APPENDIX

Contents

A1.	**Parameters for CAPO**	**54**
A1.1.	General	54
A1.2.	The Parameter File	54
A1.3.	Parameter Keys and Possible Values	54
A1.4.	Parameters for Debugging Purpose	57
A1.5.	Sample Parameter File	57
A2.	**Messages and Symbols in the Log File**	**59**
A2.1.	Classification of Loops	59
A2.2.	Construction and Optimization of Parallel Regions	60
A2.3.	Insertion of Directives in Routines	63
A2.4.	Debug Information	65
A3.	**CAPO Graphical User Interface**	**68**
A3.1.	CAPTools Main Window	68
A3.2.	Directives Browser Main Window	69
A3.3.	Loop Filters and Sub-filters	70
A3.3.1.	Loop Variable Filter Window	72
A3.4.	WhyDirectives Window	73
A3.5.	Routine Duplication Browser	78
A3.6.	Parameter Setting Window	79
A3.7.	User Loop Type Window	81
A3.8.	Reduction Operator Dialog	81
A3.9.	Updating Directives Dialog	82
A3.10.	Variable Removal Confirmation Dialog	82
A3.11.	Data Graph Window	83
A3.12.	Hookups to CAPTools	84
A4.	**CAPO Command Interface**	**87**
A4.1.	Commands for the Command Interface	87
A4.2.	Other CAPTools Commands Useful for CAPO	89
A4.3.	An Example of "capo_run.cmd"	89

A1. Parameters for CAPO

The following describes parameters available in Version 1.1.

A1.1. General

Parameters are referring to inputs that user can supply to control the behavior of directive generation in CAPO. There are default settings for all the parameters (see Section A1.3). Parameters can be defined from a file, environment variables, or the Setting box in the Directives Browser. Values from the parameter file or environment variables supersede any defaults. Values from the parameter file supersede environment variables. Changes from the Setting box (Section A3.6) in the Directives Browser are applied at last. Parameter setting can also be done from the CAPO command interface. See Section A4 for details.

A1.2. The Parameter File

The parameter filename can be defined via the environment variable CAPO_PAR. The default filename is "`capo-inp.par`" in the current directory. An example of this file is given in Section A1.5.

Format of the parameter file:

`'#'`	the sign starts a comment
`'key value'`	the pair defines an entry

A1.3. Parameter Keys and Possible Values

ENV_VARIABLE	*KEY*	*DEFAULT*	*POSSIBLE VALUES*
CAPO_PAR		capo-inp.par	
CAPO_LOG	log-file	on	(off on stdout)
CAPO_LOGNAME	log-file-name	codeoutput.log	
CAPO_LOGINFO	log-info	std	(min std more debug)
CAPO_PLOOP	loop-granularity	6	(0 1 2 ...)
CAPO_TYPE	directive-type	omp	(omp sgi sgix no)
CAPO_REGION	region-type	default	(loop bloop one join full)
CAPO_OPTIMIZE	optimize-type	o2	(off on o2 o3)
CAPO_USERLOOP	user-loop-file	user-loop.par	
CAPO_DIRCLEAR	directive-clear	default-list	(off on filename)
CAPO_TPRIV	tpriv-directive	on	(off on)
CAPO_COMMENT	comment-type	f90	(f77 f90)
CAPO_USEPARTI	use-parti-loop	no	(no yes)
CAPO_RDUPTYPE	rdup-type	region	(loop region)
CAPO_UNKSIZE	allow-unksize	false	(false true)
CAPO_PIO	allow-pio	no	(no incall write noread any)

Description of the parameters:

- "`log-file`" type is one of

  `off` — Logging to file is off, only minimum messages are printed on screen
  `on` — Information are logged to the log-file
  `stdout` — Information are printed to stdout (screen)

- "`log-file-name`" defines the name for a log file. If no name is defined, CAPO will use the output filename from the code generation to form a log filename. Contents of the log file are described in Section A2.

- "`log-info`" type is one of

  `min` — Only minimum information are logged or printed
  `std` — Print standard set of log information
  `more` — Print more detailed log information, including region and loop numbers in the final Fortran file
  `debug` — Print debugging information, probably more than you want, including region and loop numbers in the final Fortran file

- The loop granularity is based only on the loop iterations at this point. Future extension to include profile information can easily be added.

- Currently supported directive types are

  `omp` — Produce OpenMP directives (default)
  `sgi` — Produce SGI native directives
  `sgix` — Produce OpenMP directives with SGI extensions. Currently, only the 'NEST' directive is supported
  `no` — Do not insert directives in code generation (useful for comparison).

- Different region types

  `loop` — consider only one loop for one region (no pipeline)
  `bloop` — consider one block + one loop for one region (no pipeline)
  `one` — consider one region (region not joined, no pipeline)
  `join` — consider joined region (outer loop nesting, no pipeline)
  `full` — consider full region (region joined and possible pipeline)

  For SGI directives, only "loop" is allowed for the region type (region-type). The default region-type is "loop" for SGI and "full" for OMP.

- Optimization type is intended for possible improvements to be applied, such as loop granularity check, synchronization overhead reduction, and loop transformation. Currently an attempt to reduce synchronization at end-of-loop is implemented. Other optimizations are less defined and / or tested.

  `off` — Do not do any optimization
  `on` — Try to reduce synchronization at end-of-loop
  `o2` — Use logical disprove (slow sometime) for affinity comparison
  `o3` — Perform additional optimization (such as loop transformation) before loop analysis and directive insertion.

- User-defined loop types are read from a file that can be defined via environment variable `CAPO_USERLOOP` or "`user-loop-file`" entry in the parameter file. If a "`userloop.par`" file exists in the current working directory, this file will be taken if the other two methods are not used. The format of this file is:

```
# starts comment
#RoutineName LoopNumber NewType
routine_name loop_count S|P|R|B[options]
```

Entries are specified line-by-line. "Routine_name" is case insensitive. For a program without the main-routine name defined, "MAIN" can be used to indicate the main routine.

"loop_count" is the loop number counted from the beginning of a given routine. A negative "loop_count" indicates the loop (defined by -loop_count) will not be considered for automatic loop transformation.

Currently the following new loop types are supported:

"S" for serial
"P" for parallel
"R" for reduction
"B" for break-type (e.g. so that a parallel region won't be formed around this loop).

The "R" type can optionally be attached with

"[OPR:VAR]" or "[OPR:VAR( )]" list

to indicate the reduction operator and the reduction variable, no space in-between. The second form indicates an array reduction.

- List of directives to be cleared can be read from a file or taken from the default list. The default list contains the following:

```
"cdir$",                /* Cray vector directive */
"cmic$",                /* Cray autotasking directive */
"c$par",                /* PCF (Parallel Computing Forum) directive */
"c$doacross", "c$&",    /* SGI multiprocessing directive */
"c$ ", "c$\t",
"c$omp",                /* OMP directive */
"c$sgi"                 /* SGI OMP extension */
```

The default setting is to use the above list. The 'clearing' action may be turned off by setting CAPO_DIRCLEAR to 'off'. Additional directives may be added to the default list by prefixing a '+' in front of the filename for CAPO_DIRCLEAR.

A dirclear-list file contains simply a list of directives (keywords) to be considered. A keyword should lead with one of ['C', '!', '*']. A '-' sign can be added to the front of a keyword to indicate the corresponding directive should not be cleared (i.e. keep its original form), otherwise, the directive will be commented out (cleared).

- The THREADPRIVATE directive will be generated by default. If the option is turned off via CAPO_TPRIV (=off), CAPO will use an alternative method to treat private variables used in a common block.

  off — Use an alternative method to handle private variables
  on — Try to produce THREADPRIVATE directives

- The comment type refers to the leading character to be used for directives. The 'C' character is for the f77 type and the '!' character is for the f90 type. Default is '!'.

- By default, if a loop is partitioned in a message-passing program, the loop will not be considered for directives (CAPO_USEPARTI=no). This is equivalent to a two-level parallelization. If a partitioned loop is intended for directives as well, CAPO_USEPARTI can be set to 'yes'. This would be a one-level parallelization with mixed type. The option is only

meaningful when CAPTools is first used to generate message-passing program and CAPO is then applied to insert directives.

- Two types of routine duplication (RDUP) can be selected:

  `loop` — as the type for RDUP if a routine is used both inside and outside parallel loop(s).
  `region` — as the (default) type for RDUP if a routine is used inside a parallel loop and inside parallel region but outside parallel loop.

  The first option removes any nesting of parallel regions. The second option allows nested parallel regions in such a form that a parallel region can be nested inside a parallel loop but not inside a non-worksharing section of a parallel region.

- The environment variable `CAPO_UNKSIZE` controls how unknown-size private variable (USPV) is treated. A unknown-size variable has its last dimension declared as "*" or "1" in a subroutine and is in the routine argument list. By default, if an USPV is encountered, CAPO will take effort to adjust the size of the unknown dimension. If the size cannot be adjusted, the corresponding loop will be made serial. If `CAPO_UNKSIZE` is set to "`true`", the loop with USPV will not be made serial, instead, a warning will be printed so that the user can make manual change later on.

- By default I/O statements are not allowed in the dynamic extent of parallel loops. However, one can exploit certain degrees of parallel I/O with `CAPO_PIO`.

  `no` — no I/O statements in the dynamic extent of a loop (default).
  `incall` — no I/O in the current scope of a loop, but allowed inside subroutine calls.
  `write` — allow "`WRITE(*,*)`", i.e. write to the standard output.
  `noread` — no `READ`, but allow any `WRITE`.
  `any` — allow any type of I/O statements.

A1.4. Parameters for Debugging Purpose

The following parameters are only available from the Setting box (Section A3.6) in the Directives browser. By default, all these parameters are enabled. The Setting box can be used to disable them for debugging purpose.

`Generate-NOWAIT`	— enable / disable the `NOWAIT` directive
`Transform-Induction-Loop`	— enable / disable induction loop treatment
`Handle-Array-Reduction`	— enable / disable array reduction
`Remove-Old-Directives`	— enable / disable removing old directives
`Apply-UserLoop-Type`	— enable / disable applying userloop types
`Setup-Pipeline-Loop`	— enable / disable pipeline loop

A1.5. Sample Parameter File

```
# env: CAPO_PAR
# Parameters for CAPTools-based Parallelizer with OpenMP (CAPO)
# They apply to version 1.1

# env: CAPO_LOG
# defines if log-information is wanted
log-file                on      (off on stdout)
# env: CAPO_LOGNAME
```

```
# defines log-file name when log-file = on
log-file-name                   (default: codeoutput.log)

# env: CAPO_LOGINFO
# defines type of information to be logged
log-info                std     (min std more debug)

# env: CAPO_PLOOP
# defines granularity (min. no. of iters.) for parallel loops
loop-granularity        6       (0 1 2 ...)

# env: CAPO_TYPE
# defines type of directives to be produced
directive-type          omp     (omp sgi sgix no)

# env: CAPO_REGION
# defines type of parallel regions to be considered
region-type             full    (loop bloop one join full)

# env: CAPO_OPTIMIZE
# defines optimization type for parallel regions
optimize-type           o2      (off on o2 o3)

# env: CAPO_USERLOOP
# defines the file name for user-defined loop types
user-loop-file                  (default: user-loop.par)

# env: CAPO_DIRCLEAR
# defines the file name for directives to be cleared
directive-clear         Default (off on filename)

# env: CAPO_TPRIV
# switches on/off the generation of THREADPRIVATE
tpriv-directive         on      (off on)

# env: CAPO_COMMENT
# chooses a comment type for directives
comment-type            f90     (f77 f90)

# env: CAPO_USEPARTI
# uses partitioned loops for directives
use-parti-loop          no      (no yes)

# env: CAPO_RDUPTYPE
# defines routine duplication type
rdup-type               region  (loop region)

# env: CAPO_UNKSIZE
# allows unknown-size variables
allow-unksize           false   (false true)

# env: CAPO_PIO
# allows parallel I/O
allow-pio               no      (no incall write noread any)
```

A2. Messages and Symbols in the Log File

By default, the process of automatic insertion of directives is logged to the log-file "`code-output.log`". Information in this file may be examined after directives are added. There are three main sections in the log file, as outlined in the following subsections. Depending on the log-info type as described in Section A1, different levels of information details may be logged. In general, the log-info type controls:

1) `min` — only minimum amount of information, such as WARNING and INFO messages,
2) `std` — information from `min`, plus summary for each routine and each region,
3) `more` — information from `std`, plus more detailed results for each loop and each region,
4) `debug` — information from `more`, plus additional debug information that are probably too much for an ordinary user.

In the case of "`more`" and "`debug`", additional labels (region# and loop#) are added as comments for parallel loops in the generated parallel code. Regions and loops are labeled within a given routine, sequentially.

