PerfExpert v4.2

An Easy-to-Use Automatic Performance Diagnosis and Optimization Tool for HPC Applications

User Manual

Antonio Gómez-Iglesias
agomez@tacc.utexas.edu

Leonardo Fialho
fialho@utexas.edu

Ashay Rane
ashay.rane@tacc.utexas.edu

James Browne
browne@cs.utexas.edu

Texas Advanced Computing Center
The University of Texas at Austin

October 2015
Revision 1

Contents

Chapter 1

Introduction

1.1 Purpose

HPC systems are notorious for operating at a small fraction of their peak performance. The ongoing migration to multi-core and multi-socket compute nodes further complicates performance optimization. The previously available performance optimization tools require considerable effort to learn and use. To enable wide access to performance optimization, TACC and its Technology Insertion partners have developed PerfExpert, a tool that combines a simple user interface with a sophisticated analysis engine to:

- Detect and diagnosis the causes for any core-, socket-, and node-level performance bottlenecks in each procedure and loop of an application.
- Apply pattern-based software transformations on the application source code to enhance performance on identified bottlenecks.
- Provide performance analysis report and suggestions for bottleneck remediation for application's performance bottlenecks which we are unable to optimize automatically.

Applying PerfExpert requires only a single command line added to the application's usual job script. PerfExpert automates performance optimization at the core, socket and node levels as far as is possible. PerfExpert works with, and has been tested on, all Intel and AMD processors from Intel Nehalem and AMD 10h microarchitectures. PerfExpert is currently available only for the CPU portion of Stampede compute nodes but will be extended to Intel Many Integrated Cores (MICs) in the near future.

1.2 People

James Browne
Professor Emeritus of Computer Science, UT Austin
browne@cs.utexas.edu

Antonio Gómez-Iglesias
Research Associate, Texas Advanced Computing Center
agomez@tacc.utexas.edu

Leonardo Fialho
Research Scientist, UT Austin
fialho@utexas.edu

Ashay Rane
PhD student, UT Austin
ashay.rane@tacc.utexas.edu

1.3 Publications

- Leonardo Fialho, James Browne, *"Framework and modular infrastructure for automation of architectural adaptation and performance optimization for HPC systems"*, Supercomputing 2014.
- Ashay Rane, James Browne, *"Enhancing performance optimization of multicore chips and multichip nodes with data structure metrics"*, Parallel Architectures and Compilation Techniques (PACT) 2012.
- Ashay Rane, James Browne, Lars Koesterke: *"A Systematic Process for Efficient Execution on Intel's Heterogeneous Computation Nodes"*, Extreme Science and Discovery Environment (XSEDE) 2012.
- Ashay Rane, James Browne, Lars Koesterke: *"PerfExpert and MACPO: Which code segments should (not) be ported to MIC?"*, TACC-Intel Highly Parallel Computing Symposium, April 2012.
- Ashay Rane, James Browne: *"Performance Optimization of Data Structures Using Memory Access Characterization"*. CLUSTER 2011: 570-574
- Ashay Rane, Saurabh Sardeshpande, James Browne: "Determining Code Segments that can Benefit from Execution on GPUs", poster presented at Supercomputing Conference (SC) 2011
- M. Burtscher, B.D. Kim, J. Diamond, J. McCalpin, L. Koesterke, and J. Browne. *"PerfExpert: An Easy-to-Use Performance Diagnosis Tool for HPC Applications"*, SC 2010 International Conference for High-Performance Computing, Networking, Storage and Analysis. November 2010
- O. A. Sopeju, M. Burtscher, A. Rane, and J. Browne. *"AutoSCOPE: Automatic Suggestions for Code Optimizations Using PerfExpert"*, 2011 International Conference on Parallel and Distributed Processing Techniques and Applications. July 2011

1.4 Feedback

If you have problems using PerfExpert on Stampede or Lonestar or suggestions for enhancing PerfExpert, contact us: `agomez@tacc.utexas.edu`. If you are reporting a problem, please try to include in your report a compressed file of the `.perfexpert-temp.XXXXXX` directory generated by the failed execution.

1.5 Ways to Contribute

Version 4 of PerfExpert has been designed to allow third-party contributions. There are several different ways to contribute with PerfExpert, such as:

- Providing new bottleneck alleviation solutions.
- Creating new strategies to select bottleneck alleviation solutions based on performance metrics.
- Adding new performance metrics to PerfExpert.
- Writing modules to modify the source code in order to alleviate the identified bottlenecks.

Check section 7 to find some documentation on extending PerfExpert. If you would like to contribute to PerfExpert or need help to do research using PerfExpert, please contact us at `agomez@tacc.utexas.edu`. Complete directions on how to add to or modify each phase of PerfExpert can be found on the PerfExpert web site `https://github.com/TACC/perfexpert`.

1.6 Mailing List

The PerfExpert mailing list is hosted at TACC. To subscribe send a message (no content or subject is required) to: `perfexpert-subscribe@lists.tacc.utexas.edu` or access the list's webpage at: `https://lists.tacc.utexas.edu/mailman/listinfo/perfexpert`.

1.7 Funding Sources

The NSF Track 2 Ranger grant and the current NSF Stampede grant.

1.8 Acknowledgments

Chapter 2

Installation Instructions

2.1 Prerequisites

PerfExpert is based on other tools so that installation of PerfExpert requires that they be installed. These tools are:

- **PAPI** (`http://icl.cs.utk.edu/papi/software/`): PAPI is required to measure hardware performance metrics like cache misses, branch instructions, etc. The PAPI installation is mostly straightforward: download, `./configure`, `make`, and `make install`. If your Linux kernel version is 2.6.32 or higher, then PAPI will mostly likely use `perf_events`. Recent versions of PAPI (v3.7.2 and beyond) support using `perf_events` in the Linux kernel. However, if your kernel version is lower than 2.6.32, then you would require patching the kernel with either `perfctr`[1] or `perfmon`[2].
- **HPCToolkit** (`http://hpctoolkit.org/software.html`): HPCToolkit is a tool that works on top of PAPI. HPCToolkit is used by PerfExpert to run the program multiple times with specific performance counters enabled. It is also useful for correlating addresses in the compiled binary back to the source code.
- **Java Virtual Machine** and the **Java Development Kit** (`http://www.oracle.com/technetwork/java/javase/downloads/`).
- **ROSE Compiler** (`http://rosecompiler.org/`): ROSE is the framework PerfExpert uses to manipulate the applications source code. It is required only if you wish to compile PerfExpert with performance optimization capability.
- **Apache Ant** (`http://ant.apache.org/bindownload.cgi`): it is required only to compile PerfExpert
- **SQLite** version 3 (`http://www.sqlite.org/`): PerfExpert uses a SQLite database to store suggestions for bottleneck remediation and other information for automatic optimization.
- **GNU Multiple Precision Arithmetic Library** (`http://gmplib.org/`): MACPO, one of the tools which compose PerfExpert requires this package.
- **Google SparseHash** (`https://code.google.com/p/sparsehash/`): MACPO, one of the tools which compose PerfExpert requires this package.

[1] `http://user.it.uu.se/~mikpe/linux/perfctr/2.6/`
[2] `http://perfmon2.sourceforge.net/`

2.1.1 Installing Prerequisites

We provide a `INSTALL` script in the `perfexpert-externals-X.X.tar.gz` package. This script is an example of how to install some of PerfExpert prerequisites. To install PAPI, Java Virtual Machine, Java Development Kit, and SQLite follow the instructions of your Linux distribution.

