This document will guide you through the installation of RoseACC and your first "hello-world".

1 Installation

1.1 Software Dependencies

- **ROSE Compiler** is the foundation of RoseACC. It enables to transform C/C++ codes into an Intermediate Representation (**IR**). ROSE Compiler’s IR is an Abstract Syntax Tree (**AST**). RoseACC modifies the AST, replacing annotated region of codes with calls to libOpenACC and generating OpenCL C kernel. Finally, ROSE Compiler produces source-code from the edited AST. ROSE Compiler has its own dependencies:
  
  - **GCC 4.4.3 to 4.4.7** is needed to compile ROSE Compiler (these are relatively old version GCC).
  - **Boost 1.42.0 to 1.47.0** is used by ROSE Compiler (these are relatively old version Boost).

Using GCC 4.4.7, causes issues in boost/thread/xtime.h ([bug report](https://bugs.open-acc.org/1881)), a simple fix is to prefix TIME.UTC with an underscore...

- **SQLite 3** is used by RoseACC, both the compiler and libOpenACC runtime. SQL is used to store the description of generated kernels and profiling information collected by OpenCL.

- **OpenCL** is a runtime for computation accelerators.

1.1.1 Installation of the dependencies

**GCC** We assume that you have compatible version of GCC on your system (to check GCC version: `gcc -v`). If it is not the case consider to install GCC from source, it is well documented online and lead to less configuration issues than packet managers.

**Boost** is often already installed. However, you should check the version in `boost/version.hpp`. For more installation tips: see [http://www.boost.org/doc/libs/1_45_0/more/getting_started/unix-variants.html](http://www.boost.org/doc/libs/1_45_0/more/getting_started/unix-variants.html)

```bash
wget http://downloads.sourceforge.net/project/boost/boost/1.45.0/boost_1_45_0.tar.gz
tar xzf boost_1_45_0.tar.gz
cd boost_1_45_0
./bootstrap.sh --prefix="/opt/
./bjam install
```

Listing 1: Installation of Boost
SQLite3 might already be installed, else:

```bash
wget http://www.sqlite.org/2014/sqlite-autoconf-3080600.tar.gz
tar xzf sqlite-autoconf-3080600.tar.gz
cd sqlite-autoconf-3080600
./configure --prefix=/opt/
made install
```

Listing 2: Installation of SQLite3

OpenCL headers are sufficient if you want to use the RoseACC Compiler on a machine without accelerator. They can be found on Khronos OpenCL website: [https://www.khronos.org/registry/cl/](https://www.khronos.org/registry/cl/)

NVIDIA OpenCL Please refer to NVIDIA’s documentation: [https://developer.nvidia.com/opencl](https://developer.nvidia.com/opencl)


1.2 RoseACC workspace

The RoseACC workspace is a repository grouping the components of RoseACC.

- **ROSE Compiler** provides modules:
  - **DLX**: Directive-based Language eXtensions. Framework to create language extension using compiler directives.
  - **KLT**: Kernel from LoopTrees. Generates kernel from a loop-nest abstraction, need to be specialized for a programming model (ie. OpenACC) and target language (ie. OpenCL).
  - **MFB**: Multiple File Builder & **MDCG**: Model Driven Code Generation. High-level AST manipulations.

- **RoseACC Compiler** is the implementation of OpenACC using **DLX**, **KLT**, **MDCG**, and **MFB**.

- **libOpenACC** is RoseACC runtime. It provides the primitive to move data and execute code on accelerators. It uses OpenCL Runtime.

- **tests** for the runtime and the compiler.

1.2.1 Setting up RoseACC Workspace

The RoseACC workspace is hosted on GitHub at [https://github.com/tristanvdb/RoseACC-workspace](https://github.com/tristanvdb/RoseACC-workspace)

```bash
git clone git@github.com:tristanvdb/RoseACC-workspace.git
cd RoseACC-workspace
git submodule init
git submodule update
```

Listing 3: Checkout the RoseACC workspace.

We provide a setup script to build, configure, compile and install all components of RoseACC. It takes a list of pathes and the number of parallel processes to use for compilation.

```bash
./setup.sh build_dir install_dir $BOOST_HOME $OPENCL_INCDIR $OPENCL_LIBDIR $SQLITE_INCDIR $SQLITE_LIBDIR 12
```

Listing 4: Setup script usage, either set or substitute the environment variables.
2 Test RoseACC

Now that the RoseACC workspace is setup, we are going to go through the tests module.

