Brown Deer Technology

Application Note: Building COPRTHR SDK for Intel MIC

1 Introduction

The COPRTHR SDK supports the Intel MIC architecture and may be used on platforms with Intel Xeon Phi accelerators. This Application Note provides details for building and configuring COPRTHR on these platforms. Support requires the "Freewill" release of the COPRTHR SDK v1.6, as well as Standard components of the Intel software stack for MIC should also be installed prior to beginning.

This Application Note provides details on the following topics:

- Building basic COPRTHR SDK support for a MIC host
- Building COPRTHR SDK for MIC native execution
- Cross-compiling kernels for MIC native execution
- Configuring a CLRPC server for networked MIC accelerators

---

2 Building Basic COPRTHR SDK Support for a MIC Host

The basic components of the COPRTHR SDK may be used to target a MIC accelerator in the same way that it is used to target a GPU. In particular, the STDCL API and clcc tools can be used to develop portable code and taken together provide a powerful code development tool.

2.1 Configure the package

2.1.1 For root installation as a priveleged admin

If you have root access, the build itself should work with all of the defaults. The target install directory will be `/usr/local/browndeer`. If the Intel compiler is the default C/C++ compiler used, at the present time it may be necessary to disable Fortran bindings in libstdcl using `--with-fortran=` with no value provided following the equal (=) sign.

The complete commandline would simply be,

```
./configure --with-fortran= --enable-mic-cross-compile
```

If the pre-requisites libelf (libelf-0.8.13), libconfig, or libevent have been installed in non-standard locations, the options `--with-libelf`, `--with-libconfig`, and `--with-libevent` may be used to make adjustments. See `./configure --help` for a full listing of customizations.

2.1.2 For user installation as an unpriveleged user

If you do not have root access on the platform it is still possible to do a private "user install". The following paths are assumed to be defined: `LIBELF_DIR`, `LIBCONFIG_DIR`, `LIBEVENT_DIR`, corresponding to the install locations of libelf (libelf-0.8.13), libconfig, and libevent. Additionally we will assume the target installation directory is `COPRTHR_INSTALL_DIR`.

```
./configure --prefix=$COPRTHR_INSTALL_DIR --with-libelf=$LIBELF_DIR \
    --with-libconfig=$LIBCONFIG_DIR --with-libevent=$LIBEVENT_DIR \
    --with-opencl-icd-path=$COPRTHR_INSTALL_DIR/etc/OpenCL/vendors \
    --enable-user-install --with-fortran= --enable-mic-cross-compile
```

See `./configure --help` for a full listing of customizations.

2.2 Build and install the package

This part is easy. If configured for a root installation type,

```
make
sudo make install
```

If configured for a user installation type,

```
make
make install
```

That's it. Note that `make clean` will remove files generated by the build process and `make distclean` will restored things to their pre-configure state.

2.3 Update your PATH and LD_LIBRARY_PATH

Once installed, ensure that your environment variables PATH and LD_LIBRARY_PATH have been updated as directed by the messages printed to the screen when you typed `make install`.

... No, seriously, update these environment variables. Failure to do so causes at least 50% of the emails asking why the SDK does not work.

2.4 Configuration of ocl.conf

The file `ocl.conf` controls the OpenCL platforms seen by the COPRTHR loader `libocl.so` which should be used in place of the Khronos defined loader, `libOpenCL.so`. On some platforms it may make sense to alias libocl.so to libOpenCL.so,

```
ln -s libocl.so libOpenCL.so
```

This step is not done by default.

The search path for the controling ocl.conf file provides for increasing specialization from the system level to the individual user, and specifically follows:

```
./ocl.conf
./.ocl.conf
$HOME/ocl.conf
$HOME/.ocl.conf
/etc/ocl.conf
```

The ocl.conf file provides an opportunity for precisely controling the OpenCL configuration of a given system. In order to ensure that the Intel OpenCL SDK is seen by the loader, the appropriate line should be added to the platforms section of the controling ocl.conf file. Failure to properly configure this file is a common cause of problems. The ocl.conf file is very flexible and easy to setup. See Section 3 of the COPRTHR Primer for more details.

2.5 Testing

Following the installation, a quick test may be performed to indicate whether the installation was successful and properly configured.

From the root directory of the COPRTHR SDK package type,

```
make quicktest
```

The results should be self-explanatory.

---

3 Building COPRTHR SDK for MIC Native Execution

The COPRTHR SDK may be built for MIC native execution. However, some limitations arise due to the fact that the MIC accelerator cards will be running a stripped-down Linux kernel. An example is just-in-time (JIT) compilation support from an application running native since the Intel compiler will not be available on the card, only on the host as part of the standard Intel compiler installation. Nevertheless most of the run-time libraries are supported including the core libraries libstdcl, libcoprthr, libcoprthr_opencl, and libclrpc. The lack of JIT compilation support is easily overcome since COPRTHR provides robust support for pre-cross-compiled kernels.

