Oxbow Toolkit: User Guide

Oxbow Developers
oxbow-dev@email.ornl.gov

August 17, 2014

Contents

1 Introduction

HPC architectures will continue to change over the next decade in response to efforts to improve energy efficiency, reliability, and performance. At this time of significant disruption, it is critically important to understand the requirements of contemporary and future extreme-scale scientific applications, so that we can drive or adopt new architectural and software features that satisfy the requirements of our applications. e.g., integrated GPU and CPU, integrated random number generator, transactional memory, fine-grained power management, MPI collective offload.

Hence, we believe that it is essential to quantitatively measure, project, and prioritize the resource and feature requirements of our anticipated workloads on such extreme-scale systems.

The Oxbow toolkit is a collection of tools to empirically characterize application behaviours along a critical set of dimensions namely computation, communication, memory capacity and access patterns.

2 Installation

For instructions on building and installing Oxbow, see the README included with the Oxbow tools source distribution. Following these instructions should result in an installation directory structure that includes a subdirectory for the oxbow tools and (optionally) a subdirectory for third party utilities. The directory names will be determined by the vendor ID of the compiler used during the build process.

For example, using gnu compilers and installing into the prefix `/local/opt/oxbow`, will result in an installed directory structure something like the following:

```
/local/opt/oxbow/oxbow-tpls-gnu/
/local/opt/oxbow/oxbow-tpls-gnu/binutils-VER
/local/opt/oxbow/oxbow-tpls-gnu/libunwind-VER
/local/opt/oxbow/oxbow-tpls-gnu/papi-VER
/local/opt/oxbow/oxbow-tpls-gnu/pin-VER
/local/opt/oxbow/oxbow-tool-gnu/
/local/opt/oxbow/oxbow-tool-gnu/bin
/local/opt/oxbow/oxbow-tool-gnu/etc
/local/opt/oxbow/oxbow-tool-gnu/include
/local/opt/oxbow/oxbow-tool-gnu/lib
/local/opt/oxbow/oxbow-tool-gnu/share
```

The `etc` directory contains a script, `envvars.sh` to set up your environment for building and running applications using the Oxbow tools. In section 3, most of the instructions begin by sourcing this script. For example, using our installation of Oxbow on the Keeneland test system, the user environment for using Oxbow tools with Intel compilers is set up by running:

```
$ source /nics/a/proj/oxbow/oxbow-tool-intel/etc/envvars.sh
```

The other subdirectories of `oxbow-tool-vendor` follow standard conventions.

- `bin` : executables

- `etc` : system specific configuration
- `include` : header files
- `lib` : libraries
- `share` : documentation

3 Using Oxbow Tools

There are currently five tools available in the Oxbow toolkit.

- **mpiP** (3.2) profiles MPI communication. Using mpiP requires relinking your binary against the mpiP libraries in the Oxbow installation.
- **miami-imix** (3.3) collects information about the mix of micro-operations in instructions executed in an application. Recompiling is necessary only if you wish to add in start and stop calipers to limit the code sections profiled.
- **pin-imix** (3.4) collects information about the mix of opcode classes of instructions executed in an application. Recompiling is necessary only if you wish to add in start and stop calipers to limit the code sections profiled.
- **membw** (3.5) calculates the memory bandwidth (reads and writes) consumed by the application. Using membw requires adding several library calls to an application and recompiling.
- **reused** (3.6) outputs a reuse distance histogram for an application memory access. The reuse distance metric is defined as the number of intervening memory accesses between accesses to a given memory address. Recompiling is necessary only if you wish to add in start and stop calipers to limit the code sections profiled.

