Compiling Stream-Language Applications to a Reconfigurable Array Processor

Zain-ul-Abdin and Bertil Svensson
Centre for Research on Embedded Systems (CERES),
Halmstad University, Halmstad, Sweden.

Abstract

New parallel architectures are emerging to meet the increased computational demands of streaming applications. This creates a need for high-level, architecture-independent languages. One such language is StreamIt, designed around the notions of streams and stream transformers, which allows efficient mapping to a variety of architectures.

This paper presents our approach of compiling StreamIt applications to the XPP reconfigurable array architecture. We focus mainly on the compiler back end. Although StreamIt exposes the parallelism in the stream program, still a thorough analysis is needed to adapt it to the target architecture. A code generator has been designed for the XPP. It has been demonstrated that by applying optimizations, performance comparable to the low level NML implementation can be achieved. Moreover, the construction of the compiler makes it possible to port StreamIt applications to multiprocessor architectures by doing some architecture specific modifications in the back end.

1 Introduction and Motivation

Media applications, such as voice and video processing as in HDTV or baseband processing in cell phone base stations, are becoming increasingly complex and consume more and more computing power.

Programming these streaming applications faces the difficulty of making full use of the computing power available by new increasingly complex parallel architectures. New grid-based hardware technologies are emerging, which conventional programming languages are not well suited for. The compilers for conventional programming languages such as C or Java generate trees that are difficult to map onto parallel architectures. The programming language StreamIt is designed for applications with chains of data to be processed in different ways. The compiler for StreamIt produces stream graphs [1] that can easily be mapped onto parallel architectures such as grid-based architectures.

The goal of the StreamIt project at MIT is to provide the language and compilation infrastructure required by the Reconfigurable Architecture Workstation (RAW) project. RAW is a scalable processor architecture suitable for applications in which the compiler can extract and statically schedule fine-grain parallelism. It consists of 16 tile processors arranged in a 2D-mesh, each with a programmable switch for communication between tiles [2].

The eXtreme Processing Platform (XPP) from PACT XPP Technologies, with its packet oriented communication mechanism is very suitable for compute intensive streaming applications. The PACT XPP64A-1 reconfigurable processor core consists of an 8 x 8 array of ALU-PAEs (Processing Array Elements) with 2 rows of RAM-PAEs at the edges [3]. The software development tools for XPP consist of Vectorizing-C compiler for compiling C code to Native Mapping Language (NML) code, that can be compiled by XMAP mapper to generate binaries for XPP core or for simulation by XSIM simulator [4].

In this paper we present a method to compile applications developed in StreamIt to XPP. The primary focus is laid on issues related to the compiler back end and its interface to the preprocessor. It is the back end of the compiler that translates the graph representation of the

parallel program produced by the compiler front end into machine code.

The rest of the paper is organized as follows; Section 2 presents salient features of parallelizing compilers. Section 3 provides an overview of the StreamIt language. Section 4 describes the StreamIt compiler and its generated code. Section 5 examines the proposed StreamIt compilation steps for XPP. The results of FIR and FFT implementations are discussed in Section 6, and the conclusions of the study are presented in Section 7.

2 Parallelizing Compilers

For parallel architectures the compilers must be able to detect and utilize parallelism. Mapping computations onto various processing array elements and optimization of different distributions are dealt with by the compiler back end tool chain. Compared to traditional compilers, a significant feature of parallelizing compilers is instruction scheduling. The phase of mapping a parallel program to a parallel architecture starts without explicit communication and produces a program with explicit communication operations. The communication scheduling starts with data distribution and generates a code for whatever communication is required to adhere to the chosen distribution.

3 StreamIt Language

The StreamIt language is an implementation of the stream-programming model, which constructs a stream graph, containing stream transformers with a single input and a single output to enable efficient development of stream programs.

StreamIt [5] has five data types and four primitives for building stream transformers. The StreamIt primitives, i.e., filter, pipeline, split-join and feedback loop, have one input and one output stream. Filters are used for building atomic stream transformers while pipelines, split-joins and feedback loops are used for combining stream transformers as shown in Figure 1.

StreamIt, which is a portable language for compute-intensive signal processing applications, allows the compiler to map a single source program on a variety of grid-based architectures by taking away the variations and retaining the similarities [6]. The ability of the compiler to reconstruct the stream graphs enables it to combine adjacent stream transformers or split computationally intensive transformers into multiple parts to have a more parallel computation. The StreamIt compiler optimizes the stream graph to produce an efficient code, which allows the implementation of high-performance applications for parallel systems without detailed knowledge of the target architecture.

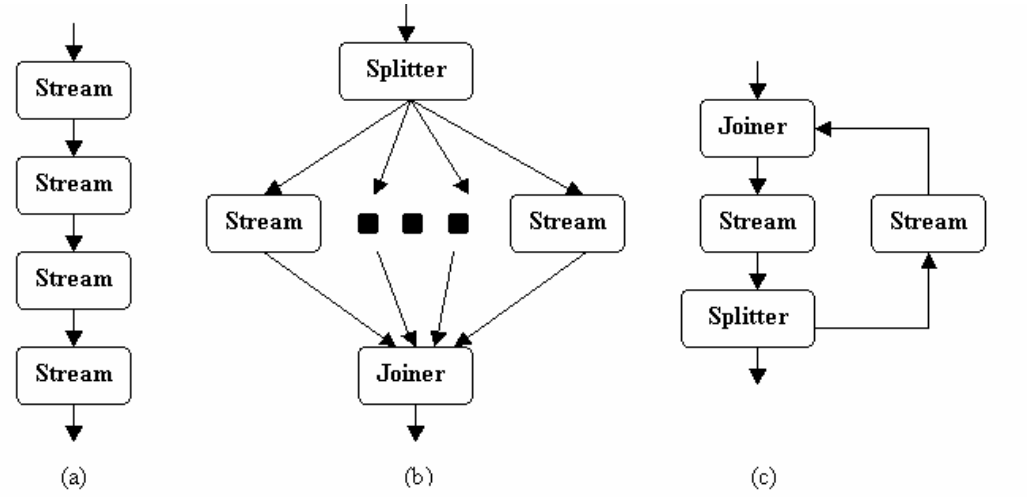

Figure 1 StreamIt combinators: (a) Pipeline, (b) Split-join, (c) Feedback loop.