A2.1. Classification of Loops

The first section lists the analysis of loops in all routines from the dependence information. For a given routine a loop is labeled with its sequence number, the group number and the loop-nesting level. The group number is defined as a sequence number for a loop-nest group at a given nesting level. Loops are classified as parallel, serial, or possible pipeline. For a parallel loop, it is further tested for granularity and is indicated if a parallel directive is to be added, provided the loop is not nested inside another parallel loop. For a serial loop, the reason of serialization as well as the first variable that causes the loop to be serialized is given. The causes of loop serialization include loop-carried dependences (true, anti and output), I/O statement inside, and breaking out of the loop. A pipeline loop is a serial loop with only loop-carried true dependences and determinable dependence vectors (see Section 2.4 for definition). The basic information for loops is as the following:

```
Routine: ROUTINE_NAME
  Loop # (loop_variable), group #, level #: parallel/serial
       TYPE? Reason for serial...
```

"TYPE?" is one of types from the loop type list:

```
   "REDU", "NPAR", "PAR", "IO", "LVAR", "SER", "ANTI", "PIPE",
   "BRK", "UPIPE", "PAREG", "INDU", "INPLP", "RDINP", "GRAN", "PARTI"
```

As an example, part of the analysis for three routines in NPB-LU is given here (with `log_info` set to MORE).

```
Routine: BUTS
 Loop 1 (J), group 1, level 1: parallel, granularity - ok
        PAR-> directives to be added for the loop <1,1>
 Loop 2 (I), group 1, level 2: parallel, granularity - ok
        INPLP? no directive, loop inside a parallel loop
 Loop 3 (M), group 1, level 3: parallel, granularity - no
 Loop 4 (J), group 2, level 1: serial
```

```
        PIPE? true dependence, pipeline loop? dvector: V[0,0,-1,0]
 Loop 5 (I), group 2, level 2: serial
        PIPE? true dependence, pipeline loop? dvector: V[0,-1,0,0]
 Loop 6 (M), group 2, level 3: parallel, granularity - no
 Loop 7 (M), group 2, level 3: parallel, granularity - no
 *** Total number of loops: 7, parallel: 5, serial: 2, directive: 1
Routine: JACU
 Loop 1 (J), group 1, level 1: parallel, granularity - ok
        PAR-> directives to be added for the loop <1,1>
 Loop 2 (I), group 1, level 2: parallel, granularity - ok
        INPLP? no directive, loop inside a parallel loop
 *** Total number of loops: 2, parallel: 2, serial: 0, directive: 1
...
Routine: SSOR
 Loop 1 (I), group 1, level 1: serial
        ANTI? loop carried output or non-exact anti dependence: ELAPSED
 Loop 2 (I), group 2, level 1: serial
        ANTI? loop carried output or non-exact anti dependence: ELAPSED
 Loop 3 (ISTEP), group 3, level 1: serial
        BRK? break out of the loop or comm-call inside the loop
 Loop 4 (K), group 3, level 2: parallel, granularity - ok
        PAR-> directives to be added for the loop <2,1>
 Loop 5 (J), group 3, level 3: parallel, granularity - ok
        INPLP? no directive, loop inside a parallel loop
 Loop 6 (I), group 3, level 4: parallel, granularity - ok
        INPLP? no directive, loop inside a parallel loop
 Loop 7 (M), group 3, level 5: parallel, granularity - no
 Loop 8 (K), group 3, level 2: serial
        SER? loop carried true dependence: ELAPSED
 Loop 9 (K), group 3, level 2: serial
        SER? loop carried true dependence: ELAPSED
 Loop 10 (K), group 3, level 2: parallel, granularity - ok
        PAR-> directives to be added for the loop <2,2>
 Loop 11 (J), group 3, level 3: parallel, granularity - ok
        INPLP? no directive, loop inside a parallel loop
 Loop 12 (I), group 3, level 4: parallel, granularity - ok
        INPLP? no directive, loop inside a parallel loop
 Loop 13 (M), group 3, level 5: parallel, granularity - no
 *** Total number of loops: 13, parallel: 8, serial: 5, directive: 2

>>>> Grand total: num_routines 25, num_loops 157
           loops: parallel 145, serial 12, directive 30
```

The label for a parallel loop with directive to be added (PAR->) is given as <level,group> pairs. In the case of a serial loop only one variable is listed for the cause of serialization. For a potential pipeline loop, the dependence vector for the first related variable is given, as the case of V[0,0,-1,0] for loop 4 (J) in routine BUTS.

The user-defined loop types are applied after the loop classification. Therefore, it is user's responsibility to ensure the correctness of user-supplied loop types.

A2.2. Construction and Optimization of Parallel Regions

This section contains first the summary from the pass-two analysis of all the routines in the outer-most loop level to decide if directives need to be added in a routine. Routines are traversed on their call

sequences. A <yes> or <no> flag is marked for each analyzed routine to indicate the addition of directives in the routine. A routine may need to be duplicated if it is called both inside and outside a parallel loop and will contain directives in itself.

```
Routine: ROUTINE_NAME <yes/no/inploop/noploop>
```

<yes> — routine is added with directives for parallel loops
<no> — routine has no directives
<inploop> — routine is called inside a parallel loop
<noploop> — routine has no parallel loop, but may contain potential pipeline loops

A sample result from the analysis of NPB-LU looks like the following.

```
Routine: APPLU <yes>
Routine: READ_INPUT <no>
Routine: DOMAIN <no>
Routine: SETCOEFF <no>
Routine: SETBV <yes>
Routine: SETIV <yes>
Routine: ERHS <yes>
Routine: SSOR <yes>
Routine: TIMER_CLEAR <no>
Routine: JACLD <yes>
Routine: BLTS <yes>
Routine: JACU <yes>
Routine: BUTS <yes>
Routine: RHS <yes>
Routine: TIMER_START <no>
Routine: L2NORM <yes>
Routine: TIMER_STOP <no>
Routine: ELAPSED_TIME <no>
Routine: WTIME <no>
Routine: ERROR <yes>
Routine: EXACT <no>
Routine: PINTGR <yes>
Routine: VERIFY <no>
Routine: PRINT_RESULTS <no>
Routine: TIMER_READ <no>
>>> Total routines: 25, checked: 24, with directives: 13
    in/outside ploop: 0, in/with ploop: 0, no ploop: 12
    Total directive loops: 30, effective: 30, in ploop: 0
```

The last line of the statistics indicates how many loops can be put with directives, how many of them are really added with directives, and how many of them are nested inside other loops with directives.

Next is to construct parallel regions based on the loop information. A parallel region includes at least one parallel loop or pipeline loop with possible basic blocks in the beginning of the loop. No nested parallel loops are considered at this point. Two neighboring regions can be joined together if no codes other than comments or nops (such as `continue`) exist between the two regions. Individual regions are labeled sequentially within a routine. For each region a number is included in () to indicate the end (or last) region of a joined area of regions. For disjointed regions, the end region is the same as the region itself. Additional information included for a region are: loops in the region and type of the region. Regions are also summarized for a routine as "`region-type-summary`."

```
Region-type:
   one ploop      — containing exactly one parallel loop (no pipeline)
   +prev-block    — one parallel loop plus any preceded basic blocks
   sub ploop      — one or more parallel loops nested at different levels
   pipeline       — potential pipeline
   <default>      — region with joined neighbors

Region-type-summary:
   DEFAULT        — routine contains normal parallel regions
   PIPE           — routine is part of a pipeline region
   UPIPE          — routine contains potential pipeline regions
```

Sample outputs from the analysis of NPB-LU:

```
Region-in-Routine: BUTS
 region-type-summary: UPIPE
 Parallel region 1 (2): loops [1-3]
 Parallel region 2 (2): loops [4-7]
 *** Total number of regions: 2, joined regions: 1
Region-in-Routine: JACU
 region-type-summary: DEFAULT
 Parallel region 1 (1): loops [1-2] one ploop
 *** Total number of regions: 1, joined regions: 1
Region-in-Routine: SSOR
 region-type-summary: DEFAULT
 Parallel region 1 (1): loops [4-7] one ploop
 Parallel region 2 (2): loops [10-13] one ploop
 *** Total number of regions: 2, joined regions: 2
```

Once the initial regions are determined, routines are then checked for possible pipeline regions across routines. If such a region is identified, the pipeline-loop limit is checked against all other parallel loops in the same pipeline region for alignment. If a discrepancy is found, a message will be printed out as either "not the same limit" or "low-high limit swapped." In the first case, the suggested pipeline operation may produce incorrect run-time result and further check of this generated code is needed. In the second case CAPO automatically swaps the loop limit to ensure the consistence. If pipeline loops are not desirable, set the environment variable `CAPO_REGION` to "`join`."

For LU, routines `BUTS` and `JACU` were identified to be part of a pipeline region in routine `SSOR` and information was generated as follows.

```
Region-in-Routine: BUTS
 region-type-summary: PIPE
 pipeloop: DO J=JEND,JST,-1 (BUTS)
 thisloop: DO J=JEND,JST,-1 (BUTS)
   same limit
Region-in-Routine: JACU
 region-type-summary: PIPE
 pipeloop: DO J=JEND,JST,-1 (BUTS)
 thisloop: DO J=JST,JEND,1 (JACU)
   low-high limit swapped!
Region-in-Routine: SSOR
 region-type-summary: DEFAULT
 Parallel region 1 (1): loops [4-7] one ploop
```

```
 Parallel region 2 (2): loops [8-8] pipeline
 Parallel region 3 (3): loops [9-9] pipeline
 Parallel region 4 (4): loops [10-13] one ploop
 *** Total number of regions: 4, joined regions: 4

>>>> Grand total: routines 25, regions 34, joined regions 26
```

Parallel regions are further optimized for removal of end-of-loop synchronization (use the 'NOWAIT' construct). Although more conservative approach is taken, careful examination of NOWAIT is still needed. For example, one should pay attention to the WARNING messages on "EndLoop-Sync required/re-enforced." If any problem occurs, one can always switch the optimization off (`setenv CAPO_OPTIMIZE off`).

For LU, this is the summary after region optimization:

>>>> Total number of syncs removed: 7, in 4 routines (13 checked)

A2.3. Insertion of Directives in Routines

There are four functions performed in this stage:

- clearing any old directives if CAPO_DIRCLEAR is not off (Section A1.3),
- searching for threadprivate common blocks and inserting the THREADPRIVATE directive if CAPO_TPRIV is not off,
- duplicating routines if needed, and
- inserting region/loop-level directives.

Information resulted from these four actions are not fed back to the Directives Browser except for presented as directives in the source code. Thus, once directives are inserted, the Directives Browser should not be used to do further changes.

A threadprivate common block is the one that have all its variables used as private (including copyin) for all the parallel regions in the whole program. It means even a single instance of a non-private usage of a variable can prevent the common block from becoming threadprivate. In the debug mode, causes of a common block being determined as threadprivate or shared can be examined (see Section A2.4 for details). Normally messages are printed for identified threadprivate common blocks and routines that contain them. An example is given here.

```
T_PRIV common blocks:
 -/WORK_1D/-18: SP SET_CONSTANTS EXACT_RHS INITIALIZE ADI TXINVR X_SOLVE
       NINVR Y_SOLVE PINVR Z_SOLVE LHSINIT TZETAR ADD VERIFY ERROR_NORM
       COMPUTE_RHS RHS_NORM
 -/WORK_LHS/-18: SP SET_CONSTANTS EXACT_RHS INITIALIZE ADI TXINVR X_SOLVE
       NINVR Y_SOLVE PINVR Z_SOLVE LHSINIT TZETAR ADD VERIFY ERROR_NORM
       COMPUTE_RHS RHS_NORM

>>> THREADPRIVATE directive added for 2 common blocks in 18 routines
```

Warnings may be printed for those common blocks that may potentially be threadprivate:

```
WARNING! SSOR... region 4, loop 8
```

```
/CJAC/ Type conflict: old SHARED, new PRIV - use SHARED
```

It indicates that in routine SSOR all variables in common block /CJAC/ are used as private in region 4, but the common block is shared in other places. One can trace further for where the common block is shared in the debug mode.

