Table of Contents

Under construction

In this page we describe our experience with running Geant4 applications on Intel Xeon Phi systems. This page provides instructions to compile Geant4 toolkit and applications to be run on MIC architectures.

Preliminary Consideration

Intel Xeon Phi (aka MIC) is a co-processor designed to benefit parallel applications thanks to its large hardware thread count and wide registers. While not all applications may benefit from this kind of architecture, Geant4 toolkit can be compiled for Xeon Phi architectures and applications can be run on the co-processor. In its current implementation its form factor is a PCI express card that is added to a Intel Xeon host. Differently from other accelerators (e.g. GPGPUs) the MIC has an embedded linux and it can be addressed as an additional node in the system (i.e. it is possible to `ssh` to the card).

To use a Xeon Phi coprocessor you need access to a machine with at least one Xeon Phi available and the intel C/C++ compilers (`icpc`). Currently only Intel compiler can cross-compile software for MIC architectures.

In the following we assume you have access to a Xeon Phi and you can login into it. For a general introduction to Xeon Phi, installation guides, code examples, and best practices we suggest the book Parallel Programming and Optimization with Intel Xeon Phi Coprocessors by Colfax International.

Offload vs Native mode

Xeon Phi provide two modes of operation: *native* and *offload*. In the former an application is (cross) compiled and executed on the coprocessor. For example it is possible to install the application on the card, `ssh` to it and start the application. In the latter the application source code is instrumented with a set of directives and function calls that, similarly to what happens to GPGPUs, instruct the compiler to produce binary code that will be *downloaded* and executed on the card.

For Geant4 applications we prefer the use of native mode for a set of reasons:

1. No code modification is needed, the code that runs on the host can be, in the majority of the cases, recompiled and run on the MIC without modifications.
2. The offload mode implies some overhead due to the necessity of copying, during a Geant4 simulation, the data to the card and copying back the results. This overhead can be large compared to the actual processing on the card, only an application-specific code instrumentation can be done and no general *recipe* can be suggested.
3. Xeon Phi supports Intel MPI parallelization framework. Geant4 supports, since several years, MPI to perform process-level parallelization. In version 10.0 MPI and MT can co-work. Thus ranks can be started on the host and on the Xeon Phi to collaboratively perform a simulation. Given the small communication between ranks needed during the event loop the overhead in this case is minimal. The suggested way to run in this way is to start a single rank on the card and use multi-threading to achieve a further level of parallelization.

NFS Support for Xeon Phi

A Geant4 application needs access to some support files. For example all applications need access to data files (Geant4 DataBases) that contain, for example, electromagnetic data or neutron cross-sections. In addition other support files may be needed for specific applications: input generator files, macros files. Finally almost all applications produce at least one output file (at minimum the cout/cerr log).

It is thus necessary to provide a simple mechanism to provide to the Xeon Phi coprocessor the input files and a way to retrieve the output produced by a native application. Luckily Xeon Phi supports NFS filesystems. It is **strongly** suggested to configure the system to export one or more directories from the host to the MIC. This NFS area should at least contain the Geant4 DataBases and should contain an area where results can be written.

Histograms and ntuples

Often results from a Geant4 application are collected in the form of histograms and ntuples. A package very popular in high-energy-physics is ROOT. As any other software to be used on Xeon Phi it si required that this is recompiled for MIC architectures. At the time of writing this we have not yet had experience managing to cross-compile the entire ROOT system for Xeon Phi. While some success have been done recompiling the minimum required to write ROOT files, the process is not simple.

However, starting with Geant4 version 10.0, we provide native histograms and ntuple support in Geant4 that is **compatible with ROOT** and **supports Xeon Phi**. Using `g4analysis` module you will be able to create output files on the MIC containing histograms and simple ntuples in a variety of output formats (ROOT, AIDA-XML, ASCII-CSV). Clearly not all ROOT features are supported and you should refer to the Geant4 examples on how to use `g4analysis` (an important note: if you use this method histograms from threads are automatically merged in a single output file).

