DE GRUYTER OPEN

BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **17**, No 5
Special issue with selected papers from the workshop
"Two Years Avitohol: Advanced High Performance Computing Applications 2017"

Sofia • 2017

Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2017-0059

On the Use of Large Intel Xeon Phi Clusters for GEANT4-Based Simulations

Nevena Ilieva[1], *Elena Lilkova*[1], *Leandar Litov*[2], *Borislav Pavlov*[2], *Peicho Petkov*[2]

[1]*Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria*
[2]*Sofia University "St. Kl. Ohridski", Physics Faculty, 1000 Sofia, Bulgaria*
E-mails: *nevena.ilieva@parallel.bas.bg helen.lilkova@yahoo.com Leandar.Litov@cern.ch Borislav.Pavlov@cern.ch peicho@phys.uni-sofia.bg*

Abstract: *GEANT4 is the basic software for fast and precise simulation of particle interactions with matter. Along the way towards enabling the execution of GEANT4 based simulations on hybrid High Performance Computing* (*HPC*) *architectures with large clusters of Intel Xeon Phi co-processors, we study the performance of this software suit on the supercomputer system Avitohol@BAS, Some practical scripts are collected in the supplementary material shown in the appendix.*

Keywords: *Co-processors, multithreading, acceleration, hybrid architecture, particle-matter interactions, simulations.*

1. Introduction

GEANT4 [1] (GEometry ANd Tracking**)** is a software toolkit for simulation of the interactions of particles with matter. It exploits the object-oriented technology to achieve transparency of the physics implementation, as well as openness to extension and evolution. GEANT4 encompasses a wide set of tools for all the domains of detector simulation, including geometry modeling, detector response, run and event management, tracking, visualization and user interface. An abundant set of physics processes handles the diverse interactions of particles with matter across a wide energy range, as required by GEANT4 multi-disciplinary nature. The GEANT4 source code, libraries and user documentation are freely available [2-4].

GEANT4 is the successor of the FORTRAN-based GEANT series of software toolkits [5], which dates back to 1978. GEANT4 is the first version developed using object-oriented technology and is implemented in C^{++}. GEANT4 can be used as a stand-alone tool or linked as a library. For example, in applications like CMSSW [6] (high-energy physics) and GATE [7] (medical physics), GEANT4 is accessed through a complex framework.

The High Performance Computing (HPC) approach used is based on GRID technology. The largest research facilities in HEP (High-Energy Physics) use the Worldwide LHC Computing Grid (WLCG) infrastructure to perform such simulations. In this approach, different computers simulate different events. The events are fully independent, so no data transfer between different computers is needed. The problems along this way are two-fold:

- With the increasing data volumes, the resources become more and more insufficient, even with structures like WLCG.
- The largest HPC community worldwide does not use new architectures like those involving large number of Intel Xeon Phi accelerators efficiently.

At present, GPU usage in these large-scale computations is not foreseen due to the necessity for severe modifications of the code. For Intel Xeon Phi platform, however, there is no need for doing so in native mode. This is also the mode recommended by the developers [8]. Our long-term goal is relaxing the resources insufficiency bottleneck by opening the emerging HPC hybrid architectures with Intel Xeon Phi co-processors for GEANT4-based simulations.

2. Test architecture

A pilot implementation of the service "GEANT4 on MIC" was realized on the Sofia HPC system Avitohol@BAS [9], with 300 CPU Intel Xeon E5-2650v2, 300 co-processors Intel Xeon Phi 7120P, 9.6 TB memory and LINPACK performance of 264.2 TFlop/s. The operating system is Red Hat Enterprise Linux release 6.7. The co-processors run MPSS version 3.6. GEANT4 package requires a number of additional libraries and packages being configured and installed in a specific way in advance. File-transfer options will be discussed elsewhere, here we shall focus on the installation, compilation and performance of GEANT4 on this particular heterogeneous system.

3. GEANT4 installation

The multithread enabled GEANT4.10.00.p04 library has been installed on Avitohol supercomputer as well as the CMAKE-3.3.2 utility needed to set up and compile user code linked to the GEANT4. Several test programs were compiled and run successfully in multithread version in order to test and validate the correctness of the installation.