Directives are added by annotating the call graph and using the parallel region information obtained in A2.2. The call paths are printed as the insertion is progressing. Any routine is only visited one time.

```
Routine: APPLU
Routine: APPLU->SETCOEFF
Routine: APPLU
Routine: APPLU->SETBV
Routine: APPLU
Routine: APPLU->SETIV
Routine: APPLU
Routine: APPLU->ERHS
Routine: APPLU
Routine: APPLU->SSOR
Routine: APPLU->SSOR->RHS
Routine: APPLU->SSOR->RHS->TIMER_START
Routine: APPLU->SSOR->RHS->TIMER_START->ELAPSED_TIME
Routine: APPLU->SSOR->RHS->TIMER_START->ELAPSED_TIME->WTIME
Routine: APPLU->SSOR->RHS->TIMER_START->ELAPSED_TIME
Routine: APPLU->SSOR->RHS->TIMER_START
Routine: APPLU->SSOR->RHS
Routine: APPLU->SSOR->RHS->TIMER_STOP
Routine: APPLU->SSOR->RHS
Routine: APPLU->SSOR
Routine: APPLU->SSOR->L2NORM
INFO! Array reduction variable replaced with local critical in region 1 -
        SUM() --> SUM_CAP1()
Routine: APPLU->SSOR
Routine: APPLU->SSOR->JACLD
Routine: APPLU->SSOR
Routine: APPLU->SSOR->BLTS
Routine: APPLU->SSOR
WARNING! Potential memory conflict for shared variable in region <2,1> -
ELAPSED
Routine: APPLU->SSOR->JACU
Routine: APPLU->SSOR
Routine: APPLU->SSOR->BUTS
Routine: APPLU->SSOR
WARNING! Potential memory conflict for shared variable in region <3,1> -
ELAPSED
Routine: APPLU
Routine: APPLU->ERROR
INFO! Array reduction variable replaced with local critical in region 1 -
        ERRNM() --> ERRNM_CAP1()
Routine: APPLU
Routine: APPLU->PINTGR
Routine: APPLU
Routine: APPLU->VERIFY
Routine: APPLU
```

WARNINGs for "...variable used after a parallel region," "potential memory conflict," and INFOs on the changes made to routine arguments should be examined carefully. These are just warnings, may or may not cause any programming errors. The warnings are the cases where CAPO are uncertain of decision making and user needs to inspect the generated code at the pointed places for verification. The parallel region is labeled as <region_number, parallel_loop_number> pairs in the call path right preceding the warning message.

Meanings of keywords in the WARNING message:

"variable"	— a variable used in the current routine scope
"common-variable"	— a variable used outside the current scope, e.g. through COMMON blocks or SAVE statements in a subroutine
"Shared"	— variable shared in the current region
"Plocal"	— potential private variable in the current region
"Control"	— variable with multiple control paths, i.e. variable could be updated either inside or outside the current region
"I/O statement"	— routine called inside a parallel region contains i/o (OPEN,READ,WRITE,CLOSE) statements
"STOP statement"	— routine called inside a parallel region contains STOP/PAUSE statements
"Potential memory conflict"	— for shared variable that can cause memory conflict in a parallel region

If a private variable in a parallel region is updated via a COMMON block in a subroutine, CAPO tries to privatize such a variable by adding it to the subroutine's argument list and renaming the original variable in the COMMON block of the subroutine. CAPO will generate the following INFO messages in this process:

```
New argument () added to CALL OTHER_ROUTINE():# in ROUTINE_NAME
New symbol () added to the argument list of ROUTINE_NAME
Common block /cblk/ duplicated for ROUTINE_NAME
```

CAPO performs a code transformation automatically for a reduction variable that is an array element. The corresponding message is like:

```
Array reduction variable replaced with scalar in region # -
      OLD_ARRAY_ELEMENT --> NEW_SCALAR_VARIABLE
```

A2.4. Debug Information

More information will be logged if CAPO_LOGINFO is set to "debug." These are useful for debugging CAPO. Some of the information are included here for reference only.

- UserLoop information for user-defined loop types

  `Userloop: Defined loop # in routine ROUTINENAME - newtype`

  "newtype" is one of (S, P, R, B) as mentioned in Section A1.3.

- List of old directives to be cleared
- Summary of loop type with list of all dependence vector deltas for pipeline loops

- Three tests during region formation

```
Mem-Conflict check for region #R, loops #L-#L...
   Conflict variables: <var,var...>
Shared-Array check for region #R, loops #L-#L...Assigned <Symbol>
IO-Statement check for region #R, loops #L-#L...
   I/O or Reduction in routine <RoutineName>
```

- List of symbols and types in each region

TYPE

Private	— Local (privatizable) variable
Reduction	— Scalar reduction variable
ArrayReduction	— Array reduction variable
Shared	— Shared variable
LastPrivate	— Usage in and after the region
FirstPrivate	— Usage in and before the region
CopyInOut	— Shared but no or no proof of loop-variable dependence
ThreadPrivate	— Used in a threadprivate common block
UnknownType	— Type not defined yet

CONTROL

No-Control	— Symbol not in a control dependence
Control-Dep	— Symbol in a control dependence

SCOPE

In-Scope	— Symbol defined in the current routine
Not-in-Scope	— Symbol not defined in the current routine (defined via common block or save statement)
Not-in-Use	— Symbol passed into a subroutine but not used in the subroutine

DTYPE:DEPTH (printed in [.:.])

IO	-1, Routine Input/Output
NT	0, Non-exact True
NA	1, Non-exact Anti
NO	2, Non-exact Output
ET	3, Exact True
EA	4, Exact Anti
EO	5, Exact Output
CT	6, Control
UN	7, Unknown type

Depth = 0 for loop-independent dependence

- List of routine call types, indicating the usage of a routine inside/outside parallel regions/loops. Five bits are used:

```
bit1 [0x01] called outside parallel region
bit2 [0x02] called inside paregion but outside parallel loop
bit3 [0x04] called inside parallel loop
bit4 [0x08] called outside parallel loop (= bit1 | bit2)
bit5 [0x10] called inside parallel region
```

- Information on updating duplicated routines

```
Replace call to DROUTINE with CAP_DROUTINE in ROUTINE
Removed ROUTINE from the calledby list of DROUTINE
Added   ROUTINE to the calledby list of CAP_DROUTINE
```

- List of symbols and affine expressions for testing loop limits (such as in the removal of end-of-loop synchronizations)

```
HOME (LOOP-VAR-EXPR, #hits) Low <EXPR> High <EXPR> [A1:INDX,A2:INDX..]
     (LOOP-VAR-EXPR, #hits) Low <EXPR> High <EXPR> [B1:INDX,B2:INDX..]
OTHER (NONLOOP-EXPR, #hits) [C1:INDX,C2:INDX..]
      (NONLOOP-EXPR, #hits) [D1:INDX,D2:INDX..]
```

Here <EXPR> is a symbolic expression, A,B,C,D are array names, INDX is the relevant array index. The lists are for both source and sink.

- Summary of fields associated with the ploopinfo data struct, mainly for development purpose.

```
 Loop   Lvar    D/L     Type    G WP IP Nest Flag
Routine: ROUTINE_NAME
    #   var     ?/?     TYPE?   ?  ?  ? n/cn [321]
```

'Loop' — the loop number in a routine
'Lvar' — the loop variable name
'D' — the nesting level of the outermost DO loop containing this loop
'L' — the nesting level of the loop
'Type' — one of type strings given in Section A2.1
'G' — the loop granularity flag (internal info only)
'WP' — '1' containing parallel loop, '0' without parallel loop
'IP' — '1' inside parallel loop, '0' not inside parallel loop
'n' — this loop nest flag (containing nested parallel loop)
'cn' — child loop nest flag (part of nested parallel loops)
'Flag' — three bits for internal usage only

- Symbols and their types in common blocks (for testing threadprivate). Meanings of symbol types:

[U] — Unset
[P] — Private
[R] — Reduction
[A] — ArrayReduction
[S] — Shared (RW)
[s] — Shared (Readonly)
[L] — LastPrivate
[F] — FirstPrivate
[C] — CopyInOut

- Methods used in determining the declaration size of unknown-size variables

```
[NOT ]IDENTICAL SIZE, method 1 (caller declaration) used
MAX(e1,...), MIN(e1,...), method 2|3 (access range in routine) used
NO method - variable NOT safe - <var>
```

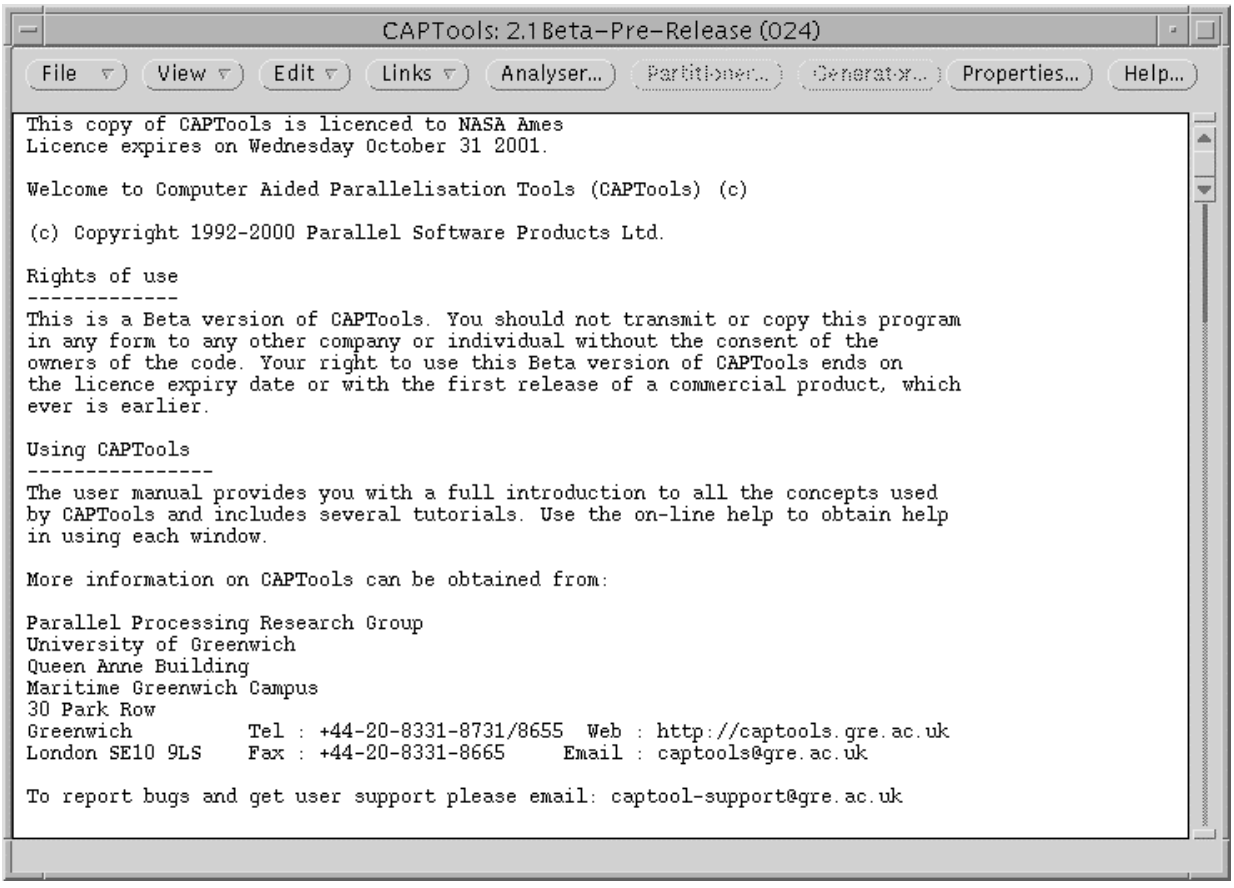
A3. CAPO Graphical User Interface

CAPO is currently integrated into CAPTools as a component to generate OpenMP directives. For CAPO-enabled CAPTools, additional items have been added to the **File**, **View** and **Edit** menus of the CAPTools main window to access the CAPO graphical user interface (GUI).

The CAPO GUI is also referred to as the *Directives Browser*. It provides an easy way for user to access information generated during the directives analysis and insertion. The browser consists of several information windows and dialog boxes as given in the following sections. It also provides hookups to the CAPTools GUI tools, such as DepGraph browser, Variable Definition browser, etc., so that one can easily navigate and interact with the parallelization process.