CAUTION:

The `INSTALL` script should be modified to fit for your environment. Do not run it out-of-the-box!

Apache Ant

The PerfExpert externals package contains another package named `apache-ant-1.9.1-bin.tar.gz`. This is the binary version of Apache Ant. As PerfExpert only needs Apache Ant during compilation there is no need to install this package in your system. Thus, you should only decompress it. To do that, run the following commands:

```
$ tar -xzvf ./apache-ant-1.9.1-bin.tar.gz
$ export ANT_HOME=`pwd`/apache-ant-1.9.1
```

these commands will decompress Apache Ant and set the `ANT_HOME` environment variable to reflect the `PATH` where such package has been decompressed.

HPCToolkit

All HPCToolkit prerequisites are provided in the package `hpctoolkit-externals-5.3.2-r4197.tar.gz`. The PerfExpert externals package provides both HPCToolkit and HPToolkit prerequisites packages. To decompress and install the HPCToolkit prerequisites package run the following commands:

```
$ tar -xzvf ./hpctoolkit-externals-5.3.2-r4197.tar.gz
$ cd hpctoolkit-externals-5.3.2-r4197/
$ ./configure
$ make
```

NOTICE:

The HPCToolkit externals package does not has to be installed in your system, HPCToolkit will take care of this. However, it should be compiled and available during HPCToolkit compilation and installation process.

After compiling the prerequisites for HPCToolkit, the HPCToolkit package can be installed. For that, run the following commands:

```
$ tar -xzvf ./hpctoolkit-5.3.2-r4197.tar.gz
$ cd hpctoolkit-5.3.2-r4197/
$ ./configure --prefix=WHERE_YOU_WANT_TO_INSTALL_IT \
    --with-externals=PATH_TO_HPCTOOLKIT_EXTERNALS_PACKAGE \
    --with-papi=PATH_TO_PAPI_INSTALLATION
$ make install
```

ROSE

ROSE has several prerequisites. One of the most important one is the compiler. You should have `GCC` version 4.4 to compile ROSE. It also requires a specific version of the Boost library (1.47). We do provide the Boost library package, but to install `GCC` version 4.4 you should follow the instruction of your Linux distribution. To compile and install the Boost library version 1.47 run the following commands:

NOTICE:

We had successfully compiled and installed ROSE using `GCC` 4.6, however according to the ROSE documentation this compiler is not supported. If you want to use this version of `GCC` do it carefully.

```
$ tar -xzvf ./boost_1_47_0.tar.gz
$ cd boost_1_47_0/
$ ./bootstrap.sh
$ ./b2 install --prefix=${INSTALL_DIR}
```

After installing successfully the Boost library run the following commands to compile and install ROSE:

```
$ tar -xzvf ./rose-0.9.5a-without-EDG-20584.tar.gz
$ cd rose-0.9.5a-20584/
$ mkdir _build
$ cd _build
$ ../configure --prefix=WHERE_YOU_WANT_TO_INSTALL_IT --with-boost=
    WHERE_BOOST_IS \
     --disable-tutorial-directory --disable-projects-directory --disable-
        opencl \
     --disable-tests-directory --disable-php --disable-cuda --disable-
        binary-analysis \
     --disable-java --with-gomp_omp_runtime_library=WHERE_LIBGOMP_IS \
     --libexecdir=/tmp --bindir=/tmp --sbindir=/tmp
$ make install
```

NOTICE:

The `configure` command we shown above avoids the compilation and installation of several features of ROSE. We do that to minimize the time required to compile ROSE. Moreover, some binaries that ROSE installs and we do not need them have been set to be installed into the `/tmp` directory. It is safe to clean the temporary directory after installing ROSE.

ATTENTION:

If you want to PerfExpert be able to optimize OpenMP applications you should set the `--with-gomp_omp_runtime_library` argument to where `GCC` has installed `libgomp.so`.

GNU Multiple Precision Arithmetic Library

Many Linux distributions already have the GNU Multiple Precision Arithmetic Library installed. To check if your system has it try to locate the `libgmp.so` file. If your system does not has the GNU Multiple Precision Arithmetic Library, run the following commands to install it:

```
$ tar -xzvf ./gmp-5.1.2.tar.gz
$ cd gmp-5.1.2/
```

```
$ ./configure --prefix=WHERE_YOU_WANT_TO_INSTALL_IT
$ make
$ make check
$ make install
```

NOTICE:

The GNU Multiple Precision Arithmetic Library is required to compile and execute MACPO, one of the tools which is part of PerfExpert. If you do not want to install MACPO you do not need to install the GNU Multiple Precision Arithmetic Library.

Google SparseHash

To install Google SparseHash you should run the following commands:

```
$ tar -xzvf ./sparsehash-2.0.2.tar.gz
$ cd sparsehash-2.0.2/
$ ./configure --prefix=WHERE_YOU_WANT_TO_INSTALL_IT
$ make install
```

NOTICE:

Google SparseHash is required to compile and execute MACPO, one of the tools which is part of PerfExpert. If you do not want to install MACPO you do not need to install the Google SparseHash.

2.2 Downloading PerfExpert

PerfExpert is an open-source project. Funding to keep researchers working on PerfExpert depends on the value of this tool to the scientific community. For that reason, it is really important to know where and who are using our tool. We would really appreciate it if you could send us a message (`agomez@tacc.utexas.edu`) telling us the institution (name and country) you are planning to install and test PerfExpert at.

The PerfExpert source code can be downloaded from the registration page: `http://www.tacc.utexas.edu/perfexpert/registration`.

We encourage people to have a look on the wiki page, send patches, and raise issues on PerfExpert's page on GitHub: `https://github.com/TACC/perfexpert`.

2.3 Setting Up PerfExpert

If you have downloaded PerfExpert from our version control system, you may have already noticed that there is no configure script available on the source code tree. To generate it run the following command:

```
$ ./autogen.sh
```

`autogen.sh` requires the following packages available:

- M4 version 1.4.13 or newer (`ftp://ftp.gnu.org/gnu/m4/`)
- Autoconf version 2.63 or newer (`ftp://ftp.gnu.org/gnu/autoconf/`)
- Automake version 1.11.1 or newer (`ftp://ftp.gnu.org/gnu/automake/`)
- Libtool version 2.2.6b or newer (`ftp://ftp.gnu.org/gnu/libtool/`)

PerfExpert comes with a `Makefile`-base source code tree to automate the entire installation process. Thus the compilation and installation of PerfExpert is similar to any other GNU package:

```
$ ./configure
$ make
$ make install
```

Optionally, you may want to run the set of test we have included into PerfExpert. To do so, just run "`make check`" after compiling your code.

If any of the prerequisites of PerfExpert are not on the `PATH` or `LD_LIBRARY_PATH`, you can specify the right locations of such files. For that, have a look on the configure script help using the following command:

```
$ ./configure --help
```

A typical command line to run `configure` may looks like this:

```
$ ./configure --prefix=WHERE_YOU_WANT_TO_INSTALL_IT --with-rose=
    WHERE_ROSE_IS \
     --with-jvm=WHERE_JVM_IS --with-papi=WHERE_PAPI_IS --with-apache-ant=
         WHERE_ANT_IS
```

Optionally, you may want to run the set of test we have included into PerfExpert. To do so, just run "`make check`" after compiling your code.

In case you have any problem installing PerfExpert, send us an email (`agomez@tacc.utexas.edu`) or use our mailing list: `perfexpert-subscribe@lists.tacc.utexas.edu`.

