Empirical Roofline Tool User's Manual

Terry J. Ligocki
Leonid Oliker
Brian Van Straalen
Sam Williams
Nicholas Wright
Matthew Cordery
Wyatt Spears
Linda Lo

November 11, 2015

Contents

1 Overview and Installation

1.1 Overview

The Roofline Performance Model for computational performance has been shown to be a useful simplification of the complexity of processor and memory systems. An overview of this model can be found at:

https://crd.lbl.gov/departments/computer-science/performance-and-algorithms-research/research/roofline

This site also contains pointers to available software, such as this tool, and references to publications pertaining to the Roofline Performance Model.

The Roofline Toolkit was created to help software developers use the Roofline Performance Model to improve the performance of their codes. One of the requirements for using the Roofline Performance Model is having a roofline characterization of the target machine.

In the past, this characterization was obtained from the machine specification data. This was time consuming and error prone. In addition, it lead to theoretical maximums that might not be achievable by any code running on the machine.

The Empirical Roofline Tool, ERT, generates the required characterization of a machine empirically. It does this by running kernel codes on the machine so the results are, by definition, attainable by some code(s) and the options used to compile and run the code are known. The ERT generates the bandwidth and gflop/sec data needed by the Roofline Performance Model.

The input to the ERT is a configuration file and the final output is a roofline graph in PostScript format and the roofline parameters in JSON format. There is also a significant amount of intermediate data generated that can be explored and used to better understand a given kernel code and machine.

Currently, the ERT can utilize MPI, OpenMP, and Cuda/GPU for parallelization.

1.2 Installation

To install ERT, you simply need to use “git” to “clone” the Roofline Toolkit located at https://bitbucket.org/berkeleylab/cs-roofline-toolkit on your local machine.

The ERT is under “Empirical_Roofline_Tool-1.1.0”. There you will find a “README” file, a PDF of this document (“ERT_Users_Manual.pdf”), the ERT executable (“ert”), some sample configuration files (“Config/config.ert.*”), and everything else used by the ERT.

To complete the installation, you need to make sure you have a few other pieces of software installed:

- Python that is at least version 2.6 and below version 3.x
- GNUplot that is at least version 4.2.x

To run in parallel, using MPI, OpenMP, and/or Cuda/GPU, you will need:

- A working installation of MPI
- A compiler and operating system that supports OpenMP code generation, compiling, linking, and running
- A compiler, operating system, and hardware that supports Cuda/GPU code generation, compiling, linking, and running

If you problems using the ERT, please read the section on troubleshooting the ERT. It contains suggestions that the developers and testers have used when trying to correct problems running the ERT on new machines with new configuraiton files.

2 Configuration

To configure the ERT, you need to create a configuration file that is setup correctly for the machine you will be testing. Several examples are provided with the ERT distribution. They are located at under “Config” in the installation directory and are named “config.*”.

In addition to the example configuration files, there are corresponding batch scripts under “Batch” (for systems with batch queues) and results under “Results”.

The examples currently provided are:

- “config.babbage.nersc.gov.01” - An example configured for testing a Knight’s Corner (KNC) node using various combinations of MPI and OpenMP to exercise the host and not the MIC .
- “config.babbage.nersc.gov.02” - An example configured for testing a Knight’s Corner (KNC) node using OpenMP to exercise the MIC and not the host.
- “config.edison.nersc.gov.01” - An example configured for a Cray XC30 node (2 x 12-core Intel ”Ivy Bridge” processors) using various combinations of MPI and OpenMP.
- “config.madonna.lbl.gov.01” - An example configured for a Linux workstation (8-core Intel Xeon CPU E5530, 2.4 GHz) using various combinations of MPI and OpenMP.
- “config.madonna.lbl.gov.02” - An example configured for a Linux workstation (8-core Intel Xeon CPU E5530, 2.4 GHz) using only OpenMP.
- “config.mira.alcf.anl.gov.01” - An example configured for a IBM Blue Gene/Q node (16 1600 MHz PowerPC A2 cores) using various combinations of MPI and OpenMP.
- “config.titan.ccs.ornl.gov.01” - An example configured for a Cray XK7 node (host: 16-core 2.2 GHz AMD Opteron 6274 (Interlagos) processor) using various combinations of MPI and OpenMP.
- “config.titan.ccs.ornl.gov.02” - An example configured for a Cray XK7 node (GPU: NVIDIA Kepler accelerator) using various number of thread-blocks and threads-per-block.

