# OTF Profile and OTF Shrink – Automatic Substitution of Similar Process Traces

Andreas Knüpfer, Matthias Jurenz, Robert Dietrich, Guido Juckeland, André Grötzsch

May 22, 2012

## 1 Overview

This document describes an approach to automatically classify similarities in parallel *trace processes* and exploit them for data reduction. This is particularly promising for massively-parallel traces. The current tool infrastructure is designed for a step-by-step evaluation of this approach and might be adapted in the future according to the lessons learned.

Terminology: In this document, the term *trace processes* denotes processes in an event trace which is under investigation. In contrast, the term *analysis processes* denotes the processes in a parallel analysis tool which handle the data in the *trace processes*.

## 2 Command Line Tools

The approach uses the following two command line tools:

First, `otfprofile[-mpi]` is collecting summarized profile information where the MPI-parallel version of this tool uses $n$ *analysis processes* to collect profile data for $m \geq n$ *trace processes*.

By default, it creates a summary report as LaTeXor PDF document. As an alternative mode, it can perform a K-means clustering based on the per-process profile data, see Section **??** for more details.

```
otfprofile[-mpi] - generate a profile of a trace in LaTeX format.

Syntax: otfprofile[-mpi] [options] <input file name>

  options:
     -h, --help           show this help message
     -V                   show OTF version
     -v                   increase output verbosity
                          (can be used more than once)
     -p                   show progress
     -f <n>               max. number of filehandles available per rank
                          (default: 50)
     -b <size>            set buffersize of the reader
                          (default: 1048576)
     -o <prefix>          specify the prefix of output file(s)
                          (default: result)
     -g <n>               max. number of process groups in LaTeX output
                          (range: 1-16, default: 16)
     -c                   do additional K-means clustering of processes/threads
     -m <mapfile>         write process mapping to <mapfile> (implies -c)
                          (default: result.map)
     -s <prefix>          call otfshrink to apply the cluster mapping to
                          input trace and produce a new trace named <prefix>
```

```
                        with symbolic links to the original (implies -c)
    --stat              read only summarized information, no events
    --[no]csv           enable/disable producing CSV output
                        (default: disabled)
    --[no]tex           enable/disable producing LaTeX output
                        (default: enabled)
    --[no]pdf           enable/disable producing PDF output
                        (implies --tex if enabled, default: enabled)
```

Second, `otfshrink` is able to apply given reduction instructions to an existing event trace. With the `-s` switch it will be called from `otfprofile[-mpi]` automatically. See Section **??** for the details and representation of the reduced data.

```
otfshrink  -  creates a new otf file that only includes
              specified processes

  options:
      -h, --help    show this help message
      -V            show OTF version
      -i <file>     specify the input trace file
      -o <file>     specify the output file
      -l "<list>"   a space-separated list of processes in quotes
                    to enable, i.e. keep in the copy,
                    e.g. '-l "1 2 3 4 8 5"'
      -v            invert setting from '-l',
                    i.e. deactivate/exclude listed processes
      -m "<list>"   map all listed processes to one representative
                    and remove all remaining ones
                    first process in list is the representative,
                    must not be mixed with '-l' and '-v'
      -f <file>     read multiple '-m' lists from the given file
                    one list/group per line, empty lines allowed
      -s <mode>     simulation mode: display all selected
                    processes, no files are created,
                    display modes: (l)ist, (r)ange or (t)able
                    defaut: range
      -p <file>     displays all processes with name and id
                    input file without ".otf"

   Multiple instances of '-l', '-m', and '-f' may be used
```

For both tools, see also the general documentation at `./docu/tools/otftools.pdf`.

# 3 K-means Clustering of Properties of Process Traces

The `otfprofile[-mpi]` tool collects profile information about subroutine calls on parallel *trace processes*. This includes the number of calls and the total exclusive runtime per subroutine. Both are stored in two feature vectors per process $p$: vector $C_p$ contains the number of calls per subroutine, vector $T_p$ contains the exclusive runtime sum per subroutine. All trace processes use the same enumeration of subroutines to create the vectors. The vectors are the input for a two-stage K-means clustering of trace processes which is performed by `otfprofile[-mpi]`.

