Dyalog ’16 Workshop TP2 Pre-requisites: Compiling ANN & Other APL Code

Overview

This document will get you up and running on your local laptop or machine with the configuration that we will be using in the Compilers Workshop at Dyalog ’16. There will be other machines at the workshop, but we encourage everyone who can to install a workable environment on their own laptops so that they have an idea of how the Co-dfns compiler will work on their local machines and setup. This document will get you all the way up to installing and configuring Co-dfns on your local machine. We will be covering the configuration and customization of Co-dfns for your local environment at the workshop, but it is good for you to have spent some time going through the basic installation process ahead of time.

Supported Operating Systems

We support the following Operating Systems:

1. Currently supported versions of Microsoft Windows, though we recommend version 10.
2. Modern Linux distribution that can support CUDA

We use recent versions of Microsoft and Linux software, so it helps to ensure that your Linux distribution or your version of Windows is up-to-date and recent. Windows 10 works very well, as does Slackware 14.2, RHEL 7+, Fedora 23+, Ubuntu 16.04, and openSUSE 13.2+. If you use some other distribution, please ensure that it is a recent version and up-to-date, as some Linux distributions have older versions of software or missing features that may conflict or cause problems with the software requirements.

We do not currently have official support for Mac OS X, and it is not recommended that you use Mac OS X for the workshop.

Software Requirements

The compiler has a few software requirements that you will want to make sure are installed on your machine. We will divide these into required packages that Co-dfns assumes are installed and will be necessary to a complete and correct Co-dfns installation and optional packages that are not required to get Co-dfns up and running, but which may help you work with Co-dfns.

The following packages are required on **all operating systems**:

- Dyalog APL 15.0 64-bit Unicode edition
- PGI C/C++ Workstation with Accelerator

- LibreSSL (preferably the latest release, currently at 2.4.2)

The following software is optional but recommended for **all operating systems**:

- NVIDIA CUDA Toolkit 7.5 or greater (CUDA 8 RC is preferred for newer distributions)

The following packages are required for **Microsoft Windows**:

- Visual Studio 2015 with C++ installed (this requires a custom installation); any version of 2015 with C++ is fine

The following packages are required for **GNU/Linux distributions**:

- A recent version of GCC

The Visual Studio and GCC requirements are to ensure that you have a basic, operating system supported compiler with which to test the Co-dfns compiler and ensure working behavior. It allows you to compile basic code to the CPU for your specific operating system. The performance of this code will be significantly less than other options, but gives a baseline for compiler interaction.

If you are not interested in compiling for the GPU, or if you have a computer that does not have a supported graphics card for doing CUDA programming, you can still see more performance by using the Intel C compiler instead. For those who wish to **only compile Co-dfns for the CPU** the following is a recommended software item:

- Latest Intel C/C++ workstation compiler suite

Obtaining PGI Compilers

The PGI compiler is critical to install and configure if you want to be able to do GPU programming with Co-dfns. If you are using Windows, PGI offers free trial software from their website:

http://www.pgroup.com/support/trial.htm

Make sure to select the C/C++ option with Accelerator support. We recommend the Workstation edition.

On Linux, NVIDIA offers the OpenACC toolkit that includes the PGI compiler, which is the recommended way of getting the compiler on Linux:

https://developer.nvidia.com/openacc-toolkit

Obtaining LibreSSL

LibreSSL is utilized by the Co-dfns compiler for random computation and other cryptographically oriented operations. It is available directly from their website:

http://www.libressl.org/

Additionally, we provide mirrors of the latest release with which the Co-dfns compiler has been tested for each Co-dfns release:

https://github.com/arcfide/Co-dfns/releases/latest

Obtaining the CUDA Toolkit

While the CUDA toolkit is not strictly required, it contains utilities and tools that we recommend you have available to you when working with Co-dfns applications. You can obtain the toolkit from NVIDIA's developer zone:

https://developer.nvidia.com/cuda-toolkit

Obtaining Intel C compiler

This is **not required** if you intend to use the PGI compiler to do GPU programming. However, if you are not able to use the PGI compiler, or if you wish to use the Co-dfns compiler for only CPU based code, then you should obtain the Intel C compiler from one of Intel's products:

https://software.intel.com/en-us/intel-parallel-studio-xe

Intel offers free trial versions of their compiler suite that you may use

Installation

All of the above software should be installed according to standard installation for your operating system. The two exceptions are that Visual Studio 2015 must be installed with the C++ software packages included, which requires a custom installation, and there is no standard installation procedure for LibreSSL on windows, so we will cover that separately in this document.