2.4 Characterizing your Machine

During the installation process, PerfExpert will run a set of benchmark applications to characterize your machine. This characterization is used to analyze the performance of the applications you want to optimize. For that reason, you should be sure the PerfExpert installation is run on a machine of the same kind you are planning to run the production code on. If it is not possible, you should run the benchmark application and generate the characterization file manually and move it to the directory where PerfExpert has been installed. To do that you should execute the `hound` command (which is available inside the `bin` directory of PerfExpert installation) and save the output of this command to a file named `machine.properties` inside the `etc` directory where PerfExpert is installed.

ATTENTION:

This is one of the most important and sensible steps of PerfExpert installation process. Be sure the content of this file reflects the characteristics of the machine where you want analyze the performance of applications. If the content of the `machine.properties` file is not accurate all the analysis, recommendations for optimization and also automatic optimization PerfExpert will do on your application could not improve it's performance.

2.5 Testing PerfExpert Installation

We provide a set of tests you may want to run before using PerfExpert to optimize your applications. To run these tests you should run the following command inside the PerfExpert build tree:

```
$ make check
```

It is normal that one of the tests (`mpi_stampede`) fails since it has been designed to be executed only on Stampede.

Chapter 3

Using PerfExpert

The objective of this chapter is to explain how to run programs using PerfExpert and how to interpret its output using a simple matrix multiplication program. In this chapter, we will use the OpenMP simple matrix multiplication program[1]. This program multiplies two matrices and prints one value from the resulting matrix.

CAUTION:

PerfExpert may, if you choose to use the full capabilities for automated optimization, change your source code during the process of optimization. PerfExpert always saves the original file with a different name (*e.g.*, `omp_mm.c.old_27301`) as well as adding annotations to your source code for each optimization it makes. We cannot, however, fully guarantee that code modifications for optimizations will not break your code. We recommend having a full backup of your original source code before using PerfExpert.

3.1 Environment Configuration

If you are using any of the TACC[2] machines, load the appropriate modules:

```
$ module load papi hpctoolkit perfexpert
```

The runs with PerfExpert should be made using a data set size for each compute node which is equivalent to full production runs but for which execution time is not more than about ten or fifteen minutes since PerfExpert will run your application multiple times (actually, three times on Stampede) with different performance counters enabled. For that reason, before you run PerfExpert you should either request iterative access to computational resources (compute node), or modify the job script that you use to run your application and specify a running time that is about 3 (for Stampede) or 6 (for Lonestar) times the normal running time of the program.

To request iterative access to a compute node on Stampede, please, have a look on the User Guide[3].

Below is an example of a job script file modified to use PerfExpert which runs PerfExpert on the application named my_program and generate the performance analysis report. Adding command line options will cause suggestions for bottleneck remediation to be generated and output and/or automatic performance optimization to be attempted.

[1] `https://computing.llnl.gov/tutorials/openMP/samples/C/omp_mm.c`
[2] `https://www.tacc.utexas.edu/`
[3] `https://portal.tacc.utexas.edu/group/tup/user-guides/stampede#running`

```
#!/bin/bash
#SBATCH -J myMPI              # job name
#SBATCH -o myMPI.o%j          # output and error filename (%j stands to
   jobID)
#SBATCH -n 16                 # total number of mpi tasks requested
#SBATCH -p development        # queue (partition) -- normal, development,
   etc.
#SBATCH -t 01:30:00           # run time (hh:mm:ss) - 1.5 hours
perfexpert 0.1 ./my_program   # run the executable named my_program
```

3.2 PerfExpert Options

There are several different options for applying PerfExpert. The following summary shows you how to choose the options to run PerfExpert to match your needs.

```
$ perfexpert -h
Usage: perfexpert <threshold> [-m target|-s sourcefile] [-r count] [-d
   database]
                  [-p prefix] [-b filename] [-a filename] [-l level] [-
                     gvch]
                  [-k card [-P prefix] [-B filename] [-A filename] ]
                  <program_executable> [program_arguments]

  <threshold>         Define the relevance (in % of runtime) of code
     fragments PerfExpert
                      should take into consideration (> 0 and <= 1)
  -m --makefile       Use GNU standard 'make' command to compile the code (
     it requires
                      the source code available in current directory)
  -s --source         Specify the source code file (if your source code has
       more than one
                      file please use a Makefile and choose '-m' option it
                         also enables
                      the automatic optimization option '-a')
  -r --recommend      Number of recommendations ('count') PerfExpert should
       show
  -d --database       Select the recommendation database file
                      (default: PERFEXPERT_VARDIR/RECOMMENDATION_DB)
  -p --prefix         Add a prefix to the command line (e.g. mpirun). Use
     double quotes to
                      specify arguments with spaces within (e.g. -p "mpirun
                          -n 2"). Use a
                      semicolon (';') to run multiple commands in the same
                         command line
  -b --before         Execute 'filename' before each run of the application
  -a --after          Execute 'filename' after each run of the application
  -k --knc            Tell PerfExpert to run the experiments on the KNC '
     card'
```

```
  -p --prefix-knc     Add a prefix to the command line (e.g. mpirun). Use
     double quotes to
                      specify arguments with spaces within (e.g. -p "mpirun
                          -n 2"). Use a
                      semicolon (';') to run multiple commands in the same
                         command line
  -B --knc-before     Execute 'filename' before each run of the application
      on the KNC
  -A --knc-after      Execute 'filename' after each run of the application
     on the KNC
  -g --clean-garbage Remove temporary files after run
  -v --verbose        Enable verbose mode using default verbose level (1)
  -l --verbose-level Enable verbose mode using a specific verbose level
     (1-10)
  -c --colorful       Enable colors on verbose mode, no weird characters
     will appear on
                      output files
  -h --help           Show this message

Use CC, CFLAGS and LDFLAGS to select compiler and compilation/linkage
    flags
```

If you select the `-m` or `-s` options, PerfExpert will try to automatically optimize your code and show you the performance analysis report & the list of suggestion for bottleneck remediation when no automatic optimization is possible.

For the `-m` or `-s` options, PerfExpert requires access to the application source code. If you select the `-m` option and the application is composed of multiple files, your source code tree should have a `Makefile` file to enable PerfExpert compile your code. If your application is composed of a single source code file, the option `-s` is sufficient for you. If you do not select `-m` or `-s` options, PerfExpert requires only the binary code and will show you only the performance analysis report and the list of suggestion for bottleneck remediation.

PerfExpert will run your application multiple times to collect different performance metrics. You may use the `-b` (or `-a`) options if you want to execute a program or script before (or after) each run. The argument `program_executable` should be the filename of the application you want to analyze, not a shell script, otherwise, PerfExpert will analyze the performance of the shell script instead of the performance of you application.

Use the `-r` option to select the number of recommendations for optimization you want for each code section which is a performance bottleneck.

CAUTION:

If your program takes any argument that starts with a "`-`" signal PerfExpert will interpret this as a command line option. To help PerfExpert handle `program_arguments` correctly, use quotes and add a space before the program's arguments (e.g., " `-s 50`").

CAUTION:

In case you are trying to optimize a MPI application, you should use the `-p` option to specify the MPI launcher and also it's arguments.

For this guide, using the OpenMP simple matrix multiply code, we will use the following command line options:

```
$ OMP_NUM_THREADS=16 CFLAGS="-fopenmp" perfexpert -s mm_omp.c 0.05 mm_omp
```

which executes PerfExpert's automatic optimizations and will also generate an OpenMP-enabled binary which will run with 16 threads. In this case, PerfExpert will compile the `mm_omp.c` code using the system's default compiler, which is GCC in the case of Stampede. PerfExpert will take into consideration only code fragments (loops and functions) that take more 5% of the runtime.