The GEANT4 pre-requisites are as follows:

- GEANT4 Version 10.0 patch-04;
- Intel C/C^{++} Version 16.X or higher. We used ICC and ICPP Version 16.0.2 (compatible with gcc Version 4.4.7);
- Intel MPSS (Manycore Platform Support Stack) Version 3.4;
- CMake Version 3.3 or higher;
- C^{++} Compiler and Standard Library supporting the C^{++}11 standard;
- Linux: GNU Compiler Collection 4.8.2 or higher.

We refer to the corresponding section of the Best Practice Guide Intel Xeon Phi v2.0 [11] for performing the following steps:

- GEANT4 configuration and installation;
- GEANT4 cross compilation as a native Xeon Phi application;
- GEANT4 user code compilation and execution on CPU;
- GEANT4 user code compilation and execution on Xeon Phi in native mode.

In the Appendices, some useful scripts are given as supplementary material.

The new version of the suite – GEANTV, which aims for complete parallelization and which is still under development – needs many specific packages to be pre-installed, most of them to be compiled and installed from scratch. The installation order is determined by the package dependencies and the compilation options have to be explicitly specified. The default system compiler on Red Hat Enterprise Linux Release 6.7 is not sufficient to compile GEANTV and even the GNU compiler should be compiled from scratch, together with some libraries needed by the compiler. Any version above 4.8.0 should be sufficient. We tested that Version 4.8.5 works. Performance of GEANTV will be studied separately.

Prior to GEANTV compilation, the following packages have to be properly configured and installed:

- ROOT6 (**https://root.cern.ch/**);
- Vc (**https://github.com/VcDevel/Vc**)GE;
- VecGeom (**https://gitlab.cern.ch/VecGeom/VecGeom**);
- Pythia8 (**http://home.thep.lu.se/~torbjorn/Pythia.html**);
- HepMC3 (**https://hepmc.web.cern.ch/hepmc/**);
- Xerces-c (**http://xerces.apache.org/xerces-c/**);
- GEANT4 (**https://geant4.web.cern.ch/geant4/**).

4. Performance analysis

Different numbers of events (1000, 10 000, 100 000 and 1 000 000) were simulated using 1, 8 and 10 threads on the CPU and 1, 60 and 240 threads on the co-processor. A beam of positive muons with energy of 1 TeV has been sent to a fixed lead target and the primary and secondary particles were traced through an experimental setup containing five detector disks. The lead target is 5 cm thick, while the detectors are 20 cm thick ionization chambers filled with Xenon. The distance between the center of the target and the first chamber is 80 cm. The distance between centers of neighboring chambers is also 80 cm. The bulk space is filled with air.

The tests were performed on 1, 8, and 10 threads for the CPU and 1, 60, and 240 threads for the co-processor. Comparing one thread on the CPU to one thread on the co-processor aims to reveal the intrinsic performance of the corresponding architectures. Eight threads correspond to the number of physical cores of the CPU at the test architecture, and 240 is the maximal number of threads for the co-processor. We performed also tests with 60 threads on the co-processor, corresponding to one thread per physical core (one core is kept for system functions). The results are presented in Fig. 1. The times per event for different number of threads are given in Table 1 and Table 2 and depicted in Fig. 2.

