

CubeLib 4.9 | Tools Guide

Description of Cube Command Line Tools

March 2025
The Scalasca Development Team
scalasca@fz-juelich.de

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	3
1.1	Abstract	3
1.2	Performance Algebra and Tools	3
1.2.1	Difference	3
1.2.2	Merge	4
1.2.3	Mean	5
1.2.4	Compare	6
1.3	POP Metrics calculation tool	6
1.3.1	Tool Usage	7
1.3.2	Only MPI analysis	7
1.3.2.1	Only-MPI POP Metrics	8
1.3.3	Hybrid Additive POP analysis	9
1.3.3.1	Hybrid Additive POP Metrics	9
1.3.4	Hybrid Multiplicative POP analysis	11
1.3.4.1	Hybrid Additive POP Metrics	11
1.3.5	Hybrid BSC POP analysis	13
1.3.5.1	Hybrid BSC POP Metrics	14
1.3.6	Additional metics	15
1.3.6.1	I/O Fficiencies	15
1.3.6.2	Additional Fficiencies	16
1.3.6.3	Control Metrics	16
1.4	Cube Remapping tool	17
1.4.1	Tool Usage	17
1.4.2	Remapping algorithm	17
1.4.3	Syntax of a .spec file	18
1.4.4	Optimization of the remapping.	21
1.5	Supplementary utilities and helping Cube command line tools	21
1.5.1	Cube Server cube_server	21
1.5.2	Sanity Check	22
1.5.3	Insertion of of derived metric into the .cubex	23
1.5.4	Inspection of a call tree	23
1.5.5	System tree partition	24
1.5.6	Region information	25
1.5.7	Cube Information	32
1.5.8	Tester	35
1.5.9	Canonizer	35
1.5.10	Clean	36
1.5.11	Reroot, Prune	37
1.5.12	Statistics	38
1.5.13	from TAU to CUBE	39
1.5.14	Common Calltree	40

Contents

1.5.15Topology Assistant	41
1.5.16Dump	44
Bibliography	53

Copyright © 1998–2024 Forschungszentrum Jülich GmbH, Germany

Copyright © 2009–2015 German Research School for Simulation Sciences GmbH,
Jülich/Aachen, Germany

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of Forschungszentrum Jülich GmbH or German Research School for Simulation Sciences GmbH, Jülich/Aachen, nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

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 Reduce system dimension, if experiments are incompatible.
- C Collapse system dimension! Overrides option -c.
- 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 experiment>

- o Name of the output file (default: merge.cube)
- c Reduce 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 experiment>

- 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 severities.

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] <cube experiment 1> <cube experiment 2>

- h Help; Output a brief help message.

1.3 POP Metrics calculation tool

CUBE provides a command line tool **cube_pop_metrics** to calculate various POP metrics. Attempting to optimize the performance of a parallel code can be a daunting task, and often it is difficult to know where to start. For example, we might ask if the way computational work is divided is a problem? Or perhaps the chosen communication scheme is inefficient? Or does something else impact performance? To help address this issue, POP has defined a methodology for analysis of parallel codes to provide a quantitative way of measuring relative impact of the different factors inherent in parallelization. This article introduces these metrics, explains their meaning, and provides insight into the thinking behind them.

Various POP models are supported:

- Only-MPI - analysis of the code using only MPI;
- Hybrid Additive - analysis of the code using hybrid parallelization MPI with OpenMP, where different metrics are related to each other as a sum;
- Hybrid Multiplicative - analysis of the code using hybrid parallelization MPI with OpenMP, where different metrics are related to each other as a product;
- Hybrid BSC - analysis of the code using hybrid parallelization MPI with "non-MPI" paradigm, developed in Barcelona Supercomputing Center;

1.3.1 Tool Usage

Usage: `./cube_pop_metrics -r [-i] [-a <pop analysis>] [-c <FOI(s)>] [-h] [-? <pop analysis>] <cube1> <cube2> ...`

- a** Name of the POP analysis. Options: mpi,hybrid-add, hybrid-mult, bsc. Default: mpi
- c** Name(s) or ID(s) of FOI(s). Default: calltree root(s). Comma separated
- i** Performs the calculation individually for every listed FOI. Default: if omit, the calculation is performed in accumulated manner
- h** Help; Output a brief help message.
- ?** Help; Output a long detailed help message.

1.3.2 Only MPI analysis

A feature of the methodology is, that it uses a hierarchy of Only-MPI Assessment, each metric reflecting a common cause of inefficiency in parallel programs. These metrics then allow a comparison of the parallel performance (e.g. over a range of thread/process counts, across different machines, or at different stages of optimization and tuning) to identify which characteristics of the code contribute to the inefficiency.

The first step to calculating these metrics is to use a suitable tool (e.g. Score-P or Extrae) to generate trace data whilst the code is executed. The traces contain information about the state of the code at a particular time, e.g. it is in a communication routine or doing useful computation, and also contains values from processor hardware counters, e.g. number of instructions executed, number of cycles.

The Only-MPI Assessment are then calculated as efficiencies between 0 and 1, with higher numbers being better. In general, we regard efficiencies above 0.8 as acceptable, whereas lower values indicate performance issues that need to be explored in detail. The ultimate goal then for POP is rectifying these underlying issues by the user. Please note, that Only-MPI Assessment can be computed only for inclusive callpaths, as they are less meaningful for exclusive callpaths. Furthermore, Only-MPI Assessment are not available in "Flat view" mode.

The approach outlined here is applicable to various parallelism paradigms, however for simplicity the Only-MPI Assessment presented here are formulated in terms of a distributed-memory message-passing environment, e.g., MPI. For this the following values are calculated for each process from the trace data: time doing useful computation, time in communication, number of instructions & cycles during useful computation. Useful computation excludes time within the overhead of parallel paradigms (Computation time).

At the top of the hierarchy is Global Efficiency (GE), which we use to judge overall quality of parallelization. Typically, inefficiencies in parallel code have two main sources:

- Overhead imposed by the parallel nature of a code
- Poor scaling of computation with increasing numbers of processes

and to reflect this we define two sub-metrics to measure these two inefficiencies. These are the Parallel Efficiency and the Computation Efficiency, and our top-level GE metric is

the product of these two sub-metrics:

$$GE = \text{Parallel Efficiency} \times \text{Computation Efficiency}$$

1.3.2.1 Only-MPI POP Metrics

Parallel Efficiency (PE) reveals the inefficiency in splitting computation over processes and then communicating data between processes. As with GE, PE is a compound metric whose components reflect two important factors in achieving good parallel performance in code:

- Ensuring even distribution of computational work across processes
- Minimizing time communicating data between processes

These are measured with Load Balance Efficiency and Communication Efficiency, and PE is defined as the product of these two sub-metrics:

$$PE = \text{Load Balance Efficiency} \times \text{Communication Efficiency}$$

Load Balance Efficiency can be computed as follows:

$$\text{Load Balance Efficiency} = \frac{\text{avg}(\text{computation time})}{\text{max}(\text{computation time})}$$

Thus it shows how big is a difference between average and maximal computation.

Communication Efficiency (CommE) is the maximum across all processes of the ratio between useful computation time and total run-time:

$$\text{CommE} = \text{maximum across processes} \frac{\text{computation time}}{\text{total runtime}}$$

CommE identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations.

CommE is composed of two additional metrics that reflect two causes of excessive time within communication:

- Processes waiting at communication points for other processes to arrive (i.e. serialisation)
- Processes transferring large amounts of data relative to the network capacity

These are measured using Serialisation Efficiency and Transfer Efficiency.

Combination of these two sub-metrics gives us Communication Efficiency:

$$\text{CommE} = \text{Serialisation Efficiency} \times \text{Transfer Efficiency}$$

To obtain these two sub-metrics we need to perform Scalasca trace analysis which identifies serialisations and inefficient communication patterns.

Serialisation Efficiency (SerE) measures inefficiency due to idle time within communications, i.e. time where no data is transferred, and is expressed as:

$$\text{SerE} = \text{maximum across processes} \left(\frac{\text{computation time}}{\text{total runtime on ideal network}} \right)$$

where total run-time on ideal network is a runtime without detected by Scalasca waiting time and MPI I/O time.

Transfer Efficiency (TE) measures inefficiencies due to time spent in data transfers:

$$TE = \frac{\text{maximum across processes}(\text{total runtime on ideal network})}{\text{maximum across processes}(\text{total measured runtime})}$$

where total run-time on ideal network is a runtime without detected by Scalasca waiting time and MPI I/O time.

1.3.3 Hybrid Additive POP analysis

Hybrid Additive This is one approach to extend POP metrics (see: 1.3.2) for hybrid (MPI+OpenMP) applications. In this approach Parallel Efficiency split into two components:

$$PE = \text{Process Efficiency} + \text{Thread Efficiency} - 1$$

- Process Efficiency shows the inefficiencies on MPI level, and can be broken down into **Computation Load Balance** and **MPI Communication Efficiency**
-
- Thread Efficiency shows the inefficiencies on OpenMP level, and can be broken down into **Amdahl's Efficiency** and **OpenMP Region Efficiency**

1.3.3.1 Hybrid Additive POP Metrics

Parallel Efficiency (PE) reveals the inefficiency in processes and threads utilization. These are measured with **Process Efficiency** and **Thread Efficiency**, and PE can be computed directly or as a sum of these two sub-metrics minus one:

$$PE = \frac{\text{avg}(\text{computationtime})}{\text{max}(\text{runtime})} = \text{Process Efficiency} + \text{Thread Efficiency} - 1$$

Process Efficiency completely ignores thread behavior, and evaluates process utilization via two components:

- Workload across processes
- Communication across processes

These two can be measured with Computation Load Balance and Communication Efficiency respectively. Process Efficiency can be computed directly or as a sum of these two sub-metrics minus one:

$$\text{Process Efficiency} = \frac{(\text{avg}(\text{time in OpenMP}) + \text{avg}(\text{serial computation}))}{\text{max}(\text{runtime})} = \text{Computation Load Balance} + \text{MPI Communication Efficiency} - 1$$

Where average time in OpenMP and average serial computation are computed as weighted arithmetic mean. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as ordinary arithmetic mean.

