PTF DVFS Plugin User's Guide

PTF Version: 1.1
DVFS Plugin Version: 1.1

Carla Guillen

13.04.2015

Contents

Chapter 1

Introduction

One of the main goals of the DVFS Plugin is to minimize the amount of energy that the HPC system uses when executing an application. One method of reducing the energy is to lower the clock frequency of the processors, which is possible in modern processors. However, manual tuning of processor frequencies for parallel applications to obtain an optimal energy to solution can be extremely time consuming. So it is important to provide an automatic tool that enables the user to:

- Measure the energy consumption of processors.
- Analyze the performance and energy data from the application.
- Provide the optimal energy to solution configuration for the given application.
- Set the new core frequency and the userspace governor during the application runtime.

The Periscope Tuning Framework (PTF) provides a mechanism that automatically extracts the information required for tuning the system. The PTF DVFS plugin provides a mechanism to tune the processor frequencies based on the analysis of the data provided by the PTF to produce the optimal energy to solution configuration for the specific parallel application.

The DVFS plugin relies on energy, power, and time models to reduce the search space for tuning. The models use performance data collected at the pre-analysis stage. The frequency that the model predicts as consuming the minimum energy consumption is calculated by the plugin. The neighboring frequencies of the predicted frequency are explored via experiments by the plugin and the best frequency among all three is given as an output.

Chapter 2

Quick Start

The DVFS Plugin is installed along with the Periscope Tuning Framework Please refer to the PTF Installation Guide for a complete description of the installation process.

2.1 Basic configuration

Set the environment variables as described in the PTF Installation Manual. Make sure you use the following module:

```
module load enopt/1.3
```

To configure the build environment for the DVFS we use the following configure command within a terminal window:

```
mkdir <build-directory>
```

```
cd <build-directory>
```

```
<path-to-source-code-directory>/configure \
--with-ace-include=$ACE_BASE --with-ace-lib=$ACE_LIBDIR \
--with-boost-include=$BOOST_BASE/include \
--with-boost-lib=$BOOST_LIBDIR \
--prefix=$HOME/install/periscope
```

The next step is to build and install the DVFS Plugin using the following commands in the terminal window:

```
make -j 16 install
```

and then

```
make install
```

It must be assured, that the PAPI library was compiled with the RAPL-component support. This PAPI component uses the RAPL library which allows in Sandy Bridge-EP and Ivy Bridge architectures the access to the energy and power sensors. To check this, the following command must be run:

```
papi_native_avail
```

If the output contains the string:

```
| rapl:::PP0_ENERGY:PACKAGE0 |
| rapl:::PP0_ENERGY:PACKAGE1 |
```

Then the PAPI library has RAPL support and thus, the energy measurements can be performed. Otherwise please refer to the PAPI documentation on how to compile PAPI with the RAPL support.

Note: Depending on the Linux kernel and kernel settings used, the command `papi_native_avail` may required privileged permissions (root) to list the above RAPL counters. If this requirement is not fulfilled for these kernels, the strings will not appear even if the energy counters are available.

2.2 Running DVFS Plugin

The DVFS plugin runs within the Periscope Tuning Framework as a plugin. It can be started using the normal `psc_frontend` command (refer to the PTF User's Guide). In that case, the tune flag must be configured in order to specify the DVFS plugin:

```
--tune=dvfs
```

For instance, running a certain application in localhost with a concrete number of tasks should look like:

```
psc_frontend --apprun="./<APPLICATION_NAME>" \
--mpinumprocs=<MPI_PROCS> --tune=dvfs --force-localhost
```

This command will start the application and the DVFS tuning strategy.

2.3 Execution results

Once the application finishes correctly and upon completion of the tuning measurements, the DVFS shows the results for three CPU frequencies. The DVFS predicts one frequency based on an energy, time and power model.

The predicted frequency as well as two other neighboring frequencies (one lower and one higher to the predicted frequency, if available) are used for the experiments. By the default, only these three frequencies are used. However, the plugin can be configured to increase the search space (by increasing the number of neighboring frequencies).

