cube
scalasca

CubeLib 4.4 | Tools Guide

Description of Cube Command Line Tools

August 2018

The Scalasca Development Team
scalasca@fz-juelich.de

JÜLICH
FORSCHUNGSZENTRUM

Attention

The Cube Tool Developer Guide is currently being rewritten and still incomplete. However, it should already contain enough information to get you started and avoid the most common pitfalls.

Contents

1 Cube Command Line Tools

1.1 Abstract

CUBE provides a set of various command line tools for different purposes.

1.2 Performance Algebra and Tools

As performance tuning of parallel applications usually involves multiple experiments to compare the effects of certain optimization strategies, CUBE offers a mechanism called *performance algebra* that can be used to merge, subtract, and average the data from different experiments and view the results in the form of a single “derived'' experiment. Using the same representation for derived experiments and original experiments provides access to the derived behavior based on familiar metaphors and tools in addition to an arbitrary and easy composition of operations. The algebra is an ideal tool to verify and locate performance improvements and degradations likewise. The algebra includes three operators—*diff*, *merge*, and *mean*—provided as command-line utilities which take two or more CUBE files as input and generate another CUBE file as output. The operations are closed in the sense that the operators can be applied to the results of previous operations. Note that although all operators are defined for any valid CUBE data sets, not all possible operations make actually sense. For example, whereas it can be very helpful to compare two versions of the same code, computing the difference between entirely different programs is unlikely to yield any useful results.

1.2.1 Difference

Changing a program can alter its performance behavior. Altering the performance behavior means that different results are achieved for different metrics. Some might increase while others might decrease. Some might rise in certain parts of the program only, while they drop off in other parts. Finding the reason for a gain or loss in overall performance often requires considering the performance change as a multidimensional structure. With CUBE's difference operator, a user can view this structure by computing the difference between two experiments and rendering the derived result experiment like an original one. The difference operator takes two experiments and computes a derived experiment whose severity function reflects the difference between the minuend's severity and the subtrahend's severity.

The possible output is presented below.

```
user@host: cube_diff scout.cube remapped.cube -o result.cube
```

```
Reading scout.cube ... done.
Reading remapped.cube ... done.
++++++++++++ Diff operation begins ++++++++++++++++++++++++++
INFO::Merging metric dimension... done.
INFO::Merging program dimension... done.
INFO::Merging system dimension... done.
INFO::Mapping severities... done.
INFO::Adding topologies...
        Topology retained in experiment.
 done.
INFO::Diff operation... done.
++++++++++++ Diff operation ends successfully +++++++++++++++++
Writing result.cube ... done.
```

Usage: cube_diff [-o output] [-c] [-C] [-h] minuend subtrahend

-o Name of the output file (default: diff.cube)

-c Do not collapse system dimension, if experiments are incompatible

-C Collapse system dimension

-h Help; Output a brief help message.

1.2.2 Merge

The merge operator's purpose is the integration of performance data from different sources. Often a certain combination of performance metrics cannot be measured during a single run. For example, certain combinations of hardware events cannot be counted simultaneously due to hardware resource limits. Or the combination of performance metrics requires using different monitoring tools that cannot be deployed during the same run. The merge operator takes an arbitrary number of CUBE experiments with a different or overlapping set of metrics and yields a derived CUBE experiment with a joint set of metrics.

The possible output is presented below.

```
user@host: cube_merge scout.cube remapped.cube -o result.cube
++++++++++++ Merge operation begins ++++++++++++++++++++++++++
Reading scout.cube ... done.
Reading remapped.cube ... done.
INFO::Merging metric dimension... done.
INFO::Merging program dimension... done.
INFO::Merging system dimension... done.
INFO::Mapping severities... done.
INFO::Merge operation...
        Topology retained in experiment.

        Topology retained in experiment.
 done.
++++++++++++ Merge operation ends successfully +++++++++++++++++
Writing result.cube ... done.
```

Usage: cube_merge [-o output] [-c] [-C] [-h] cube ...