Computation Load Balance can be evaluated directly by following formula:

$$\text{Computation Load Balance} = (\max(\text{runtime}) - \max(\text{time in OpenMP} + \text{serial computation time}) + \text{avg}(\text{time}))$$

Where average time in OpenMP and average serial computation are computed as weighted arithmetic mean. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as ordinary arithmetic mean.

MPI Communication Efficiency (CommE) can be evaluated directly by following formula:

$$\text{CommE} = \frac{(\max(\text{time in OpenMP} + \text{serial computation time}))}{\max(\text{runtime})}$$

CommE identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations. CommE is composed of two additional metrics that reflect two causes of excessive time within communication:

- Processes waiting at communication points for other processes to arrive (i.e. serialisation)
- Processes transferring large amounts of data relative to the network capacity

These are measured using Serialisation Efficiency and Transfer Efficiency. Combination of these two sub-metrics gives us Communication Efficiency:

$$\text{CommE} = \text{Serialisation Efficiency} \times \text{Transfer Efficiency}$$

To obtain these two sub-metrics we need to perform Scalasca trace analysis which identifies serialisations and inefficient communication patterns.

Serialisation Efficiency (SerE) measures inefficiency due to idle time within communications, i.e. time where no data is transferred, and is expressed as:

$$\text{SerE} = \text{maximum across processes} \left(\frac{\text{computation time}}{\text{total runtime on ideal network}} \right)$$

where total run-time on ideal network is a runtime without detected by Scalasca waiting time and MPI I/O time.

Transfer Efficiency (TE) measures inefficiencies due to time spent in data transfers:

$$\text{TE} = \frac{\text{maximum across processes}(\text{total runtime on ideal network})}{\text{maximum across processes}(\text{total measured runtime})}$$

where total run-time on ideal network is a runtime without detected by Scalasca waiting time and MPI I/O time.

Thread Efficiency considers two sources of inefficiency:

- Serial computation on the master outside OpenMP, i.e. reflects Amdahl's law
- Inefficiencies within threads, e.g. serialisation across threads

These two can be measured with Amdahl's Efficiency and OpenMP region Efficiency respectively. Thread Efficiency can be computed directly or as a sum of these two sub-metrics minus one:

$$\text{Thread Efficiency} = \frac{\max(\text{runtime}) - \text{avg}(\text{time in OpenMP}) + \text{avg}(\text{time in useful computation within OpenMP})}{\max(\text{runtime})}$$

Where average idling time of OpenMP threads considers that threads are idling if only master thread is working and can be computed by following formula

$$avg(idling\ time\ of\ OpenMP\ threads) = \sum_{process=0}^{num\ of\ processes} serial\ computation \frac{(number\ of\ threads\ per\ process - 1)}{number\ of\ all\ available\ threads}$$

Moreover, average time in OpenMP computed as weighted arithmetic mean. If number of threads is equal across processes average time in OpenMP can be computed as ordinary arithmetic mean.

Amdahl's Efficiency indicates serial computation and can be computed as follows:

$$Amdahl's\ Efficiency = \frac{(max(runtime) - avg(idling\ time\ of\ OpenMP\ threads))}{max(runtime)}$$

OpenMP Region Efficiency indicates inefficiencies within threads, and can be computed as follows:

$$OpenMP\ Region\ Efficiency = \frac{max(runtime) - avg(time\ in\ OpenMP) + avg(time\ in\ useful\ computation\ within\ threads)}{max(runtime)}$$

Where average time in OpenMP is computed as weighted arithmetic mean. If number of threads is equal across processes average time in OpenMP can be computed as ordinary arithmetic mean.

1.3.4 Hybrid Multiplicative POP analysis

Hybrid Multiplicative This is one approach to extend POP metrics (see: 1.3.2) for hybrid (MPI+OpenMP) applications. In this approach Parallel Efficiency split into two components:

$$PE = Process\ Efficiency \times Thread\ Efficiency$$

- Process Efficiency shows the inefficiencies on MPI level, and can be broken down into **Computation Load Balance** and **MPI Communication Efficiency**
- Thread Efficiency shows the inefficiencies on OpenMP level, and can be broken down into **Amdahl's Efficiency** and **OpenMP Region Efficiency**

1.3.4.1 Hybrid Additive POP Metrics

Parallel Efficiency (PE) reveals the inefficiency in processes and threads utilization. These are measured with **Process Efficiency** and **Thread Efficiency**, and PE can be computed directly or as a product of these two sub-metrics:

$$PE = \frac{avg(comp)}{max(runtime)} = Process\ Efficiency \times Thread\ Efficiency$$

Process Efficiency completely ignores thread behavior, and evaluates process utilization via two components:

- Workload across processes

- Communication across processes

These two can be measured with Computation Load Balance and Communication Efficiency respectively. Process Efficiency can be computed directly or as a product of these two sub-metrics:

$$PE = \frac{avg(timeinOpenMP) + avg(serialcomputation)}{max(runtime)} = Computation\ Load\ Balance \times MPI\ Communication$$

Where average time in OpenMP and average serial computation are computed as weighted arithmetic mean. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as ordinary arithmetic mean.

Computation Load Balance can be evaluated directly by following formula:

$$Computation\ Load\ Balance = \frac{(avg(time\ in\ OpenMP) + avg(serial\ computation))}{max(time\ in\ OpenMP + serial\ computation\ time)}$$

Where average time in OpenMP and average serial computation are computed as weighted arithmetic mean. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as ordinary arithmetic mean.

MPI Communication Efficiency (CommE) can be evaluated directly by following formula:

$$CommE = \frac{max(timeinOpenMP + serialcomputationtime)}{max(runtime)}$$

CommE identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations. CommE is composed of two additional metrics that reflect two causes of excessive time within communication:

- Processes waiting at communication points for other processes to arrive (i.e. serialisation)
- Processes transferring large amounts of data relative to the network capacity

These are measured using Serialisation Efficiency and Transfer Efficiency. Combination of these two sub-metrics gives us Communication Efficiency:

$$CommE = Serialisation\ Efficiency \times Transfer\ Efficiency$$

To obtain these two sub-metrics we need to perform Scalasca trace analysis which identifies serialisations and inefficient communication patterns.

Serialisation Efficiency (SerE) measures inefficiency due to idle time within communications, i.e. time where no data is transferred, and is expressed as:

$$SerE = maximum\ across\ processes\left(\frac{computation\ time}{total\ runtime\ on\ ideal\ network}\right)$$

where total run-time on ideal network is a runtime without detected by Scalasca waiting time and MPI I/O time.

Transfer Efficiency (TE) measures inefficiencies due to time spent in data transfers:

$$TE = \frac{maximum\ across\ processes(total\ runtime\ on\ ideal\ network)}{maximum\ across\ processes(total\ measured\ runtime)}$$

where total run-time on ideal network is a runtime without detected by Scalasca waiting time and MPI I/O time.

Thread Efficiency considers two sources of inefficiency:

- Serial computation on the master outside OpenMP, i.e. reflects Amdahl's law
- Inefficiencies within threads, e.g. serialisation across threads

These two can be measured with Amdahl's Efficiency and OpenMP region Efficiency respectively. Thread Efficiency can be computed directly or as a product of these two sub-metrics:

$$\text{ThreadEfficiency} = \frac{\text{avg}(\text{computationtime})}{(\text{avg}(\text{timeinOpenMP}) + \text{avg}(\text{serialcomputation}))} = \text{Amdahl's Efficiency} \times \text{OpenMP Region Efficiency}$$

Where average time in OpenMP and average serial computation are computed as weighted arithmetic mean. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as ordinary arithmetic mean.

Thread Efficiency considers two sources of inefficiency:

- Serial computation on the master outside OpenMP, i.e. reflects Amdahl's law
- Inefficiencies within threads, e.g. serialisation across threads

These two can be measured with Amdahl's Efficiency and OpenMP region Efficiency respectively. Thread Efficiency can be computed directly or as a product of these two sub-metrics:

$$\text{Thread Efficiency} = \frac{\text{avg}(\text{computationtime})}{(\text{avg}(\text{timeinOpenMP}) + \text{avg}(\text{serialcomputation}))} = \text{Amdahl's Efficiency} \times \text{OpenMP Region Efficiency}$$

Where average time in OpenMP and average serial computation are computed as weighted arithmetic mean. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as ordinary arithmetic mean.

OpenMP Region Efficiency indicates inefficiencies within threads, and can be computed as follows:

$$\text{OpenMPRegionEfficiency} = \frac{(\text{avg}(\text{timeinusefulcomputationwithinOpenMP}) + \text{avg}(\text{serialcomputation}))}{(\text{avg}(\text{timeinOpenMP}) + \text{avg}(\text{serialcomputation}))}$$

Where average time in OpenMP and average serial computation are computed as weighted arithmetic mean. If number of threads is equal across processes average time in OpenMP and average serial computation can be computed as ordinary arithmetic mean.

