X-Kaapi Tracing Tool

Thierry Gautier —João Lima

MOAIS
December 13, 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 that may arrise.

This documentation presents the X-Kaapi trace utilities. X-Kaapi is a library with several API. 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 synchronization, 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.
- The INRIA RT-417 presents the C API.
- The INRIA RT-418 presents the KaCC compiler that allows to write parallel program using code annotation with pragma.

X-Kaapi is composed by one runtime and several application programming interface (API). All these APIs are based on the runtime functions. With specific options it is possible to create a version of the library which 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 cannot find its information into this document, please refers to the X-Kaapi web site at http://kaapi.gforge.inria.fr.

This document is organized as following:

- The configuration options required to generate trace of execution is describe in Section 1;
- Section 2 presents how to convert internal binary representation of events to the Pajé representation.
- Section 3 describes visualisation details of X-Kaapi traces, which can be display using ViTE or Pajé.

1 Configuring X-Kaapi library and generating trace files

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 should be present. It is used to create the *Makefile* accordingly to your system configuration. Command line options can be used to modify the default behavior. You can have a complete list of the available options by running:

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

1.1 Configuration

the X-Kaapi build options to generate traces are:

- `--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`
  Specify an external up-to-date POTI library to generate Pajé traces. This option is mandatory if the visualisation tool used is Pajé. More information on POTI can be found at https://github.com/schnorr/poti.
- `--with-cupti`
  Enable the CUPTI CUDA profiler for low level GPU events. This option depends on enabling CUDA support for X-Kaapi.

X-Kaapi web site has a complete list of available options.

1.2 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
```

1.3 Activation of event's recording

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 setting, 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.

1.4 Selection of events

The X-Kaapi runtime only records events defind into the event mask, which is formed by 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 in defined 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 event number 10 and 12.

2 Converting X-Kaapi trace files to Pajé format

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

```
> katracereader --paje /tmp/events.<usename>.*
```

Its options are:

- `--paje`
  Convert X-Kaapi trace files into Pajé format and create the output file `gantt.trace`.
- `--steal-events`
  Output steal events in the convertion 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.

3 Trace visualisation using ViTE and Pajé

Pajé trace format can be displayed using:

- ViTE Trace Explorer 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. It shows the containers:

- *worker-0* and *worker-1* represent the two CPUs;
- *worker-2* contains the CPU actitivies 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.

The colors from Figure 1 represents the states:

- running tasks;

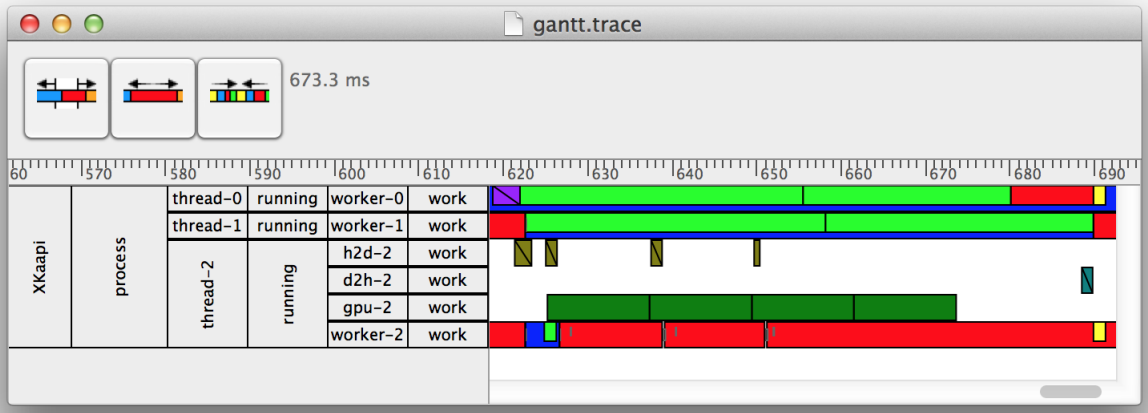

Figure 1: Gantt chart of a Pajé trace from a X-Kaapi SGEMM execution with two CPUs and one GPU.

- 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 ommited steal events for the sake of clarity.