Looking at these examples will help clarify all the descriptions and definitions below.

In general, the configuration file contains lines that define the parameters needed by the ERT to compile the specified drivers and kernels, run them over a range of MPI, OpenMP, and Cuda/GPU settings, gather the results, and generate the final roofline characterization of the machine.

Starting with an example configuration file (see above) and modifying it is probably the best way to get started. In the next sections, all the ERT configuration directives will be explained so you can understand the examples and make your own configuration files.

If you problems using the ERT with your configuration file, please read the section on troubleshooting the ERT. It contains suggestions that the developers and testers have used when trying to correct problems running the ERT with new configuraiton files.

2.1 Terminology

Anytime a "set of integers" is mentioned below it refers to a comma separated list where each entry can be a single integer or a range of integers specified by two integers separated by a hyphen. For example:

- "8" - Just the integer 8.
- "1,2,4" - The integers 1, 2, and 4.
- "1-8" - The integers 1, 2, 3, 4, 5, 6, 7, and 8.
- "1,2,6-8" - The integers 1, 2, 6, 7, and 8.

2.2 Comment Lines

Any line beginning with a "#" character is treated as a comment and will not be used by the ERT. Currently, comments and the comment character are not allowed mid-line. If they appear mid-line, they will simply be considered part of the line and used by the ERT.

2.3 Storing Results

To specify where results will be stored:

- "ERT_RESULTS" - this specifies the directory where all results of running the ERT will be kept. If this it is a relative path, it will be relative to the directory you are in when you run the ERT. If this is an absolute path, the same directory will be used every time this configuration file is specified regardless of where you are when you run the ERT.

2.4 Specification Performance Numbers

To specify roofline performance numbers based on the machine specification, i.e., theoretical performance numbers:

- "ERT_SPEC_GBYTES_L#" - Give the theoretical maximum bandwidth in GBytes/sec for the L# cache on the machine being tested.
- "ERT_SPEC_GBYTES_DRAM" - Give the theoretical maximum bandwidth in GBytes/sec for the DRAM on the machine being tested.
- "ERT_SPEC_GFLOPS" - Give the theoretical maximum computation rate in GFLOPs/sec for the machine being tested.

These numbers are passed along as metadata and are available in the final JSON roofline output generated by the ERT.

2.5 Building the Code

To specify which code to build and how to build it:

- "ERT_DRIVER" - the driver code to use with the selected kernel code (see below). There is currently only one driver, "driver1". The ERT looks for "ERT_DRIVER.c" in "Drivers" under the ERT installation directory.
- "ERT_KERNEL" - the kernel code to use with the selected driver code (see above). There is currently only one kernel, "kernel1". The ERT looks for "ERT_KERNEL.c" in "Kernel" under the ERT installation directory.
- "ERT_MPI" - "False" to compile without MPI and "True" to compile with MPI.
- "ERT_MPI_CFLAGS" - if "ERT_MPI" is "True", compilation flags specific to MPI that are needed or wanted.
- "ERT_MPI_LDFLAGS" - if "ERT_MPI" is "True", linking/loading flags specific to MPI that are needed or wanted.