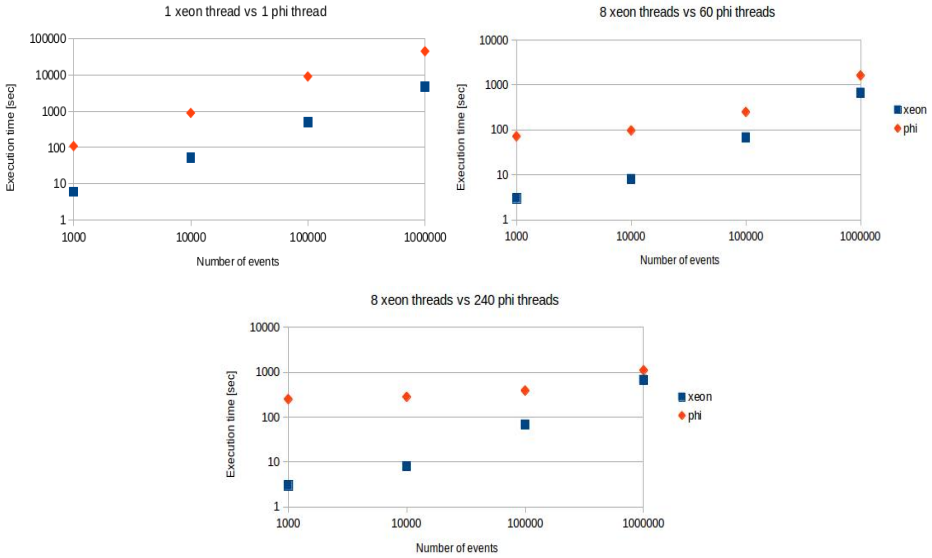

Fig. 1. Execution time vs. number of events for different number of threads on the CPU and on the co-processor

Table 1. Time per event (s) for the CPU

Threads	1K events	10K events	100K events	1000K events
1	0.006	0.0052	0.000488	0.004858
8	0.003	0.0008	0.00068	0.000672
10	0.003	0.0008	0.00056	0.000539

Table 2. Time per event (s) for Intel Xeon Phi

Threads	1K events	10K events	100K events	1000K events
1	0.109	0.0895	0.09043	0.0448
60	0.072	0.0097	0.00251	0.001633
240	0.251	0.0283	0.0038	0.001101

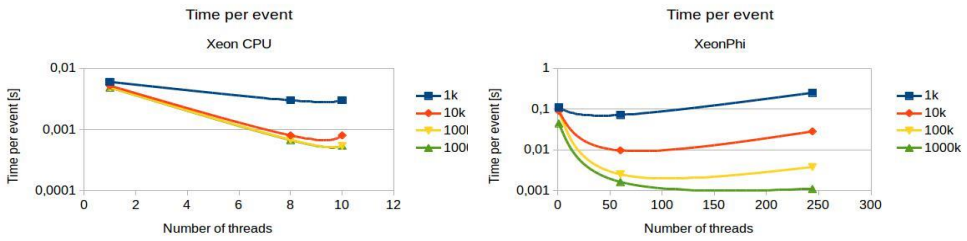

Fig. 2. Time per event (s) (double log scale) for Intel Xeon (left) and for Intel Xeon Phi (right)

We observe that the speedup through multithreading is limited both for the CPU and the co-processor and shows dependence on the number of physical cores (8 for the CPU and 60 for the co-processor) though slightly shifting towards higher threads number with the increasing of the simulated particles number. A close-up on the two computing units separately reveals a dependence, which has to be accounted for when

selecting the computing environment for particular physical tasks. For high number of simulated events, the speedup (inverse execution time ratio) through multithreading is generally higher. For simulations on the CPU, it is recommended to keep to the limitation imposed by the physical cores number. For the co-processor, however, the situation is different. For medium-size problems, the same limitation (60 threads in this case) leads to better performance results and only large-scale problems (106 events and more) are better suited for the maximal threads number – 240 (Figs 3, 4).