1.3.5 Hybrid BSC POP analysis

Hybrid Multiplicative This is one approach to extend POP metrics (see: 1.3.2) for hybrid (MPI+OpenMP) applications. In this approach Parallel Efficiency split into two components:

- Hybrid Parallel Efficiency reveals the inefficiency in processes and threads utilization and can be broken down into **Hybrid Load Balance Efficiency** and **Hybrid Communication Efficiency**
- MPI Parallel Efficiency reveals the inefficiency in MPI processes and can be broken down into **MPI Load Balance Efficiency** and **MPI Communication Efficiency**
- OpenMP Parallel Efficiency reveals the inefficiency in OpenMP threads and can be broken down into **OpenMP Load Balance Efficiency** and **OpenMP Communication Efficiency**

1.3.5.1 Hybrid BSC POP Metrics

Hybrid Parallel Efficiency (HPE) reveals the inefficiency in processes and threads utilization. These are measured with Hybrid Load Balance Efficiency and Hybrid Communication Efficiency, and HPE can be computed directly or as a product of these two sub-metrics:

$$HPE = \frac{avg(computationtime)}{max(runtime)} = Hybrid\ Load\ Balance\ Efficiency \times Hybrid\ Communication\ Efficiency$$

Hybrid Load Balance Efficiency can be computed as follows:

$$Hybrid\ Load\ Balance\ Efficiency = \frac{computation\ time}{maximum\ computation\ time}$$

Hybrid Communication Efficiency can be evaluated directly by following formula

$$Hybrid\ Communication\ Efficiency = \frac{max(computation\ time)}{max(runtime)}$$

This metric identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations.

MPI Parallel Efficiency (MPE) reveals the inefficiency in MPI processes. MPE can be computed directly or as a product of MPI Load Balance Efficiency and MPI Communication Efficiency :

$$MPE = \frac{avg(time\ outside\ of\ MPI)}{max(runtime)} = MPI\ Load\ Balance\ Efficiency \times MPI\ Communication\ Efficiency$$

MPI Load Balance Efficiency can be computed as follows:

$$MPI\ Load\ Balance\ Efficiency = \frac{avg(time\ outside\ of\ MPI)}{max(time\ outside\ of\ MPI)}$$

MPI Communication Efficiency can be evaluated directly by following formula:

$$MPI\ Communication\ Efficiency = \frac{max(time\ outside\ of\ MPI)}{max(runtime)}$$

This metric identifies when code is inefficient because it spends a large amount of time communicating rather than performing useful computations.

Serialisation Efficiency (SerE) measures inefficiency due to idle time within communications, i.e. time where no data is transferred, and is expressed as:

$$SerE = maximum\ across\ processes\left(\frac{computation\ time}{total\ runtime\ on\ ideal\ network}\right)$$

where total run-time on ideal network is a runtime without detected by Scalasca waiting time and MPI I/O time.

Transfer Efficiency (TE) measures inefficiencies due to time spent in data transfers:

$$TE = \frac{maximum\ across\ processes(total\ runtime\ on\ ideal\ network)}{maximum\ across\ processes(total\ measured\ runtime)}$$

where total run-time on ideal network is a runtime without detected by Scalasca waiting time and MPI I/O time.

OpenMP Parallel Efficiency (OMPE) reveals the inefficiency in OpenMP threads. OMPE can be computed directly or as a division Hybrid Parallel Efficiency by MPI Parallel Efficiency:

$$OMPE = \frac{avg(computation\ time)}{avg(time\ outside\ of\ MPI)} = \frac{Hybrid\ Parallel\ Efficiency}{MPI\ Parallel\ Efficiency}$$

OpenMP Load Balance Efficiency can be computed as follows:

$$OpenMP\ Load\ Balance\ Efficiency = \frac{Hybrid\ Load\ Balance\ Efficiency}{MPI\ Load\ Balance\ Efficiency}$$

OpenMP Communication Efficiency can be evaluated directly by following formula:

$$OpenMP\ Communication\ Efficiency = \frac{max(computation\ time)}{max(time\ outside\ of\ MPI)} = \frac{Hybrid\ Communication\ Efficiency}{MPI\ Communication\ Efficiency}$$

1.3.6 Additional metrics

cube_pop_metrics tool provides a series of additional metrics, which help to make better conclusion about the performance of the code

1.3.6.1 I/O Efficiencies

I/O Efficiency is an additional metric, indicating amount of time spend not in a useful computation, but in the processing of IO operations. IO Efficiency can be split into two components:

- POSIX IO Efficiency shows the fraction of time spent in POSIX IO calls
- MPI IO Efficiency shows the fraction of time spent in MPI IO calls

In this analysis IO can be computed as a sum of these two sub-metrics minus one:

$$IO = 1 - POSIX\ IO + MPI\ IO$$

, where 1 would indicate absence of any I/O operation (perfect usage of I/O is no I/O)

POSIX IO Efficiency shows the fraction of execution time spent in POSIX IO calls In this analysis POSIX IO is computed as :

$$POSIX\ IO = \frac{time\ in\ POSIX\ IO\ calls}{total\ runtime}$$

MPI IO Efficiency shows the fraction of execution time spent in MPI IO calls In this analysis MPI IO is computed as :

$$MPI\ IO = \frac{time\ in\ MPI\ IO\ calls}{total\ runtime}$$

1.3.6.2 Additional Efficiencies

Stalled resources indicates the fraction of computational cycles in user code that the processor has been waiting for some resources. If PAPI counters are available, use

$$\frac{PAPI_RES_STL}{PAPI_TOT_CYC}$$

. Otherwise, if Perf counters are available, use one of

1.

$$\frac{(stalled_cycles_backend + stalled_cycles_frontend)}{cycles}$$

2.

$$\frac{stalled_cycles_backend}{cycles}$$

3.

$$\frac{stalled_cycles_frontend}{cycles}$$

depending on what is available.

IPC Efficiency compares IPC to the reference, where lower values indicate that rate of computation has slowed. Typical causes for this include decreasing cache hit rate and exhaustion of memory bandwidth, these can leave processes stalled and waiting for data.

Instruction Efficiency is the ratio of total number of useful instructions for a reference case (e.g. 1 processor) compared to values when increasing the numbers of processes. A decrease in Instruction Efficiency corresponds to an increase in the total number of instructions required to solve a computational problem.

Computation time shows the time spent in the part of the code, identified as a useful computation. Computation time is defined in form of exclusion, namely, computation time is part of the execution time spend NOT in MPI, NOT in OpenMP, NOT in IO, not in SHMEM, not in service libraries, instrumented using the library wrapping, NOT in CUDA, NOT OpenCL, NOT in OpenACC, NOT in OpenCL and so on.

GPU Computation time shows the time spent in the part of the code, identified as a useful computation and executed on GPU.

1.3.6.3 Control Metrics

WallClock time metric is used to control the quality of the measurement and it displays :

- minimal
- average
- maximal runtime

across the system tree.

By inspecting this metric the user has an opportunity to estimate whether the measurement is made properly to be suitable for the POP analysis. If the measurement is very skewed (outliners, too wide range in the times from CPU to CPU, and similar) most likely is this measurement has been made erroneously and is not suitable for the analysis. In this case, it is advisable to redo the measurement.

1.4 Cube Remapping tool

CUBE provides an universal flexible tool, which allows to transform metric tree of an input .cubex into an arbitrary metric hierarchy according to user defined flexible rules. This can be used by performance analysis tools, such as Score-P or Scalasca. In particular one classifies different call paths according to some user defined rules.

As well one can use this tool to convert derived metrics into the data (if possible) or to add an additional metric into the current .cubex file.

During this processing step the remapper will also create topologies that can be constructed based on system tree information. For this the remapper will use reasonable default settings, for more manual control over topologies the [1.5.15](#) tool can be used.

1.4.1 Tool Usage

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 SCALASCA metrics "Idle threads" and "Limited parallelizm"
- h Help; Output a brief help message.

1.4.2 Remapping algorithm

The main idea behind the working way of the remapping tool is simple: user formulates the target hierarchy of the metrics in an external file (.spec) and defines rules how they are calculated using CubePL syntax. Remapping tool creates then an output cube with this metric hierarchy.

In particularly remapping tool performs following steps:

1. Is command-line option -c specified, remapping tool works in "Copy" mode: it creates output cube with identical metrics hierarhy. One combines it usual with the options -d and -s. Otherwise...

2. Is command-line option `-r` omit, remapping tool looks for the `.spec` file within the input `.cubex` file. Score-P starting with version v5 stores its `.spec` file within `profile.cubex`. Otherwise...
3. Is command-line option `-r` specified, remapping tool looks for `.spec` file and creates the new (working) cube object with the metric hierarchy as metric dimension specified in `.spec` file
4. Then it merges metric hierarchy of the input file into the current metric hierarchy. All metrics from the input file which present in `.spec` hierarchy gets mapped to them. All missing metrics will be added as additional metrics.
5. Is the command-line option `-s` specified, tool adds additional SCALASCA metrics "Idle threads" and "Limited parallelism"
6. Other dimensions and elements of the input `.cubex` file (topologies, mirrors and similar) get merged into the new cube object one-to-one (as the freshly created cube object doesn't have any)
7. If the command-line option `-d` is omit, remapping tool saves created cube on disk as it is. All derived metrics in `.spec` file will stay as derived metrics and will be executed only while working with the stored `.cubex` file in GUI or with command-line tools.
8. Did user specified command-line option `-d`, the second phase of the remapping is executed: calculation of derived metrics and converting them into the data. Note, that no all metrics are *convertible* (such as rates or similar) into data and should stay derived in the target file. To do so remapping tool creates an additional cube object with the identical structure (only convertible derived metrics are turned into the regular data metrics) of the intermediate working cube and copies metric-by-metric its data. As result, derived metrics are calculated and their value is stored as a data into the target cube.
9. Resulting cube object is saved on disk.

