



X-KAAPI FORTRAN programming interface

Fabien Le Mentec, Vincent Faucher, Thierry Gautier

**TECHNICAL
REPORT**

N° xxxx

November 2011

Project-Teams MOAIS



X-KAAPI FORTRAN programming interface

Fabien Le Mentec*, Vincent Faucher†, Thierry Gautier*

Project-Teams MOAIS

Technical Report n° xxxx — November 2011 — 13 pages

Abstract: This report defines the X-KAAPI FORTRAN programming interface.

Key-words: parallel computing, X-KAAPI, FORTRAN

* INRIA

† CEA

Interface de programmation FORTRAN pour X-KAAPI

Résumé : Pas de résumé

Mots-clés :

Contents

| | | |
|----------|---------------------------------------|-----------|
| 1 | Software installation | 4 |
| 2 | Initialization and termination | 5 |
| 3 | Concurrency | 6 |
| 4 | Performance | 7 |
| 5 | Independent loops | 8 |
| 6 | Dataflow programming | 10 |
| 7 | Parallel regions | 12 |
| 8 | Synchronization | 13 |

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.

In this report, only the programming model based on the C API is presented. The runtime library comes also with a full set of complementary programming interfaces: C, C++ and STL-like interfaces. The C++ and STL interfaces, at a higher level than the C interface, may be directly used for developing parallel programs or libraries.

Supported Platforms

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

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

It has been extensively tested on the following operating systems:

- GNU-Linux with x86_64 architectures,
- MacOSX/Intel processor.

There is no version for Windows yet.

X-KAAPI Contacts

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

<http://kaapi.gforge.inria.fr>

2 Initialization and termination

2.1 Synopsis

```
SUBROUTINE KAAPIF_INIT(INTEGER*4 FLAGS)
SUBROUTINE KAAPIF_FINALIZE()
```

2.2 Description

KAAPIF_INIT initializes the runtime. It must be called once per program before using any of the other routines. If successful, there must be a corresponding *KAAPIF_FINALIZE* at the end of the program.

2.3 Parameters

- *FLAGS*: if not zero, start only the main thread to avoid disturbing the execution until tasks are actually scheduled. The other threads are suspended waiting for a parallel region to be entered (refer to *KAAPIF_BEGIN_PARALLEL*).

2.4 Return value

None.

2.5 Example

Refer to examples/kaapif/foreach

```
PROGRAM MAIN
  CALL KAAPIF_INIT(1)
  ...

  CALL KAAPIF_FINALIZE()
END PROGRAM MAIN
```

3 Concurrency

3.1 Synopsis

```
INTEGER*4 FUNCTION KAAPIF_GET_CONCURRENCY()  
INTEGER*4 FUNCTION KAAPIF_GET_THREAD_NUM()
```

3.2 Description

Concurrency related routines.

3.3 Return value

KAAPIF_GET_CONCURRENCY returns the number of parallel thread available to the X-KAAPI runtime.

KAAPIF_GET_THREAD_NUM returns the current thread identifier. Note it should only be called in the context of a X-KAAPI thread.

3.4 Example

Refer to examples/kaapif/foreach

4 Performance

4.1 Synopsis

```
REAL*8 FUNCTION KAAPIF_GET_TIME()
```

4.2 Description

Capture the current time. Used to measure the time spent in a code region.

4.3 Parameters

None.

4.4 Return value

The current time, in microseconds.

4.5 Example

Refer to examples/kaapif/foreach

```
PROGRAM MAIN
  REAL*8 START
  REAL*8 STOP

  CALL KAAPIF_INIT(1)
  START = KAAPIF_GET_TIME()

  ...

  STOP = KAAPIF_GET_TIME()
  CALL KAAPIF_FINALIZE()

  WRITE(*, *) STOP - START

END PROGRAM MAIN
```

5 Independent loops

5.1 Synopsis

```

SUBROUTINE KAAPIF_FOREACH
(
  BODY,
  INTEGER*4 FIRST, INTEGER*4 LAST,
  INTEGER*4 NARGS,
  ...
)

SUBROUTINE KAAPIF_FOREACH_WITH_FORMAT
(
  BODY,
  INTEGER*4 FIRST, INTEGER*4 LAST,
  INTEGER*4 NARGS,
  ...
)

```

5.2 Description

Those routines run a parallel loop over the range $[FIRST, LAST]$ (note this is an **inclusive** interval). The loop body is defined by *BODY* whose arguments are given in parameters. It must have the following prototype:

```

SUBROUTINE BODY(I, J, TID, ...)

