A Short Guide for Analyzing OpenMP Traces with Aftermath

Andi Drebes

This is a short guide for analyzing OpenMP traces with Aftermath.

Table of Contents

1 Analyzing OpenMP Traces

This short guide provides a quick overview of how to analyze traces generated by OpenMP programs using our tool for trace-based analysis of parallel programs, called *Aftermath*. This involves the Aftermath tool itself, for visualizing and analyzing traces, and the Aftermath OpenMP run-time for generating trace files from OpenMP programs. We first show what a user can expect from both tools by summarizing the status of the OpenMP support in Aftermath and Aftermath-OpenMP. After the instructions for the configuration and installation of Aftermath and Aftermath-OpenMP and a short summary of Aftermath's terminology for OpenMP, we present short, self-contained examples for the analysis of OpenMP programs.

1.1 Status of the OpenMP Support

Aftermath has originally been designed for the analysis of dependent tasks in OpenStream. However, as many of the analyses for OpenStream programs are useful for programs from different parallel frameworks, in particular OpenMP, Aftermath has recently been extended to support traces generated by OpenMP programs. Complementary to the OpenMP support in Aftermath, we provide Aftermath-OpenMP, an instrumented OpenMP run-time library that collects performance data at execution time and that generates a trace file at program termination. This freely available run-time is based on the LLVM/clang OpenMP run-time, which in turn is based on the Intel OpenMP run-time. OpenMP programs executed with Aftermath-OpenMP and analyzed with Aftermath may use any construct implemented by the LLVM/clang OpenMP run-time, but tracing is restricted to the following features:

- Parallel loops can have a static, dynamic or guided schedule and may be chunked (explicitly or implicitly). The iteration space can be signed or unsigned, but induction variables may have at most 64 bits. Currently, only single loops are supported, multiple loops combined with a collapse clause cannot be traced properly.
- The support for tasks is currently limited to independent tasks. Dependent tasks may be used, but the run-time system does not capture data dependencies and Aftermath can only display and analyze independent tasks.
- Parallel loops and tasks may be mixed without restrictions. Parallel regions may be nested.
- As Aftermath-OpenMP captures data on a per-core basis, worker threads must be pinned to cores to avoid that the same worker traces data for multiple cores. In addition, the number of workers must be less than or equal to the number of available cores to avoid that two or more workers trace data for the same core.
- The Aftermath-OpenMP run-time requires that programs must be compiled with `clang` version 3.8.0 or higher. Any level of optimization by the compiler is allowed, but debugging symbols must be available in the final executable.

The latter restrictions regarding the mapping of workers to cores, the supported versions of LLVM/clang and the presence of debugging symbols are automatically verified and enforced by the `aftermath-openmp-trace` command distributed with the Aftermath-OpenMP run-time.

1.2 Installing Aftermath

Aftermath is distributed with the git version control system and can be cloned from our public git repository at `git://git.drebesium.org` (please make sure to clone the staging branch):

```
$ git clone git://git.drebesium.org/aftermath.git -b staging
```

The installation and configuration of Aftermath is done using the GNU Autotools. The `bootstrap` script provided by the Aftermath sources invokes the Autotools in the right order and with the appropriate parameters:

```
$ cd aftermath
$ ./bootstrap
```

This initial bootstrapping step generates the `configure` script, which checks for the presence of software packages required by Aftermath and that configures the source files. This script is invoked by executing:

```
$ ./configure
```

If all of the required packages are present, `configure` generates a make file with rules to build and install the Aftermath. The rules for the compilation and installation are interpreted by the `make` utility:

```
$ make
```