In the first stage, the processes $p \in P$ are divided into groups $G_j$ with identical $C_p$ in every group. A differing number of calls for at least one subroutine is an indicator for differing behavior. This happens for example when the master rank performs additional work compared to the regular parallel tasks of all ranks.

In the second step, the processes in each group $p \in G_j$ are subject to a k-means clustering according to the runtime vectors $T_p$. Groups with $\leq 5$ members are ignored. The pre-specified number of clusters is

$\min(n/3, 16)$ with $n = |G_j|$. This looks at the runtime distribution in the $T_P$ vectors for all processes in a group and computes a clustering with (heuristically) maximum similarity according to the K-means algorithm. The result is written to the file `result.map` unless the `-m` command line switch is used.

# 4 Complete-linkage Clustering

Since for K-means clustering the number of clusters is required, what is in general hard to determine automatically, the results are often unsatisfying. Hence the Complete-linkage method is provided as an alternative approach by the command line option `--cluster CLINKAGE`. In contrast to the K-means clustering method, this approach doesn't require any knowledge of the number of the resulting clusters, since this is determined by the algorithm.

Using the same input data as mentioned in the previous section, the clustering is done in an agglomerative hierarchical manner. By means of Complete-linkage, the distance of two clusters is defined by the maximum distance between a pair of objects taken from the first resp. the second cluster. Thus the resulting clusters are rather compact, i. e. their diameter doesn't go beyond a chosen threshold. With this threshold the quality of the resulting clusters can be influenced. The choice of the threshold is made by the command line parameter `-q` in a normalised manner: $0 \leq q \leq 1$. A maximum diameter of zero means that only processes with equal exclusive runtimes per subroutine will end up in a cluster. Contrary, a maximum diameter of one will force all trace processes into one single cluster. A reasonable value for $q$ is something in-between ($q = 0.1$ per default). As with K-means clustering, the result is written to the file `result.map` unless the `-m` command line option is used.

As a downside of the Complete-linkage method we have to mention that the quadratic complexity of the algorithm in time and space suggests to be careful when applying this method for a very large number of processes. The smartest way to handle appearing problems with time or memory is to decrease $q$ - the maximum diameter of clusters as described above. This results in less processes to be considered for clustering and thus less memory and time is required to compute the clusters. This will be done automatically in a future release.
An additional option to handle such problems is to consider only trace processes with identical number of calls per subroutine for clustering as described as stage one in the previous section. This is not done by default but can be forced by the command line switch `-H`.

# 5 Reduction of Event Traces by Substitution of Process Traces

The resulting clustering from `otfprofile[-mpi]` can be fed into the existing `otfshrink` tool to produce a reduced version of the input trace. From every K-means cluster one representing process (the one with the lowest ID) is designated as a substitute for all remaining members of the cluster which are discarded.

## Modifications on the File Level

OTF is a format with multiple files per event trace which are handled differently in `otfprofile[-mpi]`. The master file or anchor file `<prefix>.otf` is newly generated and references only the representing processes (as well as individual processes not assigned to a cluster). The global definition file `<prefix>.0.def[.z]` is also newly generated and modified. Both newly created files are usually very small in comparison to the event stream files `<prefix>.<streamid>.events[.z]` . Event stream files contain event records for one or several processes and are usually much larger than the anchor and definition files.

If an event stream of the input trace contains at least one process that remains in the output trace, then a symbolic link `<new prefix>.<streamid>.events[.z]` is created. Thus, the data is available in the reduced trace without the need for extensive copy operations. However, independent renaming or moving of the input and output traces is limited. Since re-creating the reduced trace with the stored mapping file from `otfprofile[-mpi]` is a quick operation, this a comparatively small limitation.

### Modifications on the Record Format Level

In the mapping mode (specified with `-m`) the `otfshrink` tool performs a number of modifications to definition records but not to event records.

First of all, a special `TraceContainsSubstitutes` definition record is inserted in the very beginning as a flag that a trace contains substitute information and differs from standard traces.