-o Name of the output file (default: merge.cube)

-c Do not collapse system dimension, if experiments are incompatible

-C Collapse system dimension

-h Help; Output a brief help message.

1.2.3 Mean

The mean operator is intended to smooth the effects of random errors introduced by unrelated system activity during an experiment or to summarize across a range of execution parameters. You can conduct several experiments and create a single average experiment from the whole series. The mean operator takes an arbitrary number of arguments.

The possible output is presented below.

```
user@host: cube_mean scout1.cube scout2.cube scout3.cube scout4.cube -o mean.cube
++++++++++++ Mean operation begins ++++++++++++++++++++++++++
Reading scout1.cube ... done.
INFO::Merging metric dimension... done.
INFO::Merging program dimension... done.
INFO::Merging system dimension... done.
INFO::Mapping severities... done.
INFO::Adding topologies... done.
INFO::Mean operation... done.
Reading scout2.cube ... done.
INFO::Merging metric dimension... done.
INFO::Merging program dimension... done.
INFO::Merging system dimension... done.
INFO::Mapping severities... done.
INFO::Adding topologies... done.
INFO::Mean operation... done.
Reading scout3.cube ... done.
INFO::Merging metric dimension... done.
INFO::Merging program dimension... done.
INFO::Merging system dimension... done.
INFO::Mapping severities... done.
INFO::Adding topologies... done.
INFO::Mean operation... done.
Reading scout4.cube ... done.
INFO::Merging metric dimension... done.
INFO::Merging program dimension... done.
INFO::Merging system dimension... done.
INFO::Mapping severities... done.
INFO::Adding topologies... done.
INFO::Mean operation... done.
++++++++++++ Mean operation ends successfully +++++++++++++++++
Writing mean.cube ... done.
```

Usage: cube_mean [-o output] [-c] [-C] [-h cube ...

-o Name of the output file (default: mean.cube)

-c Do not collapse system dimension, if experiments are incompatible

-C Collapse system dimension

-h Help; Output a brief help message.

1.2.4 Compare

Compares two experiments and prints out if they are equal or not. Two experiments are equal if they have same dimensions hierarchy and the equal values of the severieties.

An example of the output is below.

```
user@host: cube_cmp remapped.cube scout1.cube
Reading remapped.cube ... done.
Reading scout1.cube ... done.
++++++++++++ Compare operation begins ++++++++++++++++++++++++++
Experiments are not equal.
++++++++++++ Compare operation ends successfully +++++++++++++++++
```

Usage: cube_cmp [-h] cube1 cube2

-h Help; Output a brief help message.

1.2.5 Clean

CUBE files may contain more data in the definition part than absolutely necessary. The `cube_clean` utility creates a new CUBE file with an identical structure as the input experiment, but with the definition part cleaned up.

An example of the output is presented below.

```
user@host: cube_clean remapped.cube -o cleaned.cube
++++++++++++ Clean operation begins ++++++++++++++++++++++++++
Reading remapped.cube ... done.

        Topology retained in experiment.
++++++++++++ Clean operation ends successfully +++++++++++++++++
Writing cleaned.cube ... done.
```

Usage: cube_clean [-o output] [-h cube

-o Name of the output file (default: clean.cube|.gz)

-h Help; Output a brief help message.

1.2.6 Reroot, Prune

For the detailed study of some part of the execution, the CUBE file can be modified based on a given call-tree node. Two different operations are possible:

- The call tree may be re-rooted, i.e., only sub-trees with the given call-tree node as root are retained in the experiment.
- An entire sub-tree may be pruned, i.e., removed from the experiment. In this case, all metric values for that sub-tree will be attributed to it's parent call-tree node (= “inlined'').