To select a different compiler, you should specify the `CC` environment variable as below:

```
$ CC="icc" OMP_NUM_THREADS=16 CFLAGS="-fopenmp" perfexpert -s mm_omp.c
   0.05 mm_omp
```

If you do not want to have PerfExpert trying to optimize the application automatically, just compile your program and run the following command:

```
$ OMP_NUM_THREADS=16 perfexpert 0.05 mm_omp
```

WARNING:

If the command line you use to run PerfExpert includes the MPI launcher (*i.e.*, `mpirun -n 2 my_mpi_app my_mpi_app_arguments ...`), PerfExpert will analyze the performance of the MPI launcher instead of the performance of your application. Use the `-p` command line argument of PerfExpert to set the MPI launcher and all its arguments (*e.g.*, `-p "mpirun -n 16"`).

3.3 Passing Options to the Modules

3.3.1 make

The `make` module accept parameters that will be directly passed to the `make` command. So, if for example your code needs to be compiled with the command `make intel optim`, you will have to pass the `intel` and `optim` as arguments to the module in PerfExpert using the `args` option and the arguments between double quotes. The command will look like:

```
perfexpert --modules=make --module-option=make,args="intel optim" ./program
```

3.3.2 HPCToolkit

3.3.3 VTune

3.4 The Performance Analysis Report

This section explains the performance analysis report and the metrics shown by PerfExpert. We discuss the following sample output:

```
Loop in function compute() (99.9% of the total runtime)
================================================================================
ratio to total instrns       %  0.......25.........50.......75......100
   - floating point      :    6 ***
   - data accesses       :   33 ****************

* GFLOPS (% max)         :    7 ***
   - packed              :    0 *
   - scalar              :    7 ***
--------------------------------------------------------------------------------
performance assessment     LCPI good....okay....fair....poor....bad...
* overall               :  0.8 >>>>>>>>>>>>>>>
upper bound estimates
```

```
* data accesses          :  2.5 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - L1d hits            :  1.3 >>>>>>>>>>>>>>>>>>>>>>>>>
   - L2d hits            :  0.3 >>>>>
   - L3d hits            :  0.0 >>>>>>>>>>>>>>>>>
   - LLC misses          :  0.1 >>
* instruction accesses   :  0.3 >>>>>>
   - L1i hits            :  0.3 >>>>>>
   - L2i hits            :  0.0 >
   - L2i misses          :  0.0 >
* data TLB               :  1.5 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
* instruction TLB        :  0.0 >
* branch instructions    :  0.0 >
   - correctly predicted :  0.0 >
   - mispredicted        :  0.0 >
* floating-point instr   :  0.2 >>>>
   - fast FP instr       :  0.2 >>>>
   - slow FP instr       :  0.0 >
```

Apart from the total running time, PerfExpert performance analysis report includes, for each code segment:

- Instruction execution ratios (with respect to total instructions);
- Approximate information about the computational efficiency (GFLOPs measurements);
- Overall performance;
- Local Cycles Per Instruction (LCPI) values for the cost of memory accesses.

The program composition part shows what percentage of the total instructions were computational (floating-point instructions) and what percentage were instructions that accessed data. This gives a rough estimate in trying to understand whether optimizing the program for either data accesses or floating-point instructions would have a significant impact on the total running time of the program.

The PerfExpert performance analysis report also shows the GFLOPs rating, which is the number of floating-point operations executed per second in multiples of 109. The value for this metric is displayed as a percentage of the maximum possible GFLOP value for that particular machine. Although it is rare for real-world programs to match even 50% of the maximum value, this metric can serve as an estimate of how efficiently the code performs computations.

The next, and major, section of the PerfExpert performance analysis report shows the LCPI values, which is the ratio of cycles spent in the code segment for a specific category, divided by the total number of instructions in the code segment. The overall value is the ratio of the total cycles taken by the code segment to the total instructions executed in the code segment.

Generally, a value of 0.5 or lower for an LCPI is considered to be good. However, it is only necessary to look at the ratings (`good`, `okay`, ..., `bad`) The rest of the report maps this overall LCPI, into the six constituent categories: data accesses, instruction accesses, data TLB accesses, instruction TLB accesses, branches and floating point computations. Without getting into the details of instruction operation on Intel and AMD chips, one can say that these six categories record performance in non-overlapping ways. That is, they roughly represent six separate categories of performance for any application.

The LCPI value is a good indicator of the cost arising from instructions of the specific category. Hence, the higher the LCPI, the slower the program. The following is a brief description of each of these categories:

Data accesses:
counts the LCPI arising from accesses to memory for program variables.

Instruction accesses:
counts the LCPI arising from memory accesses for code (functions and loops).

Data TLB:
provides an approximate measure of penalty arising from strides in accesses or regularity of accesses.

Instruction TLB:
reflects cost of fetching instructions due to irregular accesses.

Branch instructions:
counts cost of jumps (i.e. if statements, loop conditions, etc.).

Floating-point instructions:
counts LCPI from executing computational (floating-point) instructions.

Some of these LCPI categories have subcategories. For instance, the LCPI from data and instruction accesses can be divided into LCPI arising from the individual levels of the data and instruction caches and branch LCPIs can be divided into LCPIs from correctly predicted and from mispredicted branch instructions. For floating-point instructions, the division is based on floating-point instructions that take few cycles to execute (*e.g.*, add, subtract and multiply instructions) and on floating-point instructions that take longer to execute (*e.g.*, divide and square-root instructions).

In each case, the classification (data access, instruction access, data TLB, etc.) is shown so that it is easy to understand which category is responsible for the performance slowdown. For instance if the overall CPI is "`poor`" and the data access LCPI is high, then you should concentrate on access to program variables and memory. Additional LCPI details help in relating performance numbers to the process architecture.

IMPORTANT:

When PerfExpert runs with automatic performance optimization enabled the performance analysis report shown reflects the performance of the code after all possible automatic optimizations have been applied.

NOTICE:

PerfExpert creates a `.perfexpert-temp.XXXXXX` directory for each time it is executed. This directory has one subdirectory for each optimization cycle PerfExpert completed or attempted. Each subdirectory includes the intermediate files PerfExpert generated during each cycle, including the performance analysis reports.

3.5 Running PerfExpert with MPI codes

TACC systems use `ibrun` to launch MPI codes, so all these examples use `ibrun`. When running on other sites, remember to change `ibrun` by `mpirun` or the launcher used at that site.