- "ERT_OPENMP" - "False" to compile without OpenMP and "True" to compile with OpenMP.
- "ERT_OPENMP_CFLAGS" - if "ERT_OPENMP" is "True", compilation flags specific to OpenMP that are needed or wanted.
- "ERT_OPENMP_LDFLAGS" - if "ERT_OPENMP" is "True", linking/loading flags specific to OpenMP that are needed or wanted.
- "ERT_GPU" - "False" to compile without the Cuda code and "True" to compile with Cuda code.
- "ERT_GPU_CFLAGS" - if "ERT_GPU" is "True", compilation flags specific to the Cuda code that are needed or wanted.
- "ERT_GPU_LDFLAGS" - if "ERT_GPU" is "True", linking/loading flags specific to the Cuda code that are needed or wanted.
- "ERT_FLOPS" - A set of integers[1] specifying the FLOPS per computational element (in the current case an 8 byte, double precision floating point number). This is specific to the current driver/kernel pair and will probably be generalized in the future.
- "ERT_ALIGN" - An integer specifies the alignment (in bits) of the data being manipulated.
- "ERT_CC" - The compiler to use to build the ERT test(s).
- "ERT_CFLAGS" - The flags to use when compiling the code.
- "ERT_LD" - The linker/loader to use to build the ERT code(s).
- "ERT_LDFLAGS" - The flags to use when linking/loading the code.
- "ERT_LDLIBS" - Explicit libraries to include when linking/loading the code.

2.6 Running the Kernel

To specify how to run the kernel once it is built:

- "ERT_RUN" - This is a command that is used almost verbatim to run the driver/kernel code that is built. There are three strings that are replaced wherever they appear in this command to customize the command:

[1]In this context, a "set of integers" is a comma separated list where each entry can be a single integer or a range of integers specified by two integers separated by a hyphen. For example, "8" or "1,2,4" or "1-8" or "1,2,6-8"

  - "ERT_OPENMP_THREADS" - The number of OpenMP threads being run.
  - "ERT_MPI_PROCS" - The number of MPI processes being run.
  - "ERT_CODE" - The current code being run.

In addition, when "ERT_GPU" is "True", two arguments are added to "ERT_RUN" command: The number of thread-blocks and the number of threads-per-block.

2.7 MPI, OpenMP, and Cuda/GPU

To specify valid combinations of MPI processes, OpenMP threads, GPU thread-blocks, and GPU thread-per-block:

- "ERT_PROCS_THREADS" - A set of integers that constrain valid products of the number of MPI processes and the number of OpenMP threads. For example, if this is '8' then only combinations of MPI processes and OpenMP threads have to multiply to give 8, e.g., 2 and 4, 8 and 1, are run.
- "ERT_MPI_PROCS" - A set of integers specifying possible number of MPI processes to run. These are subject to constraints imposed above.
- "ERT_OPENMP_THREADS" - A set of integers specifying possible number of OpenMP threads to run. These are subject to constraints imposed above.
- "ERT_BLOCKS_THREADS" - A set of integers that constrain valid products of the number of thread-blocks and the number threads-per-block. For example, if this is '28672' then only combinations of thread-blocks and threads-per-block have to multiply to give 28672, e.g., 28 and 1024, 448 and 64, are run.
- "ERT_GPU_BLOCKS" - A set of integers specifying possible number of thread-blocks to run. These are subject to constraints imposed above.
- "ERT_GPU_THREADS" - A set of integers specifying possible number of threads-per-block to run. These are subject to constraints imposed above.

2.8 Miscellaneous

Some miscellaneous parameters:

- "ERT_NUM_EXPERIMENTS" - The number of times to rerun the same code with all the parameters set the same. This is used to get a statistical sample since performance can sometimes vary without changing the local experiment.
- "ERT_MEMORY_MAX" - The maximum number of bytes to allocate/use for the entire run. These are divided up across MPI processes and OpenMP threads.
- "ERT_WORKING_SET_MIN" - The minimum size (in 8-byte, double precision floating point numbers) for an individual process/thread working set.
- "ERT_TRIALS_MIN" - The minimum number of times to repeatly run the loop over the working set. This will be the maximum number of trials when the working set is the largest and it will scale up so the product of the maximum number of trails and the working set size are approximately constant.
- "ERT_GNUPLOT" - How GNUplot is invoked.