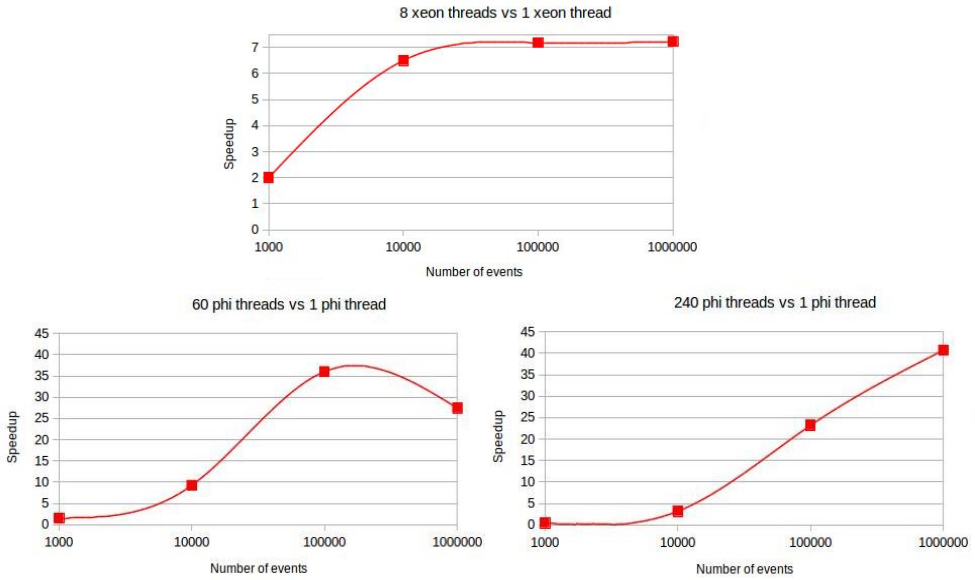

Fig. 3. Speedup vs number of events for different multithreading options on the CPU and on the co-processor

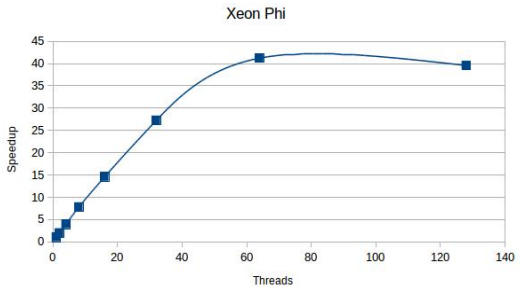

Fig. 4. Speedup with the number of threads for a simulation with 128,000 events on the co-processor

In all cases, the CPU outperforms the co-processor, however the test data in Tables 1 and 2 justifies the increasing efficiency (nominal, as well as in different processing configurations) of the co-processor for high number of simulated events, where the performances of the selected configurations are comparable. In Fig. 5, the

relative speedup (inverse execution time ratio for different processing configurations) for 60 and 240 Intel Xeon Phi threads w.r.t. 1 and 8 CPU threads is shown. As it can be seen, performing simulations on the co-processor with the maximal number of threads becomes competitive with increasing the number of the simulated events.

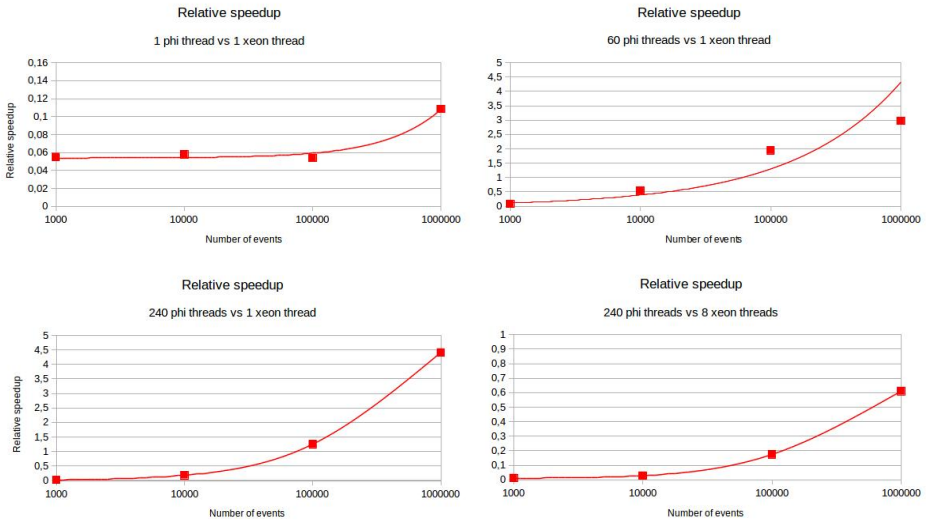

Fig. 5. Relative speedup vs number of events for different processing configurations

Finally, we consider an example with different particles – protons, though in the same experimental setup, and draw a parallel to the muon case studied so far. The important difference (in this context) is the way these particles interact with matter. Muons produce only a few secondary particles, while protons generate a whole shower. The results – presented in Fig. 6 – confirm the expectations. For muons, the execution times reach a minimum at 32 threads and then increase again, while for protons we observe a rapid decrease up to 32 threads and only a minor decrease afterwards. Thus, in the muon case, the advantages through the increased number of threads cannot compensate the increased setup time. For the larger number of events, however, the muon curve does not exhibit a minimum, but a saturation, already at half the maximal number of threads per co-processor (Fig. 2, right panel, green).