3.1 Configuration

Building the COPRTHR SDK for MIC native execution requires cross-compilation on the host using –enable-user-install configuration option. In order to support the cross-compilation the path to the location of the required MIC libraries must be specified. This should be the location where the libimf.so library built for MIC is installed on the host as part of the Intel SDK. This directory will be designated MIC_LIBS_DIR.

Prior to building COPRTHR it will be necessary to build the packages libelf (libelf-0.8.13), libconfig, and libevent for the MIC architecture, and specify the location(s) for these packages using MIC_LIBELF_DIR, MIC_LIBCONFIG_DIR, MIC_LIBEVENT_DIR, respectively.

The location where the COPRTHR SDK build should be (temporarily) installed is designated IC_COPRTHR_INSTAL_DIR. This is a temporary location since all of the packages will need to be copied over to the MIC accelerator card.

The configuration is then performed with the commandline,

```
CC=icc CFLAGS=-mmic ./configure --prefix=$MIC_COPRTHR_INSTALL_DIR \
    --with-libelf=$MIC_LIBELF_DIR --with-libconfig=$MIC_LIBCONFIG_DIR \
    --with-libevent=$MIC_LIBEVENT_DIR \
    --with-opencl-icd-path=$MIC_COPRTHR_INSTALL_DIR/etc/OpenCL/vendors \
    --enable-user-install --with-fortran= --host=k1om-unknown-linux-gnu \
    --with-lib-mic=$MIC_LIBS_DIR
```

---

4 Cross-Compiling Kernels for MIC Native Execution

The absence of JIT compilation support when running with MIC native execution can be mitigated using cross-compilation support for kernels provided by the COPRTHR SDK. This cross-compilation support is simply an extension of the robust support provided for a standard compilation model and workflow - something OpenCL lack. There are two approaches - one that works now, and one that will hopefully work in the near future. We will mention first what does not work and why.

The obvious solution for the lack of native JIT compilation support for MIC is to use the COPRTHR `clcc` compiler tools to produce a linkable object file with embedded kernels for MIC and then link this pre-compiled object file into the application cross-compiled for MIC. At present, there is an issue with this approach since the object file that `clcc` produces will be "marked" as an ELF file for the x86_64 host even if the embedded kernels are for the MIC accelerator. This reflects the standard offload use-case.

As a work-around, the ability of `clcc` to generate the raw pre-compiled kernel binary, using the `--dump-bin` option, can be exploited to accomplish something very close to what we would like to do. The key is that the STDCL `clopen()` call supports both source and binary files, and will automatically detect a raw binary and load the pre-compiled kernels dynamically.

Given kernel code in the file `my_kernel.cl`,

```
] clcc --coprthr-cc -mtarget=mic --dump-bin my_kernel.cl
```

will produce the binary file `my_kernel.clbin.3.mic`.

Then, from within host code compiled for MIC native execution, the pre-compiled binary can be loaded using a single STDCL call to `clopen()`,

```
...
void* clh = clopen(stdacc,"my_kernel.clbin.3.mic",CLLD_NOW);
...
```

For more information see the STDCL API Reference and the COPRTHR Primer.

---

5 Configuring a CLRPC server for networked MIC accelerators

MIC accelerators may be accessed directly over a network which enables the capability of setting up CLRPC servers to form multi-node networks of OpenCL devices accessible through a single STDCL context. Building the COPRTHR SDK for MIC native execution will build the necessary library and server executable by default. In this section the steps required for setting up such a network will be discussed.

The COPRTHR SDK should be installed on each MIC accelerator card which typically will be running a stripped-down Linux kernel and an essentially bare filesystem requiring you to "bring what you need" in order to execute a simple program. On each accelerator card, run a CLRPC server using,

```
] clrpcd -i eth0
```

On any host platform from which the accelerator cards are accessible, the `clprc` section of the controling `ocl.conf` file should be modified so that the section itself is not commented out, and a list of IP addresses for the MIC accelerators running CLRPC servers is provded.

As an example, assume that we have four (4) MIC accelerators within a platform configured such that the accelerator cards have IP address 10.0.2.1 - 10.0.2.4. The required modification to the `ocl.conf` file would be the addition of the section below:

```
clrpc = {
    enable = "yes";
    servers = (
        { url="10.0.2.1:2112" },
        { url="10.0.2.2:2112" },
        { url="10.0.2.3:2112" },
        { url="10.0.2.4:2112" }
    );
};
```

The above modifications will cause the `libocl.so` loader to provide an application code using OpenCL a unique OpenCL platform for each server represented the exported resources. Using the STDCL API the resources are combined to form a single context. The result is that the application code will "see" a context, `stdnpu`, containing four (4) MIC accelerators with device numbers 0,1,2, and 3.

So for example, a STDCL query to determine the number of available devices would show `ndev` equal to 4,

```
size_t ndev = clgetndev(stdnpu);
```

The STDCL context `stdnpu` may be used like any other context and applications designed with a multi-device capability will be able to use all of the networked devices.

For more information on CLRPC, see Section 7 of the COPRTHR Primer.

Document revision 16.0.0.0