3 Running

On one level, running the ERT is straightforward. Simply invoke the "ert" command with a single command line argument which specifies the file that has the configuration information. If all goes well, the tool runs to completion and at the end tells you the location of resulting the roofline graph file and JSON database file.

The "ert" command has the following command line options:

```
--version          show program's version number and exit
-h, --help         show this help message and exit
--verbose=VERBOSE  Set the verbosity of the screen output
                   [default = 1].
                   = 0 : no output,
                   = 1 : outlines progress,
                   = 2 : good for debugging (prints all
                         commands)
--quiet            Don't generate any screen output,
                   '--verbose=0'

Build options:
  --build          Build the micro-kernels [default]
  --no-build       Don't build the micro-kernels
  --build-only     Only build the micro-kernels

Run options:
  --run            Run the micro-kernels [default]
  --no-run         Don't run the micro-kernels
  --run-only       Only run the micro-kernels

Post-processing options:
  --post           Run the post-processing [default]
  --no-post        Don't run the post-processing
  --post-only      Only run the post-processing
  --gnuplot        Generate graphs using GNUplot [default]
  --no-gnuplot     Don't generate graphs using GNUplot
```

The "verbose"/"quiet" options allow the user to control the amount of diagnostic output produced when the "ert" command is executed:

```
    --verbose=0, -quiet  No diagnostic output is generated

    --verbose=1          An outline is produced as the code
                         is running showing its progress (default)

    --verbose=2          An outline and list of commands being
                         executed are produced as the code
                         is running showing the details of
                         its progress
```

The "build"/"no-build"/"build-only", "run"/"no-run"/"run-only", and "post"/"no-post"/"post-only" options allow the user to control what is done during the current execution of “ert”. By default, the micro-kernels are built, run, and the results are post-processed to generate the final roofline outputs.

It is sometimes necessary to build the micro-kernels on a host machine, run them on the target machine, and post-process them somewhere else entirely. This would be accomplished by:

- Running “ert” on the host machine with the “--build-only” option.
- Transferring the entire ERT_RESULT directory created to the target machine.
- Running “ert” on the target machine with the “--run-only” option using exactly the same configuration file as on the host.
- Transferring the entire updated ERT_RESULT directory to the post-processing machine.
- Running “ert” on the target machine with the “--post-only” option using exactly the same configuration file as on the host.

This is only one example using the build, run, and post-process options. Other permutations are possible to accommodate needs.

There is also the “gnuplot/no-gnuplot” options which can be used to control if GNUplot is used to produce graphs (in PostScript format) during the post-processing. If GNUplot isn’t available or you don’t want these graphs, simply use the “--no-gnuplot” option.

Of course, this is simply the “tip of the iceberg”. What actually happened? Where did everything get stored and processed? What do you do if final results don’t look reasonable? The sections below give more detailed information about the ERT that can help answer these questions when they arise.

Also, if you are having trouble getting the ERT to run with your configuration file use the “--verbose=2” option. This usually gives enough information to help diagnose the problem. If not, see the section on troubleshooting the ERT.

3.1 Overall Run Structure

When the ERT is run with a configuration file, e.g. “ert Config/config.edison.nersc.gov.01”, it proceeds as follows:

1. Output the version number of the release.
2. Read the configuration file.
3. Check to see if there is an existing output directory with the same configuration file. If so, use that directory. If not, create a new output directory and copy the current configuration file there.
4. For each value specified by “ERT_FLOP”:
    (a) Build the roofline characterization code with that number of FLOPs per element.
    (b) For each number of MPI processes specified by “ERT_MPI_PROCS” and each number of OpenMP threads specified by “ERT_OPENMP_THREADS:
        i. If the product of the number of MPI processes and OpenMP threads isn’t one of the numbers specified by “ERT_PROCS_THREADS” go back to the previous step.
        ii. If this is a new output directory, run the code as many times as “ERT_NUM_EXPERIMENTS” specifies and save all the results. If not, skip this step - the code has already been run in this configuration.
        iii. Process the output to produce local results including a variety of graphs generated by GNUplot in PostScript format.
5. Gather all the results to produce the final roofline results as a graph generated by GNUplot in PostScript format and a JSON output file. Report the location of these two files.