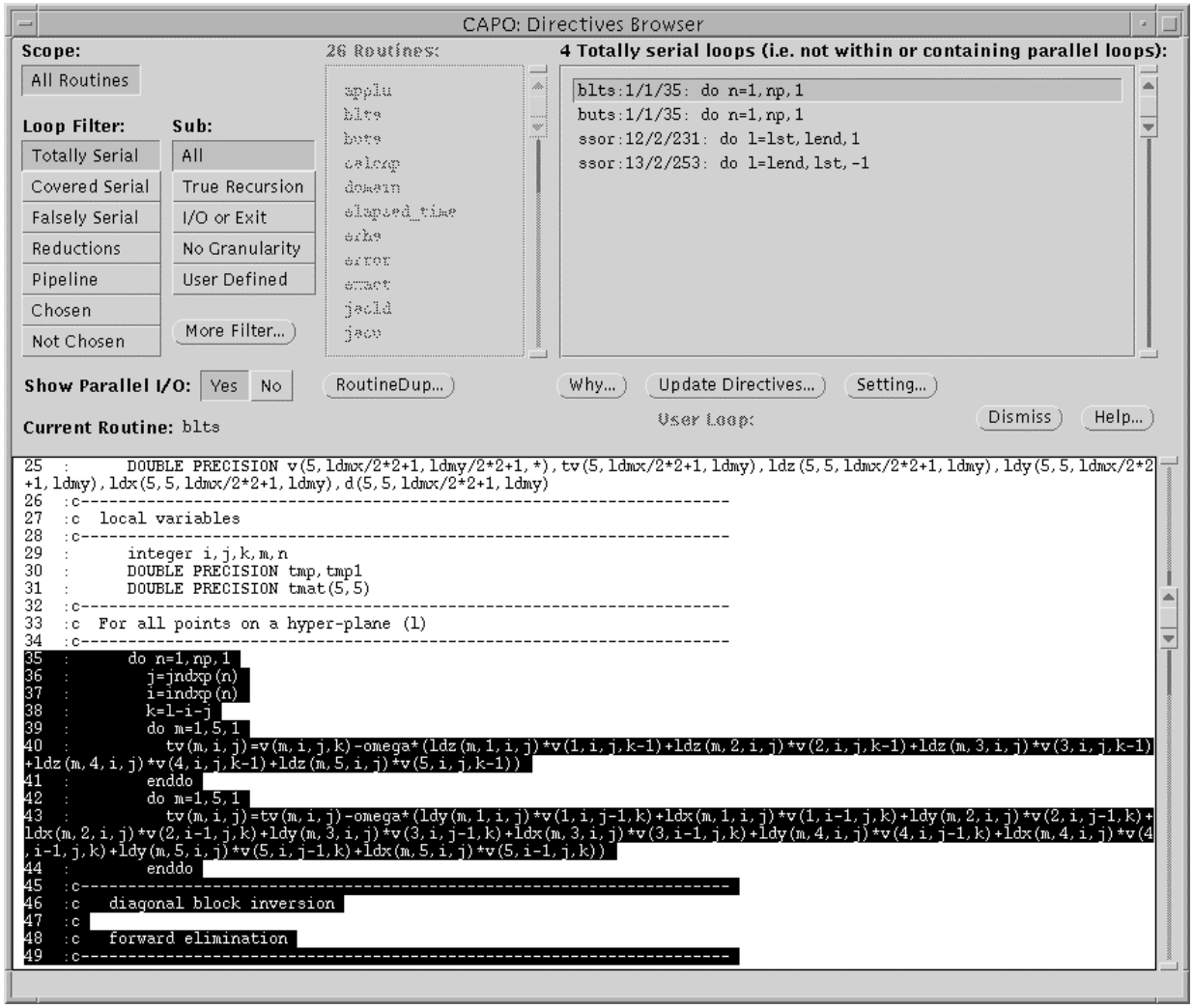
A3.1. CAPTools Main Window

This is the main GUI window the user will see after CAPO/CAPTools is started. The CAPO GUI (the Directives Browser) is started from the **View** (*Directives*) menu after a source file or a database file is loaded from the **File** menu. A summary of CAPO hookups to CAPTools is given in Section A3.12.



A3.2. Directives Browser Main Window

The main window of the Directives browser is activated by **View**–>*Directives*... from the CAPTools main window (see Sections A3.1 and A3.12) after a source or database is loaded in. It presents information from the first two phases of the directives analysis (before directives are added). It is organized around loop types and is an entry point for other browser windows, such as **WhyDirectives** and **RoutineDuplication**. Once directives are generated (via *Save OpenMP Directives Code*), the Directives browser should not be used to do further changes.



Scope [setting]: selects one routine or all routines for loop listing.

Routines [list]: a list of routines that can be selected for loop listing.

Loops [list]: a list of loops under the selected routine/loop filters. To activate the **WhyDirectives** window through the Why... button, a loop needs to be selected.

Loop Filter [list]: provides a way to focus on a particular type of loops, mainly serial or parallel, as described in details in Section A3.3.

Sub [list]: sub-loop filter to be combined with the loop filter to provide finer control of loop selection.

More Filter [button]: activates the **Loop Variable Filter** window to perform even finer loop selection (Section A3.3.1).

Show Parallel I/O [setting]: controls the way that a loop with I/O statements inside is displayed. By default (*Yes*), loops with potential parallel I/O are classified as parallel although parallel I/O with directives is not supported at this point.

RoutDup [button]: activates the **RoutineDuplication** window (Section A3.5).

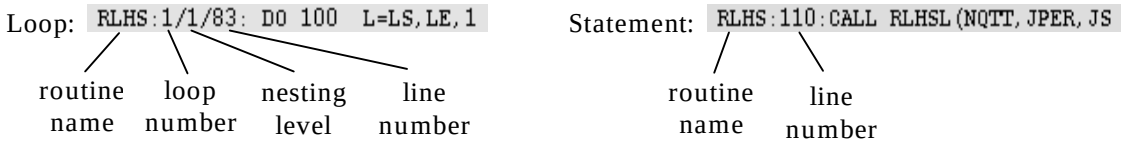
Why [button]: activates the **WhyDirectives** window (Section A3.4) after a loop is selected.

Update Directives [button]: activates the **Update** dialog box (Section A3.9) to re-perform the directives analysis, usually after settings are changed.

Setting [button]: activates the **Setting** window (Section A3.6) to reset parameters for CAPO. The window may also be launched from **Edit**–>*Directives Setting*… in the CAPTools main window.

Current Routine [textpane]: displays the source of a selected routine or a routine in which a selected loop is located. The selected loop nest is highlighted.

How a loop or a statement is labeled:



A3.3. Loop Filters and Sub-filters

Definitions of basic loop types:

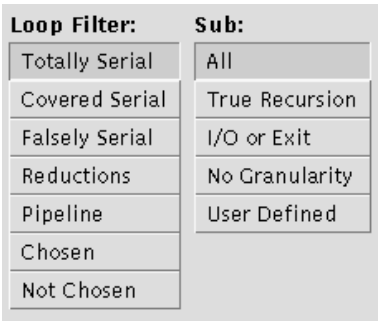
Serial loop — a loop *with* loop-carried TRUE dependence from data flow, ANTI/OUTPUT dependence from non-privatizable variables, I/O statements, and/or exit statements.

Parallel loop — a loop *without* loop-carried TRUE dependence from data flow, ANTI/OUTPUT dependence from non-privatizable variables, I/O statements, and exit statements. Such a loop can be executed in parallel.

Reduction loop — a loop, other than one or more reduction operations, that can be executed in parallel.

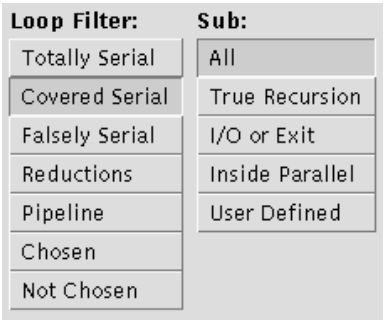
Pipeline loop — a loop that contains loop-carried TRUE dependences with determinable, non-negative dependence vectors (see definition in Section 2.4). The loop can potentially be used to set up a parallel pipeline with an outer loop.

Distributed loop — one of Parallel loop, Reduction loop or Pipeline loop.



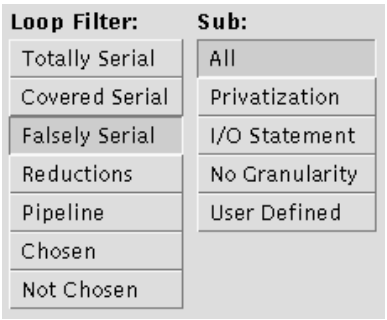
Loop Filter: *Totally Serial* —
serial loop with loop-carried TRUE dependence, containing no distributed loop and not nested inside other distributed loop. The code section in the loop will be executed sequentially.

Sub-filter: *True Recursion* — no I/O or exit statements
I/O or Exit — with I/O and/or exit statements
No Granularity — one or no iteration
User Defined — user-defined serial loop



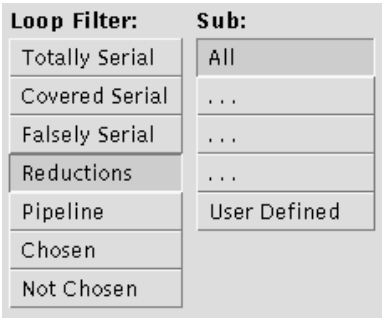
Loop Filter: *Covered Serial* —
serial loop with loop-carried TRUE dependence, containing distributed loop or nested inside other distributed loop. The code section in the loop will partially or completely be executed in parallel.

Sub-filter: *True Recursion* — no I/O or exit statements
I/O or Exit — with I/O and/or exit statements
Inside Parallel — inside other parallel loops
User Defined — user-defined serial loop



Loop Filter: *Falsely Serial* —
serial loop without loop-carried TRUE dependence, but containing ANTI/OUTPUT dependence from non-privatizable variables. Loop may contain distributed loops for parallel execution.

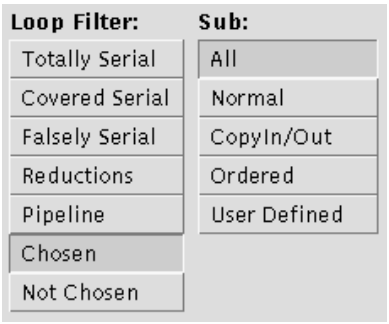
Sub-filter: *Privatization* — due to non-privatizable variables
I/O Statement — with I/O statements but no nested parallel loops
No Granularity — no granularity and no nested parallel loops
User Defined — user-defined serial loop



Loop Filter: *Reductions* —
loop with one or more reduction operations which can be executed as parallel reductions.

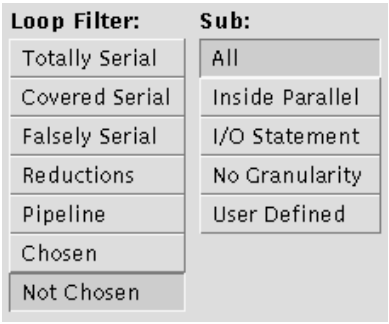
Loop Filter: *Pipeline* —
A pipeline loop as part of a parallel pipeline working with an outer loop.

Sub-filter: *All* — all loops with reductions/pipeline
User Defined — user-defined reduction loop



Loop Filter: *Chosen (Parallel)* **—**
parallel loop chosen for distribution with directives. The code section in the loop will be executed in parallel.

Sub-filter: *Normal* — regular parallel loop
CopyIn/Out — with copyin/copyout variables
Ordered — with ordered code section
User Defined — user-defined parallel loop

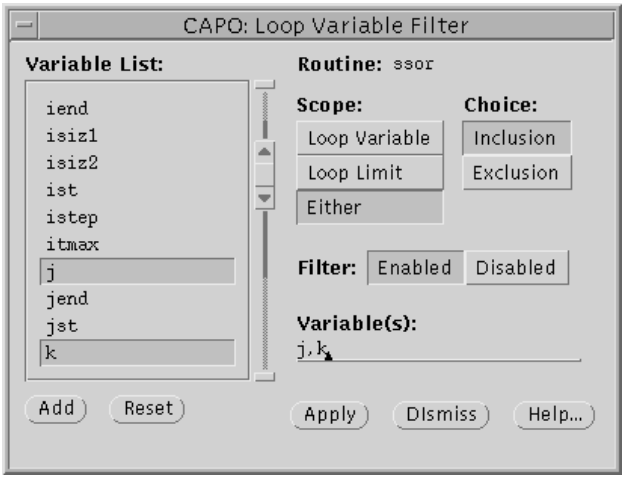


Loop Filter: *Not Chosen (Parallel)* **—**
parallel loop not chosen due to other parallel loop(s) already been chosen. The loop is either inside other distributed loop or contains distributed loops.

Sub-filter: *Inside Parallel* — inside other parallel loops
I/O Statement — with I/O statements
No Granularity — parallel but no granularity
User Defined — user-defined parallel loop

A3.3.1. Loop Variable Filter Window

The Loop Variable Filter Window controls even finer selection of loops in conjunction with the main loop filter and sub filter. The filter applies to variables used in loop heads.



Routine [label]: indicates the currently selected routine.

Variable List [list]: contains a list of variables used in the loop heads of the current routine.

Scope [setting]: controls the scope of variables.

Loop Variable — variables from loop iteration
Loop Limit — variables from loop high-low limit
Either — either of the above two cases

Choice [setting]: controls the filtering effect.