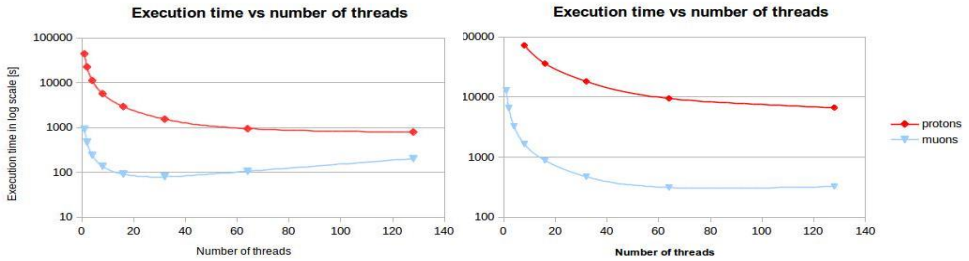

Fig. 6. Execution time vs. number of threads on Intel Xeon Phi for different particles: 10 000 events (left); 128 000 events (right)

Our results are qualitatively in a very good agreement with those reported in [11, 12]. However, a direct quantitative comparison is not possible, because of the different benchmarks used in these three studies and the different hardware. Our study is performed on Intel Xeon Phi 7120P [13], while the results reported in [11, 12] are obtained on Xeon Phi 5110P [14].

5. Conclusions

We studied the performance of GEANT4 simulations on a hybrid cluster with Intel Xeon Phi co-processors. The results show an essential speedup with the number of threads for increasing number of simulated events in all tested configurations. The performance depends on the particular physical case. The benefits of multi-threading are evident for large number of particles, no matter if secondary or primary ones. The scaling behavior can be well understood in the context of the particle properties. The results suggest choosing a high number of threads if the simulation produces a high number of secondary particles. For cases with a smaller number of secondary particles, a performance increase can be achieved in two ways:

- by finding the optimal number of threads and keeping to it for the given class of simulations;
- by performing the simulation with a sufficiently high number of primary particles (at least 1000) per thread.

Formally, the co-processor does not outperform the CPU, but for a large number of simulated events (which is the real case) the Xeon Phi performance is comparable with the CPU performance. Porting an extremely large package like Geant4 to a new architecture is a nontrivial task. Thus, this option is not considered by the developers for Nvidia CUDA as the code requires essential changes. On the contrary, the source code needs little or almost no changes in order to be compiled and executed on Intel Xeon Phi, which is a remarkable advantage, making it the only hybrid platform able to run GEANT4. In fact, GEANT4 is not optimized for parallel computations. The fully parallelized version of the suite, GEANTV, which is still under development, will expectedly provide an overall performance enhancement, including that on co-processors of the series Intel Xeon Phi. With the energy efficiency in mind, such heterogeneous architectures are a promising candidate for offloading to them part of these important and computationally expensive simulations.

***Acknowledgments*:** This work was partly supported by the PRACE project funded in part by the EU's Horizon 2020 Research and Innovation Programme (2014-2020) under grant agreement No 653838.

References

1. **https://geant4.web.cern.ch/geant4/**
2. Agostinelli, S., et al. GEANT4 − A Simulation Toolkit. − NIM, Vol. **A506**, 2003, No 3, pp. 250-303.
3. Allison, J., et al. GEANT4 Developments and Applications. – IEEE Transactions on Nuclear Science, Vol. **53**, 2006, No 1, pp. 270-278.
4. Allison, J., et al. Recent Developments in GEANT4. – NIM, Vol. **A835**, 2016, pp. 186-225.

