SHOC: The Scalable HeterOgeneous Computing Benchmark Suite

Dakar Team
Future Technologies Group
Oak Ridge National Laboratory

Version 1.1.5, November 2012

1 Introduction

The Scalable HeterOgeneous Computing benchmark suite (SHOC) is a collection of benchmark programs that tests the performance and stability of systems using computing devices with non-traditional architectures for general purpose computing, and the software used to program them. Its initial focus is on systems containing Graphics Processing Units (GPUs) and multi-core processors, and on the OpenCL [3] programming standard. It can be used on clusters as well as individual hosts.

OpenCL is an open standard for programming a variety of types of computing devices. The OpenCL specification describes a language for programming *kernels* to run on an OpenCL-capable device, and an Application Programming Interface (API) for transferring data to such devices and executing kernels on them.

In addition to OpenCL-based benchmark programs, SHOC also includes Compute Unified Device Architecture (CUDA)[5] versions of its benchmarks for comparison.

2 Supported Platforms

The Dakar team intends SHOC to be useful on any platform with an OpenCL implementation. However, the Dakar team develops and tests SHOC primarily on UNIX-like platforms, specifically Linux and Mac OS X. This section lists software versions used for much of the SHOC development on these platforms.

2.1 Linux

- A recent RedHat-family OS distribution (Fedora or RHEL).[1]
- A working OpenCL implementation. The Dakar team has tested SHOC using the following versions:
  - NVIDIA GPU Computing SDK version 4.1 or later

[1] Some Linux distributions may include a more recent GCC toolchain that is not yet supported by NVIDIA CUDA. On such platforms, an earlier version of GCC must be used to compile SHOC, and the SHOC configuration files must be modified so that the –compiler-bindir switch is passed to nvcc to indicate to nvcc the location of the GCC compiler binaries it should use.

  - AMD Accelerated Parallel Processing (APP) SDK version 2.6 or later

- (Optional) CUDA 4.1 or later

SHOC may work on other platforms with other OpenCL and CUDA versions than those listed here, but will most likely require modifications for differing OpenCL header and library paths, differing system library versions, and differing compiler versions/vendors.

2.2 Mac OS X

- Mac OS X 10.7 ("Lion") or Mac OS X 10.8 ("Mountain Lion").
- (Optional) CUDA 4.1 or later

2.3 Windows

- Note that Windows support is still experimental.
- The current development environment is Windows 7 x64 with CUDA 4.2 and Visual Studio 2010.

2.4 Clusters

In addition to individual systems, SHOC can also build parallel benchmark programs for clusters. Each cluster node must meet the requirements described earlier in this section for the OS distribution used on that node. Also, the cluster must have a working implementation of the Message Passing Interface (MPI) [1, 2] library such as OpenMPI `www.open-mpi.org` or MPICH2 `www.mcs.anl.gov/mpi/mpich`.

3 Configuring and Building SHOC

The steps needed to configure and build SHOC differ depending on whether you are building on a UNIX-like system (Linux and OS X) or Windows.

3.1 Linux and OS X

Beginning with version 1.1.5, SHOC adopts a more traditional GNU-style build process:

```
$ configure
$ make
$ make install
```

Prior versions did not use a `make install` step. Note that individual SHOC programs can be run without the install step, but the convenience script `shocdriver` (see 4) will not function correctly unless SHOC has been installed.

SHOC uses a configuration script generated by GNU autoconf. This script is located in the SHOC distribution's root directory. In the rest of this document, we assume this directory is called `$SHOC_ROOT`.

Beginning with version 1.1.5, SHOC can be built both within the SHOC source code tree and outside the source code tree. You might want to build outside the source tree, for example, if you are sharing one source code installation across several different types of systems. In this scenario, you could use the same source code installation with several platform-specific build directories. Begining with version 1.1.5, we recommend building SHOC outside the source code tree.

In the following, we will denote the directory where you build SHOC as `$BUILD_ROOT`. If you are building SHOC within its source tree, `$BUILD_ROOT` is the same as `$SHOC_ROOT`. Note that `$BUILD_ROOT` can be somewhere within the SHOC source tree hierarchy, such as at `$SHOC_ROOT/nvidia` and `$SHOC_ROOT/amd`.

Specify the directory where SHOC should be installed using the autoconf-standard `--prefix` option to the SHOC configure script. If you do not specify an installation location using `--prefix`, the configure script will assume the installation location is `$BUILD_ROOT`. In the rest of this document, we will call the installation location `$DEST_ROOT`.

