Folding
User guide manual
for version 1.0rc8

tools@bsc.es

March 16, 2015

Contents

List of Figures

List of Tables

Chapter 1

User guide

1.1 Quick start guide

The Foldingis a mechanism that provides instantaneous performance metrics, source code references and memory references[1]. This mechanism receives a trace-file (currently generated by Extrae- see further details on generating a trace-file for the Foldingin Appendix 2) and generates plots and an additional trace-file depicting the fine evolution of the performance. The Foldinguses information captured through instrumentation and sampling mechanisms and smartly combines them. In this context, the samples are gathered from scattered computing regions into a synthetic region by preserving their relative time within their original region so that the sampled information determines how the performance evolves within the region. Consequently, the folded samples represent the progression in shorter periods of time no matter the monitoring sampling frequency, and also, the longer the runs the more samples get mapped into the synthetic instance. The framework has shown mean differences up to 5% when comparing results obtained sampling frequencies that are two orders of magnitude more frequent.

1.1.1 Decompressing the package

The Foldingpackage is distributed in a .tar.bz2 file that can be uncompressed in the working directory by executing the following command:

```
# tar xvz folding-1.0rc8-x86_64.tar.bz2
```

where `folding-1.0rc8-x86_64.tar.bz2` refers to the Foldingpackage as distributed from the BSC web page[2].

1.1.2 Contents of the package

After decompressing the package, the working directory should be populated with the directories (and corresponding descriptions) as listed in Table 1.1.

1.1.3 Quick run

This section provides examples of two types of execution of the Foldingtool. These examples take benefit of the included sample trace-files from the package. For further information on how to generate trace-files for the Foldingtool, check Appendix 2.

[1] This last option is experimental at the moment of writing this document

[2] `http://www.bsc.es/computer-sciences/performance-tools/downloads`

Table 1.1: Contents of the folding package.

Directory	Contents
bin/	Binary packages
etc/	
extrae-configurations/	Minimal configuration files for Extrae
models/	Configuration files to calculate performance models
basic/	
ibm-power5/	
ibm-power7/	
ibm-power8/	
intel-haswell/	
intel-nehalem/	
intel-sandybridge/	
include/	Header files for the development of 3rd party tools
lib/	Libraries for the folding
share/	Miscellaneous files
cfg/	Configuration files for Paraver
doc/	Documentation
html	
examples/	
folding-writer/	Example on how to generate data for the folding
user-functions/	Sample tracefile with manually instrumented regions
clusters/	Sample tracefile with automatically detected regions

Applied to manually instrumented regions

This first example uses a trace-file from the 444.namd SPEC benchmark that contains manually instrumented information that is located in

```
${FOLDING_HOME}/etc/share/examples/user-functions
```

This trace-file was generated by Extraeand delimiting the main loop using the ExtraeAPI[3], more precisely the `Extrae_user_function` which emits events with label `User function` (or event type 60000019). To apply the Foldingprocess to this trace-file, simply execute the following commands:

```
# cd ${FOLDING_HOME}/etc/share/examples/user-functions
# ${FOLDING_HOME}/bin/folding 444.namd.prv "User function"
```

Applied to automatically characterized regions

This example consists of a trace-file for the Nemo application when executed in MareNostrum3. This trace-file contains information regarding automatically characterized regions. This characterization has been done using the Clustering tool[4]. This tool enriches the trace-file by adding events labeled as `Cluster ID` (and event type 90000001) into the trace-file. In this context, these events identify similar computation regions based on the event value. To apply the Foldingprocess to this trace-file, simply execute the following commands:

[3]Please refer to `http://www.bsc.es/computer-sciences/performance-tools/documentation` for the latest ExtraeUser's Guide.

[4]Please refer to `http://www.bsc.es/computer-sciences/performance-tools/documentation` for the latest documentation with respect to the Clustering tool.

```
# cd ${FOLDING_HOME}/etc/share/examples/user-functions
# ${FOLDING_HOME}/bin/folding \
  nemo.exe.128tasks.chop1.clustered.prv "Cluster ID"
```

This trace-file also contains all the necessary performance counters in order to take benefit of several performance models based on performance counters. Simply add the `-model intel-sandybridge` option to the Foldingscript to generate the plots with information of the models instead of providing each performance counter individually. The commands to execute should look like this:

```
# cd ${FOLDING_HOME}/etc/share/examples/user-functions
# ${FOLDING_HOME}/bin/folding -model intel-sandybridge \
  nemo.exe.128tasks.chop1.clustered.prv "Cluster ID"
```

1.1.4 Exploring the results

The Foldingmechanism generates two types of output inside a directory named as the trace-file given (without the `.prv` suffix). The first type of results include a set of gnuplot files where each of these represents the evolution of the performance counters within the region. The tool also generates a Paravertrace-file with synthetic information derived from the Foldingmechanism.

Using gnuplot

With respect to the gnuplot files, the Foldingmechanism generates as many files as the combination of analyzed regions (clusters, OpenMP outlined routines, taskified OmpSs routines, or manually delimited regions) and the counters gathered during the application execution. The user can easily list the generated gnuplot files calling `ls *.gnuplot` within the directory created. The name of the gnuplot files contain the trace-file prefix, the identification of the region folded, and the performance counter shown. For instance, the example described in Section 1.1.3 generates output files that can be explored by executing the command:

```
# gnuplot -persist \
  444.namd.codeblocks.fused.any.any.any.main.Group_0.PAPI_TOT_INS.\
  gnuplot
```

When executing the aforementioned command, the `gnuplot` command should open a window that resembles that in Figure 1.1[5]. The Figure shows that the application faces six phases that execute at 4,500 MIPS approximately. Most of the code occurs in three code locations (being line 76 the most observed line), and we also observe that phases related to high MIPS are related with the activity in the middle of the code-line plot.

