

Contents

1	Getting started guide	3
1.1	Prerequisites	3
1.2	Setup and usage	4
1.2.1	Step 1: Extract the archive	4
1.2.2	Step 2: Run the jar file	4
1.2.3	Understanding the switches	5
1.2.4	Current limitations	5
1.2.5	Running the generated X10 code	6
2	Artifact evaluation	6
2.1	Overview of the X10 compiler	6
2.2	Command line switch combinations for evaluating the artifact	7

List of Figures

Abstract

MATLAB is a popular dynamic array-based language commonly used by students, scientists and engineers who appreciate the interactive development style, the rich set of array operators, the extensive builtin library, and the fact that they do not have to declare static types. Even though these users like to program in MATLAB, their computations are often very compute-intensive and are better suited for emerging high performance computing systems. This paper reports on MIX10, a source-to-source compiler that automatically translates MATLAB programs to X10, a language designed for “Performance and Productivity at Scale”; thus, helping scientific programmers make better use of high performance computing systems.

There is a large semantic gap between the array-based dynamically-typed nature of MATLAB and the object-oriented, statically-typed, and high-level array abstractions of X10. This paper addresses the major challenges that must be overcome to produce sequential X10 code that is competitive with state-of-the-art static compilers for MATLAB which target more conventional imperative languages such as C and Fortran. Given that efficient basis, the paper then provides a translation for the MATLAB `parfor` construct that leverages the powerful concurrency constructs in X10.

The MIX10 compiler has been implemented using the McLab compiler tools, is open source, and is available both for compiler researchers and end-user MATLAB programmers. We have used the implementation to perform many empirical measurements on a set of 17 MATLAB benchmarks. We show that our best MIX10-generated code is significantly faster than the de facto Mathworks’ MATLAB system, and that our results are competitive with state-of-the-art static compilers that target C and Fortran. We also show the importance of finding the correct approach to representing the arrays in X10, and the necessity of an *IntegerOkay* analysis that determines which double variables can be safely represented as integers. Finally, we show that our X10-based handling of the MATLAB `parfor` greatly outperforms the de facto MATLAB implementation.

1 Getting started guide

The artifact is provided as a zipped tar archive(.tar.gz) of a directory named `MiX10`. The `README.txt` file provided in the `MiX10` directory gives a list of the various components of our artifact.

The main component of our artifact is the MIX10 compiler itself, provided as an executable jar file. It is a command-line only tool. It takes an input MATLAB file and compiles it to X10. In this section we first list the prerequisites required to run the jar file and then give step-by-step instructions to run the compiler. We also give a description of various command-line switches available with the compiler.

1.1 Prerequisites

The following list gives the various prerequisites in order to successfully run the MIX10 compiler:

Operating System: In its current version MIX10 is available only for Unix and Unix-like operating systems only. In particular, we have tested the MIX10 compiler on Ubuntu Linux 12.04 (kernel version: 3.8.0-35-generic #52 precise1-Ubuntu SMP Thu Jan 30 17:24:40 UTC 2014) and Mac OS X 10.9 (Darwin Kernel Version 13.2.0: Thu Apr 17 23:03:13 PDT 2014).

Java: The MIX10 compiler requires the Java Runtime Environment 1.8.0 (Java 8). It is incompatible with the previous versions of the Java Runtime Environment.

X10 compiler: In order to run the MiX10 compiler, one does not need to have an X10 compiler. However, to compile and run the X10 files generated by the MiX10 compiler, it is required to have a compatible version of the X10 compiler. The X10 code generated by the MiX10 compiler requires at least version 2.4 of the X10 compiler, available freely from the X10 website¹. Details about the different variations of the X10 compiler used for our experiments are discussed in Section 2

1.2 Setup and usage

Once the prerequisites are satisfied, download the artifact (`MiX10.tar.gz`) and follow the following step-by-step guide to run the MiX10 compiler with the various command-line switches. We will use bubble sort implementation in MATLAB, as an example input file.

1.2.1 Step 1: Extract the archive

Once you have downloaded the file `MiX10.tar.gz`, open a terminal and `cd` to the directory where the `MiX10.tar.gz` file is located. To extract the archive, run the following command:

```
1 $ tar -xvf MiX10.tar.gz
```

This will create a directory named `MiX10`, which will contain a `README.txt` file with a description of the contents of the directory.

1.2.2 Step 2: Run the jar file

Inside the `MiX10` directory, there is the jar file named `MiX10.jar`. This jar file can be run using a command formatted as:

```
1 $ java -jar MiX10.jar -mix10c\  
2 -arg_info "DOUBLE&1*1&REAL"\  
3 -main "path/to/the/input_matlab_file.m"\  
4 -od "path/to/the/output/directory/"\  
5 [Optional switches]
```

For example, to compile the bubble sort example located in the directory `examples/bubble/`, run the following command:

```
1 $ java -jar MiX10.jar -mix10c\  
2 -arg_info "DOUBLE&1*1&REAL"\  
3 -main "./examples/bubble/drv_bubble.m"\  
4 -od "./output/"
```

This should create the output x10 file, `drv_bubble_x10.x10` under the `output` directory. It also creates a package directory named `simpleArrayLib` which contains a file named `Mix10.x10`, the library file containing the builtins. In addition to generating the output X10 file, the above command also displays some debug information on the console. If everything worked correctly, the

¹<http://x10-lang.org/software/download-x10/latest-release.html>

debug information should end with printing the X10 program on the console, followed by a message, "No files provided, must have at least one file.". This message can be safely ignored (It will be resolved in future releases of MiX10 and McLAB [2, 1]).