As an example, a possible output of the plugin running a certain application is presented below (the output is slightly modified to fit the page).

```
Found Optimum Scenario:
 Region id: USER_REGION (file1.f90:4)   Optimum Scenario:2      Frequency: 1700
 Region id: USER_REGION (file2.f90:11)   Optimum Scenario:4      Frequency: 1900
 Region id: USER_REGION (file2.f90:27)   Optimum Scenario:0      Frequency: 1500
Search Path:
Scenario|Governor |Freq | Energy   | Runtime | EDP      | Region
        |         |(MHz)| (J)      | (s)     |          |
--------+---------+-----+----------+---------+----------+---------------------------
0       |Userspace|1500 | 2810.000 | 19.163  | 53849.154| USER_REGION (file1.f90:4)
0       |Userspace|1500 | 362.000  | 1.558   | 564.137  | USER_REGION (file2.f90:11)
0       |Userspace|1500 | 1729.000 | 17.526  | 30302.454| USER_REGION (file2.f90:27)
[...]
9       |Userspace|2400 | 2922.000 | 12.466  | 36424.775| USER_REGION (file1.f90:4)
9       |Userspace|2400 | 363.000  | 1.558   | 565.706  | USER_REGION (file2.f90:11)
9       |Userspace|2400 | 2403.000 | 10.810  | 25977.391| USER_REGION (file2.f90:27)
Other data:
Scenario|Governor |Freq |Avrg Power|TCO       | EDDP      |Region
        |         |(MHz)| (W/node) |(EUR/node)|           |
--------+---------+-----+----------+----------+-----------+-------------------------
0       |Userspace|1500 |89.859    |2.601     |1031932.878|USER_REGION (file1.f90:4)
0       |Userspace|1500 |119.354   |0.224     |879.146    |USER_REGION (file2.f90:11)
0       |Userspace|1500 |50.097    |2.191     |531080.809 |USER_REGION (file2.f90:27)
[...]
9       |Userspace|2400 |118.244   |1.788     |454060.323 |USER_REGION (file1.f90:4)
9       |Userspace|2400 |119.352   |0.224     |881.608    |USER_REGION (file2.f90:11)
9       |Userspace|2400 |115.167   |1.541     |280825.990 |USER_REGION (file2.f90:27)
```

Chapter 3

DVFS Autotuning Approach

The DVFS plugin follows the general PTF plugin approach (refer also to the *PTF User's Guide*).

3.1 Tuning parameter

The DVFS plugin employs as a tuning parameter, the available CPU frequencies of the processor (except for turbomode, which is 3.1 GHz for the Intel Sandy Bridge-EP processors). The Intel Sandy Bridge-EP microarchitecture uses frequencies between 1.2 GHz and 2.7 GHz in steps of 0.1 GHz (i.e. 1.2, 1.3, 1.4, ... , 2.6, and 2.7 GHz).

3.2 Search strategy

In order to find the best tuning of an application, a search through the tuning space has to be performed. The DVFS plugin uses a model-based search which reduces the search space to three frequencies.

3.3 Tuning scenario

Based on the strategy, consecutive *tuning scenarios* are then being generated at run time and the energy to solution of the application is evaluated for each of these scenarios.

3.4 Tuning action

Applying a scenario to the application means setting the CPU parameters before running the application. Therefore, the tuning action performed by the DVFS plugin is to set the CPU frequency.

Chapter 4

Advanced Configuration

In the following sections, the configuration of the runtime environment is described. The plugin can change the objective function and the domains to change the frequency via the two environment variables: `PSC_DVFS_TUNING_OBJECTIVE` and `PSC_FREQ_TO_ALL_NODE` respectively.

4.1 Configuration of the Objective Function

If the user does not set the `PSC_DVFS_TUNING_OBJECTIVE` environment variable the tool will calculate the model based on the Model Energy 1. Each model predicts the best frequency for the chosen tuning objective. The following models are available:

Model Energy 1: optimizes based on energy measurements directly, by using the reference frequency.

Model Energy 2: optimizes based on power and time measurements of the reference frequency. In contrast to model 1, power and time are predicted separately and their product is used.

Energy-Delay: optimizes based on the product of energy and runtime.

Total Cost of Ownership (TCO): optimizes based on the total cost of ownership. The prices of energy are based on current prices used in Germany. The cost function is adapted to the SuperMUC in the Leibniz Supercomputing Centre.

Power capping: optimizes taking into account a maximum power limit. The optimal frequency recommended by the plugin will not surpass the power limit. This power limit has been defined as 110 Watt.

Model Policy 1: optimizes the increase of performance only when it is greater than the increase of energy. Normalized energy values are used to compare the increase of energy.

Model Policy 2: optimizes the increase of performance only when it is greater than the increase of power. Normalized energy values are used to compare the increase of power.