To run PerfExpert with an MPI code, use `ibrun` as a prefix with the `-p` option as follows:

```
perfexpert 0.1 --verbose=10 --modules=lcpi,hpctoolkit -p "ibrun" -- ./
    program
```

Parameters can be passed to `ibrun` as follows:

```
perfexpert 0.1 --verbose=10 --modules=lcpi,hpctoolkit -p "ibrun -np 4" --
    ./program
```

3.5.1 Using Intel Xeon Phi

When using native MPI applications on the Intel Xeon Phi, there are two main scenarios:

HPCToolkit

This is the simplest case. All you need to do is to specify which MPI launcher you want to use as a prefix in your PerfExpert command. For example:

```
perfexpert 0.1 --modules=lcpi,hpctoolkit --module-option=lcpi,
   architecture=MIC  --module-option=hpctoolkit,mic-card=mic0 --
   module-option=hpctoolkit,prefix="ibrun.symm -m" -- ./lulesh2.0
   -i 100 -s 8
```

Intel VTune

In this case, the application that you want to analyze needs to be specified inside of a script. VTune will run then this script on the Intel Xeon Phi. For example, in the case of Lulesh, the script will look like this:

```
#!/bin/bash

export OMP_NUM_THREADS=1;  \
export LD_LIBRARY_PATH=/opt/apps/intel/15/compiler/lib/mic/:
   $LD_LIBRARY_PATH; \
/opt/apps/intel15/impi/5.0.2.044/mic/bin/mpiexec -np 8 \
/work/02658/agomez/tools/toys/lulesh2.0.2/lulesh2.0 -i 10000 -s 8
```

We first export a set of environmental variables that are required by our application to run on the Xeon Phi. Then, we need to specify an MPI launcher. In this example, we are choosing the mpiexec provided by Intel MPI 5.0 for the Intel Xeon Phi. This launcher needs the number of MPI tasks that will run on the coprocessor (8 in this example). Finally, we simply need to pass the application that we are studying with its arguments.

When calling Intel VTune with this script, the command will be like:

```
perfexpert 0.1 --verbose=10 --modules=lcpi,vtune --module-option=lcpi,
   architecture=MIC  \
--module-option=vtune,mic-card=mic0  -- ssh mic0 PATH_TO_SCRIPT/test.sh
```

3.6 List of Recommendations for Optimization

If PerfExpert runs with `-r` option enabled, it will also generate a list of suggestions for performance improvement for each bottleneck. This option is always available, it does not depend on which of the other command line option are. A list of suggestions for this example is shown below:

```
#-----------------------------------------------
# Recommendations for /work/02204/fialho/tutorial/3/mm_omp.c:14
#-----------------------------------------------
#
# Here is a possible recommendation for this code segment
#
Description: move loop invariant computations out of loop
Reason: this optimization reduces the number of executed floating-point
   operations
Code example:
```

```
loop i
  x = x + a * b * c[i];
 =====>
temp = a * b;
loop i
  x = x + temp * c[i];

#
# Here is a possible recommendation for this code segment
#
Description: change the order of loops
Reason: this optimization may improve the memory access pattern and make
   it more cache
and TLB friendly
Code example:
loop i
  loop j {...}
 =====>
loop j
  loop i {...}

#
# Here is a possible recommendation for this code segment
#
Description: componentize important loops by factoring them into their own
    subroutines
Reason: this optimization may allow the compiler to optimize the loop
   independently and
thus tune it better
Code example:
loop i {...}
...
loop j {...}
 =====>
void li() { loop i {...} }
void lj() { loop j {...} }
...
li();
...
lj();
```

Chapter 4

Using MACPO

MACPO is an acronym for Memory Access Characterization for Performance Optimization. It is a tool that has been built to assist performance tuning of single- and multi-threaded C, C++ or Fortran applications. More specifically, MACPO is designed to provide insight into an application's memory usage patterns.

4.1 A Quick Demonstration

To demonstrate the functioning of MACPO, let's consider an example program. This program uses Pthreads to calculate the value of Pi using the Monte-Carlo method.

The following code shows the function that is executed by each thread:

```
void* thread_func (void* arg) {
  int idx, i, repeat;
  float x, y, z;
  thread_info_t* thread_info = (thread_info_t*) arg;

  for (repeat = 0; repeat < REPEAT_COUNT; repeat++) {
    for (i = 0; i < ITERATIONS; i++) {
      idx = i + thread_info->tid;
      x = random_numbers[idx % RANDOM_BUFFER_SIZE];
      y = random_numbers[(1 + idx) % RANDOM_BUFFER_SIZE];
      z = x * x + y * y;
      if (z <= 1) counts[thread_info->tid]++;
    }
  }
  pthread_exit(0);
}
```

To compile the application using MACPO, indicating that we are interested in understanding the performance metrics associated with the `thread_func` function, run the following commands:

```
$ macpo.sh --macpo:instrument=thread_func -c monte-carlo.cc
$ macpo.sh --macpo:instrument=thread_func monte-carlo.o -o monte-carlo -
   lpthread -lrt
```

Runtime logs (macpo.out) are produced when the application is run:

```
$ ./monte-carlo
```

To print performance metrics, use the `macpo-analyze` program.

```
$ macpo-analyze macpo.out
```

This will produce an output that is similar to the one shown below:

```
Var "counts", seen 1668 times, estimated to cost 147.12 cycles on every
   access.
Stride of 0 cache lines was observed 1585 times (100.00%).

Level 1 data cache conflicts = 78.22% [################################
         ]
Level 2 data cache conflicts = 63.37% [##########################
                ]
NUMA data conflicts = 43.56%          [#################
                         ]

Level 1 data cache reuse factor = 97.0%
   [######################################## ]
Level 2 data cache reuse factor = 3.0%  [##
                                        ]
Level 3 data cache reuse factor = 0.0%  [
                                          ]
```

The output shows the estimated cost of memory accesses (147.12 cycles) to the variable `counts` in terms of cycles per access[1]. The output also shows stride values (0 cache lines) observed in the accesses to the variable[2]. As per the cache conflicts shown in the output, accesses to the variable `counts` suffer from both L1 and L2 cache conflicts (also known as cache thrashing). Using this knowledge, we can pad the `counts` array (*i.e.*, add dummy bytes to the array) so that each thread running on a core and, thus, sharing an L1 cache as well as the shared L2 cache, which may be shared by other cores, accesses a different cache line. This optimization reduces the running time of the `thread_func` routine from 9.14s to 3.17s.

The other information provided by MACPO is discussed below in Section 4.3.

4.2 When to Use MACPO

MACPO may be useful to you in any of the following situations:

- Perfexpert reports that L2 or L3 data access LCPI is high.
- Your program uses a lot of memory or it is memory-bound.
- CPU profiling does not show any interesting results.

If all (or most) of your application's memory accesses are irregular, you may be able to infer the optimizations applicable to your program. However, for such programs, MACPO metrics may not be able to assist you directly.

4.3 MACPO Metrics

Performance information shown by MACPO can be grouped into two parts. The first part shows information that is applicable to the entire function being profiled. The second part shows information that is specific to the important variables in the function.

[1] A large number of cycles indicates poor memory performance. Causes of large values may be explained by the following metrics.

[2] A stride of 0 cache lines indicates that subsequent references are in the same cache line; this indicates good memory performance and is good for vectorization. A stride of n indicates that subsequent references are n cache lines apart; the larger n, the worse is the memory performance.

4.3.1 Function-wide Performance Metrics

Currently, function-wide information only shows the number of streams that were seen while compiling the function. A stream is a variable or data structure that may be accessed repeatedly in a uniform manner.

4.3.2 Variable-specific Performance Metrics

For each variable that is encountered a significant number of times, MACPO shows the following performance metrics:

Estimated average cycles per access
MACPO collects metrics that allow it to calculate the approximate reuse distance of variables. The reuse distance, in turn, helps to estimate the specific level of the cache that the request would have originated from. This makes it possible to estimate the number of cycles spent in each memory access. This cost in terms of memory access is grouped by the variable name and averaged to show in this metric.

Dominant stride values and their percentages
A stride is the constant difference in bytes between the last memory access and the most recent memory access to a variable. MACPO computes the access strides in units of cache line size. This provides an indication of how well a code can be vectorized (stride of 0, *i.e.*, sequential access, is best) and how one might optimize the code for better performance.

