X-Kaapi Tracing Tool

Thierry Gautier —João V. F. Lima

MOAIS
December 14, 2012

Contents

Foreword

X-Kaapi is developed by the INRIA MOAIS team http://moais.imag.fr. The X-Kaapi project is still under development. We do our best to produce as good documentation and software as possible. Please inform us of any bug, malfunction, question, or comment.

This documentation presents the X-Kaapi trace utilities. X-Kaapi is a library with several APIs. The C interface is the lowest interface to program directly on top of the runtime. The C++ interface is an extension of Athapascan-1 interface with new features to avoid explicit declaration of shared variable. X-Kaapi may also be used with C++ through a parallel STL implementation.

About X-Kaapi

X-Kaapi is a **"high level"** interface in the sense that no reference is made to the execution support. The synchronisation, communication, and scheduling of operations are fully controlled by the software. X-Kaapi is an **explicit parallelism language**: the programmer indicates the parallelism of the algorithm through X-Kaapi's one, easy-to-learn template functions, `Spawn` to create tasks. The programming semantics are similar to those of a sequential execution in that each "read" executed in parallel returns the value it would have returned had the "read" been executed sequentially.

The following documentations exist about X-Kaapi:

- X-Kaapi comes from Athapascan interface defined in 1998 and updated in the INRIA technical report RT-276. Available at: http://hal.inria.fr/inria-00069901.
- The INRIA RT-417 presents the C API. Available at: http://hal.inria.fr/hal-00647474.
- The INRIA RT-418 presents the KaCC compiler that allows to write parallel program using code annotation with pragma. Available at: http://hal.inria.fr/hal-00648245.

X-Kaapi is composed by one runtime and several application programming interfaces (API). All these APIs are based on the runtime functions. With specific options it is possible to create a version of the library that is able to record events at runtime, and then to process them to display Gantt diagram or to have access to some statistics.

Reading this Document

This document is a developer documentation designed to describe how to use the X-Kaapi's trace utilities. If the reader can not find its information into this document, please refer to the X-Kaapi web site at http://kaapi.gforge.inria.fr.

This document is organised as following:

- The configuration options required to configure and install X-Kaapi, as well as to generate execution traces, is describe in Section 1;
- Section 2 describes how to generate X-Kaapi traces;
- Section 3 presents how to convert internal binary representation of events to the Pajé trace format;
- Section 4 describes visualisation details of X-Kaapi traces, which can be displayed using ViTE or Pajé;
- The display of PAPI performance counters is given in Section 5.

1 Software installation

X-Kaapi is both a programming model and a runtime for high performance parallelism targeting multicore and distributed architectures. It relies on the work stealing paradigm. X-Kaapi was developed in the MOAIS INRIA project by Thierry Gautier, Fabien Le Mentec, Vincent Danjean, and Christophe Laferrière in the early stage of the library.

X-Kaapi Contacts

If you wish to contact the XKaapi team, please visit the web site at:

http://kaapi.gforge.inria.fr

1.1 Supported Platforms

X-Kaapi targets essentially SMP and NUMA platforms. The runtime should run on any system providing:

- a GNU toolchain (GCC ≥ 4.3);
- the pthread library;
- Unix-based environment.

It has been extensively tested on the following operating systems:

- GNU-Linux with x86_64 architectures;
- Mac OS X with Intel processors.

NVIDIA GPU support has been tested on the following configurations:

- NVIDIA's Fermi and Kepler GPUs;
- CUDA Toolkit 5.0 and 4.2.

There is no version for Windows yet.

1.2 X-Kaapi library installation

To install X-Kaapi programming environment, you need to:

- compile the X-Kaapi library;
- install the X-Kaapi library.