As one can see user can use remapping tool for different purposes:

1. Convert derived metrics of the current `.cubex` file into the data metrics. For that it is enough to execute remapping tool as:

```
./cube_remap2 -d <input>
```

2. Add an additional derived metric into the current `.cubex` cube
3. Create a new `.cubex` with the user specified metrics hierarchy. (standard usage)

1.4.3 Syntax of a `.spec` file

Examples for `.spec` files are shipped with the CUBE package and stored in the directory `[prefix]/share/doc/cubelib/examples`

Here we provide some commented examples of `.spec` files.

1. In first case we add an additional metric IPC into the cube file

```
1 <metrics>
2   <metric type="POSTDERIVED" convertible="false">
```

```

3     <disp_name>IPC, ipc</disp_name>
4     <uniq_name>ipc</uniq_name>
5     <dtype>FLOAT</dtype>
6     <uom></uom>
7     <descr>Value of IPC (instructions per cycle),
8         computed as PAPI_TOT_INS() / PAPI_TOT_CYC()
9     </descr>
10    <cubepl>
11        metric::PAPI_TOT_INS() / metric::PAPI_TOT_CYC()
12    </cubepl>
13 </metric>
14 </metrics>

```

Here the whole hierarchy consist of only one metric IPC, which is "non convertible" into data as it is a POSTDERIVED metric and usual aggregation rules are not applicable. This metric has a data type FLOAT and requires, that input file contains metrics PAPI_TOT_CYC and PAPI_TOT_INS. Otherwise this metric will be always evaluated to 0.

2. In second case we add two sub metrics to the metric "Time", in order to classify time spend in "MPI" regions and "Other"

```

1 <metrics>
2   <metric>
3     <!-- This metric is copied from the trace analysis -->
4     <disp_name>Time</disp_name>
5     <uniq_name>time</uniq_name>
6     <dtype>FLOAT</dtype>
7     <uom>sec</uom>
8     <url>@mirror@scalasca_patterns-2.1.html#time</url>
9     <descr>Total CPU allocation time
10        (includes time allocated for idle threads)
11    </descr>
12    <metric type="PREDERIVED_EXCLUSIVE">
13      <disp_name>MPI regions</disp_name>
14      <uniq_name>mpiregions</uniq_name>
15      <dtype>FLOAT</dtype>
16      <uom>sec</uom>
17      <url></url>
18      <descr> Time spend in MPI regions </descr>
19      <cubepl>
20        {
21          ${a}=0;
22          if (${cube::region::name}[${calculation::region::id}] =~ /^MPI_/)
23            {
24              ${a}=metric::time();
25            };
26          return ${a};
27        }
28      </cubepl>
29    </metric>
30    <metric type="PREDERIVED_EXCLUSIVE">
31      <disp_name>Other regions</disp_name>
32      <uniq_name>nonmpi</uniq_name>
33      <dtype>FLOAT</dtype>
34      <uom>sec</uom>
35      <url></url>
36      <descr>Time spend in non-MPI regions</descr>
37      <cubepl>
38        {
39          ${a}=0;
40          if (not(${cube::region::name}[${calculation::region::id}] =~ /^MPI_/))
41            {
42              ${a}=metric::time();
43            };
44          return ${a};
45        }
46      </cubepl>
47    </metric>
48 </metric>
49 </metrics>

```

Here target hierarchy consist of the top metric "Time" and two sub metrics "MPI regions" and "Other regions". Clearly that they both will sum up to the total "Time" metric. Hence one expects that expanded metric "Time" will be always zero. Note that sub metrics are defined as PREDERIVED_EXCLUSIVE as their value is valid only for the one call path and not their children.

3. In third case we would like to split the time metric into two metrics and to hide the metric itself. For that we declare it as "GHOST" metric and declare sub metrics as root metrics.

```
1 <metrics>
2   <metric viztype="GHOST">
3     <!-- This metric is copied from the trace analysis -->
4     <disp_name>Time</disp_name>
5     <uniq_name>time</uniq_name>
6     <dtype>FLOAT</dtype>
7     <uom>sec</uom>
8     <url>@mirror@scalasca_patterns-2.1.html#time</url>
9     <descr>Total CPU allocation time
10      (includes time allocated for idle threads)
11     </descr>
12   </metric>
13   <metric type="PREDERIVED_EXCLUSIVE">
14     <disp_name>MPI regions</disp_name>
15     <uniq_name>mpiregions</uniq_name>
16     <dtype>FLOAT</dtype>
17     <uom>sec</uom>
18     <url></url>
19     <descr> Time spend in MPI regions </descr>
20     <cubepl>
21       {
22         ${a}=0;
23         if (${cube::region::name}[${calculation::region::id}] =~ /^MPI_/)
24         {
25           ${a}=metric::time();
26         };
27         return ${a};
28       }
29     </cubepl>
30   </metric>
31   <metric type="PREDERIVED_EXCLUSIVE">
32     <disp_name>Other regions</disp_name>
33     <uniq_name>nonmpi</uniq_name>
34     <dtype>FLOAT</dtype>
35     <uom>sec</uom>
36     <url></url>
37     <descr>Time spend in non-MPI regions</descr>
38     <cubepl>
39       {
40         ${a}=0;
41         if (not(${cube::region::name}[${calculation::region::id}] =~ /^MPI_/))
42         {
43           ${a}=metric::time();
44         };
45         return ${a};
46       }
47     </cubepl>
48   </metric>
49 </metrics>
```

As result, the target .cube file will have only two new root metrics, "MPI regions" and "Other regions". Metric "Time" will be not visible in GUI and command line tools.

1.4.4 Optimization of the remapping.

As remapping process is defined in very general way, naive approach in .spec file formulation might take significant calculation time. Using advanced technics in CubePL one can optimize remapping process. For example one can:

- Using CubePL variables in order to avoid string comparison by every calculation step
- Using CubePL initialization step in order to perform some calculation only once
- Use bitmask multiplication for the evaluation of values along the system tree

Great example of usage of the optimization technics are the .spec files provided with Score-P or Scalasca.

Internal computation is performed asynchronously via tasking paradigm. To control level of concurrency one can set the environment variable **CUBE_MAX_CONCURRENCY**. Possible values are

N - utilize maximal N thread simultaneously.

max - utilize maximal #cores threads simultaneously (default).

Value larger than #cores will lead to oversubscription of the CPUs and might lead to regression of the performance.

1.5 Supplementary utilities and helping Cube command line tools

Besides of the remapping utility and the algebra tools CUBE provides various tool for different testing, checking, inspection tasks.

1.5.1 Cube Server `cube_server`

This tool starts a server and provide cube reports "as they are" for clients without needing to copy them to the client machine. The `cube_server` performs also all required calculations, so one uses computational power of the HPC system and not of the client system (which might be insufficient).

```
user@host: cube_server
[12044] Cube Server(<name>): CubeLib-4.7.0 (e15c4008) [POSIX]
```

Usage: `cube_server` [options]

-h, -? Print this help message.

-n <name> Use <name> as a name of the server in report messages.

-v <level> Verbosity <level>, one of "mute", "launch", "full" (default "mute")

-p N Bind socket on port N (default port: 3300)

-c Close server when last task is finished.

Many hosts don't allow ports to be accessed from the outside. You may use SSH tunneling (also referred to as SSH port forwarding) to route the local network traffic through SSH to the remote host.

In the following example, `cube_server` is started with the default port 3300 on the remote server `server.example.com`. The traffic, which is sent to `localhost:3000`, will be forwarded to `server.example.com` on the same port.

If `<name>` is omitted, the name is either generated by the `cube_server` or the value of the environment variable `CUBE_SERVER_NAME` is taken.

Usually `cube_server` runs indefinitely long, until the user kills the process (Ctrl^C or command 'kill'). If one starts the `cube_server` with the command line option `"-c"`, one connection is served and if it is disconnected, `cube_server` finishes.

```
[client]$ ssh -L 3300:server.example.com:3300 server.example.com
[server.example.com]$ cube_server
Cube Server: CubeLib-4.7.0 (external) [POSIX]
cube_server[5247] Waiting for connections on port 3300.
```

1.5.2 Sanity Check

This tool performs a series of checks to confirm that stored data has a semantic sense, e.g. "non-negative time" and similar. It is being used as a correctness tool for the Score-P and Scalasca measurements.

```
user@host: cube_sanity profile.cubex
Name not empty or UNKNOWN ... 71 / 71 OK
  No ANONYMOUS functions ... 71 / 71 OK
  No TRUNCATED functions ... 71 / 71 OK
  File name not empty ... 71 / 71 OK
  Proper line numbers ... 0 / 71 OK
  No TRACING outside Init and Finalize ... 123 / 123 OK
No negative inclusive metrics ... 409344 / 440832 OK
No negative exclusive metrics ... 409344 / 440832 OK
```

Usage: `cube_sanity [flags] <cube experiment>`

- h** Help; Output a brief help message.
- n** Disables the (time consuming) check for negative inclusive metric values.
- x** Disables the (time consuming) check for negative exclusive metric values.
- l** Disables checks for line numbers.
- f <file.filt>** Checks whether a node's name is matched by a pattern in `file.filt`.
- o <output_file>** Path of the output file. If no output file is given, detailed output will be suppressed. A summary will always be printed out to `stdout`.
- v <0|1|2|3>** Verbosity level of output

1.5.3 Insertion of of derived metric into the .cubex

One small tool, which does nothing but adding an additional user-defined derived metric into the input .cubex file. Functionality of this tool is provided by remapping tool 1.4 tool as well, but from historical reasons we still deliver this tool.

```
user@host: cube_derive -t postderived -r exclusive -e "metric::time()/metric::visits(e)" -p root kenobi

user@host: cube_dump -m kenobi -c 0 -t aggr -z incl result.cubex
===== DATA =====
Print out the data of the metric kenobi

All threads
-----
MAIN_(id=0) 1389.06
```