Furthermore, all *process definition records* of substituted processes are removed. Directly before the definition records of remaining processes, a *process substitution record* is written which enumerates all other processes to be represented by a given process. (The representing processes are included in the list for convenience).

All other definition records that reference IDs of substituted processes are not modified, because this would complicate the decompression. For example, process group definitions representing MPI communicators are not modified and still contain substituted process. Analysis tools that read traces with substitution information but without decompression need to tolerate invalid process tokens.

## 6 Further Handling of Reduced Event Traces by Analysis Tools

For the analysis of substitution-mode traces by consumer tools, there are two scenarios.

In the first scenario, the consumer explicitly considers substitution information and reads the reduced trace. By means of the substitution definition records, it s able to transfer observations based on the representing processes to the substituted processes which are assumed to be *similar enough* (TM). This scenario directly passes on the advantage of the reduced data volume to the consumer.

In the second scenario, the consumer relies on the OTF library to decompress the reduced representation. For this, OTF provides an special set of call back handlers (`SubstitutionHandlers`) which the consumer has to register to the OTF reader object. The `SubstitutionHandlers` will deliver the uncompressed information to another set of call back handlers that pass it to the consumer like in the standard OTF read mode.

The `SubstitutionHandlers` will consume the process substitution definitions and store them internally. In the following, all records for representing processes are multiplied – they are delivered to the consumer several times on behalf of the representing process and on behalf of all processes represented by it. Assumed the representing processes are similar enough, this recreates the original trace in a *similar* way. However, it also restores the original data size. Therefore, the read speed may increase but the over-all data volume delivered to the consumer is the same as for the original trace.

## 7 Example

An example with a trace of NAS parallel benchmarks BT with 64 ranks is shown below. The command

```
mpirun -np 4 otfprofile-mpi -p --nopdf nas_bt_64.otf -s reduced
```

produces the following mapping:

```
3 : 4 5 6 23
9 : 17 22 25
11 : 12 13 46
14 : 52
15 : 59 60 61 63
18 : 24 26 32 34 40 48
19 : 27 38
20 : 31 39 45
21 : 28 35 53 54 55
29 : 33 36 41 62
30 : 37 44
43 : 51
```

The processes at the beginning of each line represent all processes after the colon. All missing process IDs remain separate.

In the present working directory, it produces the situation shown in Figure **??**. The reduced version uses symbolic links to selected files of the original and adds almost nothing to the total storage size of this direcrtory.

In Figure **??**, both traces are shown in Vampir. While the original (Figure **??** top) reflects the regular visualization, the reduced counterpart (Figure **??** bottom) has random gaps which produce a irregular impression. Note that processes representing substituted processes are marked with a `#<number>` in their name, where the number shows the size of their cluster.

Future versions of Vampir will be able to restore the regular visualization by interpreting the substitute information and filling the gaps with duplicated information from the representing processes.

# 8 Future Extensions

Future extensions to this experimental approach will include more statistical input data for the clustering process, e.g. MPI message sizes, I/O data volumes, etc. This will help to differentiate otherwise similar behavior of processes.