This file refers to the user routine `main` (which was manually instrumented) of the trace-file 444.namd.prv and provides information of the total graduated instructions (`PAPI_TOT_INS`). The user will notice that there are additional files for the different performance counters and they can explore them individually. The Foldingalso generates an additional plot that combines the metrics of all the counters into a single plot. This plot mainly provides information with respect to the MIPS rate (referenced on the right Y-axis), and ratio of the remaining performance

[5]Warning! If the user has problems to open the gnuplot, they should check whether the gnuplot installation is compatible and supports the default terminal. Otherwise, simply select the appropriate terminal (or leave it blank) in the first four lines in the gnuplot script

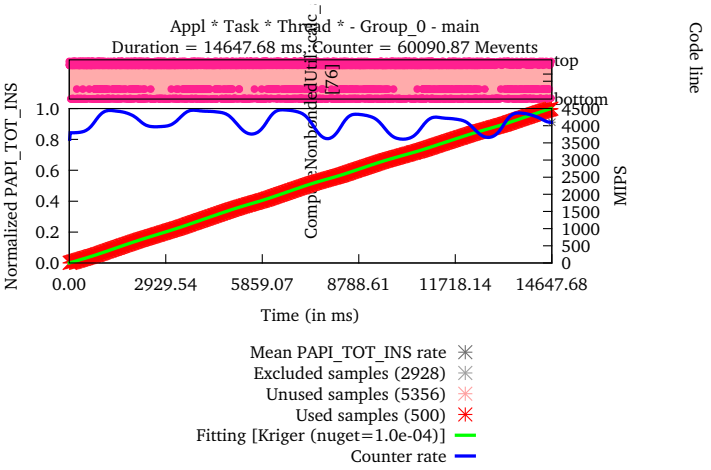

Figure 1.1: Evolution of graduated instructions for 444.namd.

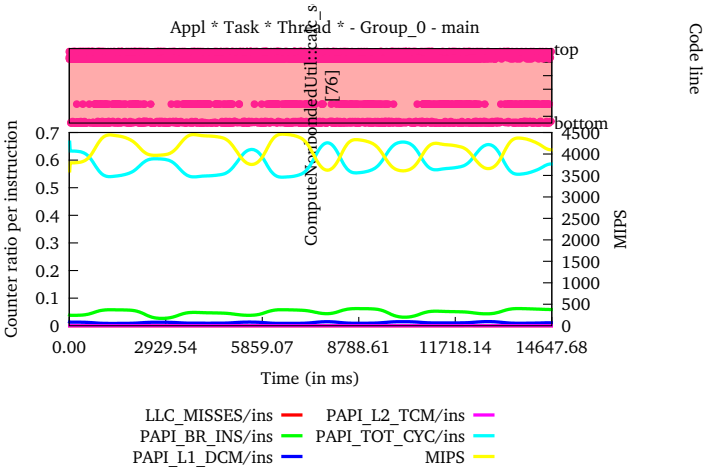

Figure 1.2: Evolution of multiple counters for 444.namd.

counters per instruction (referenced on the left Y-axis). For the particular case of the example from Section 1.1.3, this plot can be explored calling:

```
# gnuplot -persist \
  444.namd.codeblocks.fused.any.any.any.main.\
  Group_0.ratio_per_instruction.gnuplot
```

This command should generate an output combining all the performance counter slopes as shown in Figure 1.2.

The aforementioned instructions also apply to the automatically delimited example described in Section 1.1.3. In this case, the region names are numbered as `Cluster_1` to `Cluster_11`, but they also contain the trace-file prefix and the performance counters to explore them individually. If the user requested the performance models, then additional gnuplot files are created to provide information regarding these models. For the particular case of the Intel SandyBridge model, it generates three models that always generate the MIPS rate and add different metrics:

- *instruction-mix*
  Gives insight of the type of instructions executed along the region.

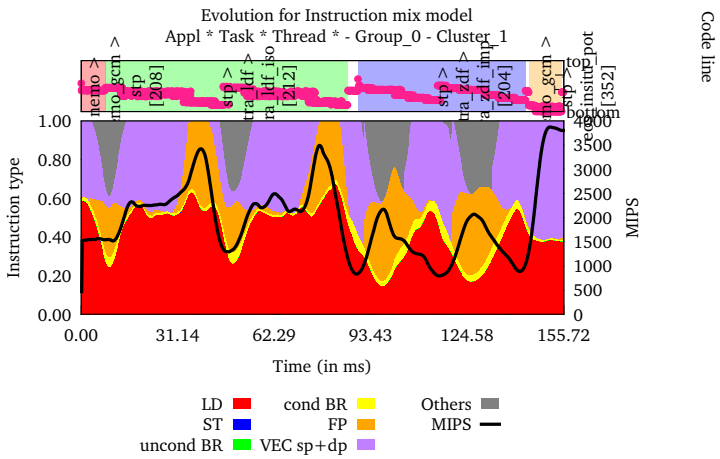

Figure 1.3: Instruction mix decomposition for Cluster 1 of Nemo.

- *architecture-impact*
  Provides information regarding to the cache misses at different levels and the branch mispredicts along the region.

- *stall-distribution*
  This plot shows information regarding on which components of the processor are stalling the processor pipeline.

For instance, to open the instruction mix for the region labeled as Cluster 1 of the Nemo application executed in Section 1.1.3, the user needs to open the plot invoking the commands below and should obtain a plot similar to Figure 1.3. The reader may see that the application shows two distinctive phases (green and blue) and within each of them there are two repetitions of the same performance.

```
# gnuplot -persist \
  nemo.exe.128tasks.chop1.clustered.codeblocks.fused.any.any.any.\
  Cluster_1.Group_0.instructionmix.gnuplot
```

The tool also provides a GUI-based tool to explore the plots. the user may invoke a visualizer named `wxfolding-viewer`, by invoking it from the newly created directory such as:

```
# ${FOLDING_HOME}/bin/wxfolding-viewer *.wxfolding
```

Using Paraver

The Foldingprocess generates a trace-file with a suffix `.folded.prv` that lets Paraverto display some parts of the folded results. The Foldingpackage includes several configuration files in the `${FOLDING_HOME}/share/cfg` directory for Paraverto help analysing the results. From the configuration files contained in that directory, we outline the following:

- *views/*
  - `win_folded_type.cfg`
    Generates a time-line that shows in which instances the Foldingresults have been

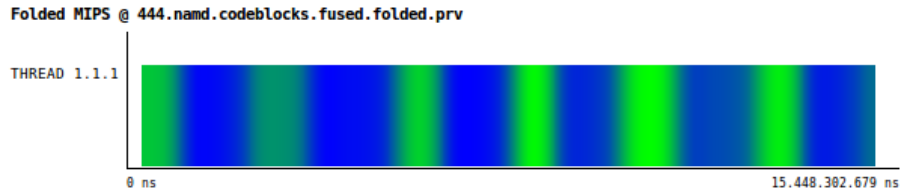

Figure 1.4: Evolution of graduated instructions for 444.namd in Paraver.