3.1 Using Convenience Scripts

The sections below contain specific instructions for building and running each tool. Rather than run the tools directly, you may use convenience scripts. These are installed under:

`/path/to/oxbow/oxbow-tool-vendor/bin/util`

All of the scripts are named for the tool they invoke, and have similar usage:

```
$ export PATH=$PATH:/path/to/oxbow/oxbow-tool-vendor/bin/util
$ do-tool-name.sh [FLAGS] OUTDIR -- [MPIRUN] -- COMMAND
```

These are meant to be common case run scenarios, so the `FLAGS` here are not specific flags to the tool being run. There is only one flag value of interest, and it applies only to miami-imix, pin-imix, and reused. For these tools, `-unmarked` is used to tell the script that the binary being run does not contain caliper functions.

These tools all produce various output files. Sometimes quite a lot of files are generated, as for multithreaded applications or MPI processes with many ranks. The `OUTDIR` argument to the convenience script tells the tool where to place all output. This directory will also contain a log of the exact commands issued by the

script when invoking the command, as well as any output and error messages. The log will be located in OUTDIR/do-tool-name.log.

If the application is launched with an mpirun (or aprun or mpiexec) type command, enclose this command and any arguments between two sets of dashes. Follow the MPIRUN command with the actual command and arguments to be executed.

Example invocations of convenience scripts

Run an unmodified MPI binary using the pin-imix tool. Put output in myoutdir.

```
$ do-pin-imix.sh -unmarked myoutdir -- mpirun -n 4 -- ./myprog arg1 arg2
```

Run an unmodified serial binary using the reused tool. Put the output in `$HOME/work`.

```
$ do-reused.sh -unmarked $HOME/work -- -- echo "hello"
```

Run a modified serial binary that has caliper functions added for pin-imix. Put the output in `$HOME/work`.

```
$ do-pin-imix.sh $HOME/work -- -- ./myprog-modified arg1 arg2
```

Run a binary that has been relinked for mpiP. Put the output in myoutdir.

```
$ do-mpip.sh myoutdir -- aprun -B -- ./mympi-modified arg1 arg2
```

3.2 mpiP: MPI Communication profiling

mpiP is a lightweight profiling library for MPI applications. Because it only collects statistical information about MPI functions, mpiP generates considerably less overhead and much less data than tracing tools. All the information captured by mpiP is task-local. It only uses communication during report generation, typically at the end of the experiment, to merge results from all of the tasks into one output file.

For extensive information about configuring and using mpiP, see the mpiP user guide. It can be accessed online at:

http://mpip.sourceforge.net/

A copy of the user guide is also installed in Oxbow under:

```
/path/to/oxbow/oxbow-tool-vendor/share/doc/mpip/
```

To use mpiP to characterize your application's communication patterns, you will need to relink your application against the mpiP libraries and its third party library dependencies. Add the following link flags:

```
-L${OXBOW_TOOLS_DIR}/lib -lmpiP
-L${LIBUNWIND_DIR}/lib -lunwind
-L${BINUTILS_DIR}/lib -lbfd
-L${BINUTILS_DIR}/lib64 -liberty
```

The locations of the required libraries can be added to your environment by sourcing the environment setup script installed in oxbow. For example:

```
$ source /path/to/oxbow/oxbow-tool-vendor/etc/envvars.sh
$ mpicc -g obj1.o obj2.o -o myprog-mpip -L${OXBOW_TOOLS_DIR}/lib -lmpiP \
-L${LIBUNWIND_DIR}/lib -lunwind -L${BINUTILS_DIR}/lib -lbfd -L${BINUTILS_DIR}/lib64 -liberty
```

Once the application is relinked, launch as normal. The application will output the results of profiling the MPI communication. To configure what output is produced, set the `MPIP` environment variable. The variable stores flags with similar syntax to command line flags. See the user guide for information on specific flags.

If you are using the provided `do-mpip.sh` convenience script, `MPIP` will be set in the script unless you set it yourself before running the script. The setting for `MPIP` in the convenience script will output results for both collective communication, point-to-point communication, as well as a collective communication matrix.

```
$ do-mpip.sh myoutdir -- mpirun -n 64 -- ./myprog-mpip arg1 arg2
```

3.3 miami-imix: Micro-operation Instruction Mix PIN Tool

The Miami imix tool profiles an application run using the Intel PIN profiling infrastructure. Each instruction is broken down into micro-operations: inividual reads, writes, integer, float, and SIMD operations. The tool output prints the counts of each micro-operation type in a comma-seperated-value file. The rows of the csv indicate which binary module the instructions resulted from.