Visualization

For Geant4 application all (interactive) visualization support with the exception of defaults, should be **turned off** (e.g. no Qt or other advanced drivers). However drivers that produce a file output and do not have external dependencies (HepRep, VRML, ASCIITree) should be supported (*not tested yet*).

Building Geant4 toolkit for Xeon Phi

The process of compiling Geant4 toolkit is the same as for the host, with the difference that the `-mmic` compilation flag should be added and `icpc` should be used as compiler. CMake is used to configure the compilation of Geant4 and the process is carried out on the host.

In the following we assume that:

1. The directory `/geant4-sw/data` is NFS exported to the Xeon Phi and already contains Geant4 databases
2. To simplify the handling of the binaries a **static** installation is suggested. Only `.a` libraries are built.
3. A bash family shell will be used as an example.
4. Intel tools (compilers, libraries) are installed in the default location under `/opt/intel`. If this is not the case, you will need to modify this path in all the following instructions.

Important note on compatibility

We have compiled and tested Geant4 with Intel Compiler (icc / icpc) version 16 and MPSS (Intel Manycore Platform Support Stack) version 3.4. Please note that due to a bug in icc Geant4 **does not** compile with the latest MPSS version 3.6.

Please note that to compile latest versions of C++ you need compatibility layer of icc being setup with gcc 4.9.x, note the code will not compile if you have compatibility layer older or never than that. To see your icc compatibility layer type:

```
iccp -v
```

. The output should match:

```
icpc version 16.0.1 (gcc version 4.9.3 compatibility)
```

gcc version **must** match 4.9.x

See: Geant4 hypernews

NEW **Preliminary tests seem to indicate that the most recent MPSS version 3.6.1 has solved the problems reported with 3.6 and the bug has been solved** NEW

Configuration

CMake supports cross-compilation if you provide a **toolchain file**. The purpose of this file is to instruct cmake where the utilities needed to cross-compile are located and to setup the environment.

The content of the toolchain file we use:

```
# this one is important
SET(CMAKE_SYSTEM_NAME Linux)
#this one not so much
SET(CMAKE_SYSTEM_VERSION 1)

# specify the cross compiler
SET(CMAKE_C_COMPILER icc)
SET(CMAKE_CXX_COMPILER icpc)
# where is the target environment
SET(CMAKE_FIND_ROOT_PATH  /opt/intel/composerxe)

# search for programs in the build host directories
SET(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
# for libraries and headers in the target directories
SET(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
SET(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
```

Copy and adapt the above lines in a text file (e.g. `mic-toolchain-file.cmake`).

Before proceeding to configure Geant4 with cmake you must setup the environment as follows:

```
$ export CC=icc
$ export CXX=icpc
$ export LD=/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-ld
$ export AR=/usr/linux-k1om-4.7/bin/x86_64-k1om-linux-ar
$ export LDFLAGS=-mmic
$ export CXXFLAGS=-mmic
$ export CFLAGS=-mmic
```

Note: you may need to specify another archiver/linker (e.g. `xiar`) if you want to use some advanced compilation options (e.g. `-ipo`). However the ones in the example are good for general purpose configuration.

You can now proceed with the Geant4 configuration in a build directory:

```
cmake -DGEANT4_BUILD_MULTITHREADED=ON \
           -DGEANT4_USE_SYSTEM_EXPAT=OFF \
           -DGEANT4_INSTALL_DATA=OFF -DGEANT4_INSTALL_DATADIR=/geant4-sw/data \
           -DCMAKE_C_COMPILER=${CC} -DCMAKE_CXX_COMPILER=${CXX} -DCMAKE_LINKER=${LD} -DCMAKE_AR=$
           -DCMAKE_TOOLCHAIN_FILE=<path-to-toolchain-file>/mic-toolchain-file.cmake \
           -DBUILD_SHARED_LIBS=OFF -DBUILD_STATIC_LIBS=ON \
          [... other G4 configuration options ...] \
          <path-to-g4-source>
```

If you need GDML support read the advanced section on Xerces-C.

