INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

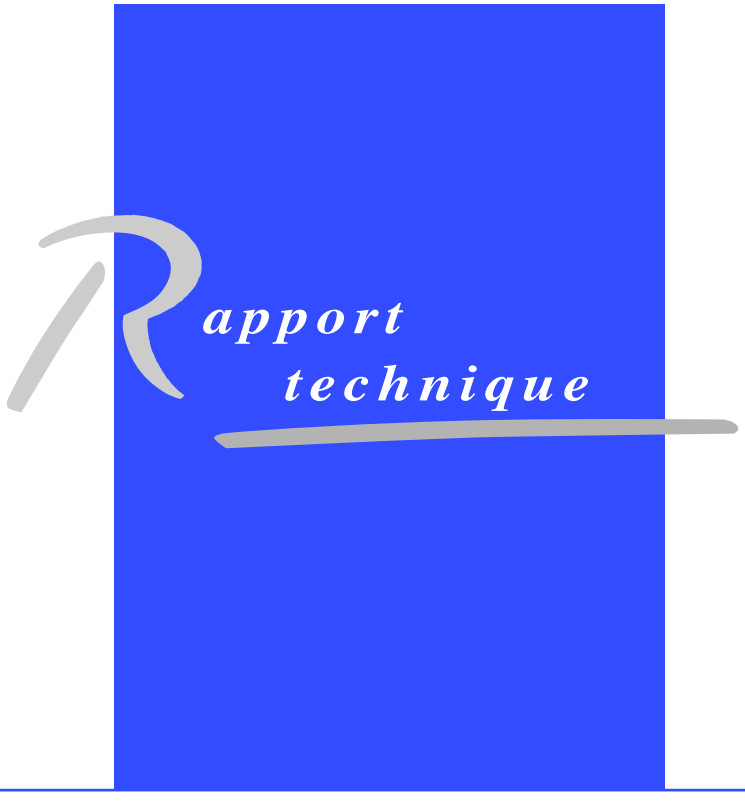
X-KAAPI *FORTRAN programming interface*

Thierry Gautier — Vincent Faucher — Bruno Raffin — Fabien Le Mentec

N° ????

~November 2011~

~Distributed and High Performance Computing~



ISSN 0249-0803 ISRN INRIA/RT--????--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

X-Kaapi FORTRAN programming interface

Thierry Gautier , Vincent Faucher , Bruno Raffin , Fabien Le Mentec

Theme˜: Distributed and High Performance Computing
Équipes-Projets˜MOAIS

Rapport technique n° ???? — November 2011 — 12 pages

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

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

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

X-Kaapi FORTRAN programming interface

Résumé : Pas de résumé

Mots-clés :

Contents

1 Initialization and termination

1.1 Synopsis

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

1.2 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.

1.3 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).

1.4 Return value

None.

1.5 Example

Refer to examples/kaapif/foreach

```
PROGRAM MAIN
  CALL KAAPIF_INIT(1)

  ...

  CALL KAAPIF_FINALIZE()
END PROGRAM MAIN
```

2 Concurrency

2.1 Synopsis

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

2.2 Description

Concurrency related routines.

2.3 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.

2.4 Example

Refer to examples/kaapif/foreach

3 Performance

3.1 Synopsis

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

3.2 Description

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

3.3 Parameters

None.

3.4 Return value

The current time, in microseconds.

3.5 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
```

4 Independent loops

4.1 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,
  ...
)
```

4.2 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

4.3 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.

4.4 Return value

None.

4.5 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
```

5 Dataflow programming

5.1 Synopsis

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

5.2 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,
- KAPAIF_TYPE_REAL=2,
- KAPAIF_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.

5.3 Parameters

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

5.4 Return value

None.

5.5 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
```

6 Parallel regions

6.1 Synopsis

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

6.2 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. Wether threads are suspendable or not is controlled according by the KAAPIF_INIT parameter.

6.3 Parameters

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

6.4 Return value

None.

6.5 Example

Refer to examples/kaapif/dfg

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

7 Synchronization

7.1 Synopsis

```
SUBROUTINE KAAPIF_SCHED_SYNC()
```

7.2 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.

7.3 Return value

None.

7.4 Example

Refer to examples/kaapif/dfg

```
PROGRAM MAIN
  ...

  CALL KAAPIF_SYNC()

  ...
END PROGRAM MAIN
```

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP~101 - 54602~Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153~Le~Chesnay~Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042~Rennes~Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP~93 - 06902~Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP~105 - 78153~Le~Chesnay~Cedex (France)
http://www.inria.fr
ISSN 0249-0803