An example of the output is presented below.

```
user@host: cube_cut -r inner_auto_ -p flux_err_ -o cutted.cube remapped.cube
Reading remapped.cube ... done.
++++++++++++ Cut operation begins ++++++++++++++++++++++++++

        Topology retained in experiment.
++++++++++++ Cut operation ends successfully +++++++++++++++++
Writing cutted.cube ... done.
```

Usage: cube_cut [-h] [-r nodename] [-p nodename] [-o output cube

-o Name of the output file (default: cut.cube|.gz)

-r Re-root call tree at named node

-p Prune call tree from named node (= ''inline'')

-h Help; Output a brief help message.

1.2.7 Remap (version 2)

A more flixible implementation of the tool `cube_remap` is the `cube_remap2`.

This tool takes a remapping specification file as a command line argument and perform recalculation of the metric values according to the specified rules, expressed in CubePL syntax.

This tool can be used to convert all derived metrics into usual metrics, which are holding data (notice, that POSTDERIVED metrics became invalid while this conversion).

CUBE provides examplees of remapping specification files for SCOUT and Score-P. They are stored in the directory `[prefix]/share/doc/cube/examples`

Usage: ./cube_remap2 -r <remap specification file> [-o output] [-d] [-s] [-h] <cube experiment>

-r Name of the remapping specification file. By omitting this option the specification file from the cube experiment is taken if present.

-c Create output file with the same structure as an input file. It overrides option "-r"

-o Name of the output file (default: remap)

-d Convert all prederived metrics into usual metrics, calculate and store their values as a data.

-s Add hardcoded SCALSCA metrics "Idle threads" and "Limited parallelizm"

-h Help; Output a brief help message.

1.2.8 Statistics

Extracts statistical information from the CUBE files.

```
user@host: ./cube_stat  -m time,mpi -p remapped.cube -%
MetricRoutine     Count    Sum       Mean      Variance  Minimum      ...  Maximum
time INCL(MAIN__)    4  143.199101  35.799775  0.001783   35.759769   ...  35.839160
time   EXCL(MAIN__)  4    0.078037   0.019509  0.000441    0.001156   ...   0.037711
time   task_init_    4    0.568882   0.142221  0.001802    0.102174   ...   0.181852
time   read_input_   4    0.101781   0.025445  0.000622    0.000703   ...   0.051980
time   decomp_       4    0.000005   0.000001  0.000000    0.000001   ...   0.000002
time   inner_auto_   4  142.361593  35.590398  0.000609   35.566589   ...  35.612125
time   task_end_     4    0.088803   0.022201  0.000473    0.000468   ...   0.043699

mpi  INCL(MAIN__)    4   62.530811  15.632703  2.190396   13.607989   ...  17.162466
mpi    EXCL(MAIN__)  4    0.000000   0.000000  0.000000    0.000000   ...   0.000000
mpi    task_init_    4    0.304931   0.076233  0.001438    0.040472   ...   0.113223
mpi    read_input_   4    0.101017   0.025254  0.000633    0.000034   ...   0.051952
mpi    decomp_       4    0.000000   0.000000  0.000000    0.000000   ...   0.000000
pi    inner_auto_    4   62.037503  15.509376  2.194255   13.478049   ...  17.031288
mpi    task_end_     4    0.087360   0.021840  0.000473    0.000108   ...   0.043333
```

```
user@host: ./cube_stat  -t33 remapped.cube -p -m time,mpi,visits
Region        NumberOfCalls ExclusiveTime InclusiveTime      time       mpi      visits
sweep_                   48     76.438435    130.972847   76.438435  0.000000      48
MPI_Recv              39936     36.632249     36.632249   36.632249  36.632249   39936
MPI_Send              39936     17.684986     17.684986   17.684986  17.684986   39936
MPI_Allreduce           128      7.383530      7.383530    7.383530   7.383530     128
source_                  48      3.059890      3.059890    3.059890   0.000000      48
MPI_Barrier              12      0.382902      0.382902    0.382902   0.382902      12
flux_err_                48      0.380047      1.754759    0.380047   0.000000      48
TRACING                   8      0.251017      0.251017    0.251017   0.000000       8
MPI_Bcast                16      0.189381      0.189381    0.189381   0.189381      16
MPI_Init                  4      0.170402      0.419989    0.170402   0.170402       4
snd_real_             39936      0.139266     17.824251    0.139266   0.000000   39936
MPI_Finalize              4      0.087360      0.088790    0.087360   0.087360       4
initialize_               4      0.084858      0.168192    0.084858   0.000000       4
initxs_                   4      0.083242      0.083242    0.083242   0.000000       4
MAIN__                    4      0.078037    143.199101    0.078037   0.000000       4
rcv_real_             39936      0.077341     36.709590    0.077341   0.000000   39936
inner_                    4      0.034985    142.337220    0.034985   0.000000       4
inner_auto_               4      0.024373    142.361593    0.024373   0.000000       4
task_init_                4      0.014327      0.568882    0.014327   0.000000       4
read_input_               4      0.000716      0.101781    0.000716   0.000000       4
```

```
octant_                 416      0.000581      0.000581    0.000581   0.000000     416
global_real_max_         48      0.000441      1.374712    0.000441   0.000000      48
global_int_sum_          48      0.000298      5.978850    0.000298   0.000000      48
global_real_sum_         32      0.000108      0.030815    0.000108   0.000000      32
barrier_sync_            12      0.000105      0.383007    0.000105   0.000000      12
bcast_int_               12      0.000068      0.189395    0.000068   0.000000      12
timers                    2      0.000044      0.000044    0.000044   0.000000       2
initgeom_                 4      0.000042      0.000042    0.000042   0.000000       4
initsnc_                  4      0.000038      0.000050    0.000038   0.000000       4
task_end_                 4      0.000013      0.088803    0.000013   0.000000       4
bcast_real_               4      0.000010      0.000065    0.000010   0.000000       4
decomp_                   4      0.000005      0.000005    0.000005   0.000000       4
timers_                   2      0.000004      0.000048    0.000004   0.000000       2
```

