Lightweight Performance Instrumentation (lwperf)

Eric Anger - eanger@gatech.edu

Contents

1 Overview

The Lightweight Performance Instrumentation Library (lwperf) collects performance data from applications and logs them to various backends including CSV files, to an Eiger database, or calls to Eiger models for running as a skeleton within the SST/macro simulator.

1.1 Introduction

Lightweight Performance Instrumentation Library (lwperf) is a tiny collection of simple, portable functions aimed at making it easy to gather high-level compute cost numbers and the driving algorithm parameters from individual cluster nodes running real applications. The author's intended use of these numbers is to construct models that support interpolation-based estimates of compute costs at other parameter values.

This tool is not intended to collect data for code segments containing inter-node communication code, particularly MPI code. Rather, it is intended for characterizing node-level serial, locally multi-threaded (OpenMP nests in an MPI/OMP hybrid), or locally accelerated code sections. It is usually the performance of the node or local (co)processor group in aggregate rather than individual thread performance which is of interest, as it is the group which provides the total workload to shared local resources such as memory. Single-node but multi-rank instances of MPI-only codes may also be usefully profiled with this tool. There is no technical reason lwperf cannot be used to profile MPI calls, but there are better tools for profiling MPI widely available.

1.2 Background

The driving need for this tool is portability: numerous superior (and heavy-weight) but proprietary performance analysis tools exist. As with mini-applications, we need mini-analysis tools. Free MPI-oriented profiling tools already exist, but they usually stop at profiling communications and do not collect the parameters that control local workloads. Source-level insight is required to identify and capture the influential parameters.

2 Implementation

The lwperf library is written in C++, but only exposes a simple C API. All functionality is implemented internally, with the details hidden from the end user.

The library also provides optional support for PAPI, Eiger, and SST/macro. These features are conditionally compiled into the library and a macro describes if these features are included.

2.1 Dependencies

C++ The library is written in C++. The implementation requires features from C++11 (i.e. auto variables, range-based for loops, `std::chrono` timing functions); however, since none of these features are externally visible, the correct runtime library can be statically linked and hidden from the end user.

C A C99 compiler is all that is required to use the library interface.

PAPI ***optional*** PAPI version 5.3 or newer is required to tally hardware performance counters. Configure with –enable-papi to turn on.

Eiger ***optional*** Eiger version 3.0 or newer is required to dump data to the Eiger database. Configure with –enable-eiger to turn on.

SST/macro ***optional*** SST/macro version 2.4 or newer is required to make calls to Eiger models.

3 Application Programming Interface

This section documents the API for interacting with the lwperf library.

3.1 Setup

The library API is specified by a single header, `lwperf.h`. This header pulls in implementation details depending on which output format to use. Several preprocessor flags describe where the logging information should be output. If no flags are selected, no logging will be performed.

`LWPERF_USE_CSV` Output to CSV files, one per site.

`LWPERF_USE_EIGER` Output to Eiger database, if supported.

`LWPERF_USE_SSTMAC` Evaluate the models and increment SST/macro simulated time, if supported

The library is initialized with a call to `lwperf_init`, which takes the name of the machine, application, and database (i.e. output location) as parameters. This returns a `lwperf_t` object containing all state for that lwperf instance. After execution, call `lwperf_finalize` to clean up the `lwperf_t` object.

Any invariants about execution, implying no variation during the course of application execution, can be documented with the `lwperf_add_invariant` function. This takes the name and value for the metric to be recorded.

3.2 Profile sites

At any site (usually a loop nest) where data collection is wanted, the collection point is defined at the beginning of the site with a unique name. The profile site starts with a call to `lwperf_log` and ends with a call to `lwperf_stop`. Execution time (and optionally PAPI counters, if supported) are recorded for each `log`/`stop` pair. Parameters that are of interest for this region can be added with the `lwperf_add_site_param` function. Each parameter has a name and an associated value. This are recorded for each `log`/`stop` pair.

A helper macro, `lwperf_add_site_params`, is included to help ease adding multiple metrics to a profile site. Any number of name/value metrics can be added in a single call.

The following is an example from a molecular dynamics code:

```
lwperf_t perf = lwperf_init("testbed", "md-app", "output_db");
lwperf_add_site_param(perf, "Tenergy", "nlocal", nlocal);
lwperf_add_site_param(perf, "Tenergy", "nneigh", nneigh);

// site Tenergy
lwperf_log(perf, "Tenergy");
for (int i = 0; i < nlocal; i++) { // outer loop
  for (int j=0; j< numneigh; j++) {
      ... // computational work
  }
}
lwperf_stop(perf, "Tenergy");
```

3.3 Hardware Performance Counters

Hardware performance counters may be collected in addition to the system clock time and used as nondeterministic metrics. lwperf uses the PAPI interface for collecting hardware performance counters, support for which is disabled by default. Configure with the –enable-papi flag to compile the library with PAPI support. The `lwperf_init_papi` function is used to turn on PAPI support during execution.

4 Running

The timing data obtainable should be reasonably accurate for measured non-overlapping compute sections, but the overall application performance may be strongly influenced by the extra I/O.

5 Analysis

Lwperf makes collecting large amounts of performance data easy. Analysis of the data is beyond the scope of this document. Studying the impact of application

parameters and compiler options on the runtime performance of specific code segments (loop nests) requires careful construction of a set of benchmark runs spanning some relevant parameter space. The generated CSV files can be easily aggregated and then imported or translated for use in any preferred tool (a spreadsheet, gnuplot, etc).

The default data collected includes the wall-clock time. This enables production of time-lines which capture what code section was active when, not just how much time was spent in each call.

6 Skeletonization

lwperf can be used in conjunction with the SST/macro simulator to facilitate the creation of application "skeletons", versions of the application fit for accelerated simulation by removing all time-intensive memory acquisition and computation. Each profiling site can be viewed as a way to replace the execution of that code with an equivalent Eiger model. Setting the compiler flag `LWPERF_USE_SSTMAC` will cause the `lwperf_stop` function make an appropriate call to the Eiger model that would be created by running this code. Currently only C++ applications are supported. While this may not handle the entirety of the skeletonization effort, it allows for seamless integration of Eiger models and skeleton construction. Removal of the actual computation performed (to speed up simulation execution time) is left to the application writer.

Skeletonization assumes a naming convention for models created with Eiger, <*site name*>.*model*, where <site name> is the second parameter to the `lwperf_log` function.