Cache conflicts (thrashing) for L1 and L2 data caches
Cache conflicts arise when multiple processors, each with at least one private cache, repeatedly claim exclusive access to a portion of memory. This metric shows the percentage of requests to each level of the cache that conflicted with another access to the same variable.

NUMA conflicts
Most modern processors exhibit non-uniform memory access costs. The cost of memory access depends on whether the processor that hosts the memory controller managing the memory address is the same as the processor accessing the memory address. This metric displays the percentage of observed memory accesses that conflicted with another memory access to the same variable at the NUMA level.

Reuse factors for L1, L2 and L3 data caches
From the observed reuse distance and a probabilistic model of set-associative caches, MACPO estimates whether a given memory access would be served out of L1 or would overflow the size of the cache, resulting in the memory access being served out of a higher (L2 or possibly L3) level of cache. This analysis permits MACPO to calculate the multicore reuse factor, which is a count of the number of times a given cache line is reused in a specific level of the cache.

4.4 Improving Application Performance by Interpreting MACPO Metrics

This section explains how to translate the MACPO metrics into source code changes to improve the performance of your application.

Estimated average cycles per access
The cycles per access metric provides an overview of the performance of memory accesses to a specific variable. It makes it possible to identify whether a particular variable is suffering from memory performance bottleneck problems.

Dominant stride values and their percentages

Programs that have unit strides or small regular stride values generally execute faster than programs that have long or irregular access strides. There are several factors giving better memory access performance. Since data is typically fetched from memory as cache lines, unit strides increase reuse. Hardware prefetchers can recognize small regular patterns in data accesses and bring data into caches before it is referenced, thus reducing data access penalty. Virtual address to physical address translation can also be serviced more efficiently (using TLBs) when the code exhibits unit strides.

Cache conflicts (thrashing) for L1 and L2 data caches

Cache conflicts indicate thrashing between cache lines. By padding the data structures with additional (dummy) bytes, each thread can be made to access a different cache line, effectively removing cache conflicts.

NUMA conflicts

Most operating systems implement a first-touch policy by which the memory controller associated with the processor that first accesses the memory address "owns" the memory address. Memory accesses for the same address by a different processor result in NUMA conflicts. As a result, NUMA conflicts typically arise when one thread initializes a portion of memory that is then accessed by different threads. To avoid NUMA conflicts, have each processor initialize its own memory.

Reuse factors for L1, L2 and L3 data caches

A low reuse factor indicates that a line is frequently evicted from the cache. The reuse factor can be improved by reducing the reuse distance of memory accesses.

4.5 Using MACPO from PerfExpert

MACPO has been integrated into PerfExpert. It can be passed as an option to the list of active modules. For example:

```
perfexpert 0.1 --verbose=10 --modules=make,lcpi,hpctoolkit,macpo --module-
    option=macpo,main=Main.c -p mpirun -- ./XSBench  -t 4 -s small -l
    100000
```

The previous command will execute the make, LCPI, HPCToolkit and MACPO modules. MACPO will use the information produced by HPCToolkit to instrument the most relevant functions and loops.

The MACPO module accepts a set of optional arguments, the most relevant being the `main` argument. This argument is used to pass to MACPO the fullpath and the filename of where the entry point to the code of interest is located (i.e. the `main` function). This will ensure a correct and full instrumentation of the code.

4.6 Running MACPO as a standalone program

The various analyses in MACPO and the corresponding options are shown below:

Analysis	MACPO option
Memory analysis	`--macpo:instrument`
Alignment checking	`--macpo:check-alignment`
Measure loop trip counts	`--macpo:record-tripcount`
Count branch outcomes	`--macpo:record-branchpath`
Measure gather/scatter accesses from vectorization	`--macpo:vector-strides`

Each of these options accepts a program location (i.e. either a function or a loop specification). Function names can be used as identifiers for instrumentation (e.g. `--macpo:check-alignment=foo`). To tell MACPO to instrument a loop, pass either the name of the function that contains the loop of interest

or the name of the file that contains the loop and the line number at which the loop starts separated by the `#` character. For instance, to instrument the loop in `bar.c` that starts at line `273` for measuring trip counts, use the option: `--macpo:record-tripcount=bar.c#273`. You may also use the option `--macpo:record-tripcount=foo#273` to achieve the same effect.

MACPO also accepts other flags that influence it's operation. The following table shows the valid options.

MACPO option	**Purpose**
`--macpo:backup-filename`	File to backup original source code to, before transforming it.
`--macpo:no-compile`	Generate the transformed source code in `rose_*` file(s) but don't compile it.
`--macpo:enable-sampling`	Enable sampling mode when collecting measurements from instrumentation.
`--macpo:disable-sampling`	Disable sampling mode when collecting measurements from instrumentation.
`--macpo:profile-analysis`	Collect and display basic profiling information for MACPO's static analyses.

Chapter 5

List of Automatic Optimizations PerfExpert Supports

Loop Interchange

Loop Tiling (aka Loop Blocking)

Loop Fission

Chapter 6

PerfExpert for Advanced Users

This chapter covers details of PerfExpert internal operation. While its reading is not required for common users, advanced user may appreciate its content.

6.1 The Big Picture

6.2 Tools Which Are Part of PerfExpert

This section describes the usage details of each of the tools which compose PerfExpert.

6.2.1 `perfexpert`

Synopsis

perfexpert <threshold> **[-m** target **| -s** sourcefile**] [-r** count **] [-d** database**] [-p** prefix**]**
[-b before**] [-a** after**] [-l** level**] [-k** card **[-P** prefix**] [-B** before**] [-A** after**]]**
[-gchv]

Description

Command Line Arguments

`threshold`

`-m, --makefile`

`target`

`-s, --source`

`sourcefile`

`-r, --recommendations`
Set the number of optimizations perfexpert should provide.

`count`
Number of optimizations perfexpert should provide.

`-d, --database`
Full path and name of PerfExpert database file (default: `PERFEXPERT_VARDIR/RECOMMENDATION_DB`).

`database`
Database file (default: `PERFEXPERT_VARDIR/RECOMMENDATION_DB`).

`-p, --prefix`

`prefix`

`-b, --before`

`-a, --after`

`-P, --knc-prefix`

`-B, --knc-before`

`-A, --knc-after`

`-A, --knc-after`

`-g, --clean-garbage`

`-v, --verbose`
Enable verbose mode (default: level 5).

`-l, --verbose_level`
Enable verbose mode using a specific verbose level.

`level`
Verbose level (range: 1-10).

`-c, --colorful`
Enable colors on verbose mode, no weird characters will appear on output files.

`-h, --help`
Show the help message.

Environment Variables

Here is a complete list of the environment variables PerfExpert uses. The command line arguments overwrite the value set by any of the environment variables.

`PERFEXPERT_MAKE_TARGET`
: This variable has the same functionality than the `-m` command line option. It's value is passed to `make` to compile the source code while using a `Makefile` (*e.g.*, if it's blue is set to `all`, PerfExpert will compile the user application using the `make all` command). If this variable is set while the user selects the `-s` command line option an error will be generated.

`PERFEXPERT_SOURCE_FILE`
: This variable has the same functionality than the `-s` command line option. It's value should be a valid source code file which will be compiled to obtain the binary executable. If this variable is set while the user selects the `-m` command line option an error will be generated.

`PERFEXPERT_DATABASE_FILE`
: This variable has the same functionality than the `-d` command line option. It's value should be a valid PerfExpert database file. By default, PerfExpert uses the `.perfexpert.db` database file located in the user `$HOME` directory. If this file does not exist, PerfExpert will copy it from the installation directory.