To speed up compilation on multicore systems, you may use the `-j` option of `make`. For example, if your machine has four processors, you can instruct `make` to run up to four compilation jobs simultaneously by running `make -j4`. The installation step usually requires super user rights and must be ran directly from a super user shell, e.g., from a shell spawned by the `su` command, or indirectly using a program capable of elevating rights, such as `sudo`:

```
$ su
# make install
```

or

```
$ sudo make install
```

If you do not want to install Aftermath system-wide or if you want to install the program as a non-privileged user, you can specify an alternate prefix during the configuration by invoking `configure` with the `--prefix` argument. The following sequence of commands installs Aftermath into a folder named `extra` within your home directory:

```
$ ./configure --prefix=$HOME/extra
$ make
$ make install
```

We strongly recommend that you run the unit tests provided by the Aftermath sources. The `check` target of the previously generated make file executes these tests automatically:

```
$ make check
```

In addition, you may also run the tests with the `valgrind` utility (if available on your system) by invoking the `valgrind-check` target:

```
$ make valgrind-check
```

1.3 Installing the Aftermath OpenMP Run-time

In order to generate traces from an OpenMP program, it is necessary to install the instrumented Aftermath OpenMP run-time. The sources of the run-time are available from another repository on the same public git server. Start by cloning the repository with `git` (again, please make sure that you clone the staging branch):

```
$ git clone git://git.drebesium.org/aftermath-openmp.git -b staging
```

Next, configure the run-time using the `cmake` command. If you have not used any custom prefix during the configuration of Aftermath, it is sufficient to run `cmake` without any additional parameter:

```
$ cd aftermath-openmp
$ cmake .
```

Otherwise, you need to indicate where the Aftermath libraries and include files can be found. This can be done using the *CMAKE_LIBRARY_PATH* and *CMAKE_INCLUDE_PATH* variables. You may also indicate a prefix for the installation of the Aftermath OpenMP run-time through the *CMAKE_INSTALL_PREFIX:PATH* variable.

```
$ cd aftermath-openmp
$ cmake -DCMAKE_LIBRARY_PATH=/path/to/aftermath-installation/lib \
        -DCMAKE_INCLUDE_PATH=/path/to/aftermath-installation/include \
        -DCMAKE_INSTALL_PREFIX:PATH=/where/to/install/the/run-time \
        .
```

Note that there is a dot at the end of the `cmake` commands. To build and install the run-time, simply run:

```
$ make
$ make install
```

You are now ready to generate traces with the instrumented run-time by using the `aftermath-openmp-trace` command. Remember that any program must be compiled with a recent version of `clang` (>= 3.8.0) and that debugging symbols must be embedded into the executable using the `-g` switch. You may check if the installation is correct by running the tests provided in the `aftermath-tests` folder:

```
$ cd aftermath-tests
$ make all-traces
```

This generates a series of traces in `aftermath-tests/traces` that can be opened with Aftermath, e.g.:

```
$ aftermath traces/for-static.ost
```

1.4 Terminology and Concepts

In the analysis of loops and tasks, we will use the following terms in addition to the terminology of the OpenMP specification: *iteration set*, *iteration period*, and *task period*. We further refer to dynamic instances of loops and tasks simply as *loops* and *tasks*. These terms are usually associated with constructs in the source code, but overloading them allows for a simple terminology for dynamic analysis and remains unambiguous within the respective context. If a distinction with constructs and instances is necessary, we use the terms *loop construct* and *task construct*.

1.4.1 Iteration sets

We define an iteration set as a (not necessarily contiguous) portion of the iteration space of a parallel loop. This lets us define the iterations assigned to a worker in a generic way, independently from the loop's schedule and chunking. To illustrate this term, consider the following example with a single parallel loop, executed by four worker threads:

```
set_omp_num_threads(4);

#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++)
  do_something(i);
```

The static schedule and the chunk size of 10 cause chunks of ten iterations to be assigned to the four workers in a round-robin fashion. That is, the first worker executes iterations 0–9, 40–49, and 80–89, the second worker executes iterations 10–19, 50–59, 90-99, and so on. The iteration set defining all iterations executed by the first worker is thus $\{0,\ldots,9\}\cup\{40,\ldots,49\}\cup\{80,\ldots,89\}$. Aftermath displays iteration sets as lists of intervals, e.g., [0, 9], [40, 49], [80, 89].