To summarize:

$SHOC_ROOT Top-level directory of the SHOC source tree.

$BUILD_ROOT Directory where SHOC programs are built.

$DEST_ROOT Directory where SHOC programs are installed and run.

> Note: To approximate the build behavior of earlier versions of SHOC, simply change to the `$SHOC_ROOT` directory, configure without specifying a prefix, make, and make install.

By default, the configure script will try to determine whether the target system has usable OpenCL, CUDA, and MPI installations. The configure script depends on the PATH environment variable to find necessary binary programs like CUDA's `nvcc` and the MPI compiler driver program `mpicxx`, so the PATH must be set before the configure script is run. Similarly, the configure script uses CPPFLAGS and LDFLAGS to find needed headers and libraries for OpenCL and CUDA. For instance, on a system with the NVIDIA CUDA/OpenCL software installed in `/opt/cuda` (a non-default location), the PATH should be updated to include `/opt/cuda/bin` and the configure script should be run as follows so that it can find the OpenCL headers:

```
$ cd $BUILD_ROOT
$ sh $SHOC_ROOT/configure CPPFLAGS="-I/opt/cuda/include"
```

SHOC uses MPI compiler drivers (e.g., `mpicxx`) to compile and link code that uses MPI functionality. The SHOC configure script depends on the PATH environment variable to find the MPI compiler drivers for the MPI installation it will use, so it is often helpful to use a command like `which mpicxx` before configuring SHOC to ensure that the PATH is set to find the desired MPI installation. Note that, unlike previous versions of SHOC that also required configuration script options to specify the location of MPI headers and libraries, current versions of SHOC do not support such options because the MPI compiler drivers automatically add the appropriate compiler and linker flags.

If you desire not to use SHOC's OpenCL, CUDA, or MPI support (e.g., because no MPI implementation is available), use the `--without-opencl`, `--without-cuda`, and/or `--without-mpi`

configure script flags. Note, however, that support for at least one of OpenCL and CUDA must be enabled to use SHOC.

3.1.1 Optimization Flags

SHOC inherits the default autoconf behavior with respect to compiler optimization flags. In particular, by default SHOC uses `-g -O2` when it detects it is compiling with the GNU compiler and just `-g` otherwise. This may not be the behavior you want. For instance, when compiling with the PGI compiler, `-fastsse` (or `-fast` for systems with a processor that does not support SSE instructions) is desirable. Some SHOC benchmarks like MD use nested inline functions (i.e., inline functions that call other inline functions, and so on). The PGI compiler's default behavior does not inline functions to a deep enough level to provide good performance in these situations. The `-fastsse` optimization flag implies a flag telling the compiler to aggressively inline functions, which results in much better performance. Although the Intel compiler does not have the same default behavior as the PGI compiler with respect to inlined functions, using the Intel compiler's optimization flags may also be preferable to the default behavior produced by the autoconf script. Note, however, that the Intel compiler's `-fast` flag implies static linking, and some of the libraries used by SHOC such as CUDA and OpenCL are not distributed as static libraries. As a workaround, use specific Intel compiler optimization flags rather than an umbrella flag like `-fast`.

To specify optimization flags when configuring SHOC, pass it to the configure script using the CXXFLAGS, CFLAGS, and LDFLAGS environment variables. For instance, when using the PGI compiler on a system with the AMD Accelerated Parallel Processing SDK, one might use a configure command like the following:

```
$ sh $SHOC_ROOT/configure \
    CPPFLAGS="-I/opt/AMDAPP/include" \
    CXXFLAGS="-g -fastsse" \
    CFLAGS="-g -fastsse" \
    LDFLAGS="-g -fastsse -L/opt/AMDAPP/lib/x86_64" \
    --without-cuda \
    --disable-stability
```

3.1.2 Bit Widths

By default, SHOC does not explicitly specify a flag to the compiler and linker to specify whether to build 32-bit or 64-bit executables. Thus, your SHOC executables will have the default bit-ness of the compiler you used to build SHOC. If you want to specify a different bit-ness (e.g., 64-bit executables on Mac OS X with the GNU compiler) you must specify the appropriate flag for your compiler in CPPFLAGS and LDFLAGS when configuring SHOC.[2] For example,

```
$ sh $SHOC_ROOT/configure \
    CXXFLAGS="-m64" \
```

[2]Earlier versions of SHOC supported an –enable-m64 option that automatically added the appropriate flag for the GNU compiler to generate 64-bit executables. Because this flag was specific to the GNU compiler, we have deprecated that configure option.

```
    CFLAGS="-m64" \
    NVCXXFLAGS="-m64"
```

gives equivalent behavior to the `--enable-m64` configure flag supported by earlier versions of SHOC if the GNU compilers are being used.