`PERFEXPERT_REC_COUNT`
: This variable has the same functionality than the `-r` command line option. It's value should be a valid integer number higher than `0`.

`PERFEXPERT_VERBOSE_LEVEL`
: This variable has the same functionality than the `-l` command line option. It's value should be a valid integer number within `1` and `10`.

`PERFEXPERT_COLORFUL`
: This variable has the same functionality than the `-c` command line option. It's value should be `0` or `1`.

`CC`
: This variable sets the compiler PerfExpert should use to obtain the binary executable. It should be a valid executable file. It is possible to set it's value to include the compiler's full path. The value of this variable has no effect when PerfExpert runs without the `-s` command line option and the `PERFEXPERT_SOURCE_FILE` environment variable is not set. Also, note that the value of this variable has no effect when PerfExpert uses `Makefile` to obtain the binary executable.

`CFLAGS`
: This variable sets the compiler flags PerfExpert should use while compiling the source code. The value of this variable has no effect when PerfExpert runs without the `-s` command line option and the `PERFEXPERT_SOURCE_FILE` environment variable is not set. Also, note that the value of this variable has no effect when PerfExpert uses `Makefile` to obtain the binary executable.

`PERFEXPERT_CFLAGS`
: This variable is used by PerfExpert to implement optimization which depends upon compiler flags. This variable should not be used to pass any user defined compiler flags because it's content will be freely modified by PerfExpert.

`PERFEXPERT_PREFIX`
: This variable has the same functionality than the `-p` command line option. It's value should be a command that will prefix the experiment command line.

`PERFEXPERT_BEFORE`
: This variable has the same functionality than the `-b` command line option. It's value should be a command that will be executed before each experiment.

`PERFEXPERT_AFTER`
: This variable has the same functionality than the `-a` command line option. It's value should be a command that will be executed after each experiment.

`PERFEXPERT_KNC`
: This variable has the same functionality than the `-k` command line option. It's value should be the name of the KNC card (*e.g.*, "`mic0`").

`PERFEXPERT_KNC_PREFIX`
: This variable has the same functionality than the `-P` command line option. It's value should be a command that will prefix the command line that will be executed on the KNC.

`PERFEXPERT_KNC_BEFORE`
: This variable has the same functionality than the `-B` command line option. It's value should be a command that will be executed on the KNC before each experiment.

`PERFEXPERT_KNC_AFTER`
: This variable has the same functionality than the `-A` command line option. It's value should be a command that will be executed on the KNC after each experiment.

Exit Status

6.2.2 `analyzer`

Synopsis

Description

Command Line Arguments

Exit Status

6.2.3 `macpo`

6.2.4 `MACVEC`

Synopsis

MACVEC is a vectorization report analyzer for the Intel compiler. It automatically creates vectorization reports for the applications that can be directly compiled with the Intel compiler commands or a `makefile` that uses the Intel compilers.

Description

MACVEC generates a report for the hotspots in the application that can benefit of vectorizing part of the code. The hotspots are detected using a measurement module (i.e. HPCToolkit or Intel VTune).

MACVEC will recompile the code. When using a `makefile`, it is necessary that it implements a `clean` target, as `make clean` will be invoked before recompiling the code. After that first step, MACVEC will pass the Intel compiler flags for generating a vectorization report using the environment variables `CFLAGS`, `CXXFLAGS` and `FCFLAGS`. In order for this step to work, it is also necessary that the `makefile` does not overwrite those variables, but rather update them. Although MACVEC will generate vectorization reports for all the source files of the code, it will only analyze those files that contain functions or loops that represent hotspots. This will depend on the `threshold` specified when calling PerfExpert.

Command Line Arguments

MACVEC is not automatically used when running PerfExpert without specifying the modules that will be run. In order to use MACVEC, the user needs to pass it as a module:

```
perfexpert 0.1 --modules=make,lcpi,hpctoolkit,macvec ./program
    program_args
```

The previous command will analyze the code using LCPI and HPCToolkit to find the hotspots. It will then recompile the code with the `make` command and analyze the vectorization report with MACVEC.

MACVEC does not accept any extra arguments.

Exit Status

6.2.5 recommender

Description

`recommender` is part of PerfExpert. It uses PerfExpert database to select possible optimizations for hot-spots identified by `analyzer`.

Synopsis

recommender -f inputfile **[-o** outputfile**] [-a** workdir **] [-r** count**] [-d** database**] [-l** level**]**
[-m metricsfile**] [-chnv]**

Command Line Arguments

-f, --inputfile
Set the file from where performance measurements should be read (default: `STDIN`).

inputfile
File to use as input for performance measurements. File format: each application hot-spot starts with a percent sign (`%`), the `code` metrics are mandatory, the list of metrics may vary, according to the example above:

```
% 1st application hot-spot
code.section_info=Loop in function compute.omp_fn.3() at ./mm_omp.c:14
code.filename=./mm_omp.c
code.line_number=14
code.type=loop
code.function_name=compute.omp_fn.3
code.extra_info=2
code.representativeness=99.8
code.runtime=22.204
(list of metrics)
...
% 2nd application hot-spot
...
```

-o, --outputfile
File to write possible optimizations (default: `STDOUT`). If the file exists it's content will be overwrite.

outputfile
File to use as output for recommendations (default: `STDOUT`).

-a, --automatic
Enable automatic performance optimization and create files into a temporary directory (default: off) This option is available only when PerfExpert is compiled with support to ROSE (`http://rosecompiler.org/`).

`workdir`
Temporary directory where intermediary files for automatic performance optimization are created.

-r, --recommendations
Set the number of optimizations perfexpert should provide.

`count`
Number of optimizations perfexpert should provide.

-d, --database
Full path and name of PerfExpert database file (default: `PERFEXPERT_VARDIR/RECOMMENDATION_DB`).

`database`
Database file (default: `PERFEXPERT_VARDIR/RECOMMENDATION_DB`).

-m, --metricsfile
Choose a different set of performance metrics. A temporary table will be created, in other words, it automatically enables the `-n` argument.

`metricsfile`
File to define metrics different from the default one. File format: each line represents a metric.

-n, --newmetrics
Do not use the system metrics table. A temporary table will be created using the default metrics file (default: `PERFEXPERT_ETCDIR/METRICS_FILE`).

-v, --verbose
Enable verbose mode (default: level 5).

-l, --verbose_level
Enable verbose mode using a specific verbose level.

`level`
Verbose level (range: 1-10).

-c, --colorful
Enable colors on verbose mode, no weird characters will appear on output files.

-h, --help
Show the help message.

Environment Variables

Here is a complete list of the environment variables `recommender` uses. The command line arguments overwrite the value set by any of the environment variables. PerfExpert automatically sets most of these variables, but it does not overwrite their values if they are already set. Thus, it is possible to use these variables to pass options to `recommender`.

`PERFEXPERT_RECOMMENDER_INPUT_FILE`
This variable has the same functionality than the `-f` command line option. It should be a valid readable file. It is possible to set it's value to include a full path. PerfExpert automatically sets the value of this variable.

PERFEXPERT_RECOMMENDER_OUTPUT_FILE
: This variable has the same functionality than the `-o` command line option. It is possible to set it's value to include a full path. PerfExpert automatically sets the value of this variable.

PERFEXPERT_RECOMMENDER_DATABASE_FILE
: This variable has the same functionality than the `-d` command line option. It's value should be a valid PerfExpert database file. By default, PerfExpert sets the value of this variable to the database it is using.