In addition, adaptive heuristics for the number of K-means clusters will be integrated to improve the simple static number of clusters used so far.

```
total 94M
drwxr-xr-x  2  12K .
drwxr-xr-x 26 4.0K ..
-rw-r--r--  1 4.3K nas_bt_64.0.def.z
-rw-r--r--  1   29 nas_bt_64.0.marker.z
-rw-r--r--  1 1.5M nas_bt_64.10.events.z
-rw-r--r--  1 1.5M nas_bt_64.11.events.z
-rw-r--r--  1 1.5M nas_bt_64.12.events.z
-rw-r--r--  1 1.5M nas_bt_64.13.events.z
-rw-r--r--  1 1.5M nas_bt_64.14.events.z
-rw-r--r--  1 1.5M nas_bt_64.15.events.z
-rw-r--r--  1 1.5M nas_bt_64.16.events.z
-rw-r--r--  1 1.5M nas_bt_64.17.events.z
-rw-r--r--  1 1.5M nas_bt_64.18.events.z
-rw-r--r--  1 1.5M nas_bt_64.19.events.z
-rw-r--r--  1 1.5M nas_bt_64.1a.events.z
-rw-r--r--  1 1.5M nas_bt_64.1b.events.z
-rw-r--r--  1 1.5M nas_bt_64.1c.events.z
-rw-r--r--  1 1.5M nas_bt_64.1d.events.z
-rw-r--r--  1 1.5M nas_bt_64.1e.events.z
-rw-r--r--  1 1.5M nas_bt_64.1.events.z
-rw-r--r--  1 1.5M nas_bt_64.1f.events.z
-rw-r--r--  1 1.5M nas_bt_64.20.events.z
-rw-r--r--  1 1.5M nas_bt_64.21.events.z
-rw-r--r--  1 1.5M nas_bt_64.22.events.z

  <skip some>

-rw-r--r--  1 1.5M nas_bt_64.3d.events.z
-rw-r--r--  1 1.5M nas_bt_64.3e.events.z
-rw-r--r--  1 1.5M nas_bt_64.3.events.z
-rw-r--r--  1 1.5M nas_bt_64.3f.events.z
-rw-r--r--  1 1.5M nas_bt_64.40.events.z
-rw-r--r--  1 1.5M nas_bt_64.4.events.z
-rw-r--r--  1 1.5M nas_bt_64.5.events.z
-rw-r--r--  1 1.5M nas_bt_64.6.events.z
-rw-r--r--  1 1.5M nas_bt_64.7.events.z
-rw-r--r--  1 1.5M nas_bt_64.8.events.z
-rw-r--r--  1 1.5M nas_bt_64.9.events.z
-rw-r--r--  1 1.5M nas_bt_64.a.events.z
-rw-r--r--  1 1.5M nas_bt_64.b.events.z
-rw-r--r--  1 1.5M nas_bt_64.c.events.z
-rw-r--r--  1 1.5M nas_bt_64.d.events.z
-rw-r--r--  1 1.5M nas_bt_64.e.events.z
-rw-r--r--  1 1.5M nas_bt_64.f.events.z
-rw-r--r--  1  354 nas_bt_64.otf
-rw-r--r--  1  13K reduced.0.def
lrwxrwxrwx  1   54 reduced.10.events.z -> nas_bt_64.10.events.z
lrwxrwxrwx  1   54 reduced.12.events.z -> nas_bt_64.12.events.z
lrwxrwxrwx  1   54 reduced.13.events.z -> nas_bt_64.13.events.z
lrwxrwxrwx  1   54 reduced.14.events.z -> nas_bt_64.14.events.z
lrwxrwxrwx  1   54 reduced.15.events.z -> nas_bt_64.15.events.z
lrwxrwxrwx  1   54 reduced.1d.events.z -> nas_bt_64.1d.events.z
lrwxrwxrwx  1   54 reduced.1e.events.z -> nas_bt_64.1e.events.z
lrwxrwxrwx  1   53 reduced.1.events.z -> nas_bt_64.1.events.z
lrwxrwxrwx  1   54 reduced.2a.events.z -> nas_bt_64.2a.events.z
lrwxrwxrwx  1   54 reduced.2b.events.z -> nas_bt_64.2b.events.z
lrwxrwxrwx  1   53 reduced.2.events.z -> nas_bt_64.2.events.z
lrwxrwxrwx  1   54 reduced.2f.events.z -> nas_bt_64.2f.events.z
lrwxrwxrwx  1   54 reduced.31.events.z -> nas_bt_64.31.events.z
lrwxrwxrwx  1   54 reduced.32.events.z -> nas_bt_64.32.events.z
lrwxrwxrwx  1   54 reduced.38.events.z -> nas_bt_64.38.events.z
lrwxrwxrwx  1   54 reduced.39.events.z -> nas_bt_64.39.events.z
lrwxrwxrwx  1   54 reduced.3a.events.z -> nas_bt_64.3a.events.z
lrwxrwxrwx  1   53 reduced.3.events.z -> nas_bt_64.3.events.z
lrwxrwxrwx  1   54 reduced.40.events.z -> nas_bt_64.40.events.z
lrwxrwxrwx  1   53 reduced.7.events.z -> nas_bt_64.7.events.z
lrwxrwxrwx  1   53 reduced.8.events.z -> nas_bt_64.8.events.z
lrwxrwxrwx  1   53 reduced.9.events.z -> nas_bt_64.9.events.z
lrwxrwxrwx  1   53 reduced.a.events.z -> nas_bt_64.a.events.z
lrwxrwxrwx  1   53 reduced.b.events.z -> nas_bt_64.b.events.z
lrwxrwxrwx  1   53 reduced.e.events.z -> nas_bt_64.e.events.z
lrwxrwxrwx  1   53 reduced.f.events.z -> nas_bt_64.f.events.z
-rw-r--r--  1  136 reduced.otf
```