3.1.3 Stability Test

By default, SHOC builds a CUDA-based stability test. If you desire not to build the SHOC stability test (e.g., because CUDA is not available), use the `--disable-stability` configuration flag.

See the output of `sh $SHOC_ROOT/configure --help` for a full list of configuration options. Also, see the example configuration scripts in `$SHOC_ROOT\config` for examples of configuring the SHOC benchmark suite.

3.1.4 Cross compiling

> Cross compilation support is experimental. The SHOC development team has had very limited opportunity to test the functionality for a wide variety of build and host systems, and has had virtually no opportunity to thoroughly test the executables produced through cross compiling.

SHOC has **experimental** support for building executables that can run on a different type of system than was used to build them. For instance, on an x86 Linux system SHOC can be configured to build executables that run on an ARM Linux system. This technique is commonly called *cross compiling*, the system used to build the software is called the *build* system, and the system that will run the executables is called the *host*. Note that SHOC only enables cross compiling, it does not provide the compilers, linkers, libraries, and support tools needed to produce executables for the host system. You will need to obtain, install, and preferably test a cross-compilation toolchain on the build system before configuring SHOC.

To configure SHOC for cross compiling, pass a description of the target system using the `--host=`*hostspec* option to SHOC's configure script. In its simplest form, this looks something like:

```
$ sh $SHOC_ROOT/configure \
    --host=arm-none-linux-gnueabi
```

See the `config/conf-crossarm.sh` file for an example of configuring SHOC to build ARM Linux executables on an x86_64 Linux system using the Sourcery CodeBench Lite ARM toolchain.[3] Note that in that example configure script we are only configuring to build serial OpenCL benchmark programs.

[3] Sourcery CodeBench Lite Edition is available from Mentor Graphics at `http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition`.

3.1.5 Regenerating the SHOC configure script

If desired, the SHOC configure script can be regenerated on the target system. Make sure that the GNU autoconf and automake tools can be found with your PATH environment variable, and do the following:

```
$ cd $SHOC_ROOT
$ sh ./build-aux/bootstrap.sh
```

Once this command has finished building a new configure script, follow the instructions given earlier in this section to configure SHOC.

3.1.6 Building

Once the SHOC benchmark suite has been configured, it is built and installed using:

```
$ cd $BUILD_ROOT
$ make
$ make install
```

Unlike previous versions of SHOC, control over whether to build the OpenCL, CUDA, OpenCL+MPI, and CUDA+MPI versions of the benchmarks is exercised at configure time instead of build time. Therefore, commands like 'make cuda' are no longer supported.

3.2 Windows

On Windows, we currently expect that the CUDA SDK version 4.2 is installed for Visual Studio 2010.

Note that other versions of CUDA should work, as they would on Linux and OS X. However, the project files on Windows request compute capability 3.0 code generation. This will likely fail with an older CUDA installation. If you wish to attempt builds using prior CUDA versions, we recommend at a minimum removing the `compute_30` and `sm_30` flags from CUDA properties within the projects.

SHOC is a single Visual Studio 2010 solution; open the file `shoc-cuda42-vs10` in the root of the SHOC directory. This solution contains one project for each benchmark. Assuming the proper CUDA support has been installed, building the solution should build all executables. Unlike Linux or OS X, on Windows no "install" step is needed after building SHOC executables.

Note: Visual Studio 2010 will currently attempt a parallel build, but this setup does not yet support it. If you see errors during the build, simply build the project again and it will continue where it left off. (This is the F7 *Build* **command, not the** *Rebuild* **command.)**

The Windows build currently supports only serial benchmarks and will not build with MPI support.

4 Running

4.1 Linux and OS X

SHOC includes a driver script for running either the CUDA or OpenCL versions of the benchmarks. The driver script assumes MPI is in your current path, so be sure to set the appropriate environement variables.