1.4.2 Iteration periods and task periods

The execution of an iteration set can be interrupted, e.g., if the worker hits a nested barrier within a parallel loop. This is the case in the following example between the calls to `do_something` and `do_something_else`:

```
#pragma omp parallel for schedule(static)
for(int i = 0; i < 100; i++) {
  do_something(i);

  #pragma omp task
```

```
  a_task();

  #pragma omp task
  another_task();

  #pragma omp taskwait

  do_something_else(i);
}
```

We refer to contiguous periods of execution of an iteration set as an iteration period. As the execution of an iteration set might be interrupted multiple times, multiple iteration periods might be associated to a single iteration set. In the example above each iteration has two iteration periods: the first iteration period contains the call to `do_something` and the creation of the two tasks and ends at the taskwait barrier, while the second period corresponds to the code executed after the barrier, with the call to `do_something_else`.

Note that information about the progress of the execution of an iteration set is generally not captured by the run-time system. Hence, Aftermath can only determine which iterations belong to an iteration set, but it cannot determine which iterations have been executed during a specific iteration period of an iteration set. For example, when executing the code of the following listing, the run-time does not necessarily capture that the first iteration period of each iteration set comprises the first three iterations and that the second iteration period is composed of the last seven iterations:

```
#pragma omp parallel for schedule(static, 10)
for(int i = 0; i < 100; i++) {
  do_something(i);

  if(i % 10 == 3) {
    #pragma omp task
    a_task();

    #pragma omp taskwait
  }

  do_something_else(i);
}
```

Similar to iteration sets task execution may be interrupted by barriers. We thus define a task period as a contiguous periods of execution of a task.

1.5 Analyzing Parallel Loops

In the following analyses of OpenMP programs, we consider different implementations of a program that calculates the amount of prime numbers in an interval using a naive prime test. We start the analysis with the implementation below, based on a statically scheduled loop:

```
#include <stdio.h>
#include <math.h>

int isprime_naive(int n)
{
  if(n % 2 == 0 && n != 2)
    return 0;

  for(int j = 3; j <= sqrt(n); j += 2)
    if(n % j == 0)
      return 0;
```

```
  return 1;
}

int main(int argc, char** argv)
{
  int n = 1;

  #pragma omp parallel
  {
    #pragma omp for schedule(static) reduction(+:n)
    for(int i = 3; i < 1000000; i += 2)
      n += isprime_naive(i);
  }

  printf("There are %d prime numbers in the interval\n", n);

  return 0;
}
```

Paste the listing above to a file named `prime_naive.c` or copy the file from the `doc/examples` folder of the Aftermath source tree and compile the program using `clang` version 3.8.0 (or later) with debugging symbols:

```
$ clang -fopenmp -g -o prime_naive prime_naive.c -lm
```

Next, generate a trace file using the `aftermath-openmp-trace` program from the Aftermath-OpenMP run-time and open the trace file in Aftermath:

```
$ aftermath-openmp-trace -o prime_naive.ost -f -- ./prime_naive
$ aftermath prime_naive.ost
```

After starting Aftermath, you will notice that the time line appears to be empty. This is because on startup the time line is in so-called *state mode*, indicating the different run-time and application states each worker traverses over time. The Aftermath OpenMP run-time traces states for barriers, critical regions, single and master constructs, but does not associate a state to parallel loops. The reason for the absence of a loop state is that states only capture very limited information (an interval), whereas more detailed information is required for the analysis of parallel loops. Therefore, Aftermath provides specific time line modes for loops, which do not only provide a quick visual overview of loop executions, but that also let the user select portions of loop executions to obtain accurate information about loop characteristics. There are four loop-specific time line modes:

- The *loop construct mode* assigns a different color to each loop construct and visualizes iteration intervals with the color associated to their loops' constructs.
- In *loop mode* Aftermath assigns a different color to each instance of a loop. That is, if the same loop construct is executed twice, the intervals associated to each execution are visualized using a different color.
- The *iteration set mode* uses a different color for each iteration set. This means that all iteration periods of the same iteration set are visualized using the same color.
- In *iteration period mode* the tool associates a different color to each iteration period, such that the iteration periods of a given iteration set can be distinguished visually on the time line.