Usage: cube_derive [-h] [-t prederived|postderived] -r exclusive|inclusive -e <cubepl expression> -p <parent_metric> <targetmetric> <cube experiment> [-o <output experiment>]

- h** Help; Output a brief help message.
- t prederived|postderived** Specifies, whether derived metric is PREDERIVED or POSTDERIVED
- r exclusive|inclusive** Specifies, whether derived metric is EXCLUSIVE or INCLUSIVE (has no effect for POSTDERIVED metrics)
- e** CubePL expression of the derived metric
- p** New metric will be added as a child of this metric. "root" is possible
Name of the created metric. It should not clash with the existing metric.
Name of the input file
- o** Name of the output file. Default "derived.cubex"

1.5.4 Inspection of a call tree

A small tool to inspect aggregated values in call tree of a specific metric. Functionality of this tool is reproduced in one of another way by cube_dump 1.5.16 utility.

```
user@host:cube_calltree -m time -a -t 1 -i -p -c ~/Studies/Cube/Adviser/4n_256_/scorep_casino_1n_256x1
Reading /home/zam/psaviank/Studies/Cube/Adviser/4n_256_/scorep_casino_1n_256x1_sum_br/profile.cubex... do
355599 (100%)          MAIN__          USR:/MAIN__
353588 (99.435%)      + monte_carlo_          USR:/MAIN__/mont
8941.95 (2.515%)      | + monte_carlo_IP_read_wave_function_ USR:/MAIN__/mont
5980.68 (1.682%)      | | + MPI_Bcast            USR:/MAIN__/mont
343175 (96.506%)      | + monte_carlo_IP_run_dmc_          USR:/MAIN__/mont
4657.85 (1.31%)       | | + MPI_Ssend           USR:/MAIN__/mont
303193 (85.263%)      | | + dmcdmc_main.move_config_          USR:/MAIN__/mont
21668.4 (6.093%)     | | | + wfn_utils.wfn_loggrad_          USR:/MAIN__/mont
```

19722.3 (5.546%)					+ pjastrow.oneelec_jastrow_	USR:/MAIN_/n
28878.5 (8.121%)					+ wfn_utils.wfn_ratio_	USR:/MAIN_/n
6907.92 (1.943%)					+ slater.wfn_ratio_slater_	USR:/MAIN_/n
19721.7 (5.546%)					+ pjastrow.oneelec_jastrow_	USR:/MAIN_/n
241626 (67.949%)					+ energy_utils.eval_local_energy_	USR:/MAIN_/n
20785.6 (5.845%)					+ wfn_utils.wfn_loggrad_	USR:/MAIN_/n
20298.1 (5.708%)					+ pjastrow.oneelec_jastrow_	USR:/MAIN_/n
211484 (59.473%)					+ non_local.calc_nl_projection_	USR:/MAIN_/n
208516 (58.638%)					+ wfn_utils.wfn_ratio_	USR:/MAIN_/n
8588.87 (2.415%)					+ slater.wfn_ratio_slater_	USR:/MAIN_/n
176779 (49.713%)					+ pjastrow.oneelec_jastrow_	USR:/MAIN_/n
14252.4 (4.008%)					+ scratch.get_eevecs1_ch_	USR:/MAIN_/n
34390.8 (9.671%)					+ parallel.qmpi_reduce_d1_	USR:/MAIN_/n
34384.3 (9.669%)					+ MPI_Reduce	USR:/MAIN_/n

Usage: cube_calltree -a -m metricname -t threshold -c -i -u -h [-f] filename

- m **metricname** print out values for the metric <metricname>;
- a annotate call path;
- t **treashold** print out only call path with a value, which is larger than <treashold>% of the total metric value. 0<=treashold<=100;
- i calculate inclusive values instead of exclusive.
- f open <filename>;
- p diplay percent value;
- c diplay callpath for every region;
- h display this help.

1.5.5 System tree partition

Is one interested on performance profile of a PART of the system, e.g. only nodeplane 0, or ranks from 0 to N, one can set other locations to VOID and create a .cubex file, which has only data for the remaining part.

Partition some of system tree (processes) and void the remainder

```

user@host:cube_part -R 0-3,5 profile.cubex
Open the cube profile.cubex...done.
Copy the cube object...done.
+++++++ Part operation begins ++++++++
Voiding 251 of 256 processes.
+++++++ Part operation ends successfully ++++++++
Writing part ... done.

```

Usage: cube_part [-h] [-I] [-R ranks] [-o output] <cubefile>

- I Invert sense of partition
- R List of process ranks for partition (e.g., "0-3,7,13-")

- o Name of the output file (default: part)
- h Help; Show this brief help message and exit.

1.5.6 Region information

This tool displays a flat profile for the selected metrics and with the classification of regions. Similar functionality is provide by the tool [1.5.16](#)

Short call displays an overview across types of regions.

```
user@host:cube_regioninfo -m time,visits profile.cubex
done.
bl type          time      %      visits      % region
  ANY          355599 100.00  664724749 100.00 (summary) ALL
  MPI              0    0.00         0    0.00 (summary) MPI
  USR          355599 100.00  664724749 100.00 (summary) USR
  COM              0    0.00         0    0.00 (summary) COM
```

If one wants a detailed view of the flat profile, one uses -r command line option. Note that can provide list of regions for filtering (file filter.txt)

```
user@host:cube_regioninfo -l filter.txt -m time,visits -r profile.cubex
done.
bl type          time      %      visits      % region
  USR              0    0.00         0    0.00 MEASUREMENT OFF
  USR              0    0.00         0    0.00 TRACE BUFFER FLUSH
  USR              0    0.00         0    0.00 THREADS
  USR              0    0.00         0    0.00 MPI_Accumulate
  USR             68    0.02        6912    0.00 MPI_Allgather
  USR              0    0.00         0    0.00 MPI_Allgatherv
  USR             10    0.00        256    0.00 MPI_Allreduce
  USR              0    0.00         0    0.00 MPI_Alltoall
  USR              0    0.00         0    0.00 MPI_Alltoallv
  USR              0    0.00         0    0.00 MPI_Alltoallw
  USR              0    0.00         0    0.00 MPI_Attr_delete
  USR              0    0.00         0    0.00 MPI_Attr_get
  USR              0    0.00         0    0.00 MPI_Attr_put
  USR            2843    0.80        7168    0.00 MPI_Barrier
  USR            6392    1.80       51072    0.01 MPI_Bcast
  USR              0    0.00         0    0.00 MPI_Bsend
  USR              0    0.00         0    0.00 MPI_Bsend_init
  USR              0    0.00         0    0.00 MPI_Buffer_attach
  USR              0    0.00         0    0.00 MPI_Buffer_detach
  USR              0    0.00         0    0.00 MPI_Cancel
  USR              0    0.00         0    0.00 MPI_Cart_coords
  USR              0    0.00         0    0.00 MPI_Cart_create
  USR              0    0.00         0    0.00 MPI_Cart_get
  USR              0    0.00         0    0.00 MPI_Cart_map
  USR              0    0.00         0    0.00 MPI_Cart_rank
  USR              0    0.00         0    0.00 MPI_Cart_shift
```

USR	0	0.00	0	0.00	MPI_Cart_sub
USR	0	0.00	0	0.00	MPI_Cartdim_get
USR	0	0.00	0	0.00	MPI_Comm_call_errhandler
USR	0	0.00	0	0.00	MPI_Comm_compare
USR	113	0.03	1792	0.00	MPI_Comm_create
USR	0	0.00	0	0.00	MPI_Comm_create_errhandler
USR	0	0.00	0	0.00	MPI_Comm_create_group
USR	0	0.00	0	0.00	MPI_Comm_create_keyval
USR	0	0.00	0	0.00	MPI_Comm_delete_attr
USR	0	0.00	0	0.00	MPI_Comm_dup
USR	0	0.00	0	0.00	MPI_Comm_dup_with_info
USR	0	0.00	0	0.00	MPI_Comm_free
USR	0	0.00	0	0.00	MPI_Comm_free_keyval
USR	0	0.00	0	0.00	MPI_Comm_get_attr
USR	0	0.00	0	0.00	MPI_Comm_get_errhandler
USR	0	0.00	0	0.00	MPI_Comm_get_info
USR	0	0.00	0	0.00	MPI_Comm_get_name
USR	0	0.00	1024	0.00	MPI_Comm_group
USR	0	0.00	0	0.00	MPI_Comm_idup
USR	0	0.00	512	0.00	MPI_Comm_rank
USR	0	0.00	0	0.00	MPI_Comm_remote_group
USR	0	0.00	0	0.00	MPI_Comm_remote_size
USR	0	0.00	0	0.00	MPI_Comm_set_attr
USR	0	0.00	0	0.00	MPI_Comm_set_errhandler
USR	0	0.00	0	0.00	MPI_Comm_set_info
USR	0	0.00	0	0.00	MPI_Comm_set_name
USR	0	0.00	512	0.00	MPI_Comm_size
USR	55	0.02	256	0.00	MPI_Comm_split
USR	0	0.00	0	0.00	MPI_Comm_split_type
USR	0	0.00	0	0.00	MPI_Comm_test_inter
USR	0	0.00	0	0.00	MPI_Compare_and_swap
USR	0	0.00	0	0.00	MPI_Dims_create
USR	0	0.00	0	0.00	MPI_Dist_graph_create
USR	0	0.00	0	0.00	MPI_Dist_graph_create_adjacent
USR	0	0.00	0	0.00	MPI_Dist_graph_neighbors
USR	0	0.00	0	0.00	MPI_Dist_graph_neighbors_count
USR	0	0.00	0	0.00	MPI_Exscan
USR	0	0.00	0	0.00	MPI_Fetch_and_op
USR	0	0.00	0	0.00	MPI_File_c2f
USR	0	0.00	0	0.00	MPI_File_call_errhandler
USR	0	0.00	0	0.00	MPI_File_close
USR	0	0.00	0	0.00	MPI_File_create_errhandler
USR	0	0.00	0	0.00	MPI_File_delete
USR	0	0.00	0	0.00	MPI_File_f2c
USR	0	0.00	0	0.00	MPI_File_get_amode
USR	0	0.00	0	0.00	MPI_File_get_atomicsity
USR	0	0.00	0	0.00	MPI_File_get_byte_offset
USR	0	0.00	0	0.00	MPI_File_get_errhandler
USR	0	0.00	0	0.00	MPI_File_get_group
USR	0	0.00	0	0.00	MPI_File_get_info
USR	0	0.00	0	0.00	MPI_File_get_position
USR	0	0.00	0	0.00	MPI_File_get_position_shared
USR	0	0.00	0	0.00	MPI_File_get_size