Figure 1: Listing of working directory after the clustering and shrinking steps.

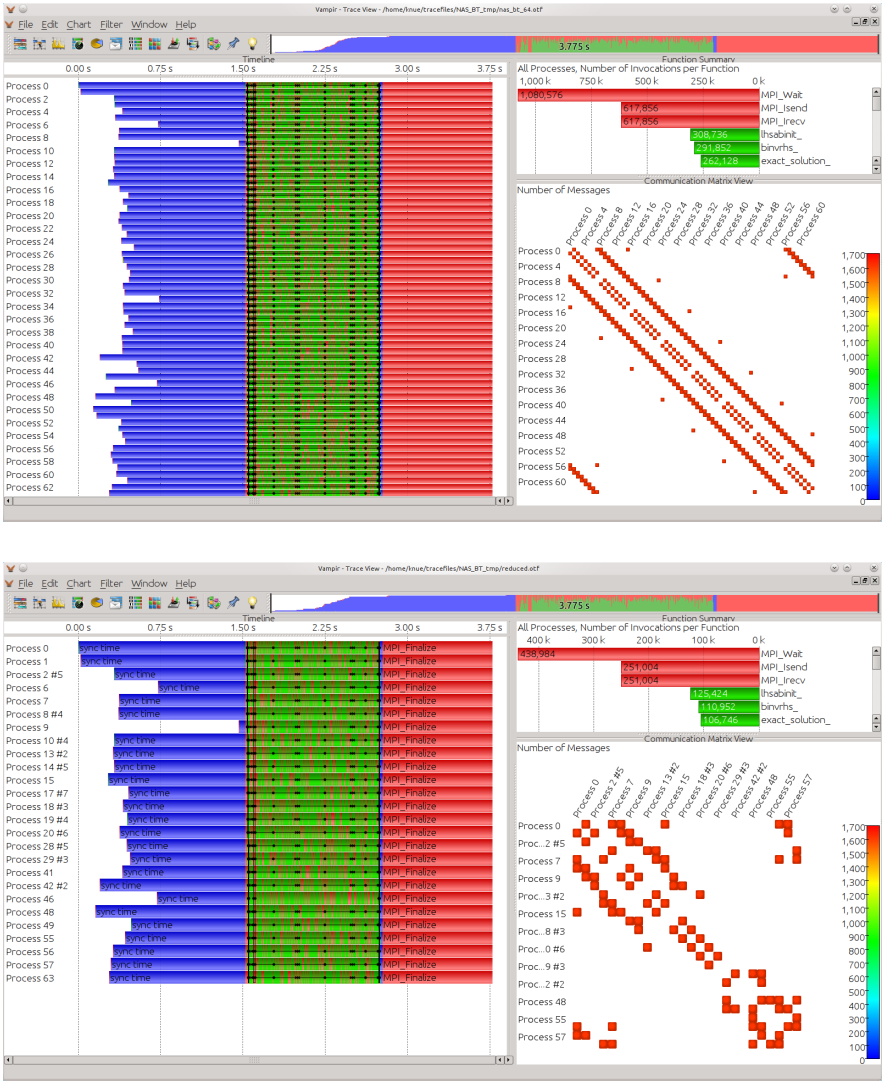


Figure 2: Comparison of the Vampir vissualization of the original trace (top) and the reduced one (bottom).