The X-Kaapi libraries and runtime are available at http://kaapi.gforge.inria.fr (tarball). X-Kaapi libraries and runtime are distributed under a CeCILL-C license[1]:

> *CeCILL-C is well suited to libraries and more generally software components. Anyone distributing an application which includes components under the CeCILL-C license must mention this fact and make any changes to the source code of those components available to the community under CECILL-C while being free to choose the licence of its application.*

Currently, X-Kaapi must be installed from sources. Other packagings (Debian or RPM packages) are under development. Please visit the X-Kaapi web site for an up-to-date information:

http://kaapi.gforge.inria.fr

Installing from sources

There are 2 ways to retrieve the sources:

- download a release snapshot at the following url:
  https://gforge.inria.fr/frs/?group_id=94.
- clone the master branch of the git repository of the project:
  `> git clone git://scm.gforge.inria.fr/kaapi/xkaapi.git xkaapi`

[1] http://www.cecill.info/index.en.html for english version

Configuration. The build system uses GNU Autotools. In case you cloned the project repository, you first have to bootstrap the configuration process by running the following script:

```
> ./bootstrap
```

The *configure* file is used to create the *Makefile* accordingly to your system configuration. Command line options can be used to modify the default behaviour. A list of the configuration options is available by running:

```
> ./configure --help
```

The options related to X-Kaapi trace support are:

- `--enable-mode=debug` or `release`
  Choose the compilation mode of the library. Defaults to release.
- `--with-cuda=<CUDA installation>`
  Enable CUDA GPU execution support.
- `--with-perfcounter`
  Enable performance counters support.
- `--with-papi`
  Enable the PAPI library for low level performance counting. More information on PAPI can be found at http://icl.cs.utk.edu/papi.
- `--with-poti`
  (Optional) Specify an external up-to-date POTI library to generate Pajé traces. More information on POTI can be found at https://github.com/schnorr/poti.
- `--with-cupti`
  Enable CUPTI profiler to generate CUDA GPU traces. **Note**: CUDA support is required.
- `--prefix=`
  Overload the default installation path.

Example:

```
> ./configure --enable-mode=release --prefix=$HOME/install
> ./configure  --prefix=$HOME/install --enable-mode=release \
         --with-cuda=/usr/local/cuda-5.0 \
         --with-cupti=/usr/local/cuda-5.0/extras/CUPTI \
         --with-perfcounter
```

If there are errors during the configuration process, you have to solve them before going further. If dependencies are missing on your system, logs will give you the names of the software to install.

Compilation and installation. On success, the configuration process generates a Makefile. The 2 following commands build and install the X-Kaapi runtime:

```
> make
> make install
```

Checking the installation. The following checks that the runtime is correctly installed on your system:

```
> make check
```

Compilation of the examples. The following compiles the sample applications:

```
> cd examples; make examples
```

Installation directory

The configure, make, make install, commands create in the prefix directory the following directory structure:

```
<prefix>/include
<prefix>/lib
<prefix>/bin
<prefix>/share
```

The `<prefix>/share` directory contains some documentation and examples.

2 Generating X-Kaapi trace files

Execution of a X-Kaapi program is controlled by environment variables. For instance, `KAAPI_CPUCOUNT` and `KAAPI_CPUSET` are used to control the number and the location of cores on the machine.

With the options presented in the previous configuration, X-Kaapi runtime is able to record events that corresponds to different activities at runtime.

The record of events is enable if the environment variable `KAAPI_RECORD_TRACE` is set. Once configured, the execution of any X-Kaapi program will record events such that:

- one file `/tmp/events.<username>.<coreid>.evt` is created for each core of the selected set (using `KAAPI_CPUCOUNT` or `KAAPI_CPUSET`).
- A file `/tmp/events.<username>.<coreid>.evt` contains the sequence of events generated by the core `<coreid>` during its execution.

Note that for multi-GPU executions, the file names follow the same file name convention.

2.1 Event selection

The X-Kaapi runtime only records events defined into the event mask, which is formed by a string containing three groups. By default the event mask contains all events. The user may change it by setting the environment variable `KAAPI_RECORD_MASK`. For instance:

```
> KAAPI_RECORD_MASK="compute"
```

specifies the event mask to contains only the events associated with computations.

The set of all events are clustered into three main classes:

- **compute**: which contains all events that are associated with computations;
- **idle**: which contains all events that are associated with idle state;
- **steal**: that defines all events involved during work stealing operation.

These names may be used into the event mask. For instance:

```
> KAAPI_RECORD_MASK="compute,idle,steal"
```

specifies an event mask that contains all events from the compute set, the idle set and steal set.

