AlphaZ Wiki

This wiki is intended to serve as the user manual of AlphaZ. If you are an AlphaZ developer, you should refer to the AlphaZ Developers wiki

Introduction

AlphaZ is an open source tool-set for program analysis, transformation and parallelization in the Polyhedral Equational Model. It is being developed by the Mélange group at CSU, and uses an equational language called Alpha/Alphabets.

AlphaZ is a general framework for analysis, transformation and code generation in the Polyhedral Equational Model. The input "program" consists of one or more mathematical equations that specify just ***what*** needs to be computed. It can be viewed as a specification. In order to produce a (conventional/imperative) program that *implements* this specification, one needs to specify a schedule (when), a processor allocation (who), and a memory allocation (where to store). Actually, even this is not strictly necessary. We also have a "memoized demand driven" code generator that produces executable code in the absence of any schedule or memory/processor allocation information.

Choosing these attributes in a way that optimizes desired performance criteria for a range of target architectures is a long-standing open research problem. AlphaZ does not aim to (immediately) solve this problem, although at times we have used the system to solve specific instances of specific optimization problems. Rather, AlphaZ provides a framework in which such problems can be solved by different users for different targets with different objective functions to optimize. This is often the immediate goals of individual students at specific times, but the goal of the AlphaZ system is to serve as the infrastructure for this research. It is therefore a system in which to explore optimization strategies, often manual but eventually automatic, to achieve higher performance. In particular, AlphaZ exposes the ability to alter memory allocation, and treats reductions as "first-class." These options are not commonly exposed in other polyhedral tools.

Related Publications

Main Components

In AlphaZ, you first read an `Alphabets` program (i.e., system of equations). In general, AlphaZ can do three things to an Alphabets program.

- AlphaZ can **transform** the program (all transformations in AlphaZ are semantics-preseving, i.e., are guaranteed to not change the meaning of the program).
- It can **analyze** the program (e.g., you may ask the system to deduce a parallel schedule for the program).
- Finally, AlphaZ can **generate code** from the equations so that the equations can be "executed."

Installing AlphaZ and Getting Started

Instruction to setup AlphaZ for CS department accounts, and to install in an external machine is available in a separate page.
Operating systems other than MacOSX and Linux are not supported at this time.
Support for 32bit Linux is limited, since we have very little access to 32bit machines.
Eclipse Setup

Source for AlphaZ are available in our repository, but our system depends on Eclipse and plug-ins developed for Eclipse Modeling Framework as well. We also use a large number of plug-ins from CAIRN team at IRISA, Rennes. The repository structure of CSU and IRISA is outlined in this page : Source Access. However, it is not a complete list, and we recommend users to download the bundle and then checkout projects corresponding to where you would like to make changes.

Alphabets and Equational Programming

Alpha is a simple equational programming language. The Alphabets grammar is just two pages long. A program is one or more *systems* of equations. The system may be *parameterized* with one or more size parameters (e.g., matrix multiplication of square matrixes has a parameter, n, the size of the matrices). A system consists of six parts (optionally preceded by a set of external function signatures):

- A system name
- A parameter declaration
- Input variable declaration
- Output variable declaration
- Local Variable Declaration
- Equations

An equation is of the form `Variable = Expression` and an Expression has the following syntax:

- **Atomic expressions** are either variable names or constants
- **Pointwise operators** are of the form `exp1 op exp2` or `op(exp1, exp2, …)`
- **Case** expressions are of the form

```
case
   exp1;
   ...
   expN;
esac
```

- **Restrict** expressions are of the form `Domain : exp`
- **Dependence** expressions are of the form `dep @ exp` or sometimes `Var[index-expr]`
- **Reduce** expressions are of the form `reduce(op, dep, exp)`

So all you need to know, is how to write Domains, and deps.

Deps are `(ListOfIndices → ListOfIndexExpressions)`

Domains are `{ListOfIndices | ListOfInequalities}`

Of course, there are a few additional subtleties, index expressions, array notation, etc. There are also subsystems (not yet implemented), but the above bit of information is all you need to be on your way. So let's get started.

Tutorial / Examples

List of Commands http://www.cs.colostate.edu/AlphaZ/AlphaZCommandRefV2.pdf
Tutorial using LU decomposition Tutorial LUD.
Tutorial on how to use external functions Tutorial External Function.
Tutorial on Check Program Check Program.
calculator
Examples of how to use Normalize.
Examples of how to use Change of Basis.
Example of how to specify Target Mapping.
Example of how to use Schedulers.
verifier
Example of how to use Schedule Code Generator.
List of Code Gen Options.
Tutorial on transformations of reductions Reduction Tutorial.
How to run compiler scripts from terminal. Command Line AlphaZ

History and Background

`AlphaZ` is similar in spirit to an earlier tool **`MMAlpha`** developed at IRISA, Rennes, France. While MMAlpha targeted FPGA based hardware synthesis, AlphaZ addresses code generation for modern (multi- and many- core) parallel processors. We also focus on optimization of programs with reduction operations. Alphabets is an extension of the language Alpha, also developed at IRISA, which was the basis of MMAlpha.

The ***polyhedral model*** is a framework for analysis and transformations of programs, extensively used for high-level loop optimizations in compilers today. The *polyhedral* ***equational*** *model* has the same goals, but focuses on equational/functional programming. The model provides the ability to reason mathematically about programs, their dependences, and semantics. This has lead to a number of very powerful tools for automatic parallelization. By design, the model is applicable to a limited class of programs: dense, regular, computations (the so-called *affine* computations). However, such programs are very widespread, and constitute the compute- and data-intensive kernels in most applications.