A Domain Specific Language for DSP

http://dsl4dsp.inf.elte.hu

Karina Bunyik, Gergely Dévai, Zoltán Gera, Zsolt Gulyavári, Zoltán Horváth,
Kálmán Karch, Krisztián Markó, Gyula Nagy, Emese Palkó, Endre Sajó, Máté Tejfel

Department of Programming Languages and Compilers
Eötvös Loránd University, Budapest

Mary Sheeran, Emil Axelsson

Computer Science and Engineering Dept.
Chalmers University of Technology, Göteborg

András Vajda, Peter Brauer,
Bo Lyckegård, Arto Mahkonen,
Anders Persson, Henrik Sahlin, Anders Wass

Ericsson

Introduction

Digital signal processing (DSP) algorithms are usually designed and described on an abstract level and than transformed to a DSP chip specific C code by expert programmers. The problem is that the gap between the abstract description and the platform dependent code is huge and even the C code optimized for two different chips differ a lot. This makes it expensive to rewrite the algorithms each time a new chip is targeted.

The ***DSL4DSP*** project is being carried out by Ericsson, ELTE University (Budapest) and Chalmers University (Göteborg, Sweden). The goal of the project is to develop a high level domain specific language designed for digital signal processing algorithms and to implement a prototype compiler and related tools (eg. debugger) for the language.

We expect that a high level domain specific language will make the implementation of algorithms easier and a compiler together with platform-specific code generator and optimizer modules will take the burden of target dependent low level programming off the programmers.

Language Design

The goal is a high level language that allows compact and easy to understand definition of algorithms and has precise semantics. It has to be completely hardware platform independent so that it may later be recognized as a standard.

The targeted application area of the language are DSP algorithms like speech- and video codecs, (de)modulators etc. A clear C interface will be defined so that source code written in the high level language can easily fit to existing code in C and use the services of DSP libraries.

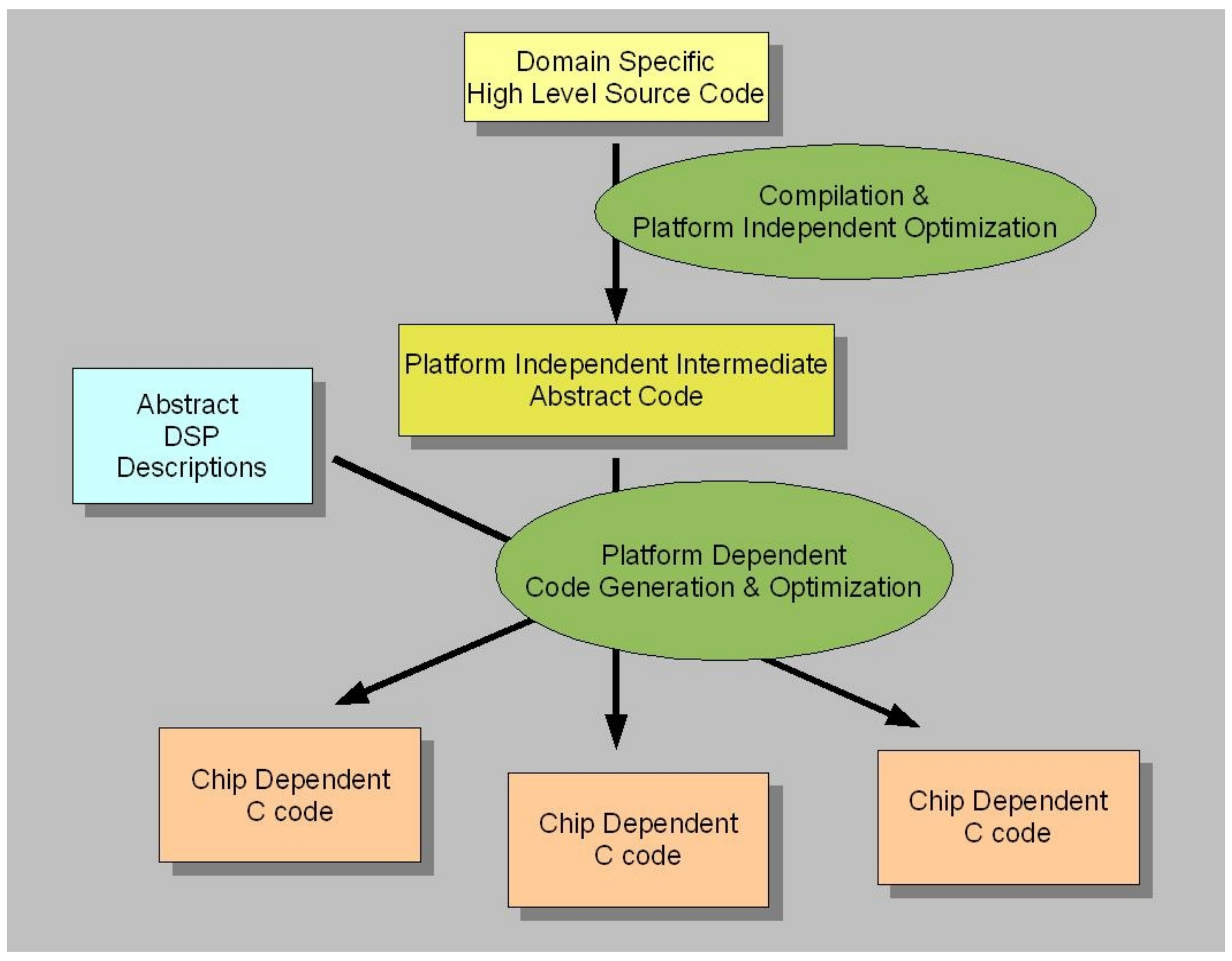
The language will be based on two programming paradigms: dataflow descriptions and functional programming. On an abstract level, DSP algorithms are usually described as data flow computations and we plan to combine this technique with the clarity and high abstraction level of functional programming.