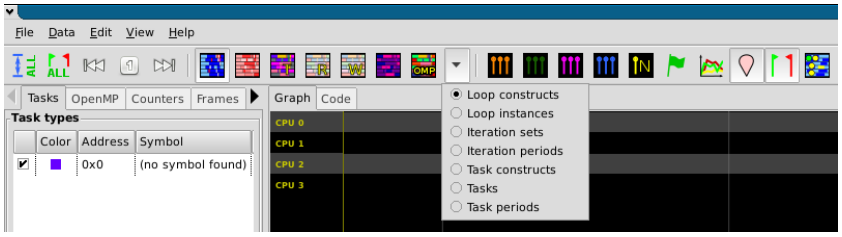

Figure 1.1: Selection of the time line mode for the visualization of parallel loops

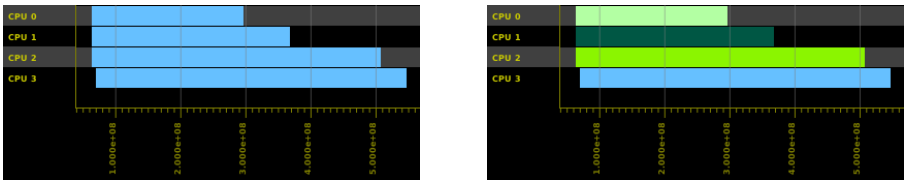

Figure 1.2: The time line in loop construct mode (left) and iteration set mode (right)

Loop-specific modes can be selected by using the drop down menu next to the OpenMP button in the tool bar, as shown in Figure 1.1. Figure 1.2 shows the time line in loop construct mode (left) and iteration set mode (right) for the `prime_naive` application. As the loop in the application is executed only once, the trace contains only one loop per loop construct. Hence, the loop mode yields the same visualization as the loop construct mode and is therefore omitted in this document. Similarly, the iteration set mode is identical to the iteration period mode, since the execution of an iteration set is never interrupted, such that there is exactly one iteration period per iteration set. As all workers are involved in the schedule and as all of them execute the same loop, the loop construct mode shows the same color for all workers. The iteration set mode displays a different color for each worker, since each worker is provided with its own part of the iteration space and thus its own iteration set.

Both visualizations immediately reveal that there is a strong imbalance between workers. This is the result of an inappropriate use of the static schedule: each worker executes the same number of iterations, but the amount of work grows with each iteration, leaving the workers with very different workloads. The partitioning of the iteration space and the amount of work can be inspected by clicking on an iteration period on the time line.

The contents of the OpenMP loop tab in the detailed text view below the time line for each worker is shown in Figure 1.3. As expected, the frames for loop constructs and for loops show the same information for all workers. The loop construct frame states that each worker has executed the same loop ("For addr: 0x400d60"), with a static schedule ("Schedule: static") and without a specified chunk size ("Chunked: No"). The loop frame indicates that the iteration space is represented by the interval $[0, 499998]$ with an increment of one. This does not reflect the original specification in the source code, which indicated the interval $[3, 999999]$ with an increment of two. However, the trace file is generated at execution time and thus after all transformations by the compiler. Hence, the iteration space and increment shown in Aftermath reflect the transformed loop characteristics. In addition to this information, the tab also indicates the number of workers involved in the execution of the loop ("#Workers: 4"), the number of chunks ("#Chunks: 4"), the number of iteration sets ("#Iteration sets: 4"), the wall clock time from the beginning of the first iteration period to the end of the last iteration period ("Wall clock time: 481.70 Mcycles"), the total, aggregated time spent by all workers ("Total time: 1.45 Gcycles") as well as the chunk load balance and the parallelism efficiency. The chunk load imbalance is calculated as follows:

$$B_{\text{itset}} = 1 - \frac{\max\{t_{\text{tot}}^{\text{itset}}\} - \text{avg}\{t_{\text{tot}}^{\text{itset}}\}}{\max\{t_{\text{tot}}^{\text{itset}}\}}$$

where $\max\{t_{\text{tot}}^{\text{itset}}\}$ and $\text{avg}\{t_{\text{tot}}^{\text{itset}}\}$ are the maximum and average total durations of the iteration sets, respectively. A value close to one indicates that there is little difference between those values and that the partitioning of work leads to iteration sets with similar durations. A value lower than one indicates an imbalance between iteration sets. The parallelism efficiency is defined as:

$$E_{\text{par}} = \frac{t_{\text{tot}}^{\text{loop}}}{t_{\text{wct}}^{\text{loop}} \cdot N_{\text{cpus}}}$$

with $t_{\text{tot}}^{\text{loop}}$ being the total, aggregated time spent in the loop by all workers, $t_{\text{wct}}^{\text{loop}}$ being the wall clock time from the start of the loop to its end, and N_{cpus} being the number of cores involved in the execution. A value close to one indicates that the total amount of work of the loop could be distributed uniformly among the workers, while a value closer to zero indicates that the loop needed more time to execute that the total amount of work divided by the number of workers, indicating a load imbalance. Note that the chunk load balance and the parallelism efficiency can differ substantially. For example, if a loop contains a barrier, the chunk balance can be high, while the parallelism efficiency of the loop might be low.

The iteration set frames show different values for each worker: the first worker executes iterations 0 to 124999, the second worker executes iterations 125000 to 249999, the third worker executes iterations 250000 to 374999, and the last worker is responsible for iterations 375000 to 499998. As expected, each iteration set is composed of a single iteration period ("#Iteration periods: 1"). The time for each iteration period varies between 231.05 million cycles and 475.09 million cycles.

Finally, the iteration period frames show the exact timestamps of the beginning and end of each iteration period. A click on a timestamp centers the time line at the respective position.

Figure 1.3: Detailed information for core 0 to core 3 (top left to bottom right)

Let us now investigate the influence of the schedule of the parallel loop on the load imbalance between workers, starting with a dynamic schedule. As the loop in the example performs several hundred thousand iterations, the default chunk size of 1 for dynamic schedules is likely to cause significant synchronization overhead. To reduce this overhead, we will use a chunk size of 1,000. To apply the new schedule and chunk size, copy the previous code to a new file named `prime_naive_dynamic.c` and replace the pragma above the loop with the following line (or use the file with the same name in the `doc/texi` folder of the Aftermath source code):

```
#pragma omp for schedule(dynamic, 1000) reduction(+:n)
```

Then, compile the new program, create a new trace file named `prime_naive_dynamic.ost` and open the trace with Aftermath:

```
$ clang -fopenmp -g -o prime_naive_dynamic prime_naive_dynamic.c -lm
$ aftermath-openmp-trace -o prime_naive_dynamic.ost -f -- ./prime_naive_dynamic
$ aftermath prime_naive_dynamic.ost
```

Figure 1.4 shows the time line for the resulting trace in loop construct mode and iteration set mode. Both modes show that there is significantly less imbalance between workers. This is the

result of the dynamic assignment of many small iteration sets, as illustrated by the numerous intervals with different colors in iteration set mode. The detailed text view for the dynamic schedule in Figure 1.5 confirms that the wall clock execution time of the loop could be reduced to about 389 million cycles. The parallelism efficiency has been increased significantly to almost 100%. The chunking leads to iteration sets with a highly different execution time, which results in a reduced chunk load balance of less than 10%.