Usage: cube_stat [-h] [-p] [-m metric[,metric...] -%] [-r routine[,routine...] cubefile

OR

cube_stat -h] [-p] [-m metric[,metric...] -t topN cubefile

-h Display this help message

-p Pretty-print statistics (instead of CSV output)

-% Provide statistics about process/thread metric values

-m List of metrics (default: time)

-r List of routines (default: main)

-t Number for topN regions flat profile

1.2.9 from TAU to CUBE

Converts a profile generated by the TAU Performance System into the CUBE format. Currently, only 1-level, 2-level and full call-path profiles are supported.

An example of the output is presented below.

```
user@host: ./tau2cube3 tau2 -o b.cube
    Parsing TAU profile...
    tau2/profile.0.0.2
    tau2/profile.1.0.0
    Parsing TAU profile...          done.
    Creating CUBE profile...
    Number of call paths : 5
    Childmain int (int, char **)
    Number of call paths : 5
    ChildsomeA void (void)
    Number of call paths : 5
    ChildsomeB void (void)
    Number of call paths : 5
    ChildsomeC void (void)
    Number of call paths : 5
    ChildsomeD void (void)
```

```
Path to Parents : 5
Path to Child : 1
Number of roots : 5
Call-tree node created
Call-tree node created
Call-tree node created
Call-tree node created
Call-tree node created
value time :: 8.0151
value ncalls :: 1
value time :: 11.0138
value ncalls :: 1
value time :: 8.01506
value ncalls :: 1
value time :: 11.0138
value ncalls :: 1
value time :: 5.00815
value ncalls :: 1
value time :: 11.0138
value ncalls :: 1
value time :: 0.000287
value ncalls :: 1
value time :: 11.0138
value ncalls :: 1
value time :: 0
value ncalls :: 0
value time :: 9.00879
value ncalls :: 1
done.
```

Usage: tau2cube [tau-profile-dir][-o cube]

1.2.10 Common Calltree

Common Calltree is a tool to reduce call trees of a set of cube files to the common part.

