

Details

Operators legally allowed in a reduction

Operator	Initialization value
+	0
*	1
-	0
&	~0
	0
^	0
&&	1
	0

Schedule types for the loop construct

- static** Iterations are divided into chunks of size *chunk_size*, and the chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.
- dynamic** Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain to be distributed.
- guided** Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain to be assigned. The chunk sizes start large and shrink to the indicated *chunk_size* as chunks are scheduled.
- auto** The decision regarding scheduling is delegated to the compiler and/or runtime system.
- runtime** The schedule and chunk size are taken from the run-sched-var ICV.

Copyright © 1997-2008 OpenMP Architecture Review Board. Permission to copy without fee all or part of this material is granted, provided the OpenMP Architecture Review Board copyright notice and the title of this document appear. Notice is given that copying is by permission of the OpenMP Architecture Review Board. Products or publications based on one or more of the OpenMP specifications must acknowledge the copyright by displaying the following statement: “OpenMP is a trademark of the OpenMP Architecture Review Board. Portions of this product/publication may have been derived from the OpenMP Language Application Program Interface Specification.”

Runtime Library Routines

Execution environment routines affect and monitor threads, processors, and the parallel environment. Lock routines support synchronization with OpenMP locks. Timing routines support a portable wall clock timer. Prototypes for the runtime library routines are defined in the file “omp.h”.

Execution Environment Routines

void omp_set_num_threads(int num_threads);

Affects the number of threads used for subsequent **parallel** regions that do not specify a **num_threads** clause.

int omp_get_num_threads(void);

Returns the number of threads in the current team.

int omp_get_max_threads(void);

Returns maximum number of threads that could be used to form a new team using a “parallel” construct without a “num_threads” clause.

int omp_get_thread_num(void);

Returns the ID of the encountering thread where ID ranges from zero to the size of the team minus 1.

int omp_get_num_procs(void);

Returns the number of processors available to the program.

int omp_in_parallel(void);

Returns *true* if the call to the routine is enclosed by an active **parallel** region; otherwise, it returns *false*.

void omp_set_dynamic(int dynamic_threads);

Enables or disables dynamic adjustment of the number of threads available.

int omp_get_dynamic(void);

Returns the value of the *dyn-var* internal control variable (ICV), determining whether dynamic adjustment of the number of threads is enabled or disabled.

void omp_set_nested(int nested);

Enables or disables nested parallelism, by setting the *nest-var* ICV.

int omp_get_nested(void);

Returns the value of the *nest-var* ICV, which determines if nested parallelism is enabled or disabled.

void omp_set_schedule(omp_sched_t kind, int modifier);

Affects the schedule that is applied when **runtime** is used as schedule kind, by setting the value of the *run-sched-var* ICV.

**void omp_get_schedule(omp_sched_t *kind,
int *modifier);**

Returns the schedule applied when **runtime** schedule is used.