1.5 Supplementary utilities and helping Cube command line tools

USR	0	0.00	0	0.00	MPI_File_get_type_extent
USR	0	0.00	0	0.00	MPI_File_get_view
USR	0	0.00	0	0.00	MPI_File_iread
USR	0	0.00	0	0.00	MPI_File_iread_all
USR	0	0.00	0	0.00	MPI_File_iread_at
USR	0	0.00	0	0.00	MPI_File_iread_at_all
USR	0	0.00	0	0.00	MPI_File_iread_shared
USR	0	0.00	0	0.00	MPI_File_iwrite
USR	0	0.00	0	0.00	MPI_File_iwrite_all
USR	0	0.00	0	0.00	MPI_File_iwrite_at
USR	0	0.00	0	0.00	MPI_File_iwrite_at_all
USR	0	0.00	0	0.00	MPI_File_iwrite_shared
USR	0	0.00	0	0.00	MPI_File_open
USR	0	0.00	0	0.00	MPI_File_preallocate
USR	0	0.00	0	0.00	MPI_File_read
USR	0	0.00	0	0.00	MPI_File_read_all
USR	0	0.00	0	0.00	MPI_File_read_all_begin
USR	0	0.00	0	0.00	MPI_File_read_all_end
USR	0	0.00	0	0.00	MPI_File_read_at
USR	0	0.00	0	0.00	MPI_File_read_at_all
USR	0	0.00	0	0.00	MPI_File_read_at_all_begin
USR	0	0.00	0	0.00	MPI_File_read_at_all_end
USR	0	0.00	0	0.00	MPI_File_read_ordered
USR	0	0.00	0	0.00	MPI_File_read_ordered_begin
USR	0	0.00	0	0.00	MPI_File_read_ordered_end
USR	0	0.00	0	0.00	MPI_File_read_shared
USR	0	0.00	0	0.00	MPI_File_seek
USR	0	0.00	0	0.00	MPI_File_seek_shared
USR	0	0.00	0	0.00	MPI_File_set_atomicity
USR	0	0.00	0	0.00	MPI_File_set_errhandler
USR	0	0.00	0	0.00	MPI_File_set_info
USR	0	0.00	0	0.00	MPI_File_set_size
USR	0	0.00	0	0.00	MPI_File_set_view
USR	0	0.00	0	0.00	MPI_File_sync
USR	0	0.00	0	0.00	MPI_File_write
USR	0	0.00	0	0.00	MPI_File_write_all
USR	0	0.00	0	0.00	MPI_File_write_all_begin
USR	0	0.00	0	0.00	MPI_File_write_all_end
USR	0	0.00	0	0.00	MPI_File_write_at
USR	0	0.00	0	0.00	MPI_File_write_at_all
USR	0	0.00	0	0.00	MPI_File_write_at_all_begin
USR	0	0.00	0	0.00	MPI_File_write_at_all_end
USR	0	0.00	0	0.00	MPI_File_write_ordered
USR	0	0.00	0	0.00	MPI_File_write_ordered_begin
USR	0	0.00	0	0.00	MPI_File_write_ordered_end
USR	0	0.00	0	0.00	MPI_File_write_shared
USR	349	0.10	256	0.00	MPI_Finalize
USR	0	0.00	0	0.00	MPI_Finalized
USR	0	0.00	25	0.00	MPI_Gather
USR	0	0.00	0	0.00	MPI_Gatherv
USR	0	0.00	0	0.00	MPI_Get
USR	0	0.00	0	0.00	MPI_Get_accumulate
USR	0	0.00	0	0.00	MPI_Get_library_version

1.5 Supplementary utilities and helping Cube command line tools

USR	0	0.00	0	0.00	MPI_Issend
USR	0	0.00	0	0.00	MPI_Keyval_create
USR	0	0.00	0	0.00	MPI_Keyval_free
USR	0	0.00	0	0.00	MPI_Mprobe
USR	0	0.00	0	0.00	MPI_Mrecv
USR	0	0.00	0	0.00	MPI_Neighbor_allgather
USR	0	0.00	0	0.00	MPI_Neighbor_allgatherv
USR	0	0.00	0	0.00	MPI_Neighbor_alltoall
USR	0	0.00	0	0.00	MPI_Neighbor_alltoallv
USR	0	0.00	0	0.00	MPI_Neighbor_alltoallw
USR	0	0.00	0	0.00	MPI_Probe
USR	0	0.00	0	0.00	MPI_Put
USR	0	0.00	0	0.00	MPI_Query_thread
USR	0	0.00	0	0.00	MPI_Raccumulate
USR	302	0.08	24728	0.00	MPI_Recv
USR	0	0.00	0	0.00	MPI_Recv_init
USR	34384	9.67	6400	0.00	MPI_Reduce
USR	0	0.00	0	0.00	MPI_Reduce_local
USR	0	0.00	0	0.00	MPI_Reduce_scatter
USR	0	0.00	0	0.00	MPI_Reduce_scatter_block
USR	0	0.00	0	0.00	MPI_Register_datarep
USR	0	0.00	0	0.00	MPI_Request_free
USR	0	0.00	0	0.00	MPI_Rget
USR	0	0.00	0	0.00	MPI_Rget_accumulate
USR	0	0.00	0	0.00	MPI_Rput
USR	0	0.00	0	0.00	MPI_Rsend
USR	0	0.00	0	0.00	MPI_Rsend_init
USR	0	0.00	0	0.00	MPI_Scan
USR	0	0.00	0	0.00	MPI_Scatter
USR	0	0.00	0	0.00	MPI_Scatterv
USR	0	0.00	0	0.00	MPI_Send
USR	0	0.00	0	0.00	MPI_Send_init
USR	0	0.00	0	0.00	MPI_Sendrecv
USR	0	0.00	0	0.00	MPI_Sendrecv_replace
USR	4695	1.32	24728	0.00	MPI_Ssend
USR	0	0.00	0	0.00	MPI_Ssend_init
USR	0	0.00	0	0.00	MPI_Start
USR	0	0.00	0	0.00	MPI_Startall
USR	0	0.00	0	0.00	MPI_Test
USR	0	0.00	0	0.00	MPI_Test_cancelled
USR	0	0.00	0	0.00	MPI_Testall
USR	0	0.00	0	0.00	MPI_Testany
USR	0	0.00	0	0.00	MPI_Testsome
USR	0	0.00	0	0.00	MPI_Topo_test
USR	0	0.00	0	0.00	MPI_Wait
USR	0	0.00	2982	0.00	MPI_Waitall
USR	0	0.00	0	0.00	MPI_Waitany
USR	0	0.00	0	0.00	MPI_Waitsome
USR	0	0.00	0	0.00	MPI_Win_allocate
USR	0	0.00	0	0.00	MPI_Win_allocate_shared
USR	0	0.00	0	0.00	MPI_Win_attach
USR	0	0.00	0	0.00	MPI_Win_call_errhandler
USR	0	0.00	0	0.00	MPI_Win_complete