Usage: cube_commoncalltree cubefile1 cubefile2 ... cubefileN

The tool *cube_commoncalltree* takes set of input cubefiles

(*cubefile1 cubefile2 ... cubefileN*)

and creates corresponding set of cube files

(*cubefile1_commoncalltree cubefile2_commoncalltree ... cubefileN_commoncalltree*).

Output cube files *cubefileX_commoncalltree* do have the equal system and metric dimensions like corresponding *cubefileX* file.

Call trees among *cubefileX_commoncalltree* files are reduced to the maximal (up to a special case in region naming scheme) common part. Inclusive value of the "missing" part is added as a exclusive value to its parent (which is a part of common part of call tree)

This tool is particulary useful for comparison of exprerimens with the different recursion

depth or with the additional sub call trees depending on some loop iteration index.

1.2.11 Topology Assistant

Topology assistant is a tool to handle topologies in cube files. It is able to add or edit a topology.

Usage: cube_topoassist {**OPTION**} cubefile

The current available options are:

- To create a new topology in an existing cube file,
- To [re]name an existing virtual topology, and
- To [re]name the dimensions of a virtual topology.

The command-line switches for this utility are:

-c: creates a new topology in a given cube file.

-n: displays a numbered list of the existing topologies in the given cube file, and lets the user choose one to be named or renamed.

-d: displays the existing topologies, and lets the user name the dimensions of one of them.

The resulting CUBE file is named topo.cube[.gz], in the current directory.

As mentioned abot, when using the **-d** or **-n** command-line options, a numbered list of the current topologies will appear, showing the topology names, its dimension names (when existing), and the number of coordinates in each dimension, as well as the total number of threads. This is an example of the usage:

```
$ cube_topoassist topo.cube.gz -n
Reading topo.cube.gz . Please wait... Done.
Processes are ordered by rank. For more information about this file,
use cube_info -S <cube experiment>

This CUBE has 3 topologie(s).
0. <Unnamed topology>, 3 dimensions: x: 3, y: 1, z: 4. Total = 12 threads.
1. Test topology, 1 dimensions: dim_x: 12. Total = 12 threads.
2. <Unnamed topology>, 3 dimensions: 3, 1, 4. Total = 12 threads. <Dimensions are not named>


Topology to [re]name?
1
New name:
Hardware topology
Topology successfully [re]named.

Writing topo.cube.gz ... done.
```

The process is similar for [re]naming dimensions within a topology. One characteristic is that either all dimensions are named, or none.

One could easily create a script to generate the coordinates according to some algorithm/equation, and feed this to the assistant as an input. The only requirement is to answer the questions in the order they appear, and after that, feed the coordinates. Coordinates are asked for in rank order, and inside every rank, in thread order.

The sequence of questions made by the assistant when creating a new topology (the **-c** switch) is:

- New topology's name
- Number of dimensions
- Will the above dimensions be named? (Y/N)
- If yes, asks the name. Empty is not valid.
- Number of coordinates in that dimension
- Asks if this dimension is either periodic or not (Y/N)
- Repeat the previous three steps for every dimension
- After that, it expects the coordinates for each thread in this topology, separated by spaces, in the order described above.

This is a sample session of the assistant:

```
$ cube_topoassist -c experiment.cube.gz
Reading experiment.cube.gz. Please wait... Done.
Processes are ordered by rank. For more information about this file, use cube_info -S <cube experiment

So far, only cartesian topologies are accepted.
Name for new topology?
Test topology
Number of Dimensions?
3
Do you want to name the dimensions (axis) of this topology? (Y/N)
y
Name for dimension 0
torque
Number of elements for dimension 0
2000
Is dimension 0 periodic?
y
Name for dimension 1
rotation
Number of elements for dimension 1
1500
Is dimension 1 periodic?
n
Name for dimension 2
period
Number of elements for dimension 2
50
Is dimension 2 periodic?
n
Alert: The number of possible coordinates (150000000) is bigger than the number of threads
 on the specified cube file (12). Some positions will stay empty.
Topology on THREAD level.
```

```
Thread 0’s (rank 0) coordinates in 3 dimensions, separated by spaces
0 0 0
0 0 1
0 0 2
...
...
...
Writing topo.cube.gz ... done.
$
```