Note: SHOC must be *installed* for the driver script to operate correctly. Be sure to `make install` to install SHOC into `$DEST_ROOT` before trying to run the driver script.

```
$ export PATH=$PATH:/path/to/mpi/bin/dir
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/mpi/lib/dir
```

To run the benchmarks on a single device, execute the following. Be sure to specify `-cuda` or `-opencl` and a device number `-d n` where `n` is the device you want to test. The example below shows how to test device zero with a small problem using CUDA.

```
$ cd $DEST_ROOT
$ ./bin/shocdriver -s 1 -cuda -d 0
```

To run on more than one node, supply the script with the number of nodes and a comma separated list of the devices that you want to use on each node. For example, if running on 4 nodes that each have 2 devices, execute the following:

```
$ cd $DEST_ROOT
$ ./bin/shocdriver -cuda -n 4 -d 0,1 -s 1 -cuda
```

Or, if on those same four nodes, you only wanted to test device one on each node:

```
$ cd $DEST_ROOT
$ ./bin/shocdriver -cuda -n 4 -d 1 -s 1 -cuda
```

These scripts output benchmark results to a file in comma separated value (CSV) format.

One can also run individual benchmarks without the driver script. To run single-process benchmark programs, use commands like:

```
$ cd $DEST_ROOT
$ ./bin/Serial/OpenCL/Scan
```

Run parallel benchmark programs with commands like:

```
$ cd $DEST_ROOT
$ mpirun -np 8 ./bin/EP/Scan
```

Use 1 MPI rank per GPU.

4.2 Windows

The driver script has not yet been ported to Windows. Instead, run individual benchmarks as described above. The binaries in Windows are output to the same location as on Linux and OS X.

```
$ cd $DEST_ROOT
$ ./bin/Serial/CUDA/Scan
```

To make running from Visual Studio more convenient, by default the benchmarks will prompt the user to press the return key before they exit. If you wish to skip this prompt, pass the Windows-specific `--noprompt` command-line flag.

5 Benchmark Programs

The SHOC benchmark suite currently contains benchmark programs, categoried based on complexity. Some measure low-level "feeds and speeds" behavior (Level 0), some measure the performance of a higher-level operation such as a Fast Fourier Transform (FFT) (Level 1), and the others measure real application kernels (Level 2).

- Level 0
  - **BusSpeedDownload**: measures bandwidth of transferring data across the PCIe bus to a device.
  - **BusSpeedReadback**: measures bandwidth of reading data back from a device.
  - **DeviceMemory**: measures bandwidth of memory accesses to various types of device memory including global, local, and image memories.
  - **KernelCompile**: measures compile time for several OpenCL kernels, which range in complexity
  - **MaxFlops**: measures maximum achievable floating point performance using a combination of auto-generated and hand coded kernels.
  - **QueueDelay**: measures the overhead of using the OpenCL command queue.
- Level 1
  - **BFS**: a breadth-first search, a common graph traversal. Requires a a device which supports atomic operations. (CC $>$ 1.2)
  - **FFT**: forward and reverse 1D FFT.
  - **MD**: computation of the Lennard-Jones potential from molecular dynamics
  - **MD5Hash**: computation of many small MD5 digests, heavily dependent on bitwise operations.
  - **Reduction**: reduction operation on an array of single or double precision floating point values.
  - **SGEMM**: matrix-matrix multiply.
  - **Scan**: scan (also known as parallel prefix sum) on an array of single or double precision floating point values.
  - **Sort**: sorts an array of key-value pairs using a radix sort algorithm
  - **Spmv**: sparse matrix-vector multiplication
  - **Stencil2D**: a 9-point stencil operation applied to a 2D data set. In the MPI version, data is distributed across MPI processes organized in a 2D Cartesian topology, with periodic halo exchanges.
  - **Triad**: a version of the STREAM Triad benchmark, implemented in OpenCL and CUDA. This version includes PCIe transfer time.
- Level 2
  - **QTC**: quality threshold clustering
  - **S3D**: A computationally-intensive kernel from the S3D turbulent combustion simulation program[4].

To see the options each program supports and their default values, run *program* `--help` for serial versions and `mpirun -np 1` *program* `--help` for parallel versions.

