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 16, 2014

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 has been 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 is 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.

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-0.8.0". 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 and/or OpenMP, you will need:

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

2 Configuration

To configure 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.ert.*".

The examples currently provided are:

- "config.ert.Linxu.Intel.pureOpenMP" - An example configured for a Linux workstation only uses OpenMP and the Intel compilers.
- "config.ert.Edison.flatMPI" - An example configured to run on NERSC's Edison supercomputer. This example only uses MPI on a single computational node.

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

In general, the configuration file contains lines that define the parameters needed by the ERT to compile specified drivers and kernels, run them over

a range of OpenMP and MPI 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.

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 has a relative path, it will be relative to the directory you are in when you run the ERT. If this has 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 on 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 on 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_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 test(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:
  - "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.

[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 "1-4,6-8"

2.7 MPI and OpenMP

To specify valid combinations of MPI processes and OpenMP threads:

- "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 the only combinations of MPI processes and OpenMP threads run have to multiply to give 8, e.g., 2 and 4, 8 and 1.
- "ERT_MPI_PROCS" - A set of integers specifying possible numbers of MPI processes to run. These are subject to constraints imposed above.
- "ERT_OPENMP_THREADS" - A set of integers specifying possible numbers of OpenMP threads 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 trails to repeat running over each working set size. 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 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.

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.

3.1 Overall Run Structure

When the ERT is run with a configuration file, e.g. "ert Config/config.ert.Linux.Intel.pureOpenMP", 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 number 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.

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 printing it does as it works. This either goes to your window or can be redirected to a file of your choice. Redirecting it 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 outputs go 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 - one with 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 a directory for each MPI process number used, "MPI.#". Under each of these, the ERT creates a directory for each OpenMP thread count used, "OpenMP.#".

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 "FLOPS.#/MPI.#/OpenMP.#" directory. 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

At this point, all the configurations of all the code variations have been run at least once and the results have been stored. Next these result 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 directory of the form "FLOPS.#/MPI.#/OpenMP.#", the independent run/experiments are combined into one result by finding the minimum, median, and maximum times 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:

- "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 overtimes outside the trial loop is overcome.
- "graph2.ps" - a graph of all the bandwidth data in "pre" that shows working set size versus gflops/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 gflop/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 choice.

3.4 Final Results

All the individual results are combined into two final results. 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 lower level.

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. 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 and OpenMP threads, 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.

A Specific Architectures/Machines

This appendix contains details specific to some of the architectures/machines where the ERT has been run.

A.1 Linux Workstations

Figure 1 shows the roofline graph obtained running "ert" with the "Config/config.ert.Linux.Intel.pureOpenMP" configuration file on a Linux workstation. The workstation had two Intel Xeon E5530 CPUs, 4 cores each, running at 2.4 GHz with hyperthreading turned on. These results were also recorded in JSON format (see Section 3.4).

A.2 NERSC Edison

Figure 2 shows the roofline graph obtained running "ert" with the "Config/config.ert.Edison.flatMPI" configuration file on NERSC Edison. These results were also recorded in JSON format (see Section 3.4).

A.3 Intel MIC

A.4 ALCF Mira

A.5 NERSC Hopper

A.6 OLCF Titan

A.7 nVidia GPU

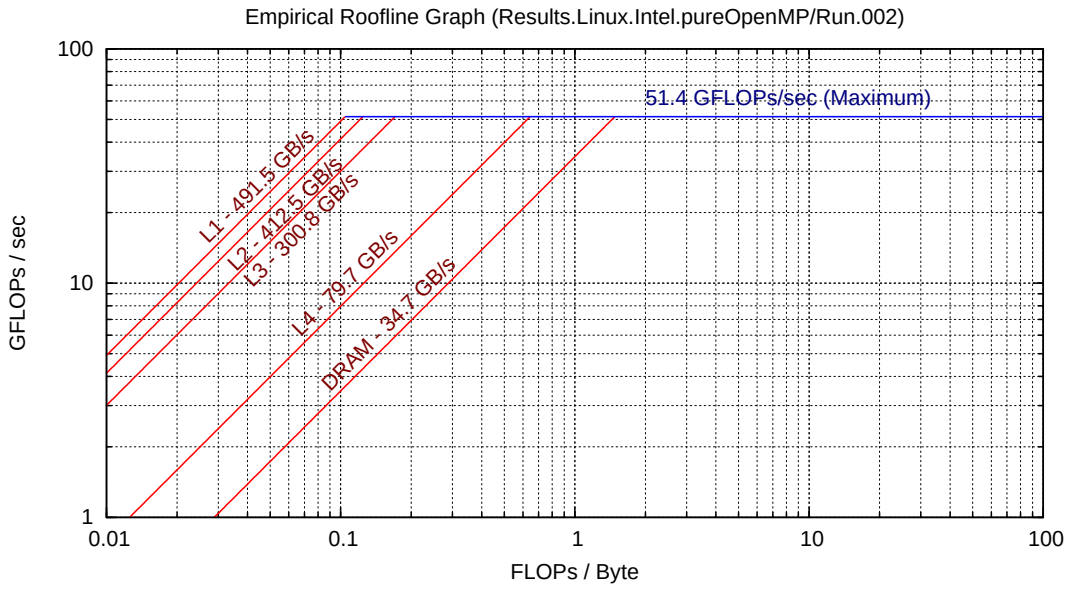

Figure 1: Roofline graph for a Linux workstation

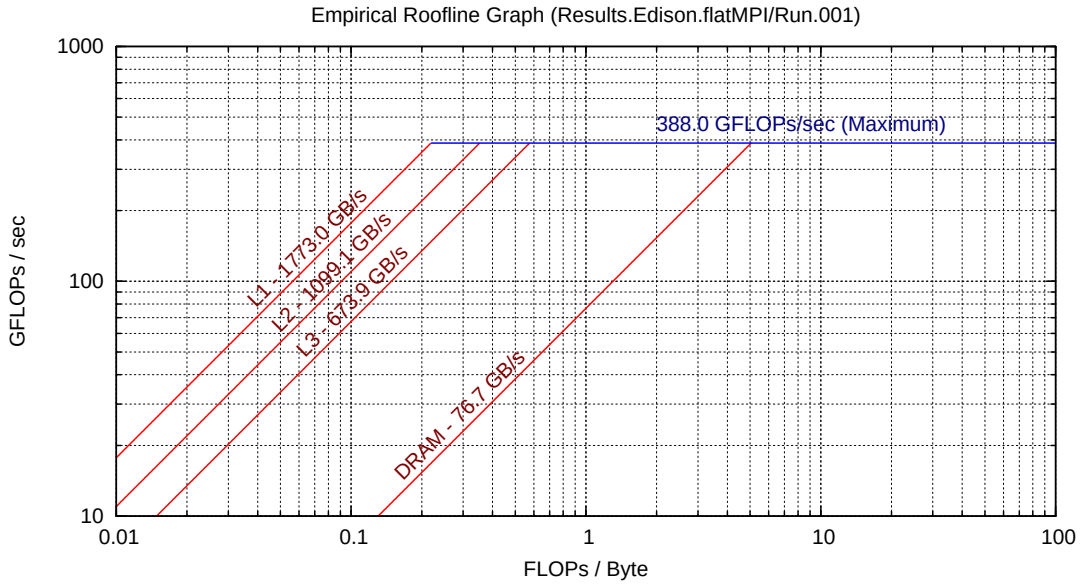

Figure 2: Roofline graph for NERSC Edison