So, a possible input file for this cube experiment could be:

```
Test topology
3
y
torque
2000
y
rotation
1500
n
period
50
n
0 0 0
0 0 1
0 0 2
... (the remaining coordinates)
```

And then call the assistant: cube_topoassist -c cubefile.cube < input.txt

1.2.12 Dump

To export values from the cube report into another tool or to examine internal structure of the cube report CUBE framework provedes a tool `cube_dump` tool, which prints out different values. It calculates inclusive and exclusive values along metric tree and call tree, agregates over system tree or displays values for every thread separately. In addition it provides user to define new metrics(see file CubeDerivedMetrics.pdf). Results are calculated and shown. For convenience user can invoke defined metrics along with new once in any order. For doing so one lists unique names of metrics separated by commas. For access to more than one callpaths, user can specify the ids or a range of them like "2-9". This also can be done for threads. Additionally provides a calculation of the flat profile values.

-m <**metrics**>|<**new metrics**>|**all**|<**filename**> Select one or more of metrics (unique names) for data dump.

By giving a CubePL expression one can define one or more new metrics by giving correspond formula. If the expression prefixed with "\< name \>:", < name > is used as a unique name for the new metric.

<filename> - takes a CubePL expression from file <filename> and defines a derived metric with it

all - all metrics will be printed.

Combination of these three is possible.

-c <**cnode ids**>|**all**|**leafs**|**roots**|**level**>**X**|**level=X**|**level**<**X**|**name=/regexp/**|<**filename**>

Select one or more call paths to be printed out .

<cnode ids> - list or range of call path ids: 0,3,5-10,25

all - all call paths are printed

leafs - only call paths without children are printed

roots - only root cnodes are printed

level<X, level=X or level>X - only cnodes of the level more, equal or less than N are printed

name=/regexp/- only cnodes with the region name matching to the regular expression *regexp*

<filename> - takes a CubePL expression from file <filename> and evaluates it for every callpath.

If the result is non-zero - call path is included into the output.

-x incl|**excl** Selects, if the data along the metric tree should be calculated as an inclusive or an exclusive value.

(Default value: incl).

-z incl|**excl**|**stored** Selects, if the data along the call tree should be calculated as an inclusive or an exclusive value.

(Default value: excl).

-t <**thread id**>|**aggr** Show data for one or more selected threads or aggregated over system tree.

-r Prints aggregated values for every region (flat profile), sorted by id.

-f < **name** > Selects a stored data with the name < name > to display

-d Shows the coordinates for every topology as well.

-y Disables expansion of clusters and shows bare stored meta structure.

-w Prints out the information about structure of the cube.

-o <**filename**>|**-** Uses a device or STDOUT for the output. If omit, STDOUT is used.

-s human|**gnuplot**|**gnuplot2**|**csv**|**csv2**|**R** Uses either human readable form, GNUPLOT or CSV (two different layouts) format or binary R matrix for data export.

-h|**-?** Help; Output a brief help message.

This is examples of the usage.

Example 1:

```
$cube_dump -m time,"metric::visits(e)","metric::time(i)/metric::visits(e)" -c 0 \
  -z incl -s gnuplot  profile.cubex

# ====================== DATA ==================
# Print out the data of the metric time
#main(id=0)
0   0   80.549003343
0   1   44.115097986
0   2   43.486614165
0   3   43.940738098
0   4   80.539393011
0   5   42.723353088
0   6   42.61159706
0   7   43.108220977
0   8   80.635220741
0   9   43.788284208
0   10  43.831524441
0   11  43.652044759
0   12  80.629776666
0   13  42.692885677
0   14  42.719330066
0   15  42.732487708

# Print out the data of the metric New Metric1
#main(id=0)
0   0   80.549003343
0   1   1.79769313486e+308
0   2   1.79769313486e+308
0   3   1.79769313486e+308
0   4   80.539393011
0   5   1.79769313486e+308
0   6   1.79769313486e+308
0   7   1.79769313486e+308
0   8   80.635220741
0   9   1.79769313486e+308
0   10  1.79769313486e+308
0   11  1.79769313486e+308
0   12  80.629776666
0   13  1.79769313486e+308
0   14  1.79769313486e+308
0   15  1.79769313486e+308

# Print out the data of the metric New Metric2
#main(id=0)
0   0   1
0   1   0
0   2   0
0   3   0
0   4   1
0   5   0
0   6   0
0   7   0
0   8   1
```

```
0   9   0
0   10  0
0   11  0
0   12  1
0   13  0
0   14  0
0   15  0
```