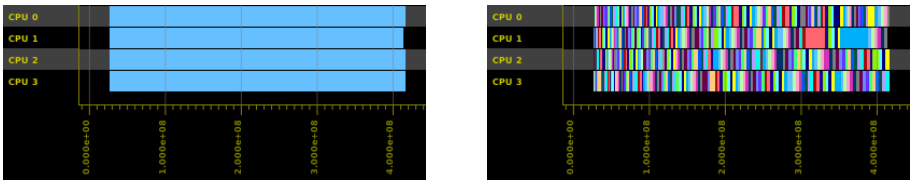

Figure 1.4: The time line in loop construct mode (left) and iteration set mode (right) for the dynamic schedule

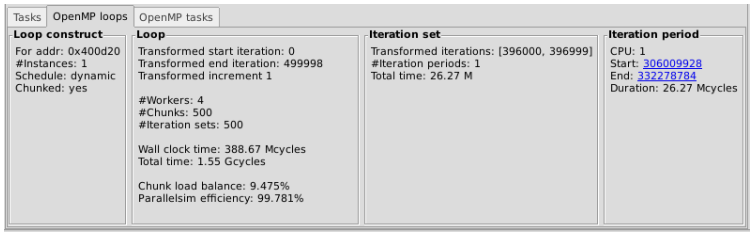

Figure 1.5: Detailed text view for the dynamic schedule

Let us now investigate how the *guided* schedule behaves. This schedule assigns larger iteration sets to the workers at the beginning of the execution of the loop and decreases the size towards its end. This allows the run-time to compensate a load imbalance caused by an increasing amount of work per iteration, just like the workload of the example. For this last version of the prime number test, copy the code to another file named `prime_naive_guided.c` and replace the loop construct with:

```
#pragma omp for schedule(guided) reduction(+:n)
```

or copy the file with the same name from the `doc/examples` directory. Compile and execute the program in order to generate a new trace file, `prime_naive_guided.ost`, and open the generated trace with Aftermath:

```
$ clang -fopenmp -g -o prime_naive_guided prime_naive_guided.c -lm
$ aftermath-openmp-trace -o prime_naive_guided.ost -f -- ./prime_naive_guided
$ aftermath prime_naive_guided.ost
```

Figure 1.6 shows again the time line in loop construct mode and iteration set mode. The work is now almost perfectly balanced between workers, with a parallel efficiency of 99.995%, shown in Figure 1.7. The time line in iteration set mode also indicates that the the increasing amount of work per iteration is in fact overcompensated by the decreasing size of iteration sets, since the duration for the execution of each set decreases.

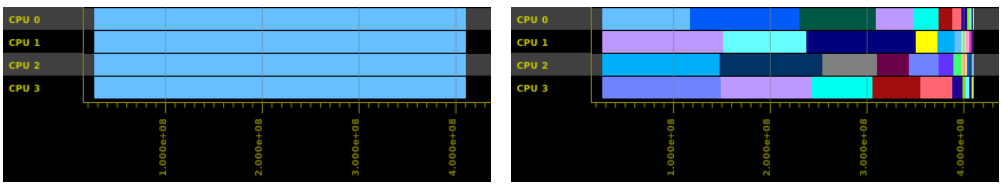

Figure 1.6: The time line in loop construct mode (left) and iteration set mode (right) for the guided schedule