Inclusion — show loops when variables appear
Exclusion — show loops when variables do not appear

Filter [setting]: disables or enables the loop variable filter.

Variable(s) [textfield]: contains a list of the currently filtered variables.

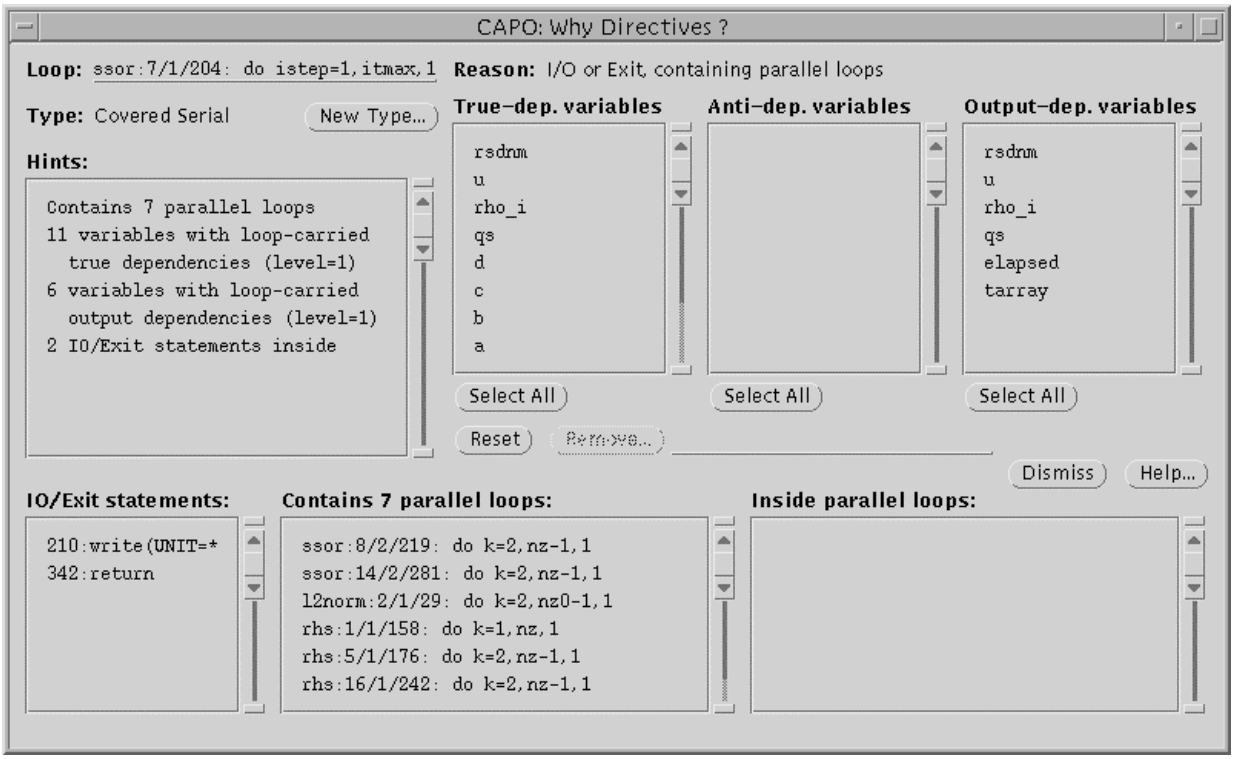
Add [button]: adds the selected variables in the *Variable List* to the filtered variable list.

Reset [button]: resets variable selection.

Apply [button]: applies the current filter to the display.

A3.4. WhyDirectives Window

The **WhyDirectives** window is displayed for a selected loop after the Why… button is clicked in the **Directives** main window. It presents detailed information for the selected loop, in particular, reasons and hints on why the loop was classified as serial or parallel. The window can be used to remove false dependences identified by the user and to redefine the loop type. Depending on the current loop type, the three variable lists may show different types of variables and the two loop lists may present different information. The displayed window is for a loop of the *Covered Serial* type.



The following items are common for All Loop Types.

Loop [textfield]: currently selected loop with routine name and loop labels (see the end of Section A3.3).

Type [textfield]: loop type as described in Section A3.3.

Reason [textfield]: one sentence summarizing why the loop was classified to its type.

Hints [textarea]: more detailed summary of the usage of the relevant variables in the loop and whether the loop contains I/O statements, exit statements, etc.

New Type [button]: activates the **New Loop Type** dialog box (Section A3.7).

Select All [button]: selects all variables in the corresponding variable list.

Reset [button]: deselects all variables in the variable lists.

Remove [button]: activates the **Variable Removal** dialog box (Section A3.10) for the selected variables.

IO/Exit statements [list]: list of I/O and exit statements in the selected loop nest.

The following list is common for Totally Serial and Covered Serial.

True-dep. variables [list]: list of variables causing loop-carried TRUE dependences, *removable*. An "[x]" followed a variable indicates the dependence vector length for this variable.

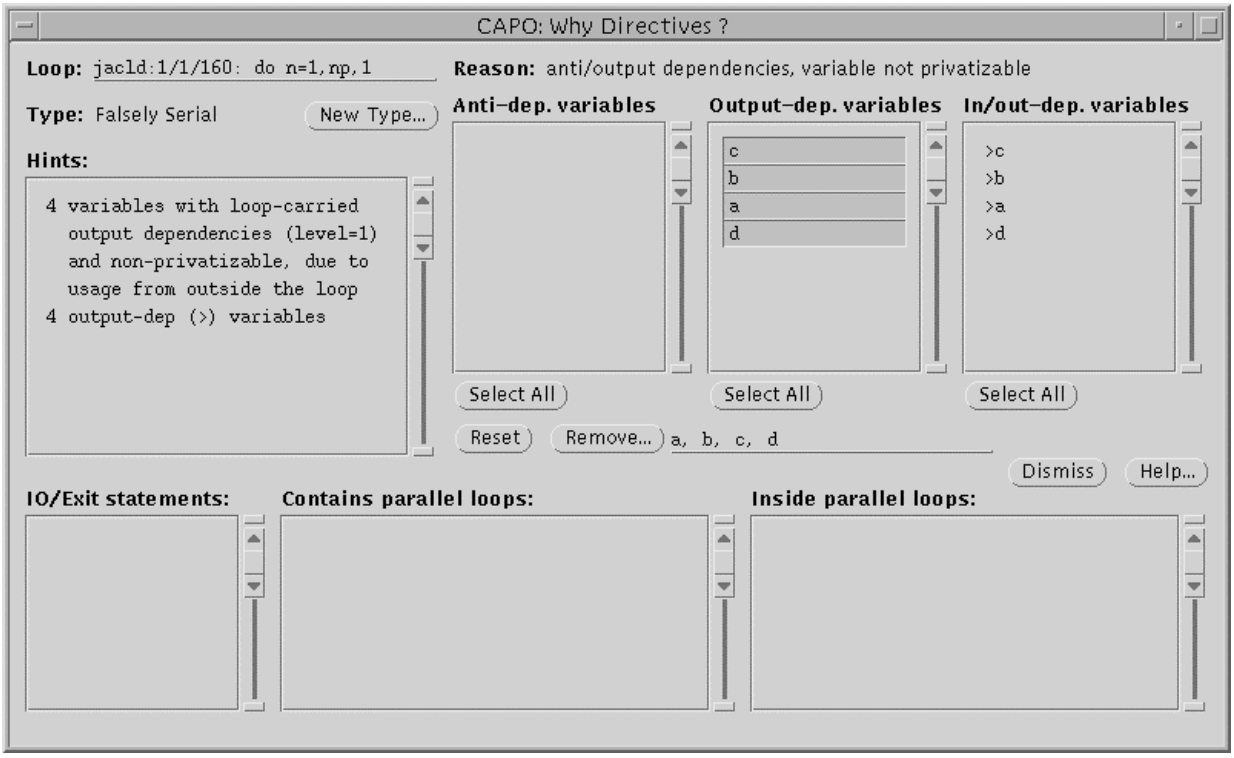
The following lists are common for Totally Serial, Covered Serial and Falsely Serial.

Anti-dep. variables [list]: list of variables causing loop-carried ANTI dependences and the variables cannot be privatized, *removable*.

Output-dep. variables [list]: list of variables causing loop-carried OUTPUT dependences and the variables cannot be privatized, *removable*.

Contains parallel loops [list]: list of parallel loops that are nested inside the current loop.

Inside parallel loops [list]: list of parallel loops that contain the current loop.



The above window is for a *Falsely Serial* loop.

The following list is for Falsely Serial.

In/out-dep. variables [list]: list of variables that have TRUE data dependences from the outside the loop, *removable*. A "<" sign in front of a variable indicates loop entry dependence on this variable, while a ">" sign indicates loop exit dependence on this variable.

The following lists are common for Reductions, Pipeline, Chosen, and Not Chosen.

Private variables [list]: list of privatizable variables in the loop nest, *not removable*.

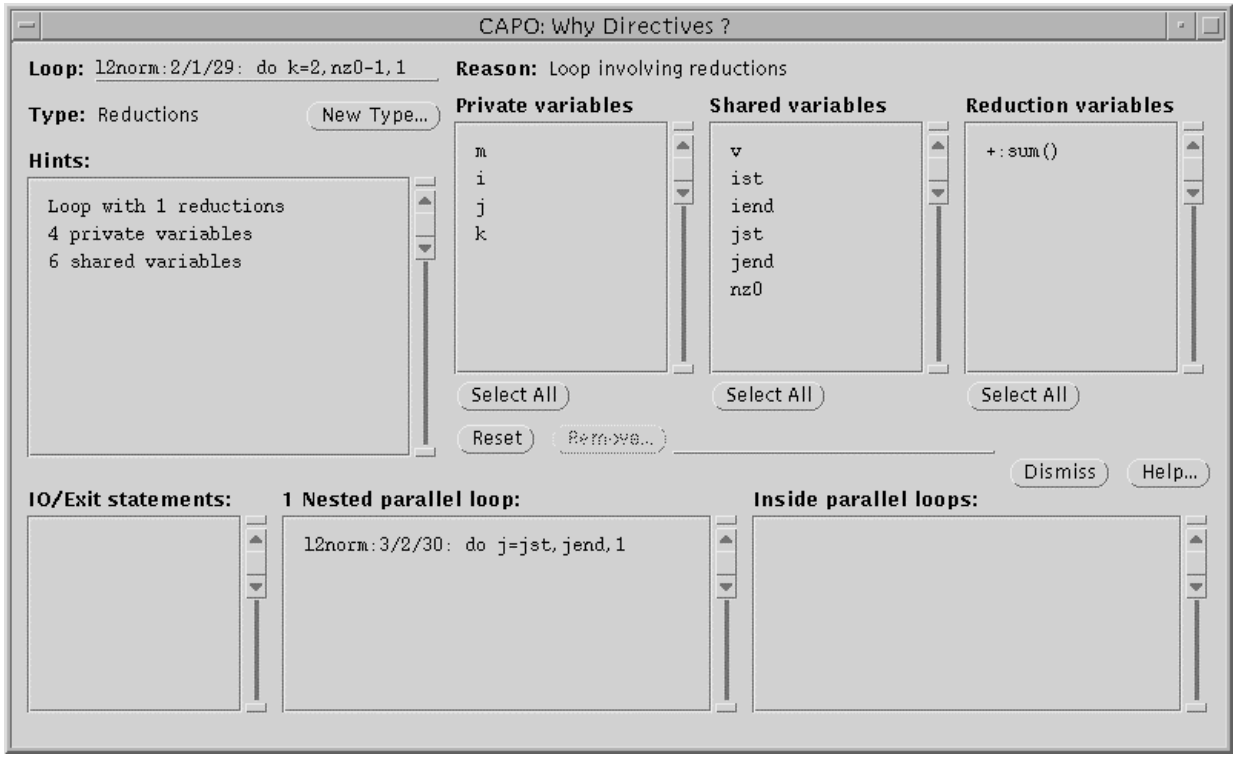
Shared variables [list]: list of shared variables in the loop nest, *not removable*.

Nested parallel loops [list]: list of secondary parallel loops that are nested inside the current loop.

Inside parallel loops [list]: list of parallel loops that contain the current loop (except for Pipeline).

The following list is only for Reduction Loop.

Reduction variables [list]: list of variables for reductions in the loop nest, *not removable*. Reduction variables are preceded with labels indicating reduction operators or intrinsic functions. A "()" after a variable indicates an array reduction.