Example 2:

```
$cube_dump -m time -s gnuplot2  profile.cubex

# ====================== CONFIGURE GNUPLOT ==================
set logscal x ; set logscal y ; set grid ; set xrange [16:300000]
; set terminal png size "600,400"

# ====================== DATA ==================
# Print out the data of the metric time

;set output "0.png"
; plot 6.4e-18*x**(5.0/2.0) + 5.2e-05 t "setPrecision(int, PrecisionFormat)(id=18)" ,
1.2e-17*x**(7.0/3.0) + 3.6e-05 t "setRoundNr(int, PrecisionFormat)(id=19)" ,
2.1e-17*x**(9.0/4.0) + 3.1e-05 t "setUpperExpNr(int, PrecisionFormat)(id=20)" ,
7.1e-19*x**(5.0/2.0) + 8.8e-06 t "getInstance()(id=21)" ,
2.8e-18*x**(2.0/1.0)*log(x) + 3.3e-06 t "getCubePluginCount()(id=24)"

;set output "1.png"
; plot 1.3e-20*x**(5.0/2.0)*log(x) + 4.2e-06 t "PluginList()(id=25)" ,
9.7e-18*x**(7.0/3.0) + 1.1e-05 t "loadContextFreePlugin(PluginData&)(id=28)" ,
1e-17*x**(9.0/4.0) + 8.8e-06 t "loadCubePlugin(PluginData&)(id=29)" ,
1e-18*x**(9.0/4.0) + 1.1e-06 t "name() const(id=35)" ,
2.6e-18*x**(2.0/1.0)*log(x) + 3.7e-06 t "getCubePlugin(int)(id=44)"
```

Example 3:

```
$cube_dump -m time,"metric::visits(e)","metric::time(i)/metric::visits(e)" -c 0 \
 -t aggr  -z incl -s human  profile.cubex

 ====================== DATA ==================
Print out the data of the metric time

            All threads
-------------------------------------------------------------------------------
main(id=0)  841.755571994
Print out the data of the metric New Metric1

            All threads
-------------------------------------------------------------------------------
main(id=0)  210.438892999
Print out the data of the metric New Metric2
```

```
            All threads
-------------------------------------------------------------------------------
main(id=0)  4
```

Example 4:

```
$cube_dump -m time,"metric::visits(e)","metric::time(i)/metric::visits(e)" -c 20 \
  -z incl -s csv  profile.cubex

20,0,8.9812e-05
20,1,0
20,2,0
20,3,0
20,4,9.7463e-05
20,5,0
20,6,0
20,7,0
20,8,0.000132327
20,9,0
20,10,0
20,11,0
20,12,7.2788e-05
20,13,0
20,14,0
20,15,0

20,0,8.9812e-05
20,1,-1.79769313486e+308
20,2,-1.79769313486e+308
20,3,-1.79769313486e+308
20,4,9.7463e-05
20,5,-1.79769313486e+308
20,6,-1.79769313486e+308
20,7,-1.79769313486e+308
20,8,0.000132327
20,9,-1.79769313486e+308
20,10,-1.79769313486e+308
20,11,-1.79769313486e+308
20,12,7.2788e-05
20,13,-1.79769313486e+308
20,14,-1.79769313486e+308
20,15,-1.79769313486e+308

20,0,1
20,1,0
20,2,0
20,3,0
20,4,1
20,5,0
20,6,0
20,7,0
20,8,1
20,9,0
```

```
20,10,0
20,11,0
20,12,1
20,13,0
20,14,0
20,15,0
```