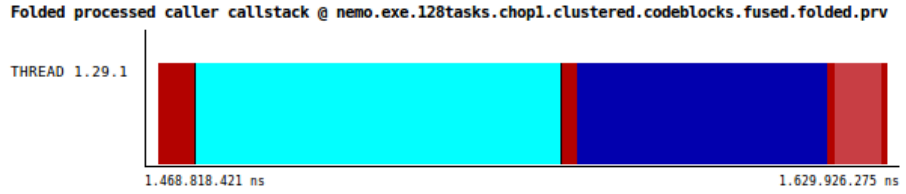

Figure 1.5: Paraver time-line showing the callers for Cluster 1 of Nemo.

integrated. This helps correlating the original trace-file and its contents with the folded trace-file. Notice that only one instance per type (where type refers to function, cluster, *etc...*) is folded.

- `win_folded_mips.cfg`
  Generates a time-line showing a signal of the MIPS rate within the folded instances. See in Figure 1.4 a time-line depicting the MIPS rate achieved in the example trace-file for 444.namd.
- `win_folded_processed_call-stack_caller.cfg`
  Generates a time-line showing the most time-dominant routines as they have been executed within the folded instances. Figure 1.5 shows a time-line depicting the called routines for the Nemo example.
- `win_folded_processed_call-stack_callerline.cfg`
  Generates a time-line showing the most time-dominant source code references (as pair of line and file) as they have been executed within the folded instances.

- *histograms/*
  - `3dh_folded_mips_per_caller.cfg`
    Generates a histogram that shows the achieved MIPS rate depending on the caller (columns) for a particular region folded.
  - `3dh_folded_mips_per_callerline.cfg`
    Generates a histogram that shows the achieved MIPS rate depending on the source code references (pair of line and file in columns) for a particular region folded.

1.2 Configuration, build and installation

This section describes how to build and install the Foldingpackage. The Foldingpackage (and its dependencies) requires the Boost library (only the headers suffice), a C compiler, a Fortran compiler and a C++ compiler that supports the C++ 2011 specification (such as g++ version 4.8). This package optionally uses the `strucchange` package from the R statistical application (and may execute in parallel if the `doParallel` is available) to use the piece-wise linear regression

interpolation mechanism. Additionally, the Foldingpackage requires the libtools package to be installed. This package helps on the parsing of Paraver trace-files and can be downloaded from the BSC download web page.

1.2.1 Libtools package

This package can be downloaded from the BSC web page and requires the boost header files[6]. If the boost header files are located in the system's default, simply run the following command:

```
# ./configure --prefix=/home/harald/aplic/libtools/1.0 \
  && make && make install
```

where `--prefix` indicates the destination folder for this package.

If the boost header files are located elsewhere in the system, run the following command:

```
# ./configure --prefix=/home/harald/aplic/libtools/1.0 \
  --with-boost=/path/to/boost \
  && make && make install
```

1.2.2 Folding package

The most basic configuration for the Foldingpackage honors the following commands:

```
# ./configure --with-libtools=$HOME/aplic/libtools/1.0 \
  --prefix=$HOME/aplic/folding/1.0rc8 && \
  make && make install
```

where `--with-libtools` refers to the location of the libtools package installed in Section 1.2.1 and `--prefix` indicates where to install the Foldingtool. If the compilation and installation succeed, the contents of the target installation should look like as the contents defined in Section 1.1.2.

The Foldingtool supports several compilation flags that modify the behavior or enable additional functionalities of the tool. The following list groups the flags according to the behavior they enable.

- `--with-clustering-suite=<DIR>`
  The Foldingtool relies on the similarity between the folded instances in order to generate its results. By default, the Foldingtool includes two mechanisms to reduce the noise that appear from using instances with significant different behavior. However, this flag allows using the BSC Clustering suite as a third alternative in order to reduce the noise.

- `--with-R=<DIR1>, --with-cube=<DIR2>, --with-clang=<DIR3>`
  Enables the usage of piece-wise linear regressions on top of the strucchange package[7] from the R statistical application[8]. This functionality requires the clang compiler[9] and can generate input files for the Cube3 performance analysis package[10].

[6] The libtools package has been successfully tested against version from 1.48 to 1.54.
[7] `http://cran.r-project.org/web/packages/strucchange/index.html`
[8] `http://www.r-project.org`
[9] `http://clang.llvm.org`
[10] `http://www.scalasca.org/software/cube-3.x/download.html`

- `--with-boost=<DIR>`
  This flag lets the Foldingto use a given Boost installation package.

- `--enable-gui`
  The results of the Foldingtool is a set of gnuplot files that have to explored manually. If this flag is given at the configure step, the Foldingpackage would include a GUI written in Python that helps exploring all the results from the tool.

- `--enable-callstack-analysis`
  This flag enables the call-stack analysis of the segments captured during the measurement step. Enabling this option results in gnuplot files that depict the performance progression collocated with the source code progression.

- `--enable-reference-analysis`
  This flag enables the memory references analysis of the references captured during the measurement step (currently, through the `perf` system tool). Enabling this option results in gnuplot files that depict the performance progression collocated with the memory address space and the sampled references.

Chapter 2

Generate a trace-file for the Folding

This chapter covers the minimum and necessary steps so as to configure Extrae[1] in order to use its resulting trace-files for the Folding process. There are three requirements when monitoring an application with Extraein order to take the most benefit from the Foldingtool. First, it is necessary to enable the sampling mechanism in addition to the instrumentation mechanism (see Section 2.1). Second, it is convenient to collect the appropriate performance counters for the underlying processor (see Section 2.2). Finally, Extraeneeds to capture a segment of the call-stack in order to allow the Foldingto provide information regarding the progression of the executed routines. The forthcoming sections provide information on how to enable these functionalities through the XML tags for the Extraeconfiguration file.

2.1 Enabling the sampling mechanism

Extraeis an instrumentation package that by default collects information from different parallel runtimes, including but not limited to: MPI, OpenMP, pthreads, CUDA and OpenCL (and even combinations of them). Extraecan be configured so that it also uses sampling mechanisms to capture performance metrics on a periodic basis. There are currently two alternatives to enable sampling in Extrae: using alarm signals and using performance counters. For the sake of simplicity, this document only covers the alarm-based sampling. However, if the reader would like to enable the sampling using the performance counters they must look at section 4.9 in the ExtraeUser's Manual for more details.