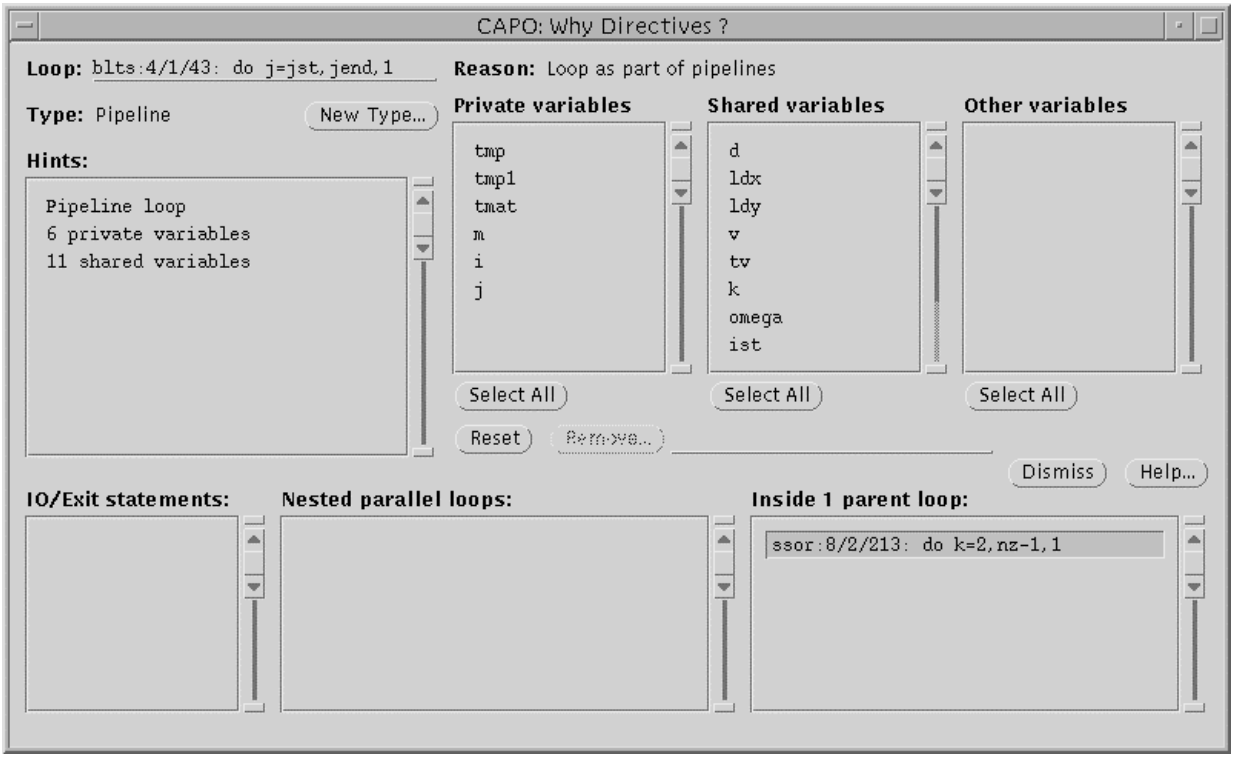


The above window is for a *Reduction* loop with reduction array variable "`sum()`." A reduction operator or intrinsic is one of those defined in Section A3.8 or `IMAX/IMIN` (`MAX/MIN` expressed with an `IF` statement block).

The following lists are only for Pipeline Loop.

Inside parent loops [list]: list of loops that are nested above the current pipeline loop to form pipelines. Appropriate synchronization directives and statements will be inserted at the code generation. A parent loop is usually a serial loop without I/O and exit statement inside.

Other variables [list]: list of variables other than private and shared, such as **CopyIn/CopyOut** variables, *not removeable*.

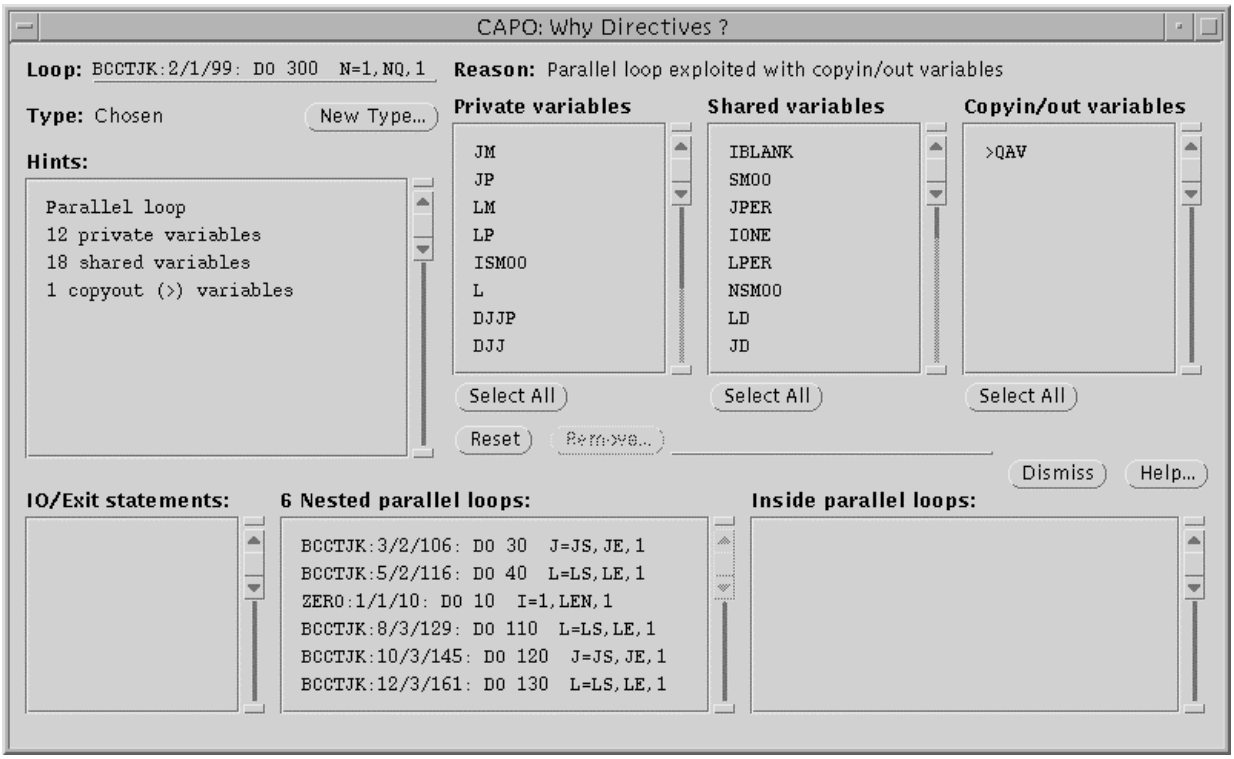


The above window is for a *Pipeline* loop with the parent loop highlighted.

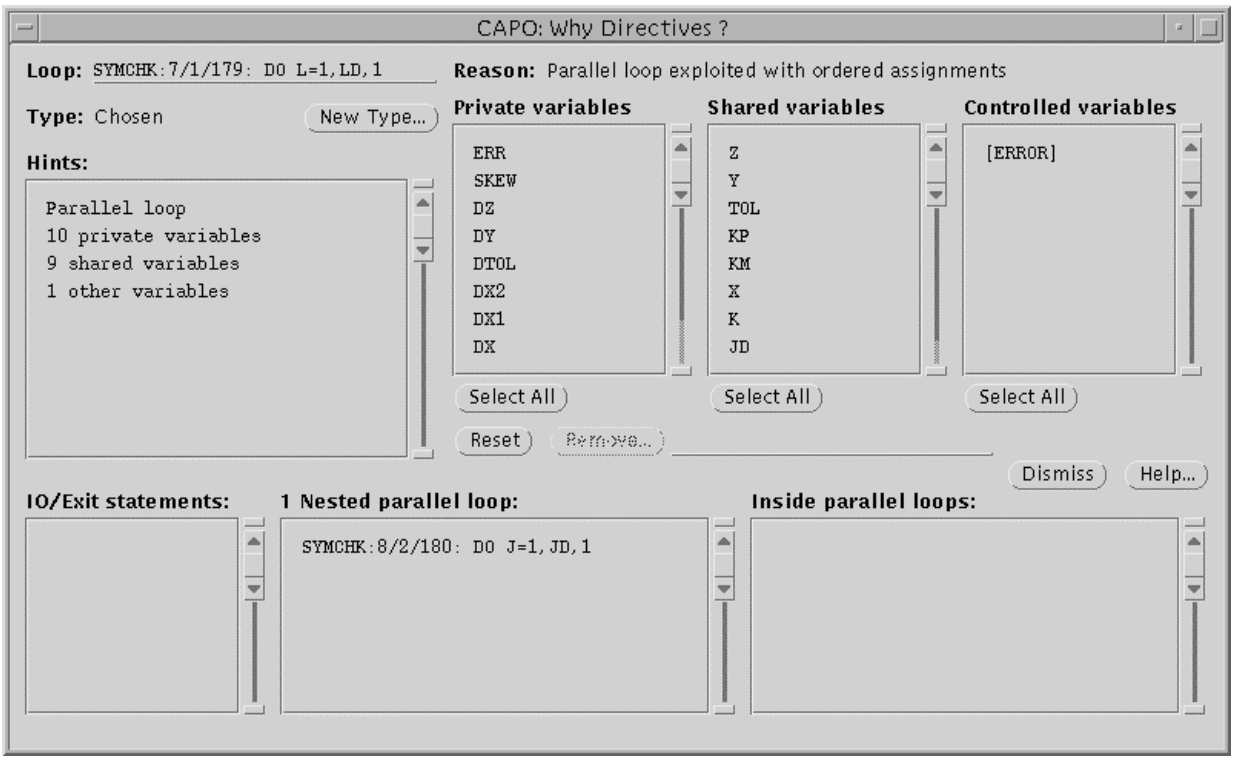
The following lists are only for Chosen Parallel Loop.

Copyin/out variables [list]: list of variables that will be declared as **CopyIn** (FIRSTPRIVATE, marked by "<") and/or **CopyOut** (LASTPRIVATE, marked by ">") due to potential conflict in updating the same memory location and the variable(s) having usage outside the loop. It might arise, for example, from an induction variable that is assigned before the loop and used after the loop. It could also indicate a programming bug.

Controlled variables [list]: list of variables that will be placed inside an "ORDERED" code section. These variables are usually inside IF conditional statements and the corresponding assignments need to be executed in a designated order as is in sequential.



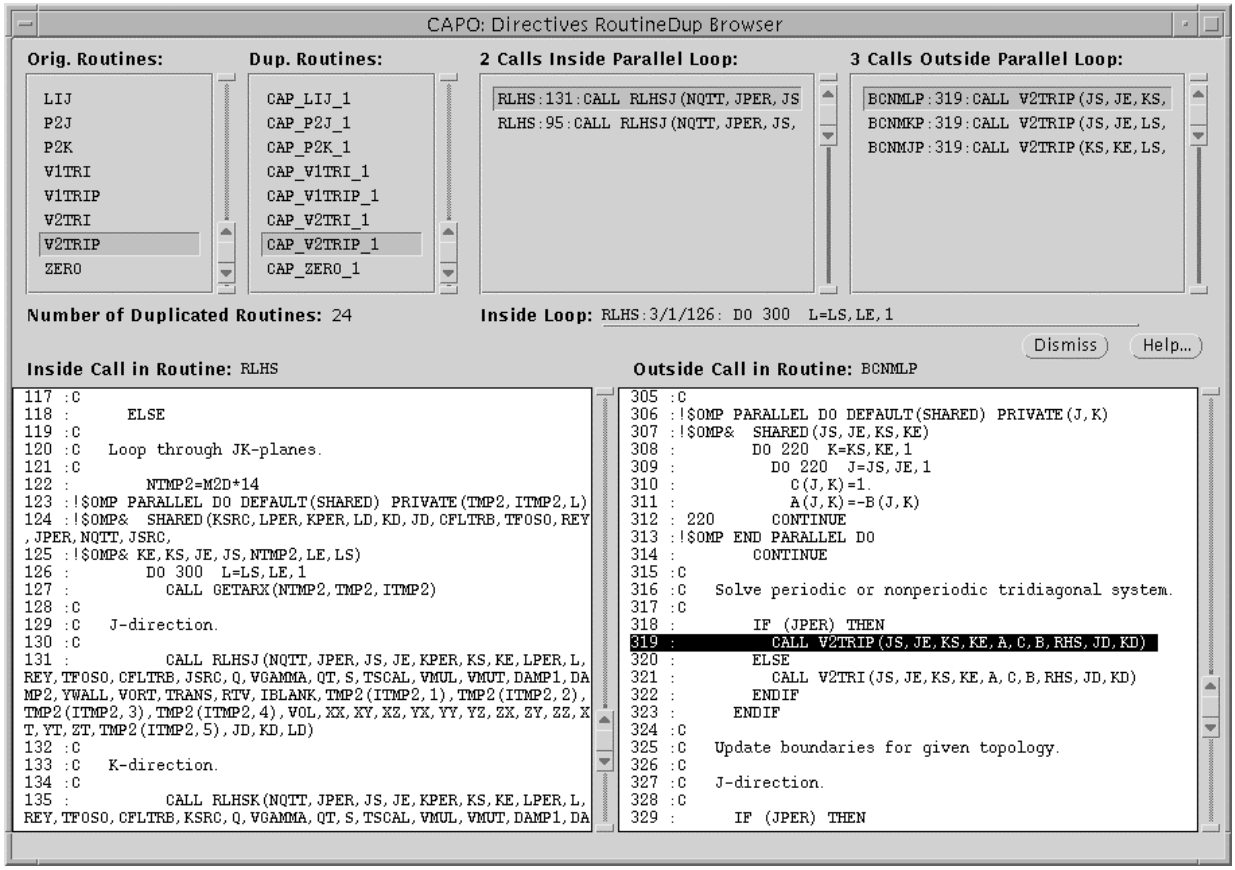
The above windows is for Chosen parallel loop with Copyin/out variables.