	OpenCL			*CUDA*		
Program	*S*	*EP*	*TP*	*S*	*EP*	*TP*
BusSpeedDownload	x	x		x	x	
BusSpeedReadback	x	x		x	x	
DeviceMemory	x	x		x	x	
KernelCompile	x	x				
MaxFlops	x	x		x	x	
QueueDelay	x	x				
BFS	x	x		x	x	
FFT	x	x		x	x	
MD	x	x		x	x	
MD5Hash	x	x		x	x	
Reduction	x	x	x	x	x	x
QTC				x		x
S3D	x	x		x	x	
SGEMM	x	x		x	x	
Scan	x	x	x	x	x	x
Sort	x	x		x	x	
Spmv	x	x		x	x	
Stencil2D	x		x	x		x
Triad	x	x		x	x	
BusCont		x			x	
MTBusCont		x			x	

Table 1: Programming APIs and parallelism models of SHOC programs

Many SHOC benchmark programs test both single precision and double precision arithmetic. For programs that support both precisions, the program first runs the single precision benchmark test, then attempts to determine if the OpenCL or CUDA device being used supports double precision arithmetic. If so, the program runs the double precision test. If the target device does not support double precision arithmetic, the driver script reports "NoResult" as the result. If some error occurred when running the benchmark, the driver script reports "BenchmarkError." In that case, please see the benchmark log files in `$DEST_ROOT/tools/Logs` for more information about the error. Please also report such situations (including the contents of the appropriate log file) to the SHOC developers the following email address `shoc-help@elist.ornl.gov`.

Benchmarks are built not only as serial programs (S) but also as embarrassingly parallel (EP) or true parallel (TP) programs. The following table indicates which versions of each program that SHOC builds.

6 Source Tree

SHOC is distributed as a compressed tar archive. Let `$SHOC_ROOT` represent the directory that will hold the SHOC source tree. The SHOC archive can be uncompressed and extracted using

```
$ cd $SHOC_ROOT
$ tar xvzf shoc-x.y.tar.gz
```

The SHOC source tree directory structure is as follows:

```
$SHOC_ROOT
    bin     # benchmark executables may be built here for in-tree builds
        EP  # "embarrassingly parallel" benchmarks
            CUDA
            OpenCL
        TP  # true parallel benchmarks
            CUDA
            OpenCL
        Serial  # single-node benchmarks
            CUDA
            OpenCL
    config  # SHOC configuration files
    doc     # SHOC documentation files
    lib     # SHOC auxiliary libraries are built here
    src     # SHOC source files
        common  # programming-model independent helper code
        cuda    # CUDA-based benchmarks
            level0  # low-level CUDA benchmarks
            level1  # higher-level CUDA benchmarks
            level2  # application level CUDA benchmarks
        mpi     # MPI-specific benchmarks
            common  # code needed by programs using MPI
            contention  # a contention benchmark
            contention-mt  # a multithreaded version of the contention benchmark
        opencl  # OpenCL benchmarks
            common  # code needed for all OpenCL benchmarks
            level0  # low-level OpenCL benchmarks
            level1  # higher-level OpenCL benchmarks
            level2  # application-level OpenCL benchmarks
        stability   # a CUDA stability test
```

7 Support

Support for SHOC is provided on a best-effort basis by the Dakar team members and eventually by its user community via several mailing lists.

- `shoc-announce@elist.ornl.gov`: mailing list for announcements regarding new versions or important updates.
- `shoc-help@elist.ornl.gov`: email address for requesting help in building or using SHOC, or for providing feedback about the benchmark suite.
- `shoc-dev@elist.ornl.gov`: mailing list for internal development discussions by the SHOC development team.

Revision History

- 0.1 September 2009
- 0.2 December 2009
- 0.3 June 2010
- 0.4 September 2010
- 0.5 October 2010
- 1.0 November 2010
- 1.01 January 2011
- 1.0.2 March 2011
- 1.0.3 March 2011
- 1.1.0 June 2011
- 1.1.1 July 2011
- 1.1.2 November 2011
- 1.1.3 December 2011
- 1.1.4 May 2012
- 1.1.5 November 2012

References

[1] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface,* 2nd edition. MIT Press, Cambridge, MA, 1999.

[2] William Gropp, Ewing Lusk, and Rajeev Thakur. *Using MPI-2: Advanced Features of the Message-Passing Interface.* MIT Press, Cambridge, MA, 1999.

[3] The Khronos Group. The OpenCL specification, version 1.0, document revision 43. specification, The Khronos Group, 2009.

[4] Evatt R Hawkes, Ramanan Sankaran, James C Sutherland, and Jacqueline H Chen. Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models. *Journal of Physics: Conference Series*, 16(1):65, 2005.

[5] NVIDIA. NVIDIA CUDA reference manual, version 2.3. manual, 2009.