To obtain the instruction mix, you can either:

1. Use the provided convenience script
2. OR Perform a two step process using the miamicfg tool and the miami-imix tool

3.3.1 Option 1: Use the do-miami-imix.sh script

For code that does not have caliper functions around a section of interest:

```
$ do-miami-imix.sh -unmarked myoutdir -- mpirun -n 4 -- ./myprog arg1 arg2
```

For code that has had caliper functions added:

```
$ do-miami-imix.sh myoutdir -- mpirun -n 4 -- ./myprog arg1 arg2
```

3.3.2 Option 2: Run both steps of miami-imix manually

Obtaining the instruction mix is a two step process.

1. Use the miamicfg tool to collect a control flow graph of the application run
2. Use the miami-imix tool to process the control flow graphs to obtain instruction mix information

Step 1: Control Flow Graph Info

First, you need to obtain the control flow group information by profiling the application in the following manner.

```
${OXBOW_TOOLS_DIR}/bin/miamicfg [options] -- <your_application> <your_arguments>
```

The double dash "–" is important as it separates the instruction mix tool's options from the target application and its parameters.

If this is an MPI application, then place the instruction mix tool in the position where you place your executable name.

```
mpirun -np 16 ${OXBOW_TOOLS_DIR}/bin/miamicfg [options] -- <your_application> <your_arguments>
```

No additional options are required for the wrapper. This step creates a .cfg file per process. By default, the output files are named: ExecName-MpiRank-ProcessPid.cfg

Optionally, you can resume and pause data collection dynamically. For this, you must modify the application's source code to insert calls to two, user defined empty functions, one for starting and one for stopping data collection. You can choose any name for these two caliper functions.

Once you identified suitable functions, you should pass use the folowing parameters to the instruction mix tool.

```
-q -start <name_of_start_function> -stop <name_of_stop_function>
```

Step 2: Instruction Mix

Once you have the control flow graph information from step 1, you should specify one resulting .cfg file to the miami-imix static tool as follows:

```
$OXBOW_TOOLS_DIR/bin/miami-imix -c <one_cfg_file>
```

This command outputs two files:

- ExecName-PID-imix.csv: The imix file has the restricted instruction categories.
- ExecName-PID-ibins.csv: The ibins file includes a more detailed classification.

Note: The '.cfg' files contain information mapped to binary addresses. For this reason, they are valid only with the original executable that you used to collect those files. The CFG file contains paths to the executable and all the shared libraries used during the profiling step.

The second step uses those paths to locate the binaries and decode the instructions. You should not delete or move your binaries before running the second step.

3.4 pin-imix: Opcode Instruction Mix PIN Tool

The pin-imix tool outputs counts of instructions categorized by opcode. This tool can be run with or without modification to your program.

3.4.1 Running with unmodified binaries

Convenience script use for unmarked code:

```
$ do-pin-imix.sh -unmarked myoutdir -- mpirun -n 64 -- ./myprog arg1 arg2
```

To run directly (no convenience script) on unmarked code:

```
$ source /path/to/oxbow/oxbow-tool-vendor/etc/envvars.sh
$ mpirun -n 64 ${PIN_DIR}/intel64/bin/pinbin -follow_execv -t \
  ${OXBOW_TOOLS_DIR}/bin/imix.pin -category -i -- myprog arg1 arg2
```

3.4.2 Adding caliper functions

Caliper functions for the various Oxbow tools are provided in an interface library in the oxbow installation.