PERFEXPERT_RECOMMENDER_METRICS_FILE
: This variable has the same functionality than the `-m` command line option. It should be a valid readable file. It is possible to set it's value to include a full path.

PERFEXPERT_RECOMMENDER_REC_COUNT
: This variable has the same functionality than the `-r` command line option. It's value should be a valid integer number higher than `0`. By default, PerfExpert sets the value of this variable to value it is set to run with.

PERFEXPERT_RECOMMENDER_WORKDIR
: The variable controls where `recommender` will generate intermediary files used for automatic optimization. The value of this variable is automatically set by PerfExpert and it the same working directory the `perfexpert` itself uses. Usually, it's value is like `.perfexpert-temp.XXXXXX`.

PERFEXPERT_RECOMMENDER_PID
: This variable is used by PerfExpert to identify calls to `recommender` during the same execution but in a different optimization cycle. It is automatically set by PerfExpert and it's value is the PID of the `perfexpert` process.

PERFEXPERT_RECOMMENDER_VERBOSE_LEVEL
: This variable has the same functionality than the `-l` command line option. It's value should be a valid integer number within `1` and `10`. By default, PerfExpert sets the value of this variable to value it is set to run with.

PERFEXPERT_RECOMMENDER_COLORFUL
: This variable has the same functionality than the `-c` command line option. It's value should be `0` or `1`. By default, PerfExpert sets the value of this variable to value it is set to run with.

Exit Status

`recommender` exits 0 on success, 1 for general errors, and 2 when there is no optimization recommendations available. Any other exit code, even if not specified here, should be considered as an error.

6.2.6 perfexpert_ct

Synopsis

Description

Command Line Arguments

Environment Variables

Here is a complete list of the environment variables `perfexpert_ct` uses. The command line arguments overwrite the value set by any of the environment variables. PerfExpert automatically sets most of these variables, but it does not overwrite their values if they are already set. Thus, it is possible to use these environment variables to pass options to `perfexpert_ct`.

`PERFEXPERT_CT_INPUT_FILE`
: This variable has the same functionality than the `-f` command line option. It should be a valid readable file. It is possible to set it's value to include a full path. PerfExpert automatically sets the value of this variable.

`PERFEXPERT_CT_OUTPUT_FILE`
: This variable has the same functionality than the `-o` command line option. It is possible to set it's value to include a full path. PerfExpert automatically sets the value of this variable.

`PERFEXPERT_CT_DATABASE_FILE`
: This variable has the same functionality than the `-d` command line option. It's value should be a valid PerfExpert database file. By default, PerfExpert sets the value of this variable to the database it is using.

`PERFEXPERT_CT_WORKDIR`
: The variable controls where `recommender` will generate intermediary files used for automatic optimization. The value of this variable is automatically set by PerfExpert and it the same working directory the `perfexpert` itself uses. Usually, it's value is like `.perfexpert-temp.XXXXXX`.

`PERFEXPERT_CT_PID`
: This variable is used by PerfExpert to identify calls to `perfexpert_ct` during the same execution but in a different optimization cycle. It is automatically set by PerfExpert and it's value is the PID of the `perfexpert` process.

`PERFEXPERT_CT_VERBOSE_LEVEL`
: This variable has the same functionality than the `-l` command line option. It's value should be a valid integer number within `1` and `10`. By default, PerfExpert sets the value of this variable to value it is set to run with.

`PERFEXPERT_CT_COLORFUL`
: This variable has the same functionality than the `-c` command line option. It's value should be `0` or `1`. By default, PerfExpert sets the value of this variable to value it is set to run with.

Exit Status

Pattern Recognizers Interface

Code Transformers Interface

6.2.7 Other Auxiliary Tools

`hound`

`sniffer`

6.3 PerfExpert Temporary Directory

Every single time PerfExpert is executed it generates a temporary directory. The name of the directory varies, but it always is in the form of `.perfexpert-temp.FPAEx6` and is located in the directory from where you called the `perfexpert` command. Generally, PerfExpert does not remove the temporary directory, unless the user specify in contrary using the `-g` command line option.

The temporary directory may be useful in many situations, such as:

- Re-analyzing results from a previous PerfExpert run.
- Checking the output of the binary executable.
- Comparing the performance of different version of the same code.
- Searching for execution errors.

The basic directory tree of the temporary directory looks like the following:

```
.perfexpert-temp.FPAEx6
.perfexpert-temp.FPAEx6/1
.perfexpert-temp.FPAEx6/1/database
.perfexpert-temp.FPAEx6/1/database/src
.perfexpert-temp.FPAEx6/1/measurements
.perfexpert-temp.FPAEx6/2
.perfexpert-temp.FPAEx6/2/database
.perfexpert-temp.FPAEx6/2/database/src
.perfexpert-temp.FPAEx6/2/measurements
  (...)
```

where for each optimization cycle there is a sub-directory named as cardinal numbers. The sub-directories inside each of the optimization cycle sub-directories are user for:

`database`
: Stores the `experiment.xml` file and the sub-directory for the source code. The experiment file, which is generated by `hpcprof` (a tool from HPCToolkit), is the application performance profile.

`database/src`
: Stores the source code extracted from the binary executable. The full path of the source code will be represented here and also the statically linked libraries.

`measurements`
: Stores the measurement files collected with `hpcrun` (a tool from HPCToolkit). There is a `.log` file for each `hpcrun` invocation and one `.hpcrun` data file for each thread of each invocation of the binary executable, as following:

```
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-000-7281ba5a-91510-0.
   log
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-000-7281ba5a-91510-0.
   hpcrun
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-001-7281ba5a-91510-0.
   hpcrun
  (...)
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-014-7281ba5a-91510-0.
   hpcrun
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-015-7281ba5a-91510-0.
   hpcrun
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-000-7281ba5a-91614-0.
   log
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-000-7281ba5a-91614-0.
   hpcrun
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-001-7281ba5a-91614-0.
   hpcrun
```

```
  (...)
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-014-7281ba5a-91614-0.
   hpcrun
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-015-7281ba5a-91614-0.
   hpcrun
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-000-7281ba5a-91718-0.
   log
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-000-7281ba5a-91718-0.
   hpcrun
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-001-7281ba5a-91718-0.
   hpcrun
  (...)
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-014-7281ba5a-91718-0.
   hpcrun
.perfexpert-temp.FPAEx6/1/measurements/mm_omp-000000-015-7281ba5a-91718-0.
   hpcrun
```

where the filenames are composed of the name of the binary executable, the MPI JOBID (if existent), the thread ID, the process ID, and other control data. In the example above, we shown the `.log` files and the `.hpcrun` files from three invocations of `hpcrun` (the number of invocations may change from one system to other) using 16 threads each of them.

Besides the sub-directories described above, there are the following files inside each of the optimization cycle sub-directory:

`recommender_report.txt`

`analyzer_report.txt`

`analyzer_metrics.txt`

`mm_omp.hpcstruct`

`hpcstruct.output`, `hpcrun.1.output`, `hpcrun.2.output`, `hpcrun.3.output`, `hpcprof.output`

6.4 Analyzing the Results from a Previous PerfExpert Run

6.5 Executing on the Intel® Xeon Phi™ Coprocessors

Chapter 7

Extending PerfExpert

7.1 PerfExpert Database Layout

7.2 Adding Metrics to PerfExpert

7.2.1 Performance Counters

7.2.2 Derived Metrics

7.3 New Recommendations for Optimization

7.4 Enabling New Automatic Optimizations

7.4.1 Pattern Recognizers Interface

7.4.2 Code Transformers Interface