1.2.3 Understanding the switches

As seen in the above example, there are four compulsory switches that are required to run the MiX10 compiler. They are described below:

- `-mix10c` This switch is required to invoke the MiX10 compiler.
- `-arg_info` This switch describes the type, shape and Real/Complex nature of the input argument that is passed to the entry point function of the MATLAB program. In the above example, the value `DOUBLE&1*1&REAL` describes that the input argument to the entry function is of type Double, is scalar (matrix of size 1*1) and is a Real numerical value.
- `-main` This switch reads the following string after it as the path to the entry point function of the input MATLAB program. The path can be relative or absolute.
- `-od` This switch reads the following string as the path to the directory where the generated X10 code should be placed. This path can be relative or absolute.
- `-class_name` This switch takes string value which is used as the name of the generated X10 program. By default, the generated class name is created by appending "_x10" to the name of the MATLAB entry point function.
- `-use_region_arrays` This is a boolean switch that tells the compiler to explicitly use the region arrays even if it is possible to use the simple arrays. Section 4 of the paper describes the default behaviour, that is to generate the simple arrays whenever possible.
- `-no_intok` This is also a boolean switch to explicitly tell the compiler to disable the IntegerOkay analysis described in section 5 of the paper.

1.2.4 Current limitations

The development of the MiX10 compiler is a work in progress and there are some limitations to how it works. These limitations are as follows:

- Exactly one argument can be passed to the entry point function. The entry point function to the input MATLAB program should take exactly one argument, whose type is described by the switch `-arg_info`. However, there is no restriction on the type or shape of this argument. none or more than one arguments will result in an exception and there will be no output.
- All the functions in the input program must be in separate files. Each function of the input MATLAB program must be present in a separate `.m` file.
- Generated `Mix10.x10` library file does not contain all the library functions used by the program. We are still in the process of building the XML file used by the builtin-handler described in Section 2 of the paper. This means that for some builtins used in the MATLAB program,

the X10 implementation is not populated automatically in the generated library file. However, we have provided a hand-written implementation of the builtin library functions in the directory named `library`.

- Add the `%!parfor` annotation for parfor loops The McLab frontend currently does not support the MATLAB parfor construct. However, adding the annotation `"%!parfor"` just before the parfor loop in the MATLAB program will trigger the MiX10 compiler to generate the parallel X10 code for the parfor construct.

1.2.5 Running the generated X10 code

There are two steps that need to be done to make the generated X10 code ready to run: (1) replace the generated `simpleArrayLib`(or `regionArrayLib`) directory by the `simpleArrayLib`(or `regionArrayLib`) directory provided inside the `library` directory mentioned above; and (2) In the `main` method in the generated X10 file, where indicated, manually add a call to the entry point function. The syntax is similar to that in Java.

2 Artifact evaluation

Once the X10 code is generated by following the steps in the previous section, it is evaluated for performance by compiling and executing it using the X10 compiler. To evaluate the effects on performance of the various features of the MiX10 compiler, appropriate switches are used to generate code with a particular feature turned on/off, as described in the getting started guide in the previous section. Similarly, different switches are used in the X10 compiler to evaluate the generated code more elaborately.

2.1 Overview of the X10 compiler

The X10 compiler provides two backends, a Java backend and a C++ backend, effectively giving two different compilers. To reproduce the X10 compiler setup used by us for evaluating the MiX10 compiler, follow the steps below:

1. Download the X10 compiler source code as described on the X10 website². We used X10 version 2.4.0, but the latest version 2.4.3 should also work.
2. As described in the section 4.1.1 of the paper, we made changes to the X10 source code to use column-major indexing in place of the default row-major indexing. To make these changes, replace the `Array_2.x10` and `Array_3.x10` files present in the `x10-trunk/x10.runtime/src-x10/x10/array` directory of the X10 compiler source code by our version of these files available from our website³. Compile the X10 source code as per the instructions given on the X10 webpage⁴. The X10 website also has documentation on how to use the X10 C++ and X10 Java compilers. We always used the `-O` switch of the X10 compiler backends for our experiments.

²<http://x10-lang.org/x10-development/building-x10-from-source.html?id=145>

³http://www.sable.mcgill.ca/mclab/mix10/x10_update/

⁴<http://x10-lang.org/x10-development/building-x10-from-source.html>

2.2 Command line switch combinations for evaluating the artifact

The optional switches provided by the MiX10 compiler can be turned on or off in combination with the `-NO_CHECKS` switch of the X10 compiler. For example, to evaluate the performance of the IntegerOkay analysis, first generate the X10 code from the MiX10 compiler using the `-no_intok` switch, which will explicitly turn off the IntegerOkay analysis. Next, generate the X10 code without using this switch to get X10 code with the analysis. These two sets of X10 code can be compiled by the X10 compiler to compare their performance for different switches of the X10 compiler (both C++ and Java). Section 5 of the paper discusses the IntegerOkay analysis and section 7.7 of the paper gives the evaluation results for it.

Similarly, the `-use_region_arrays` switch can be toggled to generate code for explicitly generating code with region arrays. Region arrays and Simple arrays are discussed in section 4.1 of the paper. Their comparison is presented in section 7.6 of the paper.

Parallel X10 code can be evaluated similarly, after setting the environment variable `X10_NTHREADS` to specify the number of worker threads used by the X10 runtime⁵

References

- [1] J. Doherty. McSAF: An Extensible Static Analysis Framework for the MATLAB Language. Master's thesis, McGill University, December 2011.
- [2] A. Dubrau and L. Hendren. Taming MATLAB. In *Proceedings of OOPSLA 2012*, pages 503–522, 2012.

⁵<http://x10-lang.org/documentation/practical-x10-programming/performance-tuning.html>