PerfRegions Documentation

Silvia Mocavero, Martin Schreiber et al.

October 14, 2016

```
(Please add yourself to the list of authors in alphabetical order if you contributed something to PerfRegi
```

Abstract

Gaining understanding in performance limitations is known to be a very challenging job. Even with the arise of automized commercial tools such as Intel Amplifier, or open source tools such as HPCToolkit, this still does not allow to get highly accurate statistics on the performance of selected areas of the codes from scientific computing area and to guarantee portability on several architectures. An alternative is to extend each program by hand with code which allows gaining insight into the performance, however this is very time consuming as well as error prone.

This document describes our development which is called PerfRegions which suggests an annotation of the program. This reveals detailed information on selected regions on the code and significantly reduces the time until insight is gained into the performance.

1 Targets

We focus on the following targets

1. **Language flexibility**: Supporting C and Fortran code
2. **Programmability**: The suggested code annotation should be easy to use
3. **Portability**: The only requirement is the PAPI library installed
4. **MPI support**: The development should support MPI-based parallelization and accumulation of the results
5. **All-in-one information**: The tool should be able to measure accurate timings as well as performance counters.
6. **No recompilation**: Since the number of performance counters to use is limited, changing the performance counters to be measured should not require recompilation.

2 Realization

The package includes 2 main folders:

1. the *src* folder, which contains the source code of perf regions. In particular, the folder includes (i) the header files, (ii) the source code for the performance measurement init and finalize, and for the region measurement start and stop and (iii) the interfaces to the PAPI libraries.
2. the *example* folder, which includes two example folders (C and Fortran examples).

3 Language extensions

The language extensions should be kept very flexible. This allows e.g. replacing existing annotations of sections/regions with the annotations of PerfRegions. This was in particular of our interest since this project was originally developed to replace timing constructs in the NEMO development in order to gain per-region information on hardware performance counters.

3.1 Example code in C

Due to the flexibility of the language extensions, we'd like to give a concrete C-code example:

```
#include <stdio.h>
#include <stdlib.h>

#pragma perf_region include

double *a;
int size;

void run_computations()
{
#pragma perf_region init

  for (int k = 0; k < iters; k++)
  {
#pragma perf_region start foo
    [... foo region computations ...]
#pragma perf_region stop foo

#pragma perf_region start bar
    [... bar region computations ...]
#pragma perf_region stop bar
  }

#pragma perf_region finalize
}
[...]
```

In this version, we use #pragma as a marker for the preprocessor to identify which parts of the code to replace with the PerfRegion code. Since the preprocessor uses regular expressions to detect such regions, this can be basically any other language extension. More information is provided in the next section.

3.2 PerfRegion C-language extensions

Using PerfRegion on C-code, the program annotation is given by

```
#pragma perf_region [identifier] [name]
```

The following table gives an overview of the language extensions:

Identifier	**Description**
include	This construct is replaced with PerfRegion header files
init	Initialize the PerfRegion library
finalize	Finalize the PerfRegion and *output* a summary of the measured performance
start [name, perf_flag]	Annotation of the start of a region to run performance measurements. 'name' has to be a unique identifier, 'perf_flag' is an indication of the performance measurement type (timing, counters or both).
stop [name]	Annotation of the end of a region to run performance measurements.'name' has to be a unique identifier and has to match to 'name' at the previous start annotation.

3.3 Fortran support

The Fortran support currently only supports replacing the timing constructs in the NEMO development. Since this shows the flexibility of PerfRegions, an example is given as follows:

[TODO: Tim, Can you please provide an example?]

4 Using PerfRegions

4.1 Compiling of PerfRegions

PerfRegions can be compiled in it's main folder by typing 'make'.

4.2 Preprocessing annotated code

The preprocessor is realized with a python script. Example scripts can be found in the example directory, e.g. 'examples/array_test_c/perf_regions_instrumentation.py'. This script instructs the preprocessor where to find the code and how to preprocess the code.

4.2.1 Preprocess

To start preprocessing the code, the script is executed with 'preprocess' as parameter:

```
./perf_regions_instrumentation.py preprocess
```

This preprocessing generates a text file 'perf_region_list.txt' which contains the names of the performance regions. This will be used for a pretty print of the performance results.

4.2.2 Reverting to original code

The preprocessing generates code which can be reverted to its original one.

To revert the PerfRegion code, call the script with 'cleanup' parameter:

```
./perf_regions_instrumentation.py cleanup
```

4.3 3rd party library, compiling and linking

PerfRegions requires the PAPI library installed.

4.3.1 Linker flags:

```
-lpapi -L[path to perf regions] -lperf_regions
```

4.3.2 Compile flags:

```
-I[path to perf regions]/src
```

4.4 Executing performance measurements

First of all, the environment variable LD_LIBRARY_PATH has to be set to the path of the PerfRegion build directory:

```
export LD_LIBRARY_PATH=../../build:$LD_LIBRARY_PATH
```

Each platform might have a different set of performance counters. The available performance counters identifiers can be determined via

```
papi_avail
```

Only a limited number of performance identifiers can be specified. PerfRegions allows to specify a list of performance counter identifiers via the environment variable such as

```
export LIST_COUNTERS=PAPI_L1_TCM,PAPI_L2_TCM,PAPI_L3_TCM
```

4.5 Example output

An example output looks as follows:

```
***********************************
Iterations: 2
Performance counters profiling:
----------------------
Section      PAPI_L1_TCM       PAPI_L2_TCM       PAPI_L3_TCM
FOO        3.2926500e+046      4.6655000e+03     1.3370000e+03
BAR          6.5657000e+04     7.7765000e+03     1.3550000e+03

Total timing (sum) :
----------------------
Wallclock time (s) 7.3830400e-03

Timing profiling:
----------------------
Section   Wallclock time (sec)   Wallclock time (%)      Frequency
FOO          3.9739800e-03            53.82                2
BAR          2.4719800e-03            33.48                2

Timing started on Sat Oct 1 12:48:13 2016
Timing ended on Sat Oct 1 12:48:13 2016
```

5 Conclusions

[Let's see how it goes and then fill in this section...]