5. **https://cds.cern.ch/record/118715?ln=en**
6. **https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBook**
7. **http://www.opengatecollaboration.org/UsersGuide**
8. **https://twiki.cern.ch/twiki/bin/view/Geant4/XeonPhiSupport**
9. **http://www.hpc.acad.bg/system-1/**
10. E. Atanassov et al., Eds. Best Practice Guide Intel Xeon Phi V2.0, January 2017. – PRACE. **http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-Intel-Xeon-Phi-1.pdf**
11. F a r r e l l, S., A. D o t t i, M. A s a i, P. C a l a f i u r a, R. M o n n a r d. Multi-Threaded GEANT4 on the Xeon-Phi with Complex High-Energy Physics Geometry. E-print: arXiv:1605.08371v1 [physics.comp-ph]. **https://arxiv.org/abs/1605.08371**
12. S c h w e i t z e r, P., S. C i p i è r e, A. D u f a u r e, H. P a y n o, Y. P e r r o t, D. R. C. H i l l, L. M a i g n e. Performance Evaluation of Multi-Threaded Geant4 Simulations Using an Intel Xeon Phi Cluster. – Scientific Programming, Vol. **2015**, 2015, Article ID 980752. 10 p. **http://dx.doi.org/10.1155/2015/980752**
13. **https://ark.intel.com/products/75799/Intel-Xeon-Phi-Coprocessor-7120P-16GB-1_238-GHz-61-core**
14. **https://ark.intel.com/products/71992/Intel-Xeon-Phi-Coprocessor-5110P-8GB-1_053-GHz-60-core**

Appendix A. Test-Code Compilation for the CPU and the Co-Processor

The test-code compilation for both cases is as follows:

```
source /opt/soft/geant4/mic/non-mic/scripts/geant4_compilers_setup_nonmic.sh
mkdir ~/test_cpu
cd ~/test_cpu
mkdir B2a
cd B2a
cmake -DCMAKE_LINKER=${LD} -DCMAKE_AR=${AR} -
DCMAKE_TOOLCHAIN_FILE=/opt/soft/geant4/mic/non-
mic/geant4.10.00.p04//mic-toolchain-file.cmake -
DGeant4_DIR=/opt/soft/geant4/mic/non-
mic/geant4.10.00.p04/geant4.10.00.p04-build/
/opt/soft/geant4/mic/non-
mic/geant4.10.00.p04/geant4.10.00.p04/examples/basic/B2/B2a/
make -j 40
```

```
source  /opt/soft/geant4/mic/scripts/geant4_compilers_setup.sh
mkdir ~/test_mic
cd ~/test_mic
mkdir B2a
cd B2a
cmake -DCMAKE_LINKER=${LD} -DCMAKE_AR=${AR} -
DCMAKE_TOOLCHAIN_FILE=/opt/soft/geant4/mic/geant4.10.00.p04/mic-
toolchain-file.cmake  -
DGeant4_DIR=/opt/soft/geant4/mic/geant4.10.00.p04/geant4.10.00.p04-build/
```

```
/opt/soft/geant4/mic/geant4.10.00.p04/geant4.10.00.p04/examples/basic/B2/B2a/
make -j 40
```

Appendix B. GEANT4 User Code Compilation and Execution on Xeon Phi in Native Mode

The following script might facilitate the compilation of the user code and linking it to the GEANT4 library:

```
#!/bin/bash
#echo $0 usage: user_install_dir user_source_code_dir
echo usage: user_install_dir user_source_code_dir
#echo $# #arguments
if [ $# -ne 2 ]
  then echo "illegal number of parameters"
 exit
fi

/opt/intel/compilers_and_libraries_2016.2.181/linux/bin/iccvars.sh intel64 -arch intel64 -platform linux
export CC=/opt/intel/compilers_and_libraries_2016.2.181/linux/bin/intel64/icc
export CXX=/opt/intel/compilers_and_libraries_2016.2.181/linux/bin/intel64/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
export PATH=/opt/soft/geant4/mic/cmake-3.3.2/bin/:${PATH}
cd $1
cmake -DCMAKE_LINKER=${LD} -DCMAKE_AR=${AR} -DCMAKE_TOOLCHAIN_FILE=/opt/soft/geant4/mic/geant4.10.00.p04/mic-toolchain-file.cmake  -DGeant4_DIR=/opt/soft/geant4/mic/geant4.10.00.p04/geant4.10.00.p04-build $2
```

where the content of mic-toolchain-file.cmake is given in [8]. The compilation is straightforward:

```
./compile_user_code_mic.sh $USR_BUILD $USR_CODE
```

where the user code is in the directory pointed by the environment variable $USR_CODE and the directory to build and store executable is pointed by $USR_BUILD.

For a complete set of compilation and installation scripts, we refer to [10].