SHOC: The Scalable HeterOgeneous Computing Benchmark Suite

Dakar Team
Future Technologies Group
Oak Ridge National Laboratory

Version 1.1.1, July 2011

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 3.2 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.

    - ATI Stream SDK version 2.2 or later

- (Optional) CUDA 3.2 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 10.6 (”Snow Leopard”) or later.
- Xcode 3.2 or later.
- (Optional) CUDA 3.1 or later

2.3 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`. An OS X 10.6 system with Xcode installed contains an OpenMPI implementation sufficient for use by SHOC without the need to install additional software.

3 Configuring

Unlike previous SHOC versions, this version of 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 presume this directory is called `$SHOC_ROOT`.

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`, 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 $SHOC_ROOT
$ sh ./configure CPPFLAGS="-I/opt/cuda/include"
```

The SHOC configuration script does not use CPPFLAGS and LDFLAGS to find MPI headers and libraries. Instead, use the `--with-mpi-libraries` and `--with-mpi-includes configuration` switches to inform the configuration script of the flags needed to use your MPI installation. Some MPI implementations can tell the user which include flags and linker flags are needed to compile and link MPI programs. For instance, OpenMPI's compiler driver programs like mpicxx accept the `-showme:compile` and `-showme:link` flags that indicate the compile and link flags needed to build MPI programs without the MPI compiler driver. As a convenience, SHOC provides example scripts in `$SHOC_ROOT\config` showing how to use the MPI compiler driver to pass compile and link flags to the SHOC configuration script. These scripts are called `conf-linux-openmpi.sh` and `conf-linux-mpich.sh` for OpenMPI and MPICH2, respectively.

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.

SHOC can be configured to build either 32-bit or 64-bit executables. By default, SHOC builds 64-bit executables on all platforms except OS X; on that platform, 32-bit executables are the default. Whether to build 32- or 64-bit executables can be controlled using the `--enable-m64` or `--disable-m64` configuration flags.

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 ./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 Regenerating the SHOC configure script

If desired, the SHOC configure script can be regenerated on the target system. Make sure that GNU autoconf and GNU automake (for aclocal) 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.

4 Building

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

```
$ cd $SHOC_ROOT
$ make
```

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.

5 Running

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.

```
$ 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 $SHOC_ROOT/tools
$ perl driver.pl -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 $SHOC_ROOT/tools
$ perl driver.pl -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 $SHOC_ROOT/tools
$ perl driver.pl -cuda -n 4 -d 1 -s 1 -cuda
```

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

After building SHOC, individual benchmark programs will be left in the directory tree rooted at `$SHOC_ROOT/bin`. Run single-process benchmark programs with commands like:

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

and parallel benchmark programs with commands like:

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

Use 1 MPI rank per GPU.

6 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
  - **FFT**: forward and reverse 1D FFT.
  - **MD**: computation of the Lennard-Jones potential from molecular dynamics
  - **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
  - **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.

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.[2]

[2] Currently, the SHOC driver script does not distinguish between lack of support for double precision and a failure when running a double precision benchmark. In both cases, double precision results are reported by the SHOC benchmark driver script as "Benchmark Error." Until the driver script is updated to distinguish these two cases, please see the benchmark log files in `$SHOC_ROOT/tools/Logs` to determine whether the "Benchmark Error" represents a lack of support for double precision or a true benchmark program failure.

	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				
FFT	x	x		x	x	
MD	x	x		x	x	
Reduction	x	x		x	x	x
S3D	x	x		x	x	
SGEMM	x	x		x	x	
Scan	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

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.

7 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 are built here
        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
```

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

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.