X-Kaapi's Installation and User's Guide

Vincent Danjean, Thierry Gautier, Christophe Laferrière, Fabien Lementec

June 10, 2010

Contents

1 Introduction **2**

2 Installation **2**
2.1 Starting with tarball archive . . . 2
2.2 Starting with git download . . . 2
2.3 Configuration . . . 2
2.4 Compilation . . . 4
2.5 Compilation of the examples . . . 4
2.6 Installation . . . 4
2.7 Supported Architectures . . . 5

3 Utilization **5**
3.1 Simple Example . . . 5
3.2 Running Code and Environnement Variables . . . 5
3.3 Monitoring performances . . . 5

1 Introduction

Here is a short manual of the X-Kaapi library. This Manual will guide you through the installation of the library and its usage. You may find contact information about people involved in this project in the `AUTHORS` file in the source directory.

2 Installation

In order to compile and use X-Kaapi's library, simply follow the next steps. You can freely download X-Kaapi at http://kaapi.gforge.inria.fr. On the forge you will have access to:

- tarball (xkaapi.tgz) file of the last stable release (please visit https://gforge.inria.fr/frs/?group_id=94).
- tarball (xkaapi.tgz) file of the daily main trunk (please visit https://gforge.inria.fr/frs/?group_id=94).
- GIT read only access to the main trunk of the source files
  `> git clone git://git.ligforge.imag.fr/git/kaapi/xkaapi.git xkaapi`

If you want to be developers of X-Kaapi, please contact one of the administrator of the X-Kaapi project on the INRIA forge. In that case, the X-Kaapi's build system uses Automake/Autoconf and Libtool tools.

2.1 Starting with tarball archive

Once you have selected the directory where you want to extract source files, enter

```
$> tar xfvz xkaapi.tgz
```

It will create in the current directory the directory xkaapi that contains all necessary files to compile and install the library. You can pass directly to the section 2.3.

2.2 Starting with git download

Once you have got the source files, you need to bootstrap the build system. We assume that xkaapi is the directory that contains the downloaded source files.

```
$> cd xkaapi
$> ./bootstrap
```

It will generate the configure script from automake/autoconf/libtool tools. If you encounter some error, please check the displayed messages and specifically check the version number of these different tools.

2.3 Configuration

Once you have extracted files from tarball or bootstrapping the build system in case of git download, you need to configure the library. Here we assume that /path/xkaapi is the source directory. The compilation will be down in the directory /path/xkaapi-build, and the installation in the director /usr/local/kaapi. In the following commands, you may change the directory depending of your requirements.

The configuration is based on the configure file in the source directory (/path/xkaapi/configure). First we create the directory for compilation:

```
$> make /path/xkaapi-build
$> cd /path/xkaapi-build
```