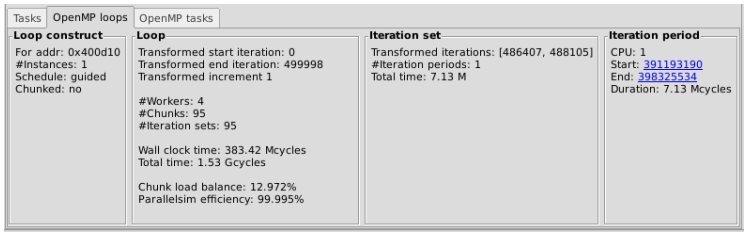

Figure 1.7: Detailed text view for the guided schedule

1.6 Analyzing Tasks

For our next example, illustrating the analysis of tasks, consider the code below, representing a task-based version of the program determining the amount of prime numbers in an interval:

```
#include <stdio.h>
#include <math.h>

static inline int imax(int a, int b)
{
  return (a > b) ? a : b;
}

int isprime_naive(int n)
{
  if(n % 2 == 0 && n != 2)
    return 0;

  for(int j = 3; j <= sqrt(n); j += 2)
    if(n % j == 0)
      return 0;

  return 1;
}

int main(int argc, char** argv)
{
  int n = 1;
  int slices = 100;
  int nloc[slices];
  int max = 1000000;
  int slice_sz = max / slices;

  #pragma omp parallel
  {
    #pragma omp for schedule(static)
    for(int i = 0; i < slices; i++) {
      nloc[i] = 0;

      #pragma omp task
      for(int j = imax(i*slice_sz, 3); j < (i+1)*slice_sz; j++)
        nloc[i] += isprime_naive(j);
```

```
    }

    #pragma omp taskwait

    #pragma omp for schedule(static) reduction(+:n)
    for(int i = 0; i < slices; i++)
      n += nloc[i];
  }

  printf("There are %d prime numbers in the interval\n", n);

  return 0;
}
```

The program divides the interval from 0 to 1,000,000 into 100 equal-sized slices and creates a task for each of the slices in a parallel loop. Each task calculates the amount of prime numbers in its associated slice and writes the result at the respective position to an array named *nloc*. After a barrier waiting for the termination of all generated tasks, a second parallel loop sums up the results of *nloc* and writes the final result to *n*. In a last step, the master thread prints the result after the end of the parallel region.

Paste the code to a new file named `prime_naive_tasks.c` or copy the file from `doc/examples`, compile it, generate a trace file and load the trace with Aftermath as usual:

```
$ clang -fopenmp -g -o prime_naive_tasks prime_naive_tasks.c -lm
$ aftermath-openmp-trace -o prime_naive_tasks.ost -f -- ./prime_naive_tasks
$ aftermath prime_naive_tasks.ost
```

Figure 1.8 shows the time line in task construct mode and task mode. The first observation on the time line in task mode is that task execution covers almost the entire duration of the program. Only a small part towards the end of the execution is not spent on task execution for some of the workers. As there is only one task construct, only one color is used. The many different colors in task mode on the right side of the figure represent the task instances.[1]

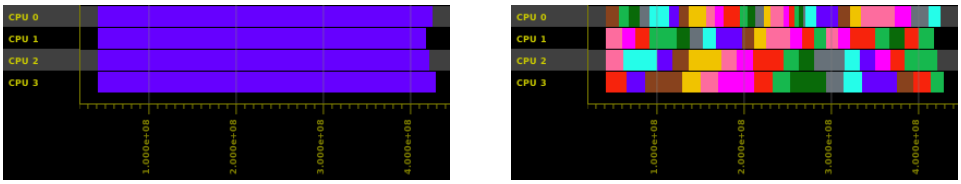

Figure 1.8: The time line in task construct mode (left) and task mode (right) for the implementation with tasks

Each task can be inspected with a click on the associated task period on the time line. Figure 1.9 shows the detailed text view for one of the tasks. Similar to the detailed text view for parallel loops, the view is split into three parts, providing information about the task construct, the task and the selected task period. The task construct frame confirms that there are exactly 100 instances of the task ("#Instances: 100"). As the tasks do not contain barriers and thus cannot be interrupted, there is only one task period per task, as shown in the task frame ("#Task periods: 1"). The executing core, the start and end timestamp as well as the exact duration of the task period are given in the rightmost frame.