Listing 2.1: Enable default time-based sampling in Extrae.

```
1 <sampling enabled="yes" type="default" period="50m" variability="10m"/>
```

The XML statements in Listing 2.1 need to be included in the Extraeconfiguration file. These statements indicate Extraethat sampling is enabled (`enabled="yes"`). They also tell Extraeto capture samples every 50 milliseconds (ms) with a random variability of 10 ms, that means that samples will be randomly collected with a periodicity of 50±10 ms. With respect to type, it determines which timer domain is used (see `man 2 setitimer` or `man 3p setitimer` for further information on time domains). Available options are: `real` (which is also the `default` value, `virtual` and `prof` (which use the SIGALRM, SIGVTALRM and SIGPROF respectively). The default timing accumulates real time, but only issues samples at master thread. To let all the threads collect samples, the type must be set to either `virtual` or `prof`.

[1]Please refer to `http://www.bsc.es/computer-sciences/performance-tools/documentation` for the latest ExtraeUser's Guide.

Additionally, the Foldingmechanism is able to combine several performance models and generate summarized results that simplify understanding the behavior of the node-level performance. Since these performance models are heavily-tighted with the performance counters available on each processor architecture and family, the following sections provide ExtraeXML configuration files ready to use on several architectures. Since each architecture has different characteristics, the user may need to tune the XML presented there to make sure that all the list performance counters are gathered appropriately.

2.2 Collecting the appropriate performance counters

The Foldingmechanism provides, among other type of information, the progression of performance metrics along a delimited region through instrumentation points. These performance metrics include the progression of performance counters of every performance counter by default. To generate these kind of reports, Extraemust collect the performance counters during the application execution and this is achieved by defining counter sets into the `<counters>` section of the Extraeconfiguration file (see Section 4.19 of the ExtraeUser's guide for more information).

There has been research that has developed some performance models based on performance counters ratios among performance counters in order to ease the analysis of the reports. Each of these performance models aims at providing insight of different aspects of the application and system during the execution. Since the availability of the performance counters changes from processor to processor (even in the same processor family), the following sections describe the performance counters that are meant to be collected in order to calculate these performance models. These sections include the minimal `<counters>` sections to be added in a previously existing Extraeconfiguration file, but the Foldingpackage also includes full Extraeconfiguration files in `${FOLDING_HOME}/etc/extrae-configurations`.

2.2.1 Intel Haswell processors

Listing 2.2: Counter definition sets for the Extrae configuration file when used on Intel Haswell procesors.

```
<cpu enabled="yes" starting-set-distribution="cyclic">
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM,PAPI_L2_DCM
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L3_TCM,RESOURCE_STALLS:SB,RESOURCE_STALLS:ROB
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_SR_INS,PAPI_BR_CN,PAPI_BR_UCN
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_BR_MSP,PAPI_LD_INS
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,RESOURCE_STALLS,RESOURCE_STALLS:RS
  </set>
</cpu>
```

The listing 2.2 indicates Extraeto arrange five performance counter sets with performance

counters that are available on Intel Haswell processors. The collection of these performance counters allows the Foldingto apply the models contained in the ${FOLDING_HOME}/etc/models/intel-sandybr that include: instruction mix, architecture impact and stall distribution. Unfortunately, the PMU of the Intel Haswell processors do not count neither floating point nor vector instructions.

2.2.2 Intel SandyBridge processors

Listing 2.3: Counter definition sets for the Extrae configuration file when used on Intel SandyBridge procesors.

```
<cpu enabled="yes" starting-set-distribution="cyclic">
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM,PAPI_L2_DCM,PAPI_L3_TCM
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_BR_MSP,PAPI_BR_UCN,PAPI_BR_CN,RESOURCE_STALLS
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_VEC_DP,PAPI_VEC_SP,PAPI_FP_INS
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_LD_INS,PAPI_SR_INS
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,RESOURCE_STALLS:LOAD,RESOURCE_STALLS:STORE,
    RESOURCE_STALLS:ROB_FULL,RESOURCE_STALLS:RS_FULL
  </set>
</cpu>
```

The listing 2.3 indicates Extraeto configure five performance counter sets with performance counters that are available on Intel SandyBridge processors. The collection of these performance counters allows the Foldingto apply the models contained in the ${FOLDING_HOME}/etc/models/intel-sandybr that include: instruction mix, architecture impact and stall distribution.

2.2.3 Intel Nehalem processors

The listing 2.4 indicates Extraeto prepare three performance counter sets with performance counters that are available on Intel Nehalem processors. The collection of these performance counters allows the Foldingto apply the models contained in the ${FOLDING_HOME}/etc/models/intel-nehalem that include: instruction mix, architecture impact and stall distribution.

2.2.4 IBM Power8 processors

The listing 2.5 indicates Extraeto arrange six performance counter sets with performance counters that are available on IBM Power8 (and similar) processors. The collection of these performance counters allows the Foldingto calculate the CPIStack model for the IBM Power8 processor which is contained in ${FOLDING_HOME}/etc/models/ibm-power8.

2.2.5 IBM Power7 processors

The listing 2.6 indicates Extraeto prepare six performance counter sets with performance counters that are available on IBM Power7 (and similar) processors. The collection of these perfor-

Listing 2.4: Counter definition sets for the Extrae configuration file when used on Intel Nehalem procesors.

```
<cpu enabled="yes" starting-set-distribution="cyclic">
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM,PAPI_L2_DCM,PAPI_L3_TCM,
    RESOURCE_STALLS:SB,RESOURCE_STALLS:ROB
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_SR_INS,PAPI_BR_CN,PAPI_BR_UCN,PAPI_BR_MSP,
    PAPI_FP_INS,RESOURCE_STALLS:LB
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_LD_INS,PAPI_VEC_SP,PAPI_VEC_DP,
    RESOURCE_STALLS,RESOURCE_STALLS:RS
  </set>
</cpu>
```

mance counters allows the Foldingto calculate the CPIStack model for the IBM Power7 processor which is contained in `${FOLDING_HOME}/etc/models/ibm-power7`.

2.2.6 IBM Power5 processors

The listing 2.7 indicates Extraeto configure six performance counter sets with performance counters that are available on IBM Power5 (and similar) processors. The collection of these performance counters allows the Foldingto calculate the CPIStack model for the IBM Power5 processor which is contained in `${FOLDING_HOME}/etc/models/ibm-power5`.

2.2.7 Other architectures

The previous definitions of counter sets included performance counters that are available on the specific stated machines. Since these performance counters may not be available on all the systems, the package also provides a group of counter sets that may be available on a variety of systems. Listing 2.8 defines three Extraecounter sets that may be available on many systems (caveat here, not all systems may provide them). With the use of these counter sets, the Foldingcan apply the models contained in the `${FOLDING_HOME}/etc/models/basic` that include: instruction mix and architecture impact.