USR	0	0.00	0	0.00	MPI_Win_create
USR	0	0.00	0	0.00	MPI_Win_create_dynamic
USR	0	0.00	0	0.00	MPI_Win_create_errhandler
USR	0	0.00	0	0.00	MPI_Win_create_keyval
USR	0	0.00	0	0.00	MPI_Win_delete_attr
USR	0	0.00	0	0.00	MPI_Win_detach
USR	0	0.00	0	0.00	MPI_Win_fence
USR	0	0.00	0	0.00	MPI_Win_flush
USR	0	0.00	0	0.00	MPI_Win_flush_all
USR	0	0.00	0	0.00	MPI_Win_flush_local
USR	0	0.00	0	0.00	MPI_Win_flush_local_all
USR	0	0.00	0	0.00	MPI_Win_free
USR	0	0.00	0	0.00	MPI_Win_free_keyval
USR	0	0.00	0	0.00	MPI_Win_get_attr
USR	0	0.00	0	0.00	MPI_Win_get_errhandler
USR	0	0.00	0	0.00	MPI_Win_get_group
USR	0	0.00	0	0.00	MPI_Win_get_info
USR	0	0.00	0	0.00	MPI_Win_get_name
USR	0	0.00	0	0.00	MPI_Win_lock
USR	0	0.00	0	0.00	MPI_Win_lock_all
USR	0	0.00	0	0.00	MPI_Win_post
USR	0	0.00	0	0.00	MPI_Win_set_attr
USR	0	0.00	0	0.00	MPI_Win_set_errhandler
USR	0	0.00	0	0.00	MPI_Win_set_info
* USR	0	0.00	0	0.00	MPI_Win_set_name
* USR	0	0.00	0	0.00	MPI_Win_shared_query
* USR	0	0.00	0	0.00	MPI_Win_start
* USR	0	0.00	0	0.00	MPI_Win_sync
* USR	0	0.00	0	0.00	MPI_Win_test
* USR	0	0.00	0	0.00	MPI_Win_unlock
* USR	0	0.00	0	0.00	MPI_Win_unlock_all
* USR	0	0.00	0	0.00	MPI_Win_wait
* USR	0	0.00	0	0.00	PARALLEL
* USR	155	0.04	256	0.00	MAIN__
* USR	0	0.00	256	0.00	get_nnpsmp_
* USR	1	0.00	126669	0.02	run_control.check_alloc_
* USR	8	0.00	256	0.00	get_smp_list_
* USR	0	0.00	256	0.00	set_shm_numablock
* USR	401	0.11	256	0.00	monte_carlo_
* USR	0	0.00	256	0.00	monte_carlo_IP.set_defaults_
* USR	124	0.03	256	0.00	monte_carlo_IP.set_input_parameters_
* USR	10	0.00	256	0.00	monte_carlo_IP.read_particles_
* USR	3	0.00	256	0.00	monte_carlo_IP.read_custom_spindep_
* USR	6	0.00	256	0.00	monte_carlo_IP.assign_spin_deps_
* USR	2	0.00	256	0.00	monte_carlo_IP.check_input_parameters_
* USR	0	0.00	256	0.00	monte_carlo_IP.input_setup_
* USR	0	0.00	1	0.00	monte_carlo_IP.print_input_parameters_
* USR	0	0.00	62	0.00	monte_carlo_IP.wout_iparam_
* USR	0	0.00	1	0.00	monte_carlo_IP.print_particles_
* USR	107	0.03	256	0.00	monte_carlo_IP.read_wave_function_
* USR	2	0.00	256	0.00	alloc_shm_
* USR	0	0.00	256	0.00	alloc_shm_sysv
* USR	0	0.00	256	0.00	monte_carlo_IP.flag_missing_dets_

1.5 Supplementary utilities and helping Cube command line tools

*	USR	0	0.00	1	0.00	monte_carlo_IP_check_wave_function_
*	USR	36	0.01	256	0.00	monte_carlo_IP_calculate_geometry_
*	USR	27	0.01	256	0.00	monte_carlo_IP_read_pseudopotentials_
*	USR	0	0.00	256	0.00	monte_carlo_IP_geometry_printout_
*	USR	1	0.00	256	0.00	monte_carlo_IP_orbital_setup_
*	USR	0	0.00	256	0.00	monte_carlo_IP_check_orbital_derivatives_
*	USR	0	0.00	256	0.00	monte_carlo_IP_setup_expectation_values_
*	USR	0	0.00	256	0.00	monte_carlo_IP_setup_interactions_
*	USR	0	0.00	256	0.00	monte_carlo_IP_check_nn_
*	USR	98	0.03	256	0.00	monte_carlo_IP_read_jastrow_function_
*	USR	0	0.00	256	0.00	monte_carlo_IP_read_backflow_
*	USR	8	0.00	256	0.00	monte_carlo_IP_init_vcpp_
*	USR	0	0.00	256	0.00	monte_carlo_IP_print_part_title_
*	USR	281	0.08	256	0.00	monte_carlo_IP_run_dmc_
*	USR	11021	3.10	25041	0.00	dmcdmc.main.move.config_
	USR	2434	0.68	45059225	6.78	wfn_utils.wfn_loggrad_
	USR	236521	66.51	175964383	26.47	pjastrow.oneelec.jastrow_
	USR	9886	2.78	144014658	21.67	wfn_utils.wfn_ratio_
	USR	15497	4.36	144014658	21.67	slater.wfn_ratio_slater_
	USR	15512	4.36	144014658	21.67	scratch.get_eevecs1_ch_
	USR	9356	2.63	25041	0.00	energy_utils.eval_local_energy_
	USR	2967	0.83	11268450	1.70	non_local.calc_nl_projection_
	USR	7	0.00	6400	0.00	parallel.qmpi_reduce_d1_
	USR	0	0.00	6400	0.00	parallel.qmpi_bcast_d1_
	USR	0	0.00	12800	0.00	parallel.qmpi_bcast_d_
	USR	16	0.00	9382	0.00	dmc.branch_and_redist_
	USR	39	0.01	6144	0.00	dmc.branch_and_redist_send_
	USR	1	0.00	2982	0.00	dmc.branch_and_redist_recv_
	USR	4	0.00	256	0.00	dmc.branch_and_redist_sendrecv_
	USR	0	0.00	256	0.00	dealloc_shm_
	USR	445	0.13	256	0.00	dealloc_shm_sysv
	USR	0	0.00	256	0.00	clean_shm_
	ANY	355599	100.00	664724749	100.00	(summary) ALL
	MPI	0	0.00	0	0.00	(summary) MPI
	USR	355599	100.00	664724749	100.00	(summary) USR
	COM	0	0.00	0	0.00	(summary) COM
	BL	12291	3.46	159199	0.02	(summary) BL
	ANY	343308	96.54	664565550	99.98	(summary) ALL-BL
	MPI	0	0.00	0	0.00	(summary) MPI-BL
	USR	343308	96.54	664565550	99.98	(summary) USR-BL
	COM	0	0.00	0	0.00	(summary) COM-BL

Usage: cube_regioninfo [-l filename] [-m filename] [-h] [-s] cubefile

-h Help; Output this help message

-r be verbose, print info about any region

-l Specify a list of regions for the blacklist

-m Specify a list of metrics, the list has to be comma-seperated, no spaces allowed

1.5.7 Cube Information

Another tool which allows to examine performance data along the call tree with ability to filter accordingly to user defined criteria. Partially same functionality is implemented by [1.5.16](#) and call tree.

```

user@host: cube_info -t -m bytes_sent profile.cubex
|           Time |           bytes_sent | Diff-Calltree
|           355598 |          14204637004 | * MAIN__
|           1411 |              0 | | * MPI_Init
|              0 |              0 | | * MPI_Comm_size
|              0 |              0 | | * MPI_Comm_rank
|              0 |              0 | | * get_nnpsmp_
|              0 |              0 | | * run_control.check_alloc_
|              69 |           917504 | | * get_smp_list_
|              0 |              0 | | | * MPI_Comm_rank
|              0 |              0 | | | * MPI_Comm_size
|              9 |          262144 | | | * MPI_Allreduce
|              0 |              0 | | | * set_shm_numablock
|             27 |          655360 | | | * MPI_Allgather
|              0 |              0 | | | * MPI_Comm_group
|              0 |              0 | | | * MPI_Group_incl
|             25 |              0 | | | * MPI_Comm_create
|              0 |              0 | | | * MPI_Comm_group
|              0 |              0 | | | * MPI_Group_incl
|             24 |              0 | | | * MPI_Comm_create
|          353588 |          14203719500 | | * monte_carlo_
|              0 |              0 | | | * monte_carlo_IP_set_defaults_
|            142 |              0 | | | * monte_carlo_IP_set_input_parameters_
|              0 |              0 | | | | * run_control.check_alloc_
|             10 |              0 | | | | * monte_carlo_IP_read_particles_
|              0 |              0 | | | | | * run_control.check_alloc_
|              3 |              0 | | | | * monte_carlo_IP_read_custom_spindep_
|              5 |              0 | | | | * monte_carlo_IP_assign_spin_deps_
|              0 |              0 | | | | | * run_control.check_alloc_
|             174 |             1024 | | | * monte_carlo_IP_check_input_parameters_
|             172 |             1024 | | | | * MPI_Bcast
|              0 |              0 | | | * monte_carlo_IP_input_setup_
|              0 |              0 | | | * monte_carlo_IP_print_input_parameters_
|              0 |              0 | | | | * monte_carlo_IP_wout_iparam_
|              0 |              0 | | | * monte_carlo_IP_print_particles_
|             68 |             12288 | | | * MPI_Bcast
|              0 |              0 | | | * run_control.check_alloc_
|              0 |              0 | | | * MPI_Comm_group
|              0 |              0 | | | * MPI_Group_incl
|             23 |              0 | | | * MPI_Comm_create
|            8941 |          12811606784 | | * monte_carlo_IP_read_wave_function_
|            2792 |              0 | | | * MPI_Barrier
|             55 |              0 | | | * MPI_Comm_split
|            5980 |          12811605760 | | | * MPI_Bcast
|              0 |              0 | | | * run_control.check_alloc_
|              5 |             1024 | | | | * alloc_shm_
|              4 |             1024 | | | | | * alloc_shm_sysv

```


1.5.8 Tester

With the tool `cube_test` one can perform various test on `.cubex` file such as range test on metric values and similar. This is used usually in the correctness tests of Score-P and SCALASCA.