Compilation

You can now proceed with compilation:

```
[g]make [... options ...]
```

Advanced: compiling Xerces-C for Xeon Phi

If your application requires the use of GDML to read/import geometry you need an additional step to cross compile Xerces-C libraries and activate GDML support in Geant4.

Perform the **setup of environment variables as previously described**. Then compile Xerces-C to be used on the MIC:

```
$ cd <xercescdir>
$ export CXXFLAGS="$CXXFLAGS -w -O2 -DNDEBUG"
$ export CFLAGS="$CFLAGS -w -O2 -DNDEBUG"
$ ./configure --prefix=<some-inst-area-for-xercesc> --host=x86_64-intel-linux CC=icc CXX=icpc --d
$ [g]make [...]
$ [g]make install
```

You can now configure and compile Geant4 as described in the previous section, adding the following cmake options: `-DGEANT4_USE_GDML=ON -DXERCESC_INCLUDE_DIR=<some-inst-area-for-xercesc>/include -DXERCESC_LIBRARY=<some-inst-area-for-xercesc>/lib/libxerces-c.a`

Building an application for Xeon Phi

Once Geant4 toolkit as been compiled you can configure, build and install the application.

Configuration

Setup the environment as described when compiling Geant4. Create a build directory for the application and then configure it with cmake:

```
cmake -DCMAKE_LINKER=${LD} -DCMAKE_AR=${AR} \
            -DCMAKE_TOOLCHAIN_FILE=<path-to-toolchain-file>/mic-toolchain-file.cmake \
            -DGeant4_DIR=<g4-build-directory-from-previous-step> \
            [... other application specific options ] \
            <path-to-application-source>
```

Compilation and installation

You can now compile the application with `[g]make [...]`. To install the application, copy the executable and needed support files, in the NFS area visible from the Xeon Phi.

Running on Xeon Phi

To run the application you can:

```
$ ssh mic0
$ #You may want to put the following in a setup.sh script
$ basepath=<where-G4-databases-are>
$ export G4LEVELGAMMADATA=${basepath}/PhotonEvaporation
```

```
$ export G4NEUTRONXSDATA=${basepath}/G4NEUTRONXS
$ export G4LEDATA=${basepath}/G4EMLOW
$ export G4NEUTRONHPDATA=${basepath}/G4NDL
$ export G4RADIOACTIVEDATA=${basepath}/RadioactiveDecay
$ export G4ABLADATA=${basepath}/G4ABLA
$ export G4PIIDATA=${basepath}/G4PII
$ export G4SAIDXSDATA=${basepath}/G4SAIDDATA
$ export G4REALSURFACEDATA=${basepath}/RealSurface
$
$  export LD_LIBRARY_PATH=/opt/intel/lib/mic:$LD_LIBRARY_PATH
$ cd <where-executable-is>
$ export G4FORCENUMBEROFTHREADS=max #or maximum number given the memory budget
$ [.... launch application as normal ...]
```

Note that `LD_LIBRARY_PATH` should contain the path where the following three libraries are installed: libimf.so, libsvml.so, libirng.so. For example `/opt/intel/lib/mic` can be itself an NFS exported path from the host.

An alternative way to run an application without an explicit `ssh` is the use of the `micnativeloadex` application. From the host, you can run an application on the first mic card as follows from the build directory.

```
$ export SINK_LD_LIBRARY_PATH=/opt/intel/lib/mic
$ micnativeloadex <Application-Name> -e "<remote-environment, for example DB databases>" -a "<app
```

Note: that in this case you should pass via the `-e` argument all Geant4 DataBases environments variables. With this method there is no need to explicitly copy the application in a NFS area. Note however that the application will be run on the card as a special user (*micuser*) in a temporary directory, thus if your application need some input files in the working directory this method will not work. Also it has been reported that with this method you should *not use* all available threads, but leave a core free (needed for host/card communication), to avoid degradation of performances. This is useful for fast test or if the only output of the application is to cout/cerr (that appears back on the host console).

-- AndreaDotti - 14 Oct 2014

This topic: Geant4 > XeonPhiSupport
Topic revision: r5 - 2016-09-02 - AndreaDotti