```

- $[I, J]$ the subrange to process (note that interval is inclusive)
- *TID* the thread identifier

5.3 Parameters

- *BODY*: the function body to be called at each iteration
- *FIRST*, *LAST*: the iteration range indices, inclusive.
- *NARGS*: the argument count
- ...: the arguments passed to *BODY*. For *KAAPIF_FOREACH_WITH_FORMAT*, refer to the *KAAPIF_SPAWN* documentation.

5.4 Return value

None.

5.5 Example

Refer to examples/kaapif/foreach

Refer to examples/kaapif/foreach_with_format

```
! computation task entry point
SUBROUTINE OP(I, J, TID, ARRAY)
  DO K = I, J
    ! process ARRAY(K)
    ...
  END DO
  RETURN
END

PROGRAM MAIN
  ...
  ! apply the OP routine on ARRAY[1:SIZE]
  CALL KAAPIF_FOREACH(OP, 1, SIZE, 1, ARRAY)
  ...
END PROGRAM MAIN
```

6 Dataflow programming

6.1 Synopsis

```
SUBROUTINE KAAPIF_SPAWN
(
  BODY,
  INTEGER*4 NARGS,
  ...
)
```

6.2 Description

Create a new computation task implemented by the function *BODY*.

BODY is called with the user specified arguments, there is no argument added by XKAAPI:

```
SUBROUTINE BODY(ARG0, ARG1, ...)
```

Each task parameter is described by 4 successive arguments including:

- the argument *VALUE*,
- the parameter *TYPE*,
- the element *COUNT*,
- the access *MODE*.

TYPE is one of the following:

- KAAPIF_TYPE_CHAR=0,
- KAAPIF_TYPE_INT=1,
- KAAPIF_TYPE_REAL=2,
- KAAPIF_TYPE_DOUBLE=3.

If a parameter is an array, *COUNT* must be set to the array size. For a scalar value, it must be set to 1.

MODE is one of the following:

- KAAPIF_MODE_R=0 for a read access,
- KAAPIF_MODE_W=1 for a write access,
- KAAPIF_MODE_RW=2 for a read write access,
- KAAPIF_MODE_V=3 for a parameter passed by value.

6.3 Parameters

- *BODY*: the task body.
- *NARGS*: the argument count.
- ...: the *VALUE*, *TYPE*, *COUNT*, *MODE* tuple list.

6.4 Return value

None.

6.5 Example

Refer to examples/kaapif/dfg

```
! computation task entry point
SUBROUTINE OP(A, B)
  ! task user specific code
  ...
  RETURN
END

PROGRAM MAIN
  ...

  ! spawn a task implemented by the OP routine
  call kaapif_spawn(fu, 2,
    ! argument[0]
    & 42
    & KAAPIF_TYPE_DOUBLE,
    & 1,
    & KAAPIF_MODE_V,
    ! argument[1]
    & 42,
    & KAAPIF_TYPE_DOUBLE,
    & 1,
    & KAAPIF_MODE_V)
  ...
END PROGRAM MAIN
```

7 Parallel regions

7.1 Synopsis

```
SUBROUTINE KAAPIF_BEGIN_PARALLEL()  
SUBROUTINE KAAPIF_END_PARALLEL(INTEGER*4 FLAGS)
```

7.2 Description

KAAPIF_BEGIN_PARALLEL and *KAAPIF_END_PARALLEL* mark the start and the end of a parallel region. Regions are used to wakeup and suspend the X-KAAPI system threads so they avoid disturbing the application when idle. This is important if another parallel library is being used. Whether threads are suspendable or not is controlled according by the *KAAPIF_INIT* parameter.

7.3 Parameters

- *FLAGS*: if zero, an implicit synchronization is inserted before leaving the region.

7.4 Return value

None.

7.5 Example

Refer to examples/kaapif/dfg

```
PROGRAM MAIN  
  ...  
  
  CALL KAAPIF_BEGIN_PARALLEL()  
  
  ...  
  
  CALL KAAPIF_END_PARALLEL(1)  
  
  ...  
END PROGRAM MAIN
```

8 Synchronization

8.1 Synopsis

```
SUBROUTINE KAAPIF_SCHED_SYNC()
```

8.2 Description

Synchronize the sequential with the parallel execution flow. When this routine returns, every computation task has been executed and memory is consistent for the processor executing the sequential flow.

8.3 Return value

None.

8.4 Example

Refer to examples/kaapif/dfg

```
PROGRAM MAIN
  ...

  CALL KAAPIF_SCHED_SYNC()

  ...
END PROGRAM MAIN
```



**RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES**

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803