3 Converting X-Kaapi trace files to Pajé format

The X-Kaapi `katracereader` converts internal traces in binary format into Pajé trace format:

```
> katracereader <options> /tmp/events.<username>.*
```

Its options are:

- `--paje`
  Convert X-Kaapi trace files into Pajé format and create the output file `paje-gantt.trace`.
- `--vite`
  Convert X-Kaapi trace files into Pajé format compatible with ViTE and create the output file `vite-gantt.trace`.
- `--steal-event`
  Output steal events in the conversion process. Disabled by default.
- `--gpu-trace`
  Output GPU thread events of kernel executions. Disabled by default.
- `--gpu-transfer`
  Ouput GPU memory transfers of each GPU thread. Disabled by default.
- `--display-data`
  Print events to standard output.

4 X-Kaapi Trace visualisation

Pajé trace format can be displayed using:

- ViTE Trace Explorer 1.2 available at http://vite.gforge.inria.fr.
- Pajé visualisation tool available at http://paje.sf.net.

Figure 1 illustrates a X-Kaapi trace of SGEMM execution with two CPUs and one GPU using Pajé Tool. It shows the containers:

- *worker-0* and *worker-1* represent the two CPUs;
- *worker-2* contains the CPU activities over the control of the GPU;
- *gpu-2* has the execution of GPU kernels;
- *h2d-2* has the transfers to the GPU and *d2h-2* the transfers from the GPU to the host memory.

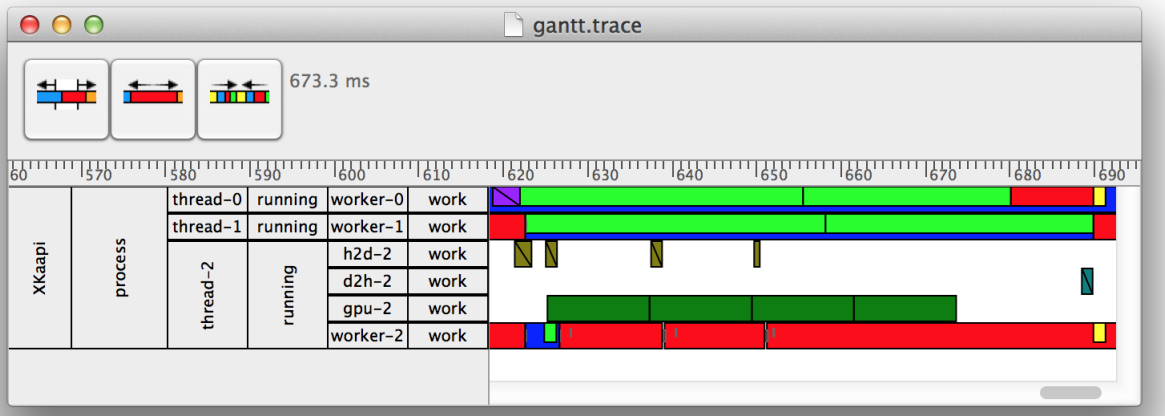

Figure 1: Gantt diagram of a Pajé trace from a X-Kaapi SGEMM execution with two CPUs and one GPU.

The colors from Figure 1 represents the states:

- **running** tasks;
- **idle** state, which means it may request tasks by steal operations, or poll devices such as GPUs;
- **active** state in which the runtime performs other activities as data validation;
- **unroll** tasks to the acceleration structure used by X-Kaapi work stealing algorithm;
- GPU **kernels**;
- GPU **host-to-device** transfers;
- GPU **device-to-host** transfers;
- memory **synchronisation**.

We use arrows to illustrate steal request from thief, and reply from its victim. Here we omitted steal events for the sake of space.

5 PAPI performance counters

In X-Kaapi source files, the example `examples/papi` shows how to use PAPI performance counters to measure performance events on a foreach algorithm. Currently PAPI is supported by the build option `--with-papi` in which the user has to give the path of the PAPI library.

The `KAAPI_PERF_PAPIES` environment variable is used to tell the runtime which PAPI counters we want to measure. An example to measure L1 cache misses, total cycles and FPU operation count is given below:

```
> KAAPI_PERF_PAPIES=PAPI_L1_DCM,PAPI_TOT_CYC,PAPI_FP_OPS ./a.out
done: 2797.519360 (ms)
perf_counters: 1048182, 1977006509, 7413173
```

Some limitations and notes of X-Kaapi counters with PAPI:

- limited to up to 3 `KAAPI_PERF_ID_PAPI_x` counters. See the man page of `papi_avail(3)` for a list of supported events.
- there is currently no way for the user to define measured sections.
- it is not possible to dynamically change the event set to measure.