To use this caliper function library, modify your C/C++ source code with the following:

```
#include <oxbow.h>
// unprofiled code section
oxbow_pin_imix_zero();   //reset statistics
oxbow_pin_imix_start();  //start profiling
// profiled code section
oxbow_pin_imix_stop();   //stop profiling
// unprofiled code section
```

When compiling, add the following include flags to your object compilation:

```
-I${OXBOW_TOOLS_DIR}/include
```

Add the following library flags during the link step:

```
-L${OXBOW_TOOLS_DIR}/lib -loxbow
```

The `OXBOW_TOOLS_DIR` variable is set using the `envvars.sh` script. So, an example compilation after adding caliper functions would be:

```
$ source /path/to/oxbow/oxbow-tool-vendor/etc/envvars.sh
$ cc -I${OXBOW_TOOLS_DIR}/include -c myprog.c
$ cc -o myprog myprog.o -L${OXBOW_TOOLS_DIR}/lib -loxbow
```

3.4.3 Running with caliper functions

Convenience script use for marked code:

```
$ do-pin-imix.sh myoutdir -- mpirun -n 64 -- ./myprog arg1 arg2
```

To run directly (no convenience script) on unmarked code:

```
$ source /path/to/oxbow/oxbow-tool-vendor/etc/envvars.sh
$ mpirun -n 64 ${PIN_DIR}/intel64/bin/pinbin -follow_execv -t ${OXBOW_TOOLS_DIR}/bin/imix.pin \
         -start_address oxbow_pin_imix_marker_start:repeat \
         -stop_address oxbow_pin_imix_marker_stop:repeat \
         -zero_stats_address oxbow_pin_imix_zero_stats:repeat \
         -emit_stats_address oxbow_pin_imix_emit_stats:repeat \
         -category -i -- myprog arg1 arg2
```

3.5 membw: Memory Bandwidth Measurement

To use the memory bandwidth tool, you need PAPI installed in your environment. The environment variable `PAPI_DIR` must be set to a suitable value for your environment.

Then you can use the following three calipers in your application:

- `init_papi_counters(int mpi_rank):`

  Initializes the PAPI library. Should be called once, before the other calipers are called. The rank is used as a suffix for the output file name.

- `start_papi_counters():`

  This needs to be inserted at the beginning of the section you wish to profile.

- `stop_papi_counters(const char* name):`

  This needs to be inserted at the end of the section you wish to profile. The 'name' parameter is used to identify the code section that you profiled.

You can call the start and stop calipers multiple times. The results are appended to the output file.

When building your application, you should

- Add `-I${OXBOW_TOOLS_DIR}/include` and `${PAPI_DIR}/include` to your include options.
- Add `-L${OXBOW_TOOLS_DIR}/lib -lmembandwidth` and `${PAPI_DIR}/lib -lpapi` to your link options.

Please ensure that you have the suitable PAPI include and link flags for your platform.

The calipers will create one output file per process. The output files are named:

`bandwidth_counts-<mpi_rank>-<proc_pid>.csv`

3.6 reused: Reuse Distance PIN Tool

You can obtain the reuse distance metrics by running:

```
$ ${PIN_DIR}/intel64/bin/pinbin -follow_execv -t ${OXBOW_TOOLS_DIR}/bin/reuse_dist_cal \
  -- mpiexec -n nprocs your_app
```

The output files are named `hist_reuse_dist_cal__xxx.txt` where xxx is the PID of process.

The tool will collect reuse distance for the whole application by default.

Similar to the Memory bandwidth tool, we can insert a few function calls in the application source to mark a portion of the application we are interested in profiling.

- `inst_begin_instrumentation():` Mark beginning of the computation section of interest.
- `inst_finish_instrumentatin():` Mark end of section.

When building your application, you should

- Add `-I${OXBOW_TOOLS_DIR}/include` to your include options.
- Add `-L${OXBOW_TOOLS_DIR}/lib -lreused` to your link options.

Acknowledgment

This research is sponsored by the Office of Advanced Scientific Computing Research in the U.S. Department of Energy. The paper has been authored by Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract #DE-AC05-00OR22725 to the U.S. Government. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.