The basic configuration is the following (to enter in the directory where you want to build xkaapi). Note that the command produces lot of output and ends by a summary of compilation options.

```
$> /path/xkaapi/configure --prefix=/usr/local/kaapi
checking build system type... i386-apple-darwin9.8.0
checking host system type... i386-apple-darwin9.8.0

... <Here we have suppress output>....

config.status: creating Makefile
config.status: creating api/Makefile
```

```
config.status: creating examples/Makefile
config.status: creating src/Makefile
config.status: creating xkaapi.pc
config.status: creating atha1.pc
config.status: creating src/misc/kaapi_version.c
config.status: creating config.h
config.status: executing depfiles commands
configure: ************************************
configure: *             Summary              *
configure: ************************************
configure: Libraries that can be used:
  * PTHREAD
  * NUMA
configure: Status for PTHREAD external library:
  Cpp flags:              -D_THREAD_SAFE
  Linker flags:           -D_THREAD_SAFE
  Static linker flags:    -D_THREAD_SAFE
configure: Status for NUMA external library:
  Not used or found
configure: ************************************
configure: Libraries that can be created:
  * XKAAPILIB
  * XKAAPIPPLIB
configure: Status for XKAAPILIB new library:
  Link information:       PTHREAD, NOT NUMA
  BUILD_CPPFLAGS:         -D_THREAD_SAFE
  BUILD_LDFLAGS:          -D_THREAD_SAFE
  BUILD_LDFLAGS_STATIC:   -D_THREAD_SAFE
  PKGCONFIG_LIBS:         -L${libdir} -lxkaapi
  PKGCONFIG_LIBS_PRIVATE: -L${libdir} -lxkaapi -D_THREAD_SAFE
  PKGCONFIG_CFLAGS:       -I${includedir}
  CONFIG_CPPFLAGS:        -I${includedir}
  CONFIG_LDFLAGS_STATIC:  -D_THREAD_SAFE
  CONFIG_LIBS:            -L${libdir} -lxkaapi
  CONFIG_LIBS_STATIC:     -L${libdir} -lxkaapi
configure: Status for XKAAPIPPLIB new library:
  Link information:       XKAAPILIB (local) [Public]
  BUILD_LDFLAGS_STATIC:   -D_THREAD_SAFE
  PKGCONFIG_REQUIRES:     xkaapi
  PKGCONFIG_LIBS:         -L${libdir} -lxkaapi++
  PKGCONFIG_LIBS_PRIVATE: -L${libdir} -lxkaapi++
  PKGCONFIG_CFLAGS:       -I${includedir}
  CONFIG_CPPFLAGS:        -I${includedir} -I${includedir}
  CONFIG_LDFLAGS_STATIC:  -D_THREAD_SAFE
  CONFIG_LIBS:            -L${libdir} -lxkaapi++ -L${libdir} -lxkaapi
  CONFIG_LIBS_STATIC:     -L${libdir} -lxkaapi++ -L${libdir} -lxkaapi
configure: ************************************
configure: Programs that can be created:
  * XKAAPIPROGS
  * XKAAPIPPPROGS
configure: Status for XKAAPIPROGS new program:
  Link information:       XKAAPILIB (local)
  BUILD_LDFLAGS_STATIC:   -D_THREAD_SAFE
  PKGCONFIG_REQUIRES_PRIVATE:
                          xkaapi
configure: Status for XKAAPIPPPROGS new program:
  Link information:       XKAAPIPPLIB (local)
  BUILD_LDFLAGS_STATIC:   -D_THREAD_SAFE
  PKGCONFIG_REQUIRES_PRIVATE:
                          xkaapi++
configure: ************************************
```

```
configure: Compilers and default flags used:
  CC: gcc
  CXX: g++
  AM_CFLAGS: -Wall
  CFLAGS: -g -O2
  AM_CXXFLAGS: -Wall
  CXXFLAGS: -g -O2
configure: ************************************
```

In case of error, please read carefully the displayed message.

X-Kaapi is a highly configurable library and you can select different flavors of the library using command line options on the previous execution of the configure. For instance:

- `--enable-target=mt`: select the target for multi-threaded implementation on top of Pthreads (default). Other implementations (gpu, mpsoc) are under development.
- `--enable-mode=debug` or `release`: choose the compilation mode of the library. The default mode is the release mode.
- `--with-perfcounter`: add support for monitoring you X-Kaapi programs.
- `--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-steal=cas` or `hybrid` or `dijkstra`: (default is `dijkstra`). Select the work stealing strategy to execute program. The `cas` is based on compare and swap atomic instruction. `dijkstra` is based on Dijkstra protocol to garante coherency without lock in most of the case (like the Cilk T.H.E. protocol). `hybrid` is a mix of `cas` and `dijkstra`.

All configuration options are self-documented. You display the options, enter:

```
$> /path/xkaapi/configure --help
```

2.4 Compilation

If everything is ok in the previous step, enter (alway in the build directory `/path/xkaapi-build`):

```
$> make
```

By default, the build system is configured in order to compiler both the static and shared library (please look at options during the configuration step).

This will compile both static and dynamic library.

Then you can check the compilation of the library on your system:

```
$> make check
```

2.5 Compilation of the examples

To compile examples, enter in directory /path/xkaapi-build/examples and enter:

```
$> make examples
```

2.6 Installation

Without error in the previous steps, enter:

```
$> make install
```

It will copy the library, include files in the directory /usr/local/kaapi specify in the option --prefix during the configuration step.

If you are not allowed to install files in this directory, either execute previous command with sudo, else contact your system administrator.

2.7 Supported Architectures

X-Kaapi currently supports SMP and NUMA architectures. It also compile well on Mac machines (32bits works fine, 64bits is in progress).

3 Utilization

In the "examples" directory, there is some examples that are compiled with the library. One may run those examples to see how things works. Note that not every example in the "examples" directory does work currently.