2.3 Capturing the call-stack at sample points

By default, the sampling mechanism captures the performance counters indicated in the `counters` section and the Program Counter interrupted at the sample point. The Foldingprovides the instantaneous progression of the routines that last at least a minimum given duration. To enable this type of analysis, it is necessary to instruct Extraeto capture a portion of the call-stack during its execution. Listing 2.9 shows how to enable the collection of the call-stack at the sample points in the Extraeconfiguration file. The mandatory lines to capture the call-stack at sample points are lines 1 and 4. Line 1 indicates that this section must be processed and Line 4 tells Extraeto capture levels 1 to 5 from the call-stack (where 1 refers to the level below to the top of the call-stack).

Listing 2.5: Counter definition sets for the Extrae configuration file when used on IBM Power8 procesors.

```
<cpu enabled="yes" starting-set-distribution="cyclic">
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL,PM_CMPLU_STALL_DCACHE_MISS,
    PM_CMPLU_STALL_THRD,PM_GRP_CMPL
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_BRU,PM_GCT_NOSLOT_CYC,
    PM_CMPLU_STALL_FXU
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_SCALAR,PM_CMPLU_STALL_LSU
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_STORE,PM_CMPLU_STALL_DMISS_L3MISS
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_VECTOR,PM_CMPLU_STALL_REJECT
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_DMISS_L2L3
  </set>
</cpu>
```

Listing 2.6: Counter definition sets for the Extrae configuration file when used on IBM Power7 procesors.

```
<cpu enabled="yes" starting-set-distribution="cyclic">
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL,PM_CMPLU_STALL_DCACHE_MISS,
    PM_CMPLU_STALL_THRD,PM_GRP_CMPL
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_DFU,PM_CMPLU_STALL_IFU,
    PM_GCT_NOSLOT_CYC
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_FXU,PM_CMPLU_STALL_SCALAR
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_LSU
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_STORE
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_RUN_INST_CMPL,PM_RUN_CYC,PM_CMPLU_STALL_VECTOR
  </set>
</cpu>
```

Listing 2.7: Counter definition sets for the Extrae configuration file when used on IBM Power5 procesors.

```
<cpu enabled="yes" starting-set-distribution="cyclic">
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_INST_CMPL,PM_CYC,PM_GCT_EMPTY_CYC,PM_LSU_LMQ_SRQ_EMPTY_CYC,
    PM_HV_CYC,PM_1PLUS_PPC_CMPL,PM_GRP_CMPL,PM_TB_BIT_TRANS
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_INST_CMPL,PM_CYC,PM_FLUSH_BR_MPRED,PM_BR_MPRED_TA,
    PM_GCT_EMPTY_IC_MISS,PM_GCT_EMPTY_BR_MPRED,PM_L1_WRITE_CYC
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_INST_CMPL,PM_CYC,PM_LSU_FLUSH,PM_FLUSH_LSU_BR_MPRED,PM_CMPLU_STALL_LSU,
    PM_CMPLU_STALL_ERAT_MISS
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_INST_CMPL,PM_CYC,PM_GCT_EMPTY_SRQ_FULL,PM_FXU_FIN,PM_FPU_FIN,
    PM_CMPLU_STALL_FXU,PM_FXU_BUSY,PM_CMPLU_STALL_DIV
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_INST_CMPL,PM_CYC,PM_IOPS_CMPL,PM_CMPLU_STALL_FDIV,PM_FPU_FSQRT,
    PM_CMPLU_STALL_FPU,PM_FPU_FDIV,PM_FPU_FMA
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PM_INST_CMPL,PM_CYC,PM_CMPLU_STALL_OTHER,PM_CMPLU_STALL_DCACHE_MISS,
    PM_LSU_DERAT_MISS,PM_CMPLU_STALL_REJECT,PM_LD_MISS_L1,PM_LD_REF_L1
  </set>
</cpu>
```

Listing 2.8: Basic counter definition sets for other processors not stated before.

```
<cpu enabled="yes" starting-set-distribution="cyclic">
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM,PAPI_L2_DCM,PAPI_L3_TCM
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_BR_CN,PAPI_BR_UCN,PAPI_LD_INS,PAPI_SR_INS
  </set>
  <set enabled="yes" domain="all" changeat-time="500000us">
    PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_VEC_SP,PAPI_VEC_DP,PAPI_FP_INS,PAPI_BR_MSP
  </set>
</cpu>
```

Listing 2.9: Collect call-stack information at sample points.

```
<callers enabled="yes">
  <mpi enabled="yes">1-3</mpi>
  <pacx enabled="no">1-3</pacx>
  <sampling enabled="yes">1-5</sampling>
</callers>
```

Chapter 3

Tool design

While the end user executes a single command to apply the Foldingtool, this command hides two major components that are executed sequentially and all the outputs are generated into a newly created directory with the name of the input trace-file. The first component processes a user-given trace-file that contains instrumented and sampled data and generates a textual file that contains sequences of instances and samples. The second component takes these sequences of instances and samples, then applies the contouring algorithm, any performance model, and the call-stack processing, and, finally, it generates the output results. Both components are grouped together within the `folding.sh` appearing to the user that the Foldingsimply consists of a single tool. The tool package contains additional components that may be capture the interest of the user.

3.1 First component: trace-file processing

The first component is divided into three phases that are executed one after another with the user-given trace-file and each of these parse the given trace-file and generates another trace-file that will be used in the subsequent phase as depicted in Figure 3.1. Each of these phases are built in a similar fashion. They parse the input trace-file and keep in memory information regarding the thread state, and eventually, add information to the output. The phases are:

1. `codeblocks` (found in `src/codeblocks`)
   This phase attributes to each sample information regarding to the loop / code region that it belongs to according to the application source code.

2. `fuse` (found in `src/fuse`)
   This phase compacts the trace-file and ensures that the resulting trace-file is well formed.

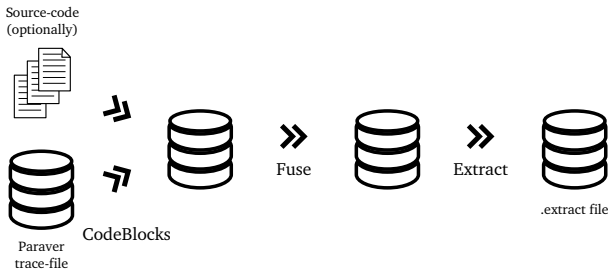

Figure 3.1: Data-flow for the first component of the folding.

3. `extract` (found in `src/extract`)
   This is the final phase and extracts information regarding the instances and samples within the trace-file.

