LAIPE

Auxiliary Subroutines

Copyright (C) 1995 - 2014
By Equation Solution

List of Contents

List of Contents …………………………………………………………………………. i
About the Manual ii

1. Auxiliary Subroutine `laipe$done` ………........…………………….…………………………………….. 1
2. Auxiliary Subroutine `laipe$getce` …………..……...…………..………………………………….. 2
3. Auxiliary Subroutine `laipe$use`………………….....…...……...………………………………... 3
4. Auxiliary Subroutine `laipe$resetUserTimes` ………….....…………………………………... 4
5. Auxiliary Subroutine `laipe$getUserTimes`…………………..………………………………….. 5
6. Auxiliary Subroutine `laipe$staySoftCore`.......………………....…………………………... 6

About This Manual

This manual introduces auxiliary subroutines. The auxiliary subroutines are not for equation or function computing, but for other purposes, for example, collecting timing results, soft core manipulations, and others. For purposes, please check the introduction of each subroutine.

The auxiliary subroutines are optional. User can completely ignore the auxiliary subroutines.

In this manual, the arguments passed to auxiliary subroutine are written with a suffix "_i" or "_o". Suffix "_i" means the argument is an input to the subroutine. Suffix "_o" means an output from the subroutine. For example, in the subroutine

```
laipe$getce(ce_o)
```

argument `ce_o` outputs the number of available computing elements.

1. Auxiliary Subroutine `laipe$done`

The subroutine releases system resources. System resources may be automatically released when application is terminated by itself or other facts. LAIPE provides the auxiliary subroutine for user to release system resources before the application exits.

The subroutine `laipe$done` should be the last LAIPE subroutine called in an application. After releasing system resource, any call to LAIPE functions is meaningless and could crash the application. The subroutine also affects other packages which are programmed in neuloop or MTASK, against which an application also links against LAIPE. A safety practice is to call the subroutine "just" before exiting an application.

1.1 Fortran Syntax for Subroutine `laipe$done`

This subroutine has no arguments. Fortran syntax is as follow:

```
CALL laipe$done
```

2. Auxiliary Subroutine `laipe$getce`

The subroutine returns the number of available computing elements. A computing element is a core in modern multicore processors or a traditional processor.

2.1 Fortran Syntax for Subroutine `laipe$getce`

Fortran syntax is as follow:

CALL `laipe$getce(ce_o)`

where

1. The argument ce_o, a 4-byte INTEGER, returns the number of available computing elements.

3. Auxiliary Subroutine `laipe$use`

The subroutine defines the number of computing elements to be used in the subsequent computing. More computing elements could be useless when there is nothing to be distributed onto the computing elements, but create more burdens for communication and synchronization. For example, one core (e.g., one computing element) can solve a (300x300) dense system equation within 0.01 seconds. There is no significant benefit to use more cores to speed up a 0.01-second computation. In this situation, more cores not only cannot show a benefit but also may become a burden for communication and synchronization. The subroutine `laipe$use` is applicable to this situation.

Especially, chunk in LAIPE solvers, which is the distribution unit for parallel computing, is not in a size of single coefficient, but is a subblock which may be in a size, for example, from (8x8) or up to (64x64). When a problem does not have a sufficient size to be distributed, more cores show no benefit. This subroutine is optional. When user wants to adjust the number of cores for its application, user can apply this subroutine. The subroutine is also needed when timing performance of a parallel code. For example,

```
!! get number of cores
call laipe$getce(numberofcores)

!! loop for collecting timing
do i = 1, numberofcores

   !! define the number of cores for the subsequent computing
   call laipe$use(i)

   !! reset the timer
   call laipe$resetUserTimes

   !! place here the parallel codes

   !! get the timing results
   call laipe$getUserTimes(et,ut,kt,tt)

   !! output the timing results

end do
```

3.1 Fortran Syntax for Subroutine `laipe$use`

Fortran syntax is as follow:

CALL `laipe$use(ce_i)`

where the argument ce_i, a 4-byte INTEGER, defines the number of computing elements for the subsequent computing. When the argument ce_i does not input a valid number, for example, a zero or negative number or a number greater than the number of available computing elements, the subroutine uses the number of available computing elements for the definition.

4. Auxiliary Subroutine `laipe$resetUserTimes`

The subroutine resets the timer for collecting timing results.

4.1 Fortran Syntax for Subroutine `laipe$resetUserTimes`

The subroutine does not have an argument. Fortran syntax is as follow:

```
CALL laipe$resetUserTimes
```

5. Auxiliary Subroutine `laipe$getUserTimes`

The subroutine returns the timing results from the latest reset.

5.1 Fortran Syntax for Subroutine `laipe$getUserTimes`

Fortran syntax is as follow:

CALL `laipe$getUserTimes( et_o, ut_o, kt_o, tt_o )`

where

1. The argument et_o, a 4-byte REAL variable, returns the elapsed time in seconds.
2. The argument ut_o, a 4-byte REAL variable, returns the CPU time in user mode in seconds.
3. The argument kt_o, a 4-byte REAL variable, returns the CPU time in kernel mode in seconds.
4. The argument tt_o, a 4-byte REAL variable, returns the total CPU time in seconds.

6. Auxiliary Subroutine `laipe$staySoftCore`

The subroutine stays each soft computing element (e.g. soft core or task) on an individual physical core (or physical computing element) where the soft core was initially scheduled. By default, operating system dynamically assigns soft computing elements onto available physical cores, and multiple soft computing elements may reside on a physical core. After calling this subroutine, soft computing element stays on the physical core where it was initially scheduled until the soft computing element is terminated.

The subroutine should be called before any call to LAIPE solvers. The subroutine also may affect other packages which are programmed in neuloop or MTASK. A safety practice is to call the subroutine in the beginning of an application.

This subroutine is optional.

6.1 Fortran Syntax for Subroutine `laipe$staySoftCore`

The subroutine does not have an argument, and the syntax is as follow:

CALL `laipe$staySoftCore`