First, verify that the `openacc` and `roseacc` are available (If not check the `PATH` and `LD_LIBRARY_PATH`).

```
./openacc -l
```

### Table: Environment Variables

<table>
<thead>
<tr>
<th>ID</th>
<th>Env. name</th>
<th>#</th>
<th>Full Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>GTX-460</td>
<td>0</td>
<td>GeForce GTX 460</td>
</tr>
<tr>
<td>1</td>
<td>E5-2620</td>
<td>0</td>
<td>Intel (R) Xeon (R) CPU E5-2620</td>
</tr>
<tr>
<td>2</td>
<td>XEONPHI</td>
<td>0</td>
<td>Intel (R) XeonPhi(TM)</td>
</tr>
</tbody>
</table>

Listing 5: The `openacc` binary can be used to check the system configuration.

If you see a warning saying "Unrecognized device: ...", it is because we "hard-code" all supported devices to permit individual devices selection. In this case please let us know, adding a device only take us a few minutes!

2.1 libOpenACC Runtime

```
make -C ~/RoseACC-workspace/build_dir/tests/libopenacc check
```

Listing 6: Executes (limited) unit tests for libOpenACC (RoseACC’s OpenACC runtime).

2.2 RoseACC Compiler

```
make -C ~/RoseACC-workspace/build_dir/tests/roseacc check
```

Listing 7: Executes RoseACC Compiler tests.

3 RoseACC: Hello-World

Finally, we will construct a small "Hello-world" example. This example shows how to use RoseACC once it is installed. It shows how to build a Makefile to have RoseACC compile an application annotated with OpenACC.

3.1 Sequential "Hello-World"

You can create the files as presented below or clone the repository `git clone git@github.com:tristanvdb/RoseACC-getting-started.git`

```makefile
all: hello-world
check: check-hello-world

hello-world.o: hello-world.c
  gcc -DSIZE=4096 -c hello-world.c -o hello-world.o

hello-world: hello-world.o
  gcc hello-world.o -o hello-world

check-hello-world: hello-world
  ./hello-world
```

Listing 8: RoseACC-getting-started/Makefile
#include <stdlib.h>

#ifndef SIZE
#error "CPP macro SIZE should be defined"
#endif

int main() {
    float a[SIZE];
    float b[SIZE];
    float c[SIZE];

    int n = SIZE;
    int i;

    // initialization loop
    for (i = 0; i < n; i++) {
        a[i] = i;
        b[i] = n-i;
        c[i] = 0.;
    }

    // compute loop
    for (i = 0; i < n; i++) {
        c[i] = a[i] + b[i];
    }

    // check loop
    float err = 0.;
    for (i = 0; i < n; i++) {
        err += abs(n-c[i]);
    }

    if (err < 10e-6) return 0;
    else return 1;
}

Listing 9: RoseACC-getting-started/hello-world.c

3.2 OpenACC version

Currently, RoseACC is focused on explicit parallelization and requires the user to provides all necessary information. The only exception is for data when they are declared with array types.

In Listing 10, we surround the compute loop by three directives:

• **data** to move data to and from the device. This directive is optional, it enables to have data persisting on the device between parallel regions.

• **parallel** to offload a region of code onto an accelerator. This is an explicitly parallelized region of code. OpenACC **kernel** regions, used for implicit parallelism are not available for RoseACC.

• **loop** to mark a loop as parallel.

Listing 10: RoseACC-getting-started/hello-world.c
In this example, the loop is distributed across the gangs (coarse grain parallelism) and workers (fine grain parallelism) of the OpenACC device. Because of RoseACC code generation technique the product of the number of gangs and workers has to be equal to the size of the loop. More details about the code generation technique:

- RoseACC-workspace/tests/roseacc/parallel/README.md
- RoseACC-workspace/tests/roseacc/extensions/tiles/README.md

To compile, link, and execute your Hello-world you can add the following to the Makefile:

```makefile
ACCFLAGS=-Droseacc:desc_format=static_data --roseacc:compile=true
ACC_INC_PATH='openacc --incpath | tail -n1'
ACC_LIB_PATH='openacc --libpath | tail -n1'
ACC_LIBS='openacc --libs | tail -n1'

all: hello-world hello-world-acc

check: check-hello-world check-hello-world-acc

# ...

rose_hello-world-acc.o: hello-world-acc.c
    roseacc -DSIZE=4096 $(ACCFLAGS) $(ACC_INC_PATH) -c hello-world-acc.c

hello-world-acc: rose_hello-world-acc.o hello-world-acc-data.o hello-world-acc.cl
    gcc -o hello-world-acc $(ACC_LIB_PATH) rose_hello-world-acc.o
    hello-world-acc-data.o $(ACC_LIBS)

check-hello-world-acc: hello-world-acc
    ./hello-world-acc
```

Now make sure that your PATH and LD_LIBRARY_PATH contains the RoseACC install directory, then make check-hello-world-acc.

If you have an issue checkout the openacc branch at https://github.com/tristanvdb/RoseACC-getting-started/tree/openacc.