The above window is for Chosen parallel loop with Controlled variables.

A3.5. Routine Duplication Browser

The **RoutineDuplication** window is used for browsing routines that are to be or were duplicated to avoid usage conflict of directives. The window is activated from the RoutDup... button in the Directives browser main window.



Orig. Routines [list]: list of original routines to be duplicated.

Dup. Routines [list]: list of duplicated routines. Before code generation, this list will be empty. After code generation, the list is filled with new routines that have one-to-one correspondence to the original routines. The matched (original, duplicated) routine pairs are selected concurrently.

Number of Duplicated Routines [numeric]: as it says.

Calls Inside Parallel Loop [list]: list of call statements (to a selected original routine) that are inside parallel loop(s).

Calls Outside Parallel Loop [list]: list of call statements (to a selected duplicated routine) that are outside any parallel loop.

Inside Loop [textfield]: the loop that contains the selected call statement to an original routine.

Inside Call in Routine [textpane]: the source for the corresponding loop for **Inside Loop**. The textpane is also used for displaying source code for the selected original routine.

Outside Call in Routine [textpane]: the source around the selected call statement from the **Call Outside Parallel Loop** list. The textpane is also used for displaying source code for the selected duplicated routine.

A3.6. Parameter Setting Window

A default setup for the **Parameter Setting** window is displayed on the left. It is launched from either the Setting... button in the Directives main window or the **Edit** → *Directives Setting...* in CAPTools main window. The window is used to reset parameters for CAPO to control the directives analysis and generation. The available parameters and their values are described in Section A1.

CAPO Version: the current version number of CAPO.

Date Built: date on which the current version of CAPO was built.

Update [button]: re-performs directives analysis with the current parameters.

Apply [button]: applies the current parameter setting without performing the directives analysis.

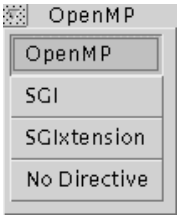
Loop Granularity [numeric]: the minimum number of iterations in a loop for the consideration as a distributed loop. If the number is 0 or if the number of iterations cannot be evaluated, there will be no check on the granularity for the loop.

For detailed information on settings and checks**,** see Section A1.3 and Section A2. The following briefly describes each setting and check box in the window.

Log Information [setting]:

Minimum — minimum log information, such as warning and info messages,
Standard — "Minimum" information plus statistics for loops and regions,
More — "Standard" information plus more detailed loop and region information,
Debug — "More" information plus much more for debugging purpose.

For both *More* and *Debug*, loop and region labels are inserted in the generated source code.



Directive Type [setting]:

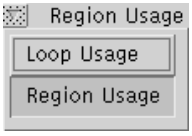
OpenMP — generate OpenMP directives (default),
SGI — generate SGI native directives,
SGIxtension— generate OpenMP directives with SGI extensions,
No Directive— create source file without directives.

Region Type [setting]:

One Loop — only one loop for one region,
Pblk + One Loop — one pre-block plus one loop for one region,
One Region — regions are not joined,
Joined Region — regions are joined, no pipeline consideration,
Full Region — consider joined region and possible pipeline (default).

Optimization [setting]:

Off — do not do any optimization,
On — try to reduce synchronization at end-of-loop,
O2 — use logical disprove (slow sometime) for affinity comparison,
O3 — enable additional optimization (such as automatic loop transformation) before directive insertion.



Routine Duplication [setting]:

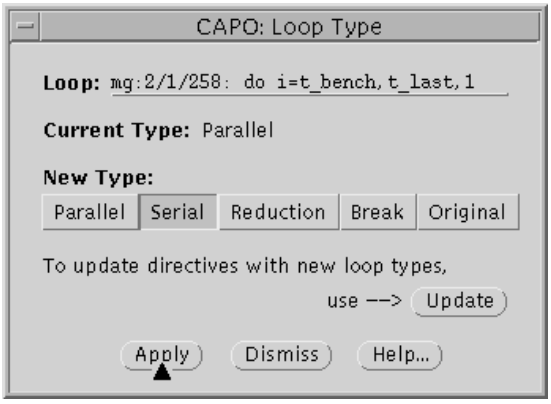
Loop Usage — routine duplicated if it is used both inside and outside parallel Loops (no nested parallel region),
Region Usage — routine duplicated if it is used inside a parallel loop and inside parallel region but outside parallel loop (allow nested parallel region).

Others [checkbox]:

Process ***THREADPRIVATE*** — enable / disable the THREADPRIVATE directive
Generate ***NOWAIT*** — enable / disable the NOWAIT directive
Transform Induction Loop — enable / disable induction loop treatment
Handle Array Reduction — enable / disable array reduction
Remove Old Directives — enable / disable removing old directives
Apply UserLoop Type — enable / disable applying userloop types
Setup Pipeline Loop — enable / disable pipeline loop
Use f77 Comment Style — use f77 (not checked) or f90 (checked) comment style
Allow Unknown-Size Array — enable / disable the use unknown-size array in PRIVATE
Use Partitioned Loop — enable / disable partitioned loop for directives

A3.7. User Loop Type Window

The loop type window is used to redefine a loop type manually. It is displayed for a selected loop by clicking on the New Type button in the **WhyDirectives** window.



Loop [textfield]: print of the selected loop.

Current Type [textfield]: the current loop type.

Update [button]: saves the newly defined loop type to the `userloop.par` file and re-performs the directives analysis with the new setting.

Apply [button]: saves the newly defined loop type to the `userloop.par` file but does not re-perform the directives analysis.

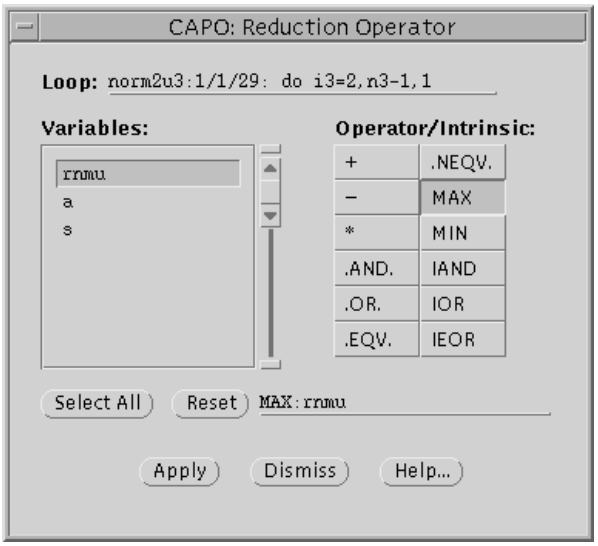
New Type [setting]: one of the selectable types.

Parallel – a parallel loop
Serial – a serial loop
Reduction – a parallel loop with reduction. The Reduction setting may activate an additional dialog box: **Reduction Operator** (See Section A3.8).
Break – a serial loop excluded from any parallel region
Original – the type originally set by CAPO.

An un-selectable type indicates a type that cannot be converted to from the current type.

A3.8. Reduction Operator Dialog

This is a dialog box to select an option (or options) for user-defined reduction loop type. The option specifies reduction operators/intrinsics and variables as part of the entry in the userloop.par file. See Section A1.3 for the description of the userloop.par file.



The dialog box is activated only if the *Reduction* setting in the LoopType window is selected and there exist potential reducible variables detected in the loop by CAPO.

Loop [textfield]: print of the selected loop.

Variables [list]: list of variables that can potentially be selected as reduction variables, *selectable*.

Operator/Intrinsic [setting]: one of the defined reduction operators or intrinsic functions.

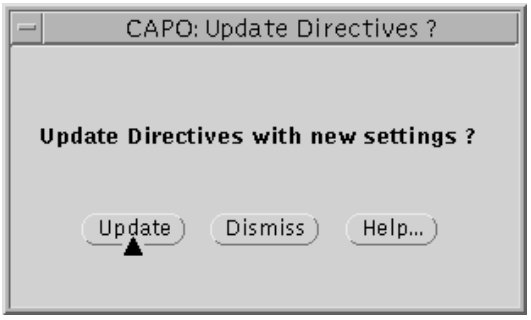
Select All [button]: selects all the variables in the variable list.

Reset [setting]: resets any previous selection. The textfield on the right lists the selected **Operator/ Intrinsic** and variables.

Apply [button]: creates an [operator/intrinsic:variables] combination and add to the option list for the currently selected loop. The option and user-loop type are only stored to the userloop.par file when the Apply or Update button in the **LoopType** window is pressed.

A3.9. Updating Directives Dialog

This is a dialog box for confirming the analysis of directives with new settings. It is popped up after the Update button in the **Directives browser** main window is pushed.



Update [button]: performs the directives analysis, including loop and region level analysis, without generating directives. The dialog will be disabled after the OpenMP directives code is generated.

A3.10. Variable Removal Confirmation Dialog

The dialog is used for confirming the removal of dependences for selected variables and types. The variables and types are determined in the **WhyDirectives** window and the dialog box is activated by pushing the Remove button. This box provides a shortcut to the **DepGraph** for quickly deleting false dependences.

Selected Vars [textfield]: list of selected variables from the **WhyDirectives** window (Section A3.4). A variable listed multiple times indicates it is selected from multiple variable lists in the **WhyDirectives** window.

Apply [button]: applies the removal action.

A3.11. Data Graph Window

The **Data Graph** window is used to create graphs for development purpose. It may have little use to a typical user, but is included for reference. The window is activated from *View→Data Graph* in the CAPTools main window. If the "Data Graph" menu item is not present, try to start CAPO with the [-capodg] option.

Scope [setting]: defines the scope of the routine list.

Graph Type [setting]: chooses from one of the predefined graph types.

Intrinsics [setting]: excludes or includes intrinsic functions in the routine list and in the graph.

Routines [list]: list of routines (name of the file containing a routine).

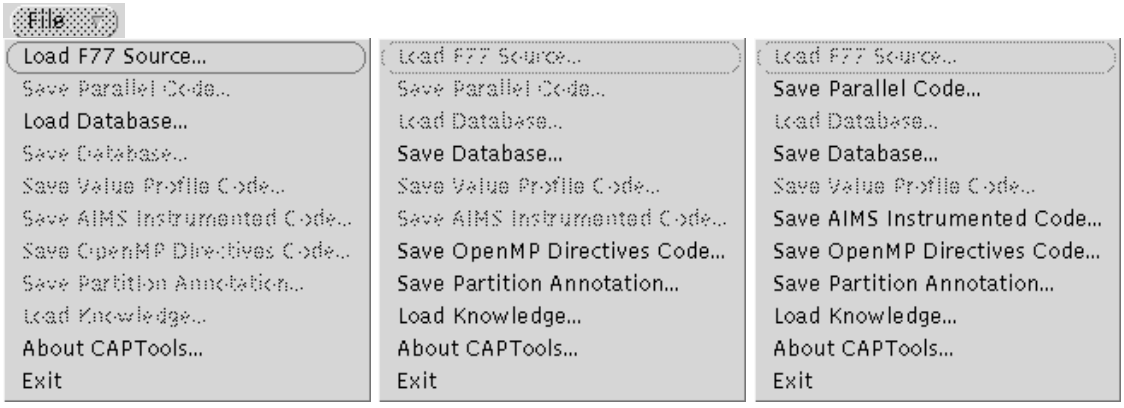
Order [setting]: defines the way routines are listed (Alphabetic, Strict, Reversed Strict).

Blocks [list]: list of basic program blocks in the selected routine.