The output of this component is a set of files containing information relative to the application. The most notable output is the `.extract` file, which contains the sequence of instances and their samples. For instance, Listing 3.1 shows the contents of the `.extract` file generated using the provided example to demonstrate the API facility. This listing contains information regarding one instance of the `FunctionA` region. The instance starts at timestamp 1,000 ns and lasts 4,500 ns, and it executes up to 2,500 instructions (PAPI_TOT_INS) and takes 5,000 cycles (PAPI_TOT_CYC) to complete. This instance has two samples associated that ocurred at timestamps 2,000 and 4,000, and each of those provides information regarding the aforementioned performance counters.

Listing 3.1: Output example for the extract phase of the Foldingmechanism.

```
1 I 2 2 2 FunctionA 1000 4500 2 PAPI_TOT_CYC 5000 PAPI_TOT_INS 2500
2 S 2000 1000 2 PAPI_TOT_CYC 2000 PAPI_TOT_INS 1000 0 0
3 S 4000 3000 2 PAPI_TOT_CYC 4000 PAPI_TOT_INS 2000 2 0 1 2 3 1 3 4 5 0
```

3.2 Second component: applying the folding

The main objective of this component relies on processing the instances and samples extracted and generate the output results. These results include the temporal evolution of the performance counters, any models requested by the user, the source code references and memory references progression, and the results are written in gnuplot and Paravertrace files. This section gives a summarized view of the folding work-flow by depicting the most notable class diagrams found in the application source code.

Figure 3.2 shows a portion of the classes that are most important within this tool. The classes *Instance* and *Sample* refer to the instances and samples as-is, without any further processing and as generated by the `extract` tool, in which each *Instance* contains a set of *Sample*, and every *Instance* belongs to an *InstanceContainer*.

After reading every *Instance*, the folding may apply a clustering algorithm (see Figure 3.3) according to the duration of each instance in order to reduce the difference between folded *Instance*. Currently, there are three alternatives regarding the grouping.

- *InstanceSeparatorNone* groups all instances into a single group.
- *InstanceSeparatorAuto* automatically groups the instances according to their duration. The grouping partitions the time-space interval defined by the shortest and longest instances and looks for group of nearby instances.
- *InstanceSeparatorDBSCAN* groups the instances according to a DBSCAN algorithm applied to the duration of the instances. The DBSCAN algorithm groups together instances that are closely packed together (instances with many nearby neighbors) in terms of time and marks as outliers those instances that lie alone in low-density regions. This grouping uses the ClusteringSuite implementation from the BSC performance tools[1].

[1] See `http://www.bsc.es/computer-sciences/performance-tools/downloads`.

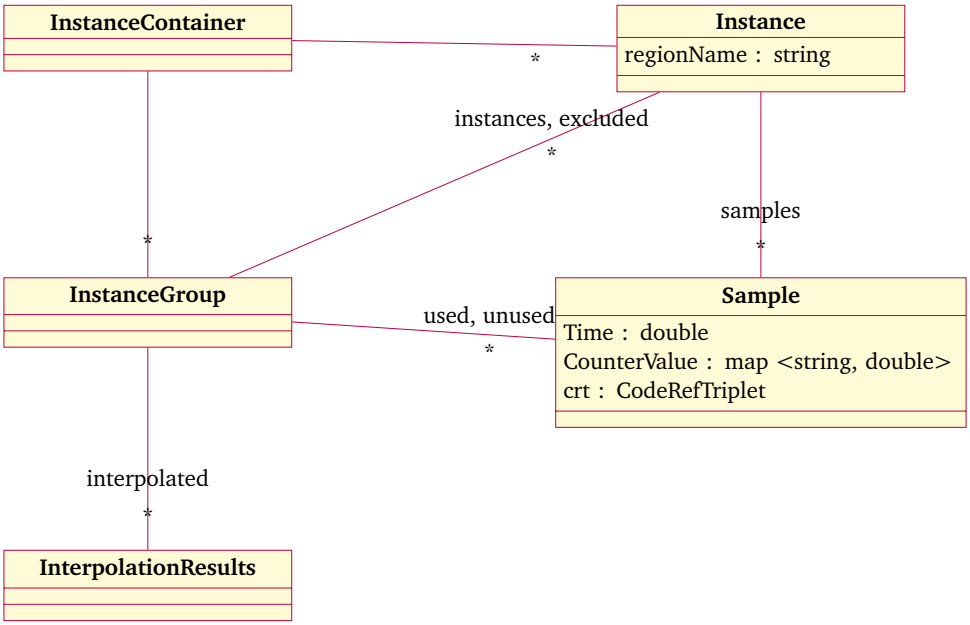

Figure 3.2: Main instances and samples related diagram classes.

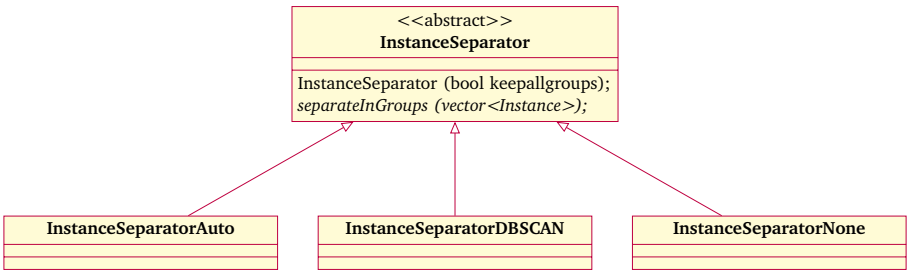

Figure 3.3: Instance selection related diagram classes.

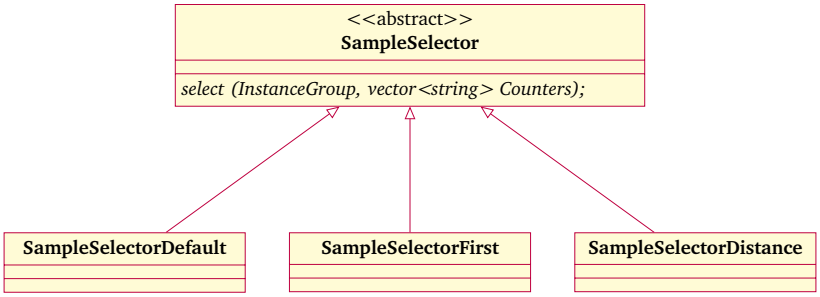

Figure 3.4: Sample selector related diagram classes.