Obtaining and Installing the Co-dfns Compiler

After downloading and installing the prerequisites, the Co-dfns distribution can be obtained by downloading the latest tarball/compressed zip package from the following link:

https://github.com/arcfide/Co-dfns/releases/latest

Download the source package in either .zip or .tar.gz form and extract this somewhere on your machine. This will be the primary working directory. Once this is complete, we must ensure that LibreSSL is visible to Co-dfns if you are on Microsoft Windows. On Linux Co-dfns will use /usr/local/lib64 or the standard paths to find LibreSSL.

Initial LibreSSL Configuration Steps for Microsoft Windows

After the Co-dfns distribution has been extracted, you will need to make sure that you have LibreSSL installed and visible to your Windows machine. The easiest way to do this is to make sure that the *libcrypto-38* files from the LibreSSL package are copied into the root of the Co-dfns directory.

CONFIGURING YOUR CO-DFNS INSTALLATION

The Co-dfns compiler has a set of variables that will need to be set to appropriate values for your operating system and development environment. Common examples of these are found in the *config.def.dyalog* file. Configure your Co-dfns compiler by copying this file to *config.dyalog* and editing the contents to set the appropriate values according to the comments in the file. In particular, make sure that you choose the compiler that you want to use as a backend compiler and, if you are on Windows, that you have the appropriate paths for compilers set.

CONFIGURING YOUR DYALOG APL ENVIRONMENT

The testing suite for Co-dfns assumes that you are using `⎕IO←0` so you will want to make sure that you have your Dyalog environment set to use `⎕IO←0` and not the default `⎕IO←1`. You will also want to make sure that your migration level and other system variables are set to their default settings.

ACCESSING THE CO-DFNS ENVIRONMENT

After successfully configuring the Co-dfns compiler, the easiest way to get the Co-dfns compiler into a workspace is to use the included *load.dyapp* SALT application. This assumes that your current working directory is the Co-dfns distribution directory.

On **Windows**, the easiest thing to do is to double-click on the *load.dyapp* file and Dyalog will automatically ensure that you are in the right working directory and load the Co-dfns compiler into a clear workspace.

On **Linux**, you should *cd* into the Co-dfns directory and then you can use the following to load the environment:

```
$ dyapp=./load.dyapp mapl
```

From there you can create files and run the testing suite, among other things.

RUNNING THE TEST SUITE

The included test suite is an useful check to see whether your system is running the way you expect or not. There are two primary methods for running tests, both of which assume that your current working directory is the Co-dfns distribution directory.

You can run the entire test suite by running the *test.dyapp* SALT application. On **Windows** this is accomplished by double clicking on the *test.dyapp* file. On Linux you can use the following command once you have *cd*'d into the Co-dfns directory:

```
$ dyapp=./test.dyapp mapl
```

Please remember that this will run the entire test suite, which is quite large.

It is often more convenient at first to test only a few tests to see that you have set up your environment correctly. To do this, load the Co-dfns environment (see the above section) and use the *util.test* function

to run individual tests found in the *tests* directory. As an example, you can test whether your configured compilers are compiling without error by running the following:

```
util.test'compile'
```

You can then do a simple test of the rest of the compiler suite by using the '*identity*' test cases. After this, you can try out the various test cases and see what works and what doesn't, or you can use the *test.dyapp* file to run the entire test suite.

You will find the intermediate files in whatever your *BUILD* directory is set to. By default this is the *build* directory in the Co-dfns distribution, but this could have been changed in your *config.dyalog* file. If you receive errors during testing, you will find a log of the details of the error in this build directory.

When running the tests, the test suite will consider tests using compilers that are not active on your configuration as passed, even though it does not run them. A future version of the test suite may mark these as skipped rather than passed.

Note: while most of the tests in the test suite should pass, some of them test features that are not currently implemented in the Co-dfns compiler. Therefore, while a large number of failed tests is cause for concern, a small number of failed tests does not necessarily mean that your compiler is misconfigured.

TROUBLESHOOTING

Here are some common problems that might arise while installing and using the Co-dfns compiler, and their remedies.

INCOMPATIBLE VERSION OF GCC FOR THE PGI/CUDA TOOLKIT

When running a recent version of GCC on some versions of Linux, the GCC version is too high for the PGI's built-in CUDA toolkit. In general, this isn't too much of a problem as long as you are using the latest CUDA toolkits from inside of the PGI compiler, but if you encounter this error in your logs, then you can fix this issue by commenting out the error lines in the host *config.h* file referenced in the error. This has not caused any issues with the current versions of GCC. Other options are to downgrade your default GCC version to something that is officially supported by CUDA.