

X-Kaapi installation and user guide

Vincent Danjean, Thierry Gautier, Christophe Laferrière, Fabien Le Mentec

June 15, 2010

Contents

1	Introduction	2
1.1	X-Kaapi runtime	2
1.2	Supported Platforms	2
2	Installation	2
2.1	Using the debian packages	2
2.1.1	Retrieving the debian packages	2
2.2	Installing from sources	2
2.2.1	Retrieving the sources	2
2.2.2	Configuration	2
2.2.3	Compilation and installation	3
2.2.4	Checking the installation	3
2.2.5	Compilation of the examples	3
3	Using X-Kaapi	3
3.1	X-Kaapi integration	3
3.2	Examples	4
3.3	Runtime environment variables	4
3.4	Monitoring performances	5

1 Introduction

1.1 X-Kaapi runtime

X-Kaapi is a runtime for fine grain parallelism on multicore architectures. It relies on workstealing paradigms. The core library comes with a full set of complementary programming interfaces. This document is a user manual containing installation instructions and runtime options, as well as a description of the components shipped with the runtime.

1.2 Supported Platforms

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

- a GNU toolchain (4.3),
- the pthread library.

It has been extensively tested on the following operating systems:

- GNU-Linux/x86_64,
- MacOSX/PowerPC.

There is no version for Windows yet.

2 Installation

There are 2 ways to install X-Kaapi:

- using the debian packages,
- installing from source.

2.1 Using the debian packages

TODO

2.1.1 Retrieving the debian packages

TODO

2.2 Installing from sources

2.2.1 Retrieving the 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 project git repository:

```
> git clone git://git.ligforge.imag.fr/git/kaapi/xkaapi.git xkaapi
```

2.2.2 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 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
```

Below is a list of the most important ones:

- `--enable-target=mt`
Select the target platform. Defaults to 'mt', for pthread.
- `--enable-mode=debug` or `release`
Choose the compilation mode of the library. Defaults to release.
- `--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/>.
- `--prefix=`
Overload the default installation path.

Example:

```
./configure --enable-mode=release --enable-target=mt --prefix=$HOME/install
```

If there are errors during the configuration process, you have to solve them before going further. It is likely there is a missing dependency on your system, in which case the log gives you the name of the software to install.

2.2.3 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
```

2.2.4 Checking the installation

The following checks the runtime is correctly installed on your system:

```
$> make check
```

2.2.5 Compilation of the examples

The following compiles the sample applications:

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

3 Using X-Kaapi

3.1 X-Kaapi integration

Integrating X-Kaapi in your project requires the following steps:

- include the header files. Example:

```
#include <kaapi.h> /* C version */
#include <kaapi++> // C++ version
```

- link with the library. Example:

```
gcc -o main main.c -lxkaapi /* C version */
g++ -o main main.cpp -lxkaapixx // C++ version
```

- the following preprocessor macro must be defined to fully disable the debugging code. It can improve the generated code:

```
-DNDEBUG
```

Refer to the API documentation for information relative to the X-Kaapi programming interfaces.

3.2 Examples

The directory *examples/* contains sample applications using X-Kaapi. Both C and C++ are used. Some directly lies on top of the core runtime, while other make use of higher level interfaces. Below is a short description for some of them:

- `fibo_xkaapi.adapt.c`
Implementation of the Fibonacci sequence generation using adaptive task stealing.
- `fibo_xkaapi.c`
Same as above, using DFG tasks.
- `fibo_kaapixx.cpp`, `fibo_kaapixx_opt.cpp`
Same as above, using the *ka* C++ interface.
- `fibo_atha.cpp`
Same as above, using the *Athapascan* interface.
- `nqueens_apikaapi`
Nqueens problem implementation using *ka* C++ interface.
- `nqueens_apiatha`
Same as above, using the *Athapascan* interface.
- `poisson3d-kaapi.cpp`
3 dimension Poisson solver implemented using the *ka* C++ interface.
- `matrix_multiply_cilk2kaapi.cpp`
Matrix multiplication using loop parallelization. Implemented using the *ka* C++ interface.

3.3 Runtime environment variables

The runtime behavior can be driven by using the following optionnal environment variables.

- `KAAPI_CPUCOUNT`
The number of process unit to be used by the runtime. No assumption is made regarding which unit is used. Example:

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

- `KAAPI_CPUSET`
The set of CPU to be used. It consists of a comma separated list of cpu indices, first index starting at 0. By default, and if no `KAAPI_CPUCOUNT` is given, all the host cpus are used. Example:

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

An index range can be used via the ':' token. Example:

```
#same as above using the range syntax:
KAAPI_CPUSET=3:6,9 ./transform 100000
```

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

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

- `KAAPI_STACKSIZE`
Size of the per thread stack, in bytes. By default, this size is set to 64 kilo bytes. Example:

```
KAAPI_STACKSIZE=4096 ./transform 100000
```

- `KAAPI_WSSELECT`
Name of the victim processor selection algorithm to use.
 - "workload": Use a user defined workload to driver the victim selection algorithm.
 - any other value: falls back to the default random victim selection algorithm.

Example:

```
KAAPI_WSSELECT=workload ./transform 100000
```

3.4 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. Example:

```
KAAPI_DISPLAY_PERF=1 ./fibonacci 30
```

- `KAAPI_PERF_PAPIES`

Assuming 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. Example:

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