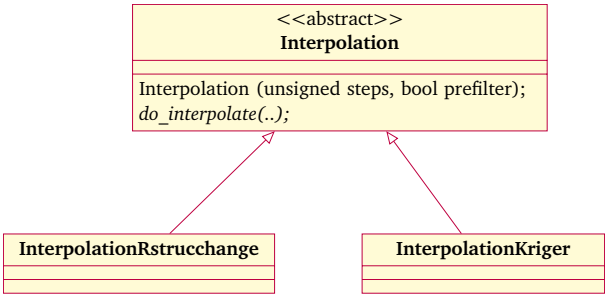

Figure 3.5: Performance counter interpolation related diagram classes.

This grouping begets the *InstanceGroup* objects which contains references to those *Instance* that belong to that particular group. Then, the folding removes the outliers to each *Instance* within every *InstanceGroup* and store the outliers and the remaining in the *excluded* and *instances* associations, respectively.

Since the complexity of the contouring algorithms depends on the number of points to connect, and therefore the number of samples to fold, the Foldingtool supports limiting the number of samples given to these algorithms. Figure 3.4 depicts the class diagram of the available *SampleSelector* mechanisms to limit the number of samples.

- *SampleSelectorDefault* the *select* method returns all of the samples within the *InstanceGroup*. This is useful when the user does not impose any limit to the number of samples to be folded.

- *SampleSelectorFirst* receives a threshold (N) in the class constructor. Then, the *select* method tags the first N samples for the processing while the rest are marked as *unused*.

- *SampleSelectorDistance* receives a threshold (N) in the class constructor. Then, the *select* method tags N samples that are equidistant within the *Instance* duration, while the rest are *unused*.

Then the Foldingrepeatedly applies the contouring algorithm to the *used* samples among the different *InstanceGroup* objects. The contouring algorithm applies to each performance counter individually, and as of writing this document, there are two approaches that honor the *Interpolation* super-class virtual method (mainly `do_interpolate`):

- *InterpolationKriger* uses the self-provided contouring algorithm based on the Kriging mechanism to implement the `do_interpolate`.

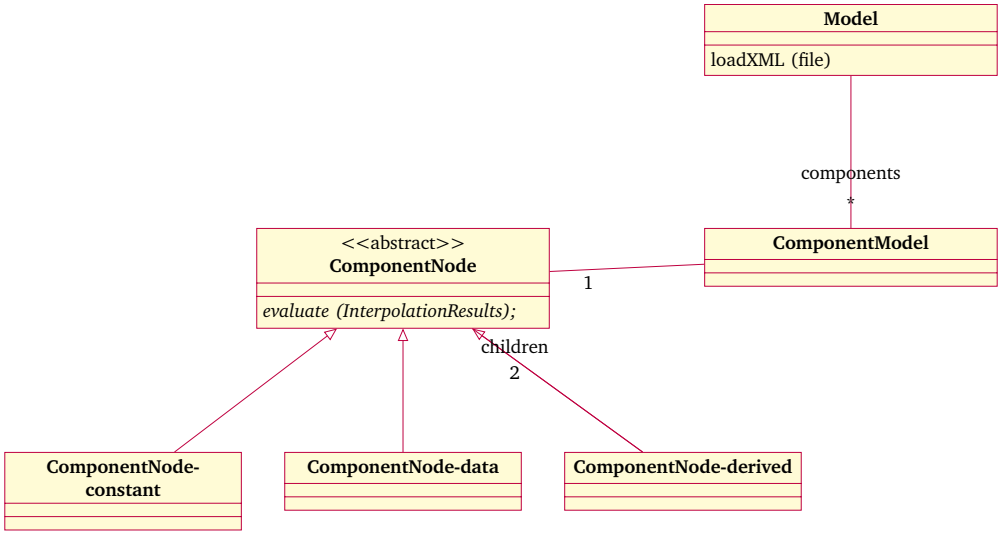

Figure 3.6: Performance models related diagram classes.

- *InterpolationRstrucchange* employs the strucchange package[2] from the R statistical package[3] to use piece-wise linear regressions to the folded samples. Additionally, this package may benefit from parallel environments if the doParallel package[4] is available on the system.

The interpolation results are stored, per performance counter, into *InterpolationResults* objects that are associated by *InstanceGroup* by the attribute *interpolated* (as depicted in Figure 3.2). The *interpolated* attribute is implemented as a hash function indexed by the performance counter, so that the interpolation results can be fetched easily.

The Foldingallows defining performance models based on performance counters using XML files (see Listing 3.2 for exemplification purposes and `${FOLDING_HOME}/etc/models` for more detailed examples). Within every XML there may be one or several components (in the last Listing these are: `l1_dcm_ratio`, `l2_dcm_ratio` and `mips`) that will be later represented in the resulting gnuplot using the selected colors and Y-axis (left [y1] or right [y2]). Each component may refer to the instantaneous value of a certain performance counter (as in the `mips` component), a constant value or the operation (addition, subtraction, multiplication and division) between two other values (as in `l1_dcm_ratio` and `l2_dcm_ratio` components). The Foldingimplements the performance models based on performance counters employing the diagram classes show in Figure 3.6. The XML model files are loaded into the *Model* class and each of them may contain multiple components (*ComponentModel*). The *ComponentModel* implements the definition of the component on top of the *ComponentNode* derived sub-classes. These sub-classes allow referencing constant values (*ComponentNode_constant*), interpolated results from a specific performance counter (*ComponentNode_data*) and operation between other two *ComponentNode* objects).

With respect to the analysis of the call-stack, the Foldingtool has implemented this analysis through the *CallstackProcessor* related-classes that receives a set of *Sample* objects to explore. Currently, the unique implementation available relies on aligning the call-stacks from the given samples and then exploring the call-stack frames at a given level whether consecutive samples refer to the same routine. If the number of samples surpasses a given threshold, then applies it

[2] `http://cran.r-project.org/web/packages/strucchange/index.html`
[3] `http://www.r-project.org`
[4] `http://cran.r-project.org/package=doParallel`

Listing 3.2: Folding example model that generates the L1D and L2D misses per instruction, in addition to the MIPS rate

```
<?xml version='1.0'?>

<model name="sample" title-name="Sample␣model"
       y1="Ratios" y2="MIPS" y1-stacked="no">

    <component name="l1_dcm_ratio" title-name="L1␣DCM" where="y1"
               color="red">
        <operation type='/'>
            <value> PAPI_L1_DCM </value>
            <value> PAPI_TOT_INS </value>
        </operation>
    </component>

    <component name="l2_dcm_ratio" title-name="L2␣DCM" where="y1"
               color="blue">
        <operation type='/'>
            <value> PAPI_L2_DCM </value>
            <value> PAPI_TOT_INS </value>
        </operation>
    </component>

    <component name="mips" title-name="MIPS" where="y2" color="black">
        <value> PAPI_TOT_INS </value>
    </component>

</model>
```

recursively to the next level until no more levels are available or the number of samples do not surpass the threshold.