Example 5:

```
$cube_dump -m time,"metric::visits(e)","metric::time(i)/metric::visits(e)" -c 1 \
  -z incl -s csv2  profile.cubex

Cnode ID, Thread ID,time,New Metric1,New Metric2
1,0,80.548967177,80.548967177,1
1,1,44.115097986,1.79769313486e+308,0
1,2,43.486614165,1.79769313486e+308,0
1,3,43.940738098,1.79769313486e+308,0
1,4,80.539359524,80.539359524,1
1,5,42.723353088,1.79769313486e+308,0
1,6,42.61159706,1.79769313486e+308,0
1,7,43.108220977,1.79769313486e+308,0
1,8,80.635176341,80.635176341,1
1,9,43.788284208,1.79769313486e+308,0
1,10,43.831524441,1.79769313486e+308,0
1,11,43.652044759,1.79769313486e+308,0
1,12,80.629742485,80.629742485,1
1,13,42.692885677,1.79769313486e+308,0
1,14,42.719330066,1.79769313486e+308,0
1,15,42.732487708,1.79769313486e+308,0
```

Example 6:

```
$cube_dump -m time,"metric::visits(e)","metric::time(i)/metric::visits(e)" -c 1 \
  -z incl -s R  profile.cubex -o output_file
```

This will generate binary file "output_file" which can be loaded in R. In consists of three matrices, each one corresponding to one metric. Each matrix is named after the metric and it contains values for all threads and nodes.

Example 7: We select only call path names, starting with "main" using the CubePL expression (stored in file name "selection.cubepl") :

```
{
${a}=0;
if (${cube::region::name}[${calculation::region::id}] =~  /^main/ )
{ ${a}=1; }; return ${a};
}
```

Then:

```
cube_dump -m time,"metric::visits(e)","metric::time(i)/metric::visits(e)"   -c selection.cubepl  \
 -z incl  -t aggr  profile.cubex

 ====================== DATA ==================
Print out the data of the metric time
                  All threads
-------------------------------------------------------------------------------
main(id=0)        841.755571994
main_loop(id=12)  840.73706946

Print out the data of the metric New Metric1
                  All threads
-------------------------------------------------------------------------------
main(id=0)        210.438892999
main_loop(id=12)  0.210184267365

Print out the data of the metric New Metric2
                  All threads
-------------------------------------------------------------------------------
main(id=0)        4
main_loop(id=12)  4000
```

Options "leafs", "roots", "level=", "level<", "level>" and "name=/regexp/" are shortcuts for a build-in CubePL expression, which is used to select a call path.

- **"leafs"** : stands for:

```
{
${a}=0;
if (${cube::callpath::#children}[${calculation::callpath::id}] == 0 )
{ ${a}=1; }; return ${a};
}
```

- **"roots"** : stands for:

```
{
${a}=0;
if (${cube::callpath::parent::id}[${calculation::callpath::id}] == -1 )
{ ${a}=1; }; return ${a};
}
```

- **"level=N"** : stands for:

```
{
${level}=N;
${index}=0;
${i}=${calculation::callpath::id};
while (${cube::callpath::parent::id}[${i}] != -1 )
{ ${i}= ${cube::callpath::parent::id}[${i}]; ${index}=${index}+1; };
${a}=0;
if (${index} ==  ${level})
{${a}=1; };
return ${a};
}
```

- **"name=/regexp/"** : stands for:

```
{
  ${a}=0;
   if ( ${cube::region::name}[${calculation::region::id}] =~  /regexp/ )
  { ${a}=1; }; return ${a};
};
```

"level<N" and "level>N" differ from "level=N" in the boolean operation in the line 8. For detailed documentation of the syntax of CubePL please see.

www.scalasca.org