3.1 Simple Example

A revoir *Let's have a look at the* `transform` *example. This code is simply the parallel version (using* X*-Kaapi of course) of the stl::transform algorithm. Inputs are a table and a unary operation, the unary operation is applied to each single element of the table and the result is stored into an output table.*

The library header has to be include (`kaapi.h`*). The work is done in the* `doit()` *function, in the while loop. In this loop, there's a call to* `kaapi_stealpoint()`*. If some steal requests have been recieved since the last call to this function, then the* `splitter()` *function is called in order to reply to these requests.*

XKaapi aims at providing a finer grain than the previous Kaapi implementation. To do so, the workstealing engine relies upon the application programmer to place its own stealing points, using the `kaapi_stealpoint()` *routine. The stealpoints provide a way to distribute the remaining computation among the thief processors. The splitting occurs dynamically by iterating over the pending stealing requests array (the* `request[]` *array argument containing* `count` *elements). It is left to the programmer to implement a splitter function, since it is assumed he knows how to partition the computation as it is in a coherent state. Inside the splitting function, the victim uses the* `kaapi_request_reply()` *routine to complete the stealing requests.*

3.2 Running Code and Environnement Variables

In order to control thread placement and global machine usage, there are two environnement variables that you may use to tune an execution :

- `KAAPI_CPUCOUNT` is simply the number of CPU to be used :

```
KAAPI_CPUCOUNT=3 ./transform 100000
```

- `KAAPI_CPUSET` is the set of CPU to be used explicitly if you only want to use a subset of a machine's cores. The value is a comma separated list. You can also specifie a range using "x:y".

```
#this will uses cores 3, 4, 5, 6 and 9 only :
KAAPI_CPUSET=3,4,5,6,9 ./transform 100000

#this will do the same using range :
KAAPI_CPUSET=3:6,9 ./transform 100000
```

  You may exclude a cpu from the set by appending the '!' token:

```
#this will uses cores from 3 to 9, but not 4:
KAAPI_CPUSET=3:9,!4 ./transform 100000
```

3.3 Monitoring performances

If configured by --with-perfcounter, the X-Kaapi library allows to output performance counters.

- `KAAPI_DISPLAY_PERF`: if defined, then performance counters are displayed at the end of the execution.

```
KAAPI_DISPLAY_PERF =1 ./fibo 30
```

- `KAAPI_DISPLAY_PERF`: if defined, then performance counters are displayed at the end of the execution.

```
KAAPI_DISPLAY_PERF =1 ./fibo 30
```

- KAAPI_PERF_PAPIES: asuming X-Kaapi was configured using –with-papi, this variable contains a comma separated list of the PAPI performance counters to use. Both counter symbolic names and numeric hexadecimal constants can be used. More information can be found on the PAPI website (http://icl.cs.utk.edu/papi/). Note that counter list cannot exceed 3 elements.

```
KAAPI_PERF_PAPIES=PAPI_TOT_INS,0x80002230,PAPI_L1_DCM ./fibo 30
```

As an example, the following is an output produced by the library as the program terminates. It includes both the per core and the cummulated performance counters:

```
KAAPI_DISPLAY_PERF=1 KAAPI_CPUCOUNT=3 KAAPI_PERF_PAPIES=PAPI_TOT_INS ./fibo 10
[KAAPI::INIT] use #physical cpu:3, start time:1266571053.688506
#op: 4196640
Fibo(10) = 55 *** Time: 1.328945e-03(s)
[KAAPI::TERM] end time:1266571053.690019
----- Performance counters, core   : 0
Total number of tasks executed     : 2, 0
Total number of steal OK requests  : 0
Total number of steal BAD requests : 0
Total number of steal operations   : 0
Total number of suspend operations : 0
Total idle time                    : 1.088381e-03
----- Performance counters, core   : 1
Total number of tasks executed     : 0, 0
Total number of steal OK requests  : 0
Total number of steal BAD requests : 1567
Total number of steal operations   : 1385
Total number of suspend operations : 0
Total idle time                    : 0.000000e+00
----- Performance counters, core   : 2
Total number of tasks executed     : 0, 0
Total number of steal OK requests  : 0
Total number of steal BAD requests : 1515
Total number of steal operations   : 1302
Total number of suspend operations : 0
Total idle time                    : 0.000000e+00
----- Cumulated Performance counters
Total number of tasks executed     : 2
Total number of steal OK requests  : 0
Total number of steal BAD requests : 3082
Total number of steal operations   : 2687
Total number of suspend operations : 0
Total idle time                    : 0.000000e+00
   sched idle time                 : 0.000000e+00
   preemption idle time            : 0.000000e+00
Average steal requests aggregation : 1.147004e+00
```