Empirical Roofline Tool Design and Use

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

October 9, 2014

Contents

1 Overview and Installation

1.1 Overview

The roofline model of computational performance has been shown to be a useful simplification of the complexity of processor and memory systems. The Empirical Roofline Tool, ERT, generates the bandwidth and gflop/sec data needed by the Roofline model by directly running benchmark codes on a given machine.

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 of intermediate data generated that may be of use to user.

1.2 Installation

To install ERT, expand the archive file, "ert.tar.gz", where you would like to have the ERT installed. In the expanded archive, you will find a "README" file, a PDF of this document ("ert_users_manual.pdf"), the ERT executable ("ert"), some sample configuration files ("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. Several examples are provided with the ERT distribution, are located at the top level of the installation directory, and are named "config.ert.*".

The three examples currently provided are:

- "config.ert.Linux" - An example configured for a Linux workstation that uses both MPI and OpenMP and the GNU compilers
- "config.ert.OSX" - An example configured to run in serial on an Apple laptop running OSX
- "config.ert.Edison" - An example configured to run on NERSC's Edison supercomputer - this is very preliminary and not validated

2.1 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 everytime this configuration file is specified regardless of where you are when you run the ERT.

2.2 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.

2.3 Running the Code

To specify how to run the code 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 "1-4,6-8"

  - "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.

2.4 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 multiply to give 8.
- "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.

2.5 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.

Looking at the configuration examples, "config.ert.*", in the top level ERT directory will give some examples that should help clarify all these descriptions and definitions.

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. It all is 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?

3.1 Storing Results

When the ERT is run with a configuration file, e.g. "ert config.ert.Linux", 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 chose. Redirecting it is a good idea so you can review 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 controling the verbosity of the output, especially what goes directly to the user.

The other ouptuts go to file in a directory structure below the "ERT_RESULTS" directory. If the "ERT_RESULTS" directory doesn't exist valid MPI process

numberthe ERT attempts to create it. Then it look 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 specificed 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 functionally 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 number not currently being used.

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

Under the "Run.#", the ERT creates one directory for each of the "ERT_FLOPS" values. This directory is 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.value" (as above). Under each of these, the ERT creates a directory for each OpenMP thread count used, "OpenMP.value".

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 each run/experiment.

The format of the "try.#" output is documented in the developer guide (a work in progress) and will be documented here in the near future.

3.2 Processing 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 "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 these set of runs/experiments. This result is stored in "sum".

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 number of trials increases as time 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 which the working set is constant.
- "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.

3.3 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 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 of 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.

3.4 Restarting/Continuing Runs

If the configuration file used is identical, an interupted/incomplete run will start from where it left off. It will repeat all of the building and post processing but it won't rerun case that have already been 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.

4 Future Work

This is a very preliminary release of the ERT. First of all, although it is internally modular, it is presented as a monolithic code. The internal modularity needs to be exposed so the ERT can be used in a more flexible way.

The ERT needs to be run and validated using many more architectures, operating systems, and compilers. This will require the ERT's design to grow and change. It will also point out problems/limitations in the current drivers/kernels being used.

Everything, externally (e.g., output formats) and internally (e.g., code comments) needs to be fully documented. The ERT needs to be cleaned up internally as there are a number of shortcuts that I took to get this release out quickly.

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

A.2 Mac OSX

A.3 NERSC Hopper

A.4 NERSC Edison

A.5 ALCF Mira

A.6 OLCF Titan

A.7 Intel MIC

A.8 nVida GPU