A motivating example is Lava, an embedded language in Haskell for hardware description and verification. Simple circuits are described in a functional style and they are combined using connection patterns (combinators) which are in fact higher order functions combining circuits according to a given scheme (eg. sequential or parallel composition). We expect that similar combinators will be useful also in the current language as they raise the level of abstraction.

An other related language is Cryptol, a domain specific language for cryptography. Cryptol faces a problem similar to ours: algorithms need to be described in an abstract way and then transformed to platform-dependent, optimized implementations. Although Cryptol is constructed for a different domain, it may inspire our language design.

In the first phase of the development, the language will be embedded in Haskell, a pure, lazy functional language. We use Haskell because it includes a wide range of language constructs that make embedding convenient. By embedding the language we get the frontend of the compiler (scanner, parser, typechecking) for free. This is especially advantageous in the design phase when the language definition is supposed to change very often.

Once the language definition is fixed, we will implement a standalone compiler. This will make it possible to adjust compile errors to the semantics of the language (instead of Haskell semantics) and provide additional tool support.


Domain Specific High Level Source Code

Compilation & Platform Independent Optimization

Platform Independent Intermediate Abstract Code

Abstract DSP Descriptions

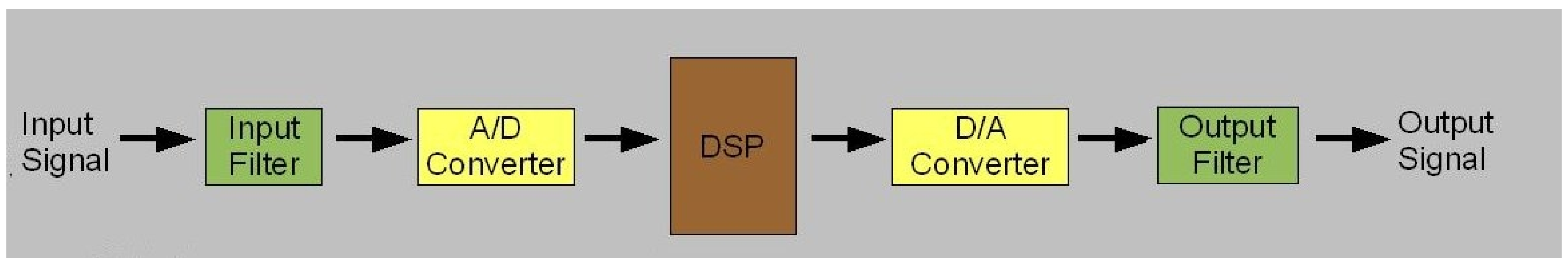
Platform Dependent Code Generation & Optimization

Chip Dependent C code

Chip Dependent C code

Chip Dependent C code

Supported by Ericsson and the
Swedish Foundation for Strategic Research.