The next sections give more details about these steps and the data they generate.

3.2 Storing Results

When the ERT is run two forms of output are generated. The obvious one is all output that goes to the screen by default. Redirecting it to a file is a good idea so you can review it later if there is a problem.

In the future, this output will probably go directly to a log file and there will be a way of controlling the verbosity of the output, especially what goes directly to the screen.

The other output goes to files in a directory structure below the "ERT_RESULTS" directory. If the "ERT_RESULTS" directory doesn't exist the ERT attempts to create it. Otherwise, it looks in the directory for any files of the form "Run.#". If it finds any, it checks the configuration file stored under them to see if they match the current configuration file specified on the command line.

At the moment, this match must be exact in every character. Eventually, this will be relaxed as much as possible to detect functional equivalence.

If a match is found, the ERT attempts to continue running and storing results under the matching "Run.#" directory. If no match is found, the ERT creates a new "Run.#" directory where "#" is the smallest positive integer not currently being used.

Once this directory has been found or created, the current configuration file is copied there. This is also the location of the roofline graph file and the JSON database file when the run completes.

Under the "Run.#", the ERT creates one directory for each of the "ERT_FLOPS" values. These directories are named "FLOPS.#" where "#" is a three digit, zero padded number. An executable is built in each of these directories as the value being used changes the code being compiled.

Under each "FLOPS.#" directory, the ERT creates directories and subdirectories for type of parallelism turned on in the configuration file. The nesting of the directories is in the order: MPI processes, OpenMP threads, Cuda/GPU thread-blocks, Cuda/GPU threads-per-block.

If any of these forms of parallelism are turned off, subdirectories are not created for them. The names of the subdirectories are: "MPI.#" for the number of MPI processes, "OpenMP.#" for the number of OpenMP threads, "GPU_Blocks.#" for the number of GPU thread-blocks, and "GPU_Threads.#" of the number of GPU threads-per-block.

At this point, the compile time and runtime environment for the code is completely specified, the code is run, and the results are stored in appropriate subdirectory. One file, "try.#", is created for the output of each experiment.

The format of the "try.#" output will be documented here in future releases of the ERT.

3.3 Processed Results

Next the individual results are processed to produce individual graphs and summaries. Then these summaries are combined to produce the final ERT output for the overall set of runs.

In addition, a significant amount of metadata is carried along through this process. This includes timestamps, the ERT configuration file, specific number for flops/element, MPI processes, OpenMP threads, etc.

In each of subdirectories under each "FLOPS.#" directory containing the "try.#" output (see above), the independent run/experiments are combined into one result by finding the minimum, median, and maximum bandwidths and GFLOP rates over the runs and storing this information in a file named "pre".

Then the maximum bandwidth and gflop rate for each working set size is extracted from "pre" and stored in "max". Finally, histograms are used to determine plateaus in the "max" data and, along with using the absolute maximum, the roofline results are determined for all of the experiments. This result is stored in "sum".

The format of all these summary files will be documented here in future releases of the ERT.

In all cases, the metadata is propagated with the processed data. Four graphs are created based on this processed data unless the "--no-gnuplot" option is specified:

- "graph1.ps" - a graph of all the bandwidth data in "pre" that shows run time versus the number of trials for each working set size. It is expected that this graph will be set of lines with slope 1.0 as the runtime should vary linearly with the number of trials once any constant overheads outside the trial loop are overcome.