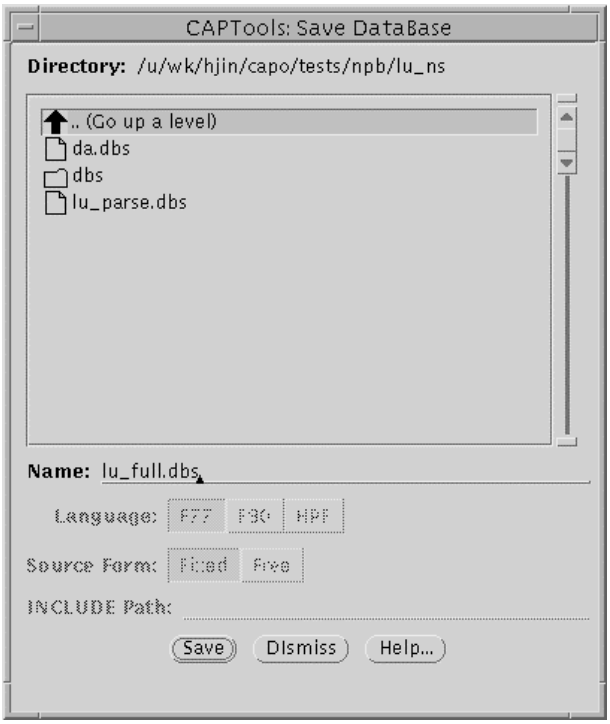
Create [button]: creates a graph for the selected routine and/or block (currently *xvcg* is used to display the graph).

A3.12. Hookups to CAPTools

For CAPO-enabled CAPTools, additional items are added to the **File** (*Save OpenMP Directives Code*), **View** (*Directives*) and **Edit** (*Directives Setting*) menus in the CAPTools main window (Section A3.1). The menu items that are relevant to directives generation are summarized here.



Before source is loaded | After source is loaded | After communication is generated



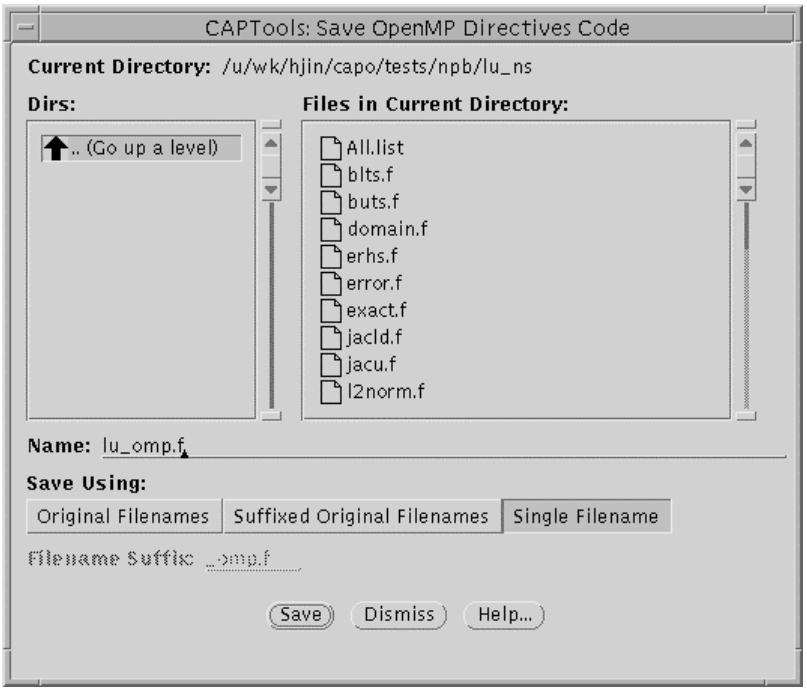
The **Save Database** dialog box.

The **File** menu:

Load F77 Source [entry]: loads Fortran 77 source (.f or .list file).

Load Database [entry]: loads a previously saved database (.dbs file).

Save Database [entry]: saves the current analysis result to a database. As of CAPO Version 1.1, the directives analysis result is not yet saved to the database. But the inserted directives are saved.



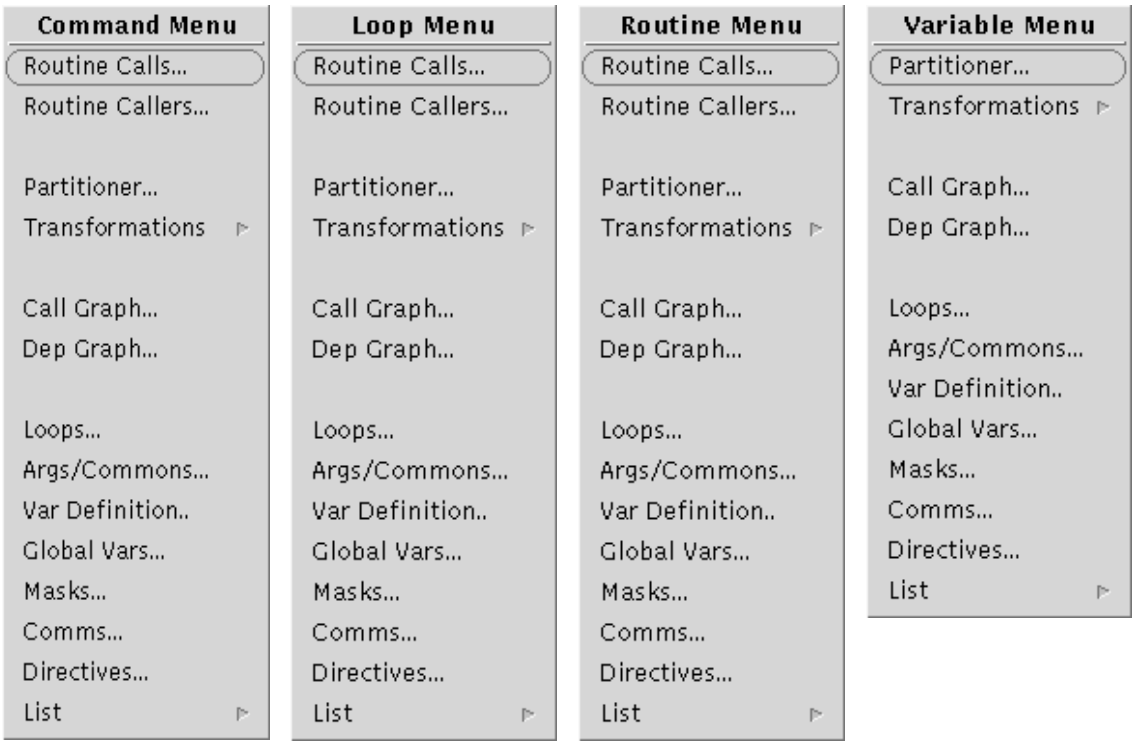
Save OpenMP Directives Code [entry]: performs the directives analysis if it has not been done and generates OpenMP directives. The code can be saved to multiple files or to a single file.

The **Save OpenMP Directives Code** dialog box.

The **View** menu:
Directives [entry]: activates the Directives browser, which performs the directives analysis (if not yet done) and presents information on directives.

The **Edit** menu:
Directives Setting [entry]: activates the Setting dialog box as given in Section A3.6. It can be used to set up parameters for CAPO before the the directives analysis is performed.

The following popup menus are hookups to various tools from selected lists or items in a GUI window, usually activated with a right-mouse-button click.



Command Menu [popup]: for a selected statement.

Loop Menu [popup]: for a selected loop.

Routine Menu [popup]: for a selected routine.

Variable Menu [popup]: for a selected variable.

A4. CAPO Command Interface

The command interface for CAPO is available in Version 1.1 and works closely with the CAPTools command interface. It provides a way to access the functionality of GUI components without starting the GUI. It serves as a means to record actions (to a log file) as a result of any user GUI activities so that these actions can be played back later. The commands in the command interface are usually recorded to a log file or a command file with

```
capo –logfile capo_run.cmd
```

and played back with

```
capo [-batch] capo_run.cmd.
```

The second line with the `[-batch]` option can be used to start a CAPO session in a batch mode.

The command interface for CAPO is different from the command-line version of CAPO, which takes simply the database as input and creates the Fortran output:

```
capo –capoc [-options] database.dbs output.f.
```

This stand-alone version is mostly for testing purpose. The command interface is the preferred method.

A4.1. Commands for the Command Interface

CAPO commands start with the keyword "`capo`" to distinguish them from CAPTools commands.

Main commands:

`load <file.dbs>`
- Load database file

`capo version 1`
- Define CAPO command version

`capo removedep <routine> <variable> <loop_number> <dtype> <fc> [<drout>]`
- Remove loop-related data dependences
  * `routine` - routine name
  * `variable` - relevant variable in the routine
  * `loop_number` - loop to be considered
  * `dtype` - dependence type: 1 for loop-carried TRUE dependences
    2 for TRUE dependences from outside loop
    3 for loop-carried ANTI dependences
    4 for loop-carried OUTPUT dependences
  * `fc` - `1` father list, `2` child list, `0` both lists
  * `[drout]` – optional field to define routine in which the variable is actually declared (if it is different from <`routine`>)

`capo update [0/1]`
- Perform directives analysis with the new setting
  `'0'` for initial analysis, `'1'` for new update

`capo passtwo`

- Re-perform the pass-two analysis

`capo generate [<file.f>]`
- Generate OpenMP directives. `<file.f>` is used to define the logfile name, i.e. <`file.log`>. If <`file.f`> is not given, "`capo-info.log`" is assumed for the logfile name.

`save source <file.f> 3 0`
- Save source code to `<file.f>`
  `'3'` indicates a single file

("`load`" and "`save`" are two CAPTools commands. See A-4.2 for details.)

Parameter setting commands:

`capo set log-file on/off/stdout`
- Turn on / off information logging, default is on

`capo set log-file-name <filename>`
- Define log filename, default is `"capo-info.log"`

`capo set log-info minl/std/more/debug`
- Select log information type, default is `std`

`capo set loop-granularity <value>`
- Set loop granularity threshold, default value = `6`

`capo set directive-type omp/sgi/sgix/no`
- Select directive type, default is `omp`

`capo set optimize-type off/o1/o2/o3`
- Set the optimization type, default is `o2`

`capo set user-loop-file <filename>`
- Define user loop file, default is `"userloop.par"`

`capo set directive-clear off/on/<filename>`
- Turn on / off old directive clearing, default is `on`
  A `<filename>` is used to define a new set of directives

`capo set comment-type f77/f90`
- Set the comment type for directive, default is `f90`

`capo set use-parti-loop yes/no`
- Allow the partitioned loop for directive, default is `no`

`capo set rdup-type loop/region`
- Select the routine duplication type, default is `region`

`capo set allow-pio no/incall/write/noread/any`
- Allow parallel I / O type, default is `no`

Setting commands for debugging purpose:

`capo set mflag <mflag_value>`
- Define the module flag
  `<mflag_value>` can be `<number>`/`<m1:m2..>` with `[+-]` sign

`capo set region-type default/loop/bloop/one/join/full`
- Set a region type, default is `full`

`capo set tpriv-directive on/off`
- Turn on / off the generation of `THREADPRIVATE`, default is `on`

`capo set allow-unksize true/false`
- Allow the use of unknown-size private variables, default is `false`

`capo set have-pipeloop true/false`
- Generate pipeline loop, default is `true`

`capo set have-induc true/false`
- Treat parallel induction loop, default is `true`

`capo set have-arreduc true/false`
- Treat array reduction, default is `true`

`capo set have-nowait true/false`
- Generate the `NOWAIT` directive, default is `true`

`capo set apply-userloop yes/no`
- Apply user defined loop types, default is `yes`

`capo set apply-dirclear yes/no`
- Apply old directive clearing, default is `yes`

A4.2. Other CAPTools Commands Useful for CAPO

`version 2`
- Define CAPTools command version

`load <file.f/file.list/file.dbs>`
- Load source / database file

`save database <file.dbs>`
- Save to database

`save source <dir/suffix/file.f> <1/2/3> 0`
- Save source with type `1, 2 or 3`
  Type `1:` Save to original files, `<dir>` is required for directory name
  Type `2:` Save to original files with `<suffix>`, `<dir/suffix>` required
  Type `3:` Save to a single file with file name `<file.f>`

```
set exact on
set scaler on
set knowledge on
set disproofs on
set interprocedural on
set logic on
```
- Settings for the analysis power

`add read knowledge applu:76:((nx-5 .GT. 0))`
- Define read user knowledge

`analyse`
- Perform dependence analysis

A4.3. An Example of "capo_run.cmd"

```
version 2
load applu_full.dbs
capo version 1
capo set log-file-name applu_omp.log
capo update 0
```

```
capo removedep setbv u 1 4 0
capo removedep setbv u 3 4 0
capo removedep setbv u 5 4 0
capo update 1
capo generate
save source applu_omp.f 3 0
```

To use the command file, do `"capo -batch capo_run.cmd"`.