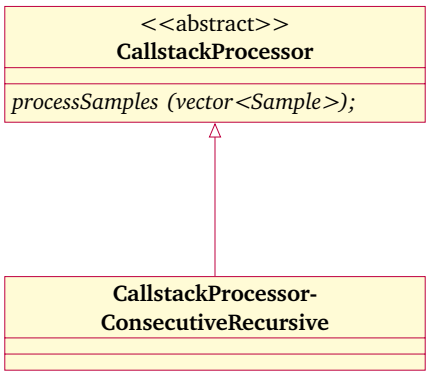

Figure 3.7: Call-stack processing related diagram classes.

Chapter 4

API

This section covers the public API available in the Foldingpackage. This API is meant to allow the Foldingtool to interact with other performance analysis tools in addition to Extrae.

4.1 Generation of input files for the Folding

4.1.1 Usage example

The directory `$FOLDING_HOME/share/examples/folding-writer` contains an example that shows how to generate an input file for the folding from a programatically point of view. The example can be compiled using the following command:

```
# cd $FOLDING_HOME/share/examples/folding-writer
# make
```

The Listing 4.1 shows the example provided in the distributed/installed package. This example demonstrates how to programatically create an `.extract` file for the `interpolate` binary of the Foldingpackage.

Listing 4.1: Example of generating an input file for the Foldingmechanism.

```
/*****************************************************************************\
 *                        ANALYSIS PERFORMANCE TOOLS                         *
 *                                  Folding                                  *
 *              Instrumentation package for parallel applications            *
 *****************************************************************************
 *     ___     This library is free software; you can redistribute it and/or *
 *    /  __          modify it under the terms of the GNU LGPL as published   *
 *   /  /  _____     by the Free Software Foundation; either version 2.1      *
 *  /  /  /     \    of the License, or (at your option) any later version.   *
 * (  (  ( B S C )                                                           *
 *  \  \  \_____/   This library is distributed in hope that it will be      *
 *   \  \__          useful but WITHOUT ANY WARRANTY; without even the        *
 *    \___           implied warranty of MERCHANTABILITY or FITNESS FOR A     *
 *                   PARTICULAR PURPOSE. See the GNU LGPL for more details.   *
 *                                                                           *
 * You should have received a copy of the GNU Lesser General Public License  *
 * along with this library; if not, write to the Free Software Foundation,   *
 * Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA          *
 * The GNU LEsser General Public License is contained in the file COPYING.   *
 *                                 ---------                                 *
 *   Barcelona Supercomputing Center - Centro Nacional de Supercomputacion   *
\*****************************************************************************/
```

```
23
24 #include "folding-writer.H"
25 #include <fstream>
26
27 using namespace std;
28
29 int main (int argc, char *argv[])
30 {
31     string nameRegion = "FunctionA";
32     unsigned long long startRegion = 1000;
33     unsigned long long durationRegion = 4500;
34
35     /* NOTE:: Counters are given in deltas from their previous read, not as absolute values *
36     map<string, unsigned long long> c1;
37     c1["PAPI_TOT_INS"] = 1000;
38     c1["PAPI_TOT_CYC"] = 2000;
39     map<unsigned, CodeRefTriplet> crt1;
40     Sample *s1 = new Sample (2000, 2000-startRegion, c1, crt1);
41
42     map<string, unsigned long long> c2;
43     c2["PAPI_TOT_INS"] = 1000;
44     c2["PAPI_TOT_CYC"] = 2000;
45     map<unsigned, CodeRefTriplet> crt2;
46     CodeRefTriplet codeinfo_l0 (1,2,3);
47     CodeRefTriplet codeinfo_l1 (3,4,5);
48     crt2[0] = codeinfo_l0;
49     crt2[1] = codeinfo_l1;
50     Sample *s2 = new Sample (4000, 4000-startRegion, c2, crt2);
51
52     /* Last sample typically coincides with end of region -- see durationRegion,
53        Folding::Write won't emit in a S entry */
54     map<string, unsigned long long> c3;
55     c3["PAPI_TOT_INS"] = 500;
56     c3["PAPI_TOT_CYC"] = 1000;
57     map<unsigned, CodeRefTriplet> crt3;
58     Sample *s3 = new Sample (4500, 4500-startRegion, c3, crt3);
59
60     vector<Sample*> vs;
61     vs.push_back (s1);
62     vs.push_back (s2);
63     vs.push_back (s3);
64
65     ofstream f("output.extract");
66     if (f.is_open())
67     {
68         FoldingWriter::Write (f, nameRegion, 1, 1, 1, startRegion,
69           durationRegion, vs);
70         f.close();
71     }
72
73     return 0;
74 }
```

The given example considers that the region `FunctionA` has been identified somehow by the underlying monitoring mechanism, starts at 1,000 ns and lasts 4,500 ns (lines 31-33). Within this period of time, three samples have occurred (*s1-s3*, created in lines 40, 50 and 58, respectively). Samples contain performance counter information and source code references. The performance counter information is given in a relative manner, thus each sample contains the difference from the previous sample (or starting point). For instance, sample *s1* captured information from two

performance counters (`PAPI_TOT_INS` and `PAPI_TOT_CYC`) that counted 1,000 and 2,000 events since the start of the region at time-stamp 2,000 ns (lines 36-40). The second sample (*s2*) does not only contain information from performance counters, but also contains a call-stack segment referencing two call-stack frames. The first frame (`codeinfo_l0`) refers to the routine coded as 1, which has source code information coded as 2, and AST-block information coded as 3 (line 46). The same applies to second frame (`codeinfo_l1`) - (line 47). These frames are mapped into depths 0 and 1 (where 0 refers to the top of the call-stack) in lines 48 and 49, and then the sample is built using the performance counter information and the call-stack information in line 50. Finally, the last sample (*s3*) only accounted 500 and 1,000 events for the `PAPI_TOT_INS` and `PAPI_TOT_CYC` performance counters respectively, but did not capture any source code reference (lines 54-58). This last sample should coincide with the end of the region (`FunctionA`), and may not be necessarily information captured from a sample point, but from an instrumentation point that indicates the end of the region. All these samples are packed together in a STL vector container (lines 60-63), and then the `FoldingWriter::Write` static method dumps all the information from the samples using the given output stream (lines 65-71).