Model Policy 3: optimizes by staying below a powercap (110W) or by having a significant performance increase (a factor 1.1 performance increase).

Model Policy 4: optimizes by considering a maximum performance degradation with respect to the nominal frequency of no more than 10% and staying below a powercap (110W). Table 4.1 shows the different available models and the corresponding ids that can be used to set them with the `PSC_DVFS_TUNING_OBJECTIVE` environment variable.

Model	PSC_DVFS_TUNING_OBJECTIVE id number
Model Energy 1	1
Model Energy 2	2
Energy-Delay Product	3
TCO	4
Power Capping	5
Model Policy 1	6
Model Policy 2	7
Model Policy 3	8
Model Policy 4	9

Table 4.1: Available tuning objectives and ids.

Example: Choosing TCO as tuning objective and running the DVFS plugin is done as follows.
Check in Table 4.1 the identification number of the TCO model (the id number is on the second column) and set is as environment variable before

running PTF with the plugin as shown below.

```
export PSC_DVFS_TUNING_OBJECTIVE=4
psc_frontend --apprun="./<APPLICATION_NAME>" \
--mpinumprocs=<MPI_PROCS> --tune=dvfs --force-localhost
```

4.2 Configuration of the Domain to Change the Frequency

The environment variable `PSC_FREQ_TO_ALL_NODE` specifies how the plugin will change the frequencies of the processors. By default, the plugin changes only the frequency of one core from a task or a thread. Intel Sandy Bridge-EP processors change the frequency at the level of the processor (each processor contains 8 cores). If the set frequencies are different in a processor, the architecture takes the highest frequency set for all the cores and applies this to the entire processor. If not all cores in a processor are being used for an application, the change of frequency may not be performed correctly. For this cases it is convenient to set the `PSC_FREQ_TO_ALL_NODE` environment variable to one, such that the master process (or the task that runs in a sequential region in the case of OpenMP) will change the frequency of the entire node (there are two processors in a node). Note that the downside of this, is the increased latency of the changes in frequency.

Example: Setting the frequency from the master process to all the node is done as follows. Exporting the environment variable should be done before running the PTF command.

```
export PSC_FREQ_TO_ALL_NODE=1
psc_frontend --apprun="./<APPLICATION_NAME>" \
--mpinumprocs=<MPI_PROCS> --tune=dvfs --force-localhost
```

4.3 Configuration of the Number of Neighbors in the Search Space

The environment variable `PSC_FREQ_NEIGHBORS` specifies the number of neighboring frequencies that will be searched in the experimentation. By default only one neighboring frequency which is higher and one neighboring frequency which is lower than the predicted frequency will be used. However, if the `PSC_FREQ_NEIGHBORS` environment variable is set, the number of neighbors to the right and to the left of the predicted frequency will be used.

Example: The following setting will allow to experiment with the frequencies in the range: $[f_{p-4}, f_{p+4}]$. In this case f_p is the predicted frequency.

```
export PSC_FREQ_NEIGHBORS=4
```

Only a maximum of seven neighboring frequencies are taken. If the environment variable exceeds this number (or is set with negative values) the plugin will use the default value (one neighbor at each side of f_p).

Chapter 5

How To Use the Tuning Advice

Once the application is successfully completed, the DVFS plugin displays the list of all tested scenarios identified each of them by its id, the specific combination of the tuning parameters applied to that region, and the energy and time measurements performed with that configuration, as well as the scenario id where the optimal values where found. In this case, the best scenario is the one that presents a better energy to solution.

One should call the specific functions to change the frequency, provided by the periscope interface, as the first code line of the main function, setting as parameters the configuration found in the optimum scenario. The user can opt to do this automatically by using the enopt library. This library uses the same interface as the interface of PTF with the application. The instrumented source code can be recompiled and linked only to the enopt library. In order to implement the advice, the user has to set the environment variable `PSC_DVFS_ADVICE_FILE` which contains one string: the path to the enopt advice (called advice_enopt_¡pid¿.txt . For example:
`export PSC_DVFS_ADVICE_FILE=/path/to/advice/advice_enopt_134.txt`
The environment variable must be set before run the application. Internally, during runtime, the application reads these two files and produces a hashed map which contains the file id and the region's first line as the key and the optimized frequency as the map's value. These two values are provided by the instrumentation, when it inserts the `start_region` and the `stop_region`, so at runtime the corresponding frequency from the advice will be set on each region.