Usage: `cube_test [flags] <cube1> <cube2>`

- ?** Help; Output a brief help message.
- h** Help; Output a long help message.
- o** Set the output file. If omitted, STDOUT is used.
- c tree1[,tree2]** Create a new copy of the tree "tree2" that is called "tree1". If "tree2" is omitted, "tree1" will be a copy of the "ALL"
- e metric[,tree]** Checks whether the metric described by the string "metric" is equal for all provided CUBE files and for all nodes of the tree "tree". If tree is omitted, all nodes are checked.
- s metric,type,tol[,tree]** Checks whether the metric described by the string "metric" is sufficiently similar for all CUBE files, according to the qualifiers type and "tol". "tol" specifies the tolerance which is given as a real number. The meaning of the tolerance value depends on the "type". "type" is either ABS or REL. If "type" is ABS than the nodes are similar if the difference between all values for that metric is below "tol". If "type" is REL than the nodes are similar if the difference between all values for that metric is below: $\sum_{i=1}^n tol * v_i^m$, where n is the number of CUBE files, i-th file and V is value of metric. If "tree" is omitted, all nodes are checked.
- a metric,thrh[,tree]** Removes all nodes from the tree "tree" whose values are below "thrh" for the metric "metric". If tree is omitted, nodes are removed from the default tree ALL.
- r metric,thrh[,tree]** Removes all nodes from the tree "tree" whose values are below "thrh" percent of the sum of all roots values. "thrh" is supposed to be a value between 0 and 1. If "tree" is omitted, nodes are removed from the default tree ALL.
- x metric,thrh[,tree]** For each node of the tree tree we consider all children N of that node. Let v_i be the value for the given metric and the node i in N. Then we select a maximal subset M (smaller than N) with the property $\sum_{i \in N} v_i \leq thrh \times \sum_{i \in M} v_i$ and remove all nodes from M. If tree is omitted, nodes are removed from the default tree ALL.
- v [0,1,2,3]** Verbosity level: 0 - silent, 1 - report only summary, 2 - report only errors, 3 - report status of all tests

WARNING: The order of the options matters.

1.5.9 Canonizer

With this tool user can convert all regions into canonical form, names into lower case, removing lines and filenames, in order to be able to compare measurements results.

Here is an example of execution

```
cube_canonize -p -m 10 -c -f -l profile.cubex canonized.cubex
```

```
cube_dump -w profile.cubex
```

```
....
----- LIST OF REGIONS -----
MEASUREMENT OFF:MEASUREMENT OFF      ( id=0, -1, -1, paradigm=user, role=artificial, url=, descr=, mod=)
TRACE BUFFER FLUSH:TRACE BUFFER FLUSH ( id=1, -1, -1, paradigm=measurement, role=artificial, url=, descr=, mod=)
THREADS:THREADS                       ( id=2, -1, -1, paradigm=measurement, role=artificial, url=, descr=, mod=)
MPI_Accumulate:MPI_Accumulate         ( id=3, -1, -1, paradigm=mpi, role=atomic, url=, descr=, mod=)
...
PARALLEL:PARALLEL                     ( id=270, -1, -1, paradigm=mpi, role=artificial, url=, descr=, mod=)
MAIN__:MAIN__                         ( id=271, -1, -1, paradigm=compiler, role=function, url=, descr=, mod=)
get_nnpsmp_:get_nnpsmp_               ( id=272, -1, -1, paradigm=compiler, role=function, url=, descr=, mod=)
run_control.check_alloc_:run_control.check_alloc_ ( id=273, -1, -1, paradigm=compiler, role=function, url=, descr=, mod=)
get_smp_list_:get_smp_list_           ( id=274, -1, -1, paradigm=compiler, role=function, url=, descr=, mod=)
set_shm_numablock:set_shm_numablock   ( id=275, -1, -1, paradigm=compiler, role=function, url=, descr=, mod=)
monte_carlo_:monte_carlo_             ( id=276, -1, -1, paradigm=compiler, role=function, url=, descr=, mod=)
...

```

```
cube_dump -w canonized.cubex
```

```
----- LIST OF REGIONS -----
off:MEASUREMENT OFF                   ( id=0, 0, 0, paradigm=user, role=artificial, url=, descr=, mod=)
flush:TRACE BUFFER FLUSH              ( id=1, 0, 0, paradigm=measurement, role=artificial, url=, descr=, mod=)
threads:THREADS                       ( id=2, 0, 0, paradigm=measurement, role=artificial, url=, descr=, mod=)
mpi_accumu:MPI_Accumulate              ( id=3, 0, 0, paradigm=mpi, role=atomic, url=, descr=, mod=)
...
parallel:PARALLEL                     ( id=270, 0, 0, paradigm=mpi, role=artificial, url=, descr=, mod=)
main:MAIN__                           ( id=271, 0, 0, paradigm=compiler, role=function, url=, descr=, mod=)
get_nnpsmp_:get_nnpsmp_               ( id=272, 0, 0, paradigm=compiler, role=function, url=, descr=, mod=)
run_contro:run_control.check_alloc_   ( id=273, 0, 0, paradigm=compiler, role=function, url=, descr=, mod=)
get_smp_li:get_smp_list_              ( id=274, 0, 0, paradigm=compiler, role=function, url=, descr=, mod=)
set_shm_nu:set_shm_numablock          ( id=275, 0, 0, paradigm=compiler, role=function, url=, descr=, mod=)
monte_carl:monte_carlo_               ( id=276, 0, 0, paradigm=compiler, role=function, url=, descr=, mod=)
...

```

Usage: cube_canonize [flags] <input> <output>

- h** Print this help
- p** Use PDT format
- m <num>** Maximum length
- c** Lower case
- f** Remove file names
- l** Remove line numbers

1.5.10 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] [-c|-C] [-h] <cube experiment>

- o** Name of the output file (default: clean.cube|.gz)
- c** Reduce system dimension, if experiments are incompatible.
- C** Collapse system dimension! Overrides option -c.
- h** Help; Output a brief help message.

1.5.11 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").

One can have multiple **-p** and **-r** options, none of regions (in **-p** and in **-r** separately) should have eachother as parents and wrong regions names are ignored.

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] [-c|-C] [-r nodename] [-p nodename] [-o output] <cube experiment >

- r** Re-root calltree at named node(s)
- p** Prune calltree from named node(s) (= "inline")
- o** Name of the output file (default: cut)

- c Reduce system dimension, if experiments are incompatible.
- C Collapse system dimension! Overrides option -c.
- h Help; Output a brief help message.

1.5.12 Statistics

Extracts statistical information from the CUBE files.

```
user@host: ./cube_stat -m time,mpi -p remapped.cube -%
```

Metric	Routine	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>...]] <cube>

OR

[-t \<topN\>] \<cube>
-t topN cube</dd>

- 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). Use mangled name, if available.
- t** Number for topN regions flat profile

Output of the tool [1.5.16](#) as

```
cube_dump -w <profile.cubex>
```

lists next to the display name of the region its mangled name as well.

1.5.13 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]

1.5.14 Common Calltree

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

Usage: cube_commoncalltree [-h] <cube experiment> ...

-h Help; Output a brief help message.

The tool *cube_commoncalltree* takes set of input cubefiles

(*cubefile1 cubefile2 ... cubefileN*)

and creates corresponding set of cube files

(*cube1_commoncalltree cube2_commoncalltree ... cubeN_commoncalltree*).

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

Call trees among *cubeX_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 particularly useful for comparison of experiments with the different recursion depth or with the additional sub call trees depending on some loop iteration index.

1.5.15 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 [opts] <cube experiment>`

The current available options are:

- To manually create a new topology in an existing cube file,
- To automatically generate new topologies based on existing system tree data,
- To [re]name an existing virtual topology, and
- To remove an existing virtual topology, and
- To [re]name the dimensions of a virtual topology.

The command-line switches for this utility are:

- c:** create a new topology in the existing CUBE file.
- g:** generate global topologies in the existing CUBE file.
- n:** [re]name an existing topology
- r:** remove an existing topology
- d:** name dimensions of a topology
- o:** output CUBE file (default: `topologies.cubex`)

The default output is a `topologies.cube[.gz]` file in the current directory.

When using the **-d**, **-r** 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 topologies.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`

For the generation option **-g** the user can choose the following options that control the generation procedure:

- Use default parameters,
- include dimensions of size 1, and

- generate separate topologies on either machine or module level.

The default settings use the split into multiple topologies, when there are multiple machines or modules in the system tree hierarchy, and discard singular dimensions as those contribute no practical topology information and increase the dimensionality. This is the same default as the remapper uses for its non-interactive generation.

The maximum amount of dimensions that can be generated for CUBE results based on Score-P measurements without splitting of topologies or removal of dimensions are these:

- CPU: Machine x Module x Node x Process x Thread
- GPU: Machine x Module x Node x Device x Context x Stream
- GPU pre Score-P 8.0: Machine x Module x Node x Process x Stream

For most standard cases and default settings, that will likely result in something like "Node x Process x Thread" and similar. The case of CUBE files prior to Score-P 8.0 is a compatibility variant to allow for the different system tree layout of older measurements.

1.5.16 Dump

To export values from the cube report into another tool or to examine internal structure of the cube report CUBE framework provides a tool `cube_dump` tool, which prints out different values. It calculates inclusive and exclusive values along metric tree and call tree, aggregates 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.

Usage: `cube_dump [-m <metrics>|<new metrics>|all|<filename>] [-c <cnode ids>|all|leafs|roots|level>X|level=X|level<X|name=/regexp/|<filename>] [-x incl|excl] [-z incl|excl|stored] [-t thread_id|aggr] [-r]] [-f name] [-w] [-g] [-s human|gnuplot|csv|csv2] [-h|-?] [-o output] <cube experiment>`

-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 of with the 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.

-g Show ghost metrics (only in combination with -w).

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

-s human|gnuplot|gnuplot2|csv|csv2|csv3|(R) Uses either human readable form, GNUPLOT or CSV format (both with different layouts) or binary R matrix (in case R is installed) 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 20 -z incl -s csv3 profile.
Cnode ID,Region Name, Thread ID,New Metric1,New Metric2,time
20,"cuDeviceGetUuid",0,5.29572e-07,1,5.29572e-07
20,"cuDeviceGetUuid",1,0,0,0
20,"cuDeviceGetUuid",2,0,0,0
20,"cuDeviceGetUuid",3,9.14558e-06,1,9.14558e-06
20,"cuDeviceGetUuid",4,0,0,0
20,"cuDeviceGetUuid",5,0,0,0
```

Example 7:

```
$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. It 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.cubep1") :

```
{
${a}=0;
if (${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.cubep1 \  
-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.

==== BASE ====

==== BASE ====