[1] Although there are a hundred tasks present in the trace, less than a hundred colors are used in the figure. This is because the number of distinct colors in Aftermath is limited, causing the tool to use the same color for different tasks. If necessary, the color of each task construct, task or task period can be changed manually.

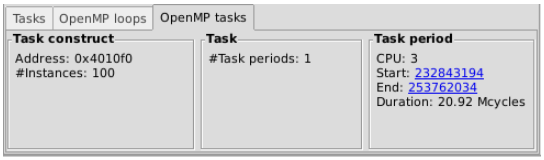

Figure 1.9: Detailed text view for the implementation with tasks

Figure 1.10 shows the time line in state mode for the entire execution of the program (left) and zoomed on the end of the execution (right). The mapping between colors and states are given in a table at the bottom of the statistics tab. The table for the example, given in Figure 1.11, shows five states:

- *barrier* (light blue), indicating time spent in an implicit barrier of a parallel loop or in explicit barriers.
- *critical* (dark blue), *single* (white), and *master* (pink), associated to time spent in a critical region, in a single construct, and in a master construct, respectively.
- *taskwait* (red), representing time spent on synchronization with tasks, either at the end of a task or in a taskwait barrier.

The time line shows short red intervals associated to task synchronization throughout the entire execution (at the end of each task) and longer red intervals for the taskwait barrier towards the end of the execution of the program. The intervals at the end of the execution are complementary to the slight imbalance between workers in the task mode in Figure 1.8, shown earlier. A zoom in this part, shown on the right side of Figure Figure 1.10, also indicates that some time was spent in the implicit barrier at the end of the second parallel loop. Exact statistics on how much time was spent in the different states can be obtained by clicking on the button labeled *Select from graph* in the statistics panel and by selecting an interval from the time line. The table showing the mapping between colors and states is then immediately updated with two values per state. The first value indicates the fraction of the selected interval that was spent in each state, while the second value indicates how many workers on average were in this state simultaneously. The values in Figure 1.11 are very low: only about 1.5% and 0.01% of the time were spent on task synchronization and barriers, respectively. The rest of the time was spent on the execution of tasks and loops.

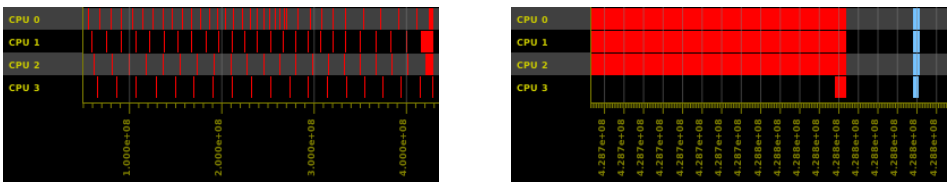

Figure 1.10: The time line in state mode (left: entire execution of the program; right: zoom on the end)

Name	Name	Per	Par
	barrier	0.01%	0.00
	critical	0.00%	0.00
	single	0.00%	0.00
	taskwait	1.46%	0.06
	master	0.00%	0.00

Figure 1.11: Statistics about the time spent in the different states

1.7 Conclusion and Perspectives

In this short guide, we showed how to install Aftermath and Aftermath-OpenMP and provided a few scenarios for the Analysis of parallel loops and tasks. We investigated the influence of the loop schedule and the use of tasks on a synthetic program, calculating the amount of prime numbers in the interval from 0 to 1,000,000. We showed how to inspect the partitioning of the iteration space of loops and how to locate imbalances between workers. We also pointed out how to quantify the time spent in different run-time states, in particular barriers for loops and tasks.

Aftermath constantly evolves and more advanced techniques for the analysis of OpenMP programs will be integrated. Aftermath and Aftermath-OpenMP are released under free software licenses. Your bug reports, suggestions for improvements and code contributions are welcome. To find out more on how to contribute and to stay informed, please visit our website at `http://www.openstream.info/aftermath`.

GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

   You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both

covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its

Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

   Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.