Input Signal

Input Filter

A/D Converter

DSP

D/A Converter

Output Filter

Output Signal

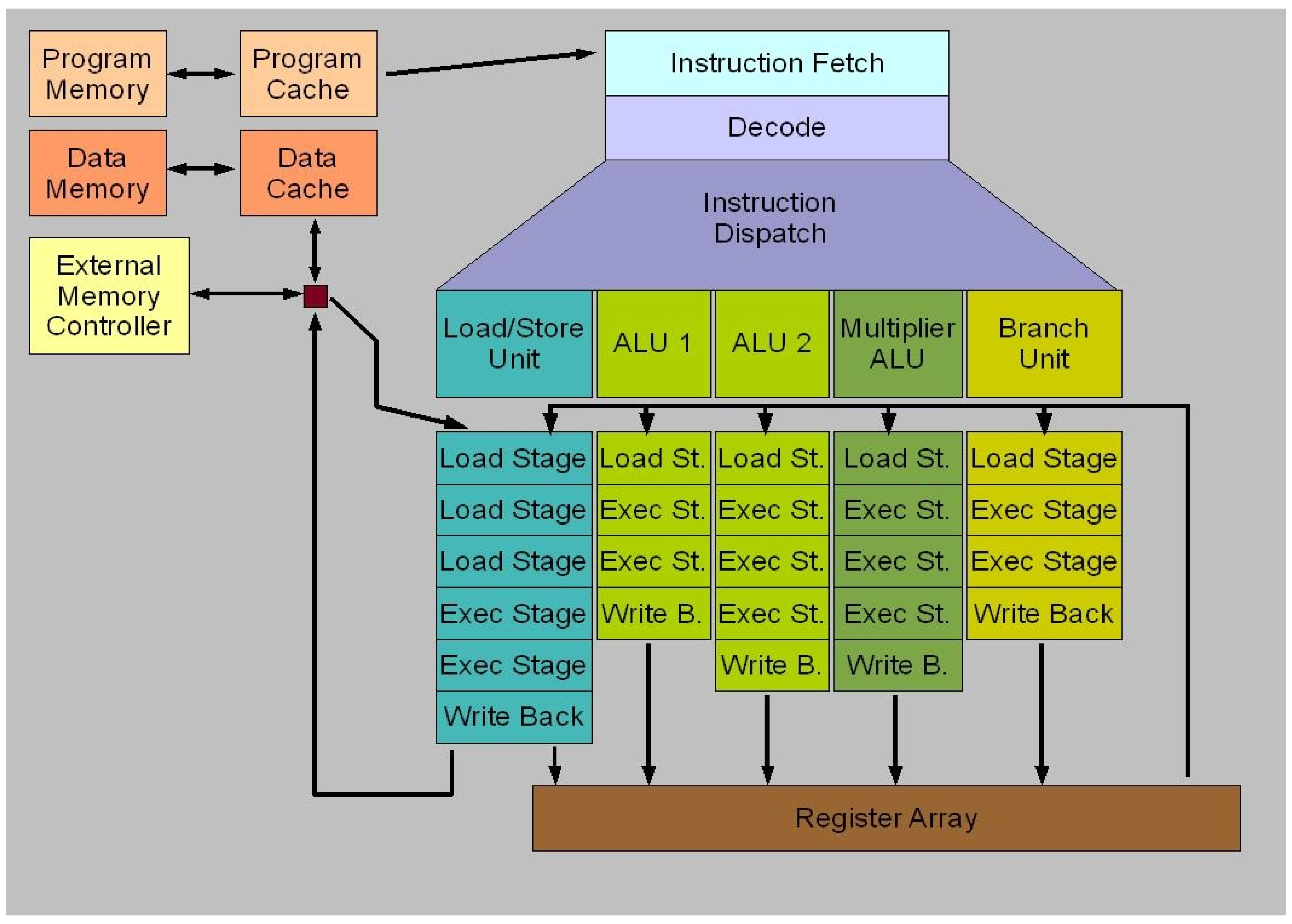
DSP Architectures

When generating code we have to take into account that the code will run mainly on DSP chips and FPGAs. In the first phase of the development we will concentrate on DSPs.

These chips usually have several cores that may be specialized for different tasks (eg. arithmetic operations, memory management etc.). This means that this architecture is implicitly parallel and we have to generate C code that allows the chip specific C compiler to identify the instructions that can run in parallel.

DSP chips are heavily using pipeline technology. There are several stages to prepare instructions for execution and then to execute them. This implies that passing the control flow (jumps, functions calls) is relatively expensive on these architectures.

Caching both code and data is an important feature of these platforms. That is why optimal implementations of DSP algorithms generally use arrays and simple loops. Dynamic memory management and linked data structures are usually present only in the programs that control the DSP algorithms and not in the algorithms themselves.


Program Memory

Program Cache

Instruction Fetch

Decode

Data Memory

Data Cache

Instruction Dispatch

External Memory Controller

Load/Store Unit	ALU 1	ALU 2	Multiplier ALU	Branch Unit
Load Stage	Load St.	Load St.	Load St.	Load Stage
Load Stage	Exec St.	Exec St.	Exec St.	Exec Stage
Load Stage	Exec St.	Exec St.	Exec St.	Exec Stage
Exec Stage	Write B.	Exec St.	Exec St.	Write Back
Exec Stage		Write B.	Write B.	
Write Back				

Register Array

Compiler Construction

The main requirement against the compiler is that the efficiency of the generated C code and hand crafted code has to be comparable. In order to accomplish this requirement, we have to take into account all the special features mentioned about the DSP chip architectures and build heavy optimization into the compiler.

We plan to compile functions according to strict semantics. Lazy semantics would require closures created dynamically in memory and some kind of garbage collection. These are too expensive techniques on the targeted platform. For the same reasons we will avoid higher order functions by extensive inlining.

Optimization techniques fall into two categories: platform dependent and platform independent techniques. Therefore we split compilation into two phases. The result of the first phase is an abstract representation of a general purpose imperative code where platform independent optimizations are already applied. The second phase receives an abstract hardware description as additional input and generates C code with annotations to guide the optimization process of the chip-specific C compiler. The abstract hardware description includes information for example on the number of cores in the chip, the length of its pipeline, and the DSP library to use.

We plan to generate human readable C code for two reasons. Optimization techniques of C compilers are constructed for human written code and we would like to get valuable feedback from domain expert C programmers about the generated code.

In cooperation with ELTE IKKK.