PerfExpert v4.0

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

User Manual

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

August 2013
Revision 4

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 TACC Staff

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

- 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: `fialho@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.

If you would like to contribute to PerfExpert or need help to do research using PerfExpert, please contact us at `fialho@utexas.edu`. Complete directions on how to add to or modify each phase of PerfExpert can be found on the PerfExpert web site `https://bitbucket.org/leonardofialho/perfexpert/`.

1.6 Mailing List

Out mailing list is hosted on Google Groups. To subscribe send a message to:
`subscribe-perfexpert@googlegroups.com` or access the group's webpage at:
`https://groups.google.com/d/forum/perfexpert`.

1.7 Funding Sources

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

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_YOU_HAVE_INSTALLED_BOOST \
    --disable-tutorial-directory --disable-projects-directory \
    --disable-tests-directory --disable-opencl --disable-php \
    --disable-cuda --disable-binary-analysis --disable-java \
    --with-gomp_omp_runtime_library=WHERE_LIBGOMP_IS_INSTALLED \
    --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=${INSTALL_DIR}
$ make install
```

NOTICE:

The 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 (`fialho@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 BitBucket:`https://bitbucket.org/leonardofialho/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 the `configure` may looks like this:

```
$ ./configure --prefix=WHERE_YOU_WANT_TO_INSTALL_IT \
    --with-rose=WHERE_ROSE_IS_INSTALLED \
    --with-jvm=WHERE_YOUR_JVM_IS_INSTALLED\
    --with-papi=WHERE_PAPI_IS_INSTALLED\
    --with-apache-ant=WHERE_YOU_HAVE_DECOMPRESSED_ANT
```

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 (`fialho@utexas.edu`) or use our mailing list: `perfexpert@googlegroups.com`.

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

First Steps on 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

# job name
#SBATCH -J myMPI

# output and error filename (%j stands to jobID)
#SBATCH -o myMPI.o%j

# total number of mpi tasks requested
#SBATCH -n 16

# queue (partition) -- normal, development, etc.
#SBATCH -p development

# run time (hh:mm:ss) - 1.5 hours
#SBATCH -t 01:30:00

# run the executable named my_program
perfexpert 0.1 ./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> [-gvch] [-l level] [-d database] [-r count] [-p prefix]
                  [-m target|-s sourcefile] [-a FILE] [-b FILE] <program_executable>
                  [program_arguments]

 <threshold>         Define the relevance (in % of runtime) of code fragments PerfExpert
                     should take into consideration (> 0 and <= 1)
 -d --database       Select the recommendation database file
 -r --recommend      Number of recommendation to show
 -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
                     automatic optimization (-a)
 -p --prefix         Add a prefix to the command line (e.g. mpirun) use double quotes to
                     specify multiple arguments (e.g. -p "mpirun -n 2"
 -b --before         Execute FILE before each run of the
                     application
 -a --after          Execute FILE after each run of the application
 -v --verbose        Enable verbose mode using default verbose level (1)
 -l --verbose_level Enable verbose mode using 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
```

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 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         :  2.4 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>+
   - L1d hits            :  1.3 >>>>>>>>>>>>>>>>>>>>>>>>>
   - L2d hits            :  0.3 >>>>>
   - L2d misses          :  0.8 >>>>>>>>>>>>>>>>
* 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.

3.4 List of Recommendations for Optimization

If PerfExpert runs with `-r` option enabled, it will generate the performance analysis report and a list of suggestions for bottleneck remediation 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 (truncated to only the most important recommendation) is shown following. Each entry in this list is similar to the following one:

```
Loop in function compute() at mm_omp.c:15 (100% of the total runtime)
*******************************************************************************
change the order of loops
this optimization may improve the memory access pattern and make it
more cache and TLB friendly.
loop i {
  loop j {...}
}
=====>
loop j {
  loop i {...}
}
```

Chapter 4

PerfExpert for Advanced Users

4.1 PerfExpert Temporary Directory

4.2 Command Line Arguments Unveiled

4.3 Using Environment Variables

PERFEXPERT_CT_VERBOSE_LEVEL

PERFEXPERT_CT_INPUT_FILE

PERFEXPERT_CT_OUTPUT_FILE

PERFEXPERT_CT_DATABASE_FILE

PERFEXPERT_CT_WORKDIR

PERFEXPERT_CT_COLORFUL

PERFEXPERT_CT_PID

PERFEXPERT_VERBOSE_LEVEL

PERFEXPERT_DATABASE_FILE

PERFEXPERT_REC_COUNT

PERFEXPERT_COLORFUL

PERFEXPERT_MAKE_TARGET

PERFEXPERT_SOURCE_FILE

CC

CFLAGS

PERFEXPERT_CFLAGS

PERFEXPERT_RECOMMENDER_VERBOSE_LEVEL

PERFEXPERT_RECOMMENDER_INPUT_FILE

PERFEXPERT_RECOMMENDER_OUTPUT_FILE

PERFEXPERT_RECOMMENDER_DATABASE_FILE

PERFEXPERT_RECOMMENDER_METRICS_FILE

PERFEXPERT_RECOMMENDER_REC_COUNT

PERFEXPERT_RECOMMENDER_WORKDIR

PERFEXPERT_RECOMMENDER_COLORFUL

PERFEXPERT_RECOMMENDER_PID

4.4 Re-Analyzing the Results from a Previous PerfExpert Run

Chapter 5

Extending PerfExpert

5.1 Adding Metrics to PerfExpert

5.2 New Recommendations for Optimization

5.3 Enabling New Automatic Optimizations