- “graph2.ps” - a graph of all the bandwidth data in “pre” that shows working set size versus gbytes/sec. The lines connect the result for various numbers of trails where the working set is constant. The maximum envelope of this graph is “graph3.ps”.
- “graph3.ps” - a graph of maximum of the bandwidth data from “max” for each working set size.
- “graph4.ps” - a graph of maximum of the GFLOPs/sec data from “max” for each working set size.

These graphs are useful in diagnosing problems in the overall results of the ERT. They are also interesting in their own right as they give detailed information about the behavior of the architecture and compiler for each parameter choices, the behavior as a function of work set size, etc.

3.4 Final Results

All the individual results are combined into two final results unless the “--no-gnuplot” option is specified then only the JSON output is generated. Both are stored directly under the “Run.#” directory:

- “roofline.ps” - the roofline graph extracted from all the runs made using the configuration file, “config.ert” (also stored in this directory).
- “roofline.json” - the parameters used to generate the roofline graph along with all the relevant metadata. The format of this data needs to be fully documented but the output is formatted in a way that makes it fairly easy to see the structural details. Also, there is separate metadata for the bandwidth results and the gflop rate results because these, typically, come from runs with different parameters. Ideally, the metadata they have in common should be factored out and placed at a higher level in JSON database structure with on the differences appearing at the lower levels.

3.5 Restarting/Continuing Runs

If the configuration file used is identical, an interrupted/incomplete run will start from where it left off. It will repeat all of the building and post processing but it won't rerun any code that has already run to completion.

This capability needs to be made more modular and flexible but it provides some needed functionality even in it's current form.

3.6 Graphs

All the graphs generated by the ERT are produced using GNUplot and a GNUplot script file. The "--no-gnuplot" option disables the production of these graphs and the associated GNUplot script files. For a given graph, "graph.ps", there will also be a GNUplot script file, "graph.gnu", in the same directory. The graph can be regenerated by running GNUplot and typing the command 'load "graph.gnu"'.

If you want to modify any graph produced by the ERT, you can use the GNUplot script as a starting point and modify it as needed to produce the graph you would like. This includes modifying titles and labels, adjusting axes, moving and adding text, etc.

4 Extended Use

This tool's uses could be extended to run user's kernels (and drivers). It basically forms a software harness that runs a code with a variety of compilations and a variety of MPI processes, OpenMP threads, and Cuda/GPU threadblocks and threads-per-block, stores the results of the runs, and compiles information about specific runs and the overall set of runs.

In future releases of the ERT, more details of the internal formats and processing will be documented so extending the ERT and using if for more general purposes will be simplified. In addition, more drivers and kernels will be released to cover a broader range of test codes.

5 Troubleshooting

This section contains a list of suggestions from the developers and testers of the ERT that have helped us get the ERT working when it hasn't successfully run to completion. Hopefully, they will be helpful if you need to troubleshoot problems running the ERT with on your own machines with your own configuration files.

The easiest step, which should probably be your first step, it do get the most output possible from the ERT. To do this use the "--verbose=2" option. This will cause the ERT to print all the commands it is attempting to execute - including the final command before the ERT failed.

Often, simply looking at the last command will help identify the problem. If not, look at the previous commands to see if they seem to be constructed correctly to build, link, and run the ERT micro-kernel.

If this doesn't indicate why things failed, you can try to run the last command interactively and see what happens. This sometimes generates errors/warnings that weren't shown when the command was run within the ERT.

If the last command is the running of the micro-kernel, all output (and errors) will be redirected to a file shown on the command output. There will be a rather long path to the file and the file will have the name "try.#" where "#" is a number. Look in this file for any and all output resulting from running the micro-kernel - including error messages.

If you are still having a problem after trying these suggestions or if it isn't clear how to follow some of them, feel free to contact us using the contact information on the WWW site where you obtained the Roofline Toolkit. We can then work with you to get the ERT running for you.