the application DSL and the block DSL. The application DSL is used to connect blocks together and map them to resources. The block DSL is used to declare and (optionally) implement the blocks.

Because the streaming application is described in a DSL embedded in Scala, Scala language constructs can be used to generate potentially large and complex application topologies and resource mappings. The application DSL thus addresses problems 1 and 2 with Auto-Pipe. ScalaPipe allows blocks to be implemented completely within the ScalaPipe block DSL. This allows one to use the same block code to operate on multiple data types and run on multiple resources. The block DSL thus addresses problems 3 and 4.

To address problem 5 with Auto-Pipe and X, ScalaPipe can be enhanced to combine multiple blocks that are connected together on the same resource. Although this feature has yet to be implemented, such an optimization would be possible to implement in ScalaPipe without requiring application or block code changes.

The ease of creating new blocks and versatile resource mapping abilities of ScalaPipe allow evaluation of alternative resource mappings. Any bottlenecks in the application can be tracked down using a performance monitoring system such as TimeTrial [4], which is integrated into ScalaPipe. These bottlenecks can then be addressed by a custom block implementation or by changing the application topology and resource mapping.

REFERENCES


This work is supported by the National Science Foundation under grant CNS-0931693.