QUARK: QUeuing And Runtime for Kernels

0.9.0

Generated by Doxygen 1.6.3

Fri Dec 16 14:12:52 2011

Contents

Chapter 1

Module Documentation

1.1 QUARK: QUeuing And Runtime for Kernels

Functions

- int QUARK_Thread_Rank (Quark *quark)
- void * QUARK_Args_List (Quark *quark)
- int QUARK_Get_RankInTask (Quark *quark)
- void * QUARK_Args_Pop (void *args_list, void **last_arg)
- Quark * QUARK_Setup (int num_threads)
- Quark * QUARK_New (int num_threads)
- void QUARK_Barrier (Quark *quark)
- void QUARK_Waitall (Quark *quark)
- void QUARK_Free (Quark *quark)
- void QUARK_Delete (Quark *quark)
- Task * QUARK_Task_Init (Quark *quark, void(*function)(Quark *), Quark_Task_Flags *task_flags)
- void QUARK_Task_Pack_Arg (Quark *quark, Task *task, int arg_size, void *arg_ptr, int arg_flags)
- unsigned long long QUARK_Insert_Task_Packed_ORIGINAL (Quark *quark, Task *task)
- unsigned long long QUARK_Insert_Task_Packed (Quark *quark, Task *task)
- unsigned long long QUARK_Insert_Task (Quark *quark, void(*function)(Quark *), Quark_Task_Flags *task_flags,...)
- int QUARK_Cancel_Task (Quark *quark, unsigned long long taskid)
- void QUARK_Worker_Loop (Quark *quark, int thread_rank)
- Quark_Sequence * QUARK_Sequence_Create (Quark *quark)
- int QUARK_Sequence_Cancel (Quark *quark, Quark_Sequence *sequence)
- Quark_Sequence * QUARK_Sequence_Destroy (Quark *quark, Quark_Sequence *sequence)
- int QUARK_Sequence_Wait (Quark *quark, Quark_Sequence *sequence)
- Quark_Sequence * QUARK_Get_Sequence (Quark *quark)
- Quark_Task_Flags * QUARK_Task_Flag_Set (Quark_Task_Flags *task_flags, int flag, intptr_t val)
- intptr_t QUARK_Task_Flag_Get (Quark *quark, int flag)
- void QUARK_DOT_DAG_Enable (Quark *quark, int enable)

1.1.1 Detailed Description

These functions are available from the QUARK library for the scheduling of kernel routines.

1.1.2 Function Documentation

1.1.2.1 void * QUARK_Args_List (Quark * *quark*)

Return a pointer to the argument list being processed by the current task and worker.

Parameters

← ***quark*** The scheduler's main data structure.

Returns

Pointer to the current argument list (icl_list_t *)

1.1.2.2 void * QUARK_Args_Pop (void * *args_list*, void ** *last_arg*)

Return a pointer to the next argument. The variable last_arg should be NULL on the first call, then each subsequent call will use last_arg to get the the next argument. The argument list is not actually popped, it is preserved intact.

Parameters

← ***args_list*** Pointer to the current arguments
↔ ***last_arg*** Pointer to the last argument; should be NULL on the first call

Returns

Pointer to the next argument

1.1.2.3 void QUARK_Barrier (Quark * *quark*)

Called by the master thread. Wait for all the tasks to be completed, then return. The worker tasks will NOT exit from their work loop.

Parameters

↔ ***quark*** The scheduler's main data structure.

1.1.2.4 int QUARK_Cancel_Task (Quark * *quark*, unsigned long long *taskid*)

Called by any thread. Cancel a task that is in the scheduler. This works by simply making the task a NULL task. The scheduler still processes all the standard dependencies for this task, but when it is time to run the actual function, the scheduler does nothing.

Parameters

↔ ***quark*** The scheduler's main data structure.
← ***taskid*** The taskid returned by a QUARK_Insert_Task

Returns

1 on success.
-1 if the task cannot be found (may already be done and removed).
-2 if the task is aready running, done, or cancelled.

1.1.2.5 void QUARK_Delete (Quark * *quark*)

Called by the master thread. Wait for all tasks to complete, then join/end the worker threads, and clean up all the data structures.

Parameters

↔ ***quark*** The scheduler's main data structure.

1.1.2.6 void QUARK_DOT_DAG_Enable (Quark * *quark*, int *enable*)

Enable and disable DAG generation in QUARK. Only to be called at the task insertion level by the master thread. Can be called multiple times to enable and disable DAG generation during the runtime. For the output to make sense, this MUST be preceeded by a sync operation such as QUARK_Barrier.

Parameters

↔ ***quark*** Pointer to the scheduler data structure

← ***enable*** Integer: 1 = enable DAG generation; otherwise disable

1.1.2.7 void QUARK_Free (Quark * *quark*)

Called by the master thread. Free all QUARK data structures, this assumes that all usage of QUARK is completed. This interface does not manage, delete or close down the worker threads.

Parameters

↔ ***quark*** The scheduler's main data structure.

1.1.2.8 int QUARK_Get_RankInTask (Quark * *quark*)

Return the rank of a thread inside a parallel task.

Parameters

← ***quark*** The scheduler's main data structure.

Returns

Pointer to the current argument list (icl_list_t *)

1.1.2.9 Quark_Sequence * QUARK_Get_Sequence (Quark * *quark*)

For the current thread, in the current task being executed, return the task's sequence value. This is the value provided when the task was Task_Inserted into a sequence.

Parameters

↔ ***quark*** Pointer to the scheduler data structure

Returns

Pointer to sequence data structure

1.1.2.10 unsigned long long QUARK_Insert_Task (Quark * *quark*, void(*)(Quark *) *function*, Quark_Task_Flags * *task_flags*, ...)

Called by the master thread. Add a new task to the scheduler, providing the data pointers, sizes, and dependency information. This function provides the main user interface for the user to write data-dependent algorithms.

Parameters

↔ ***quark*** The scheduler's main data structure.

← ***function*** The function (task) to be executed by the scheduler

← ***task_flags*** Flags to specify task behavior

← **...** Triplets of the form, ending with 0 for arg_size. arg_size, arg_ptr, arg_flags where arg_size: int: Size of the argument in bytes (0 cannot be used here) arg_ptr: pointer: Pointer to data or argument arg_flags: int: Flags indicating argument usage and various decorators INPUT, OUTPUT, INOUT, VALUE, NODEP, SCRATCH LOCALITY, ACCUMULATOR, GATHERV TASK_COLOR, TASK_LABEL (special decorators for VALUE) e.g., arg_flags INPUT | LOCALITY | ACCUMULATOR e.g., arg_flags VALUE | TASK_COLOR

Returns

A long, long integer which can be used to refer to this task (e.g. for cancellation)

1.1.2.11 unsigned long long QUARK_Insert_Task_Packed (Quark * *quark*, Task * *task*)

Called by the master thread. Add a new task to the scheduler, providing the data pointers, sizes, and dependency information. This function provides the main user interface for the user to write data-dependent algorithms.

Parameters

↔ ***quark*** The scheduler's main data structure.

↔ ***task*** The packed task structure that already has all the arguments associated with the function

Returns

A long, long integer which can be used to refer to this task (e.g. for cancellation)

1.1.2.12 unsigned long long QUARK_Insert_Task_Packed_ORIGINAL (Quark * *quark*, Task * *task*)

Called by the master thread. Add a new task to the scheduler, providing the data pointers, sizes, and dependency information. This function provides the main user interface for the user to write data-dependent algorithms.

Parameters

↔ ***quark*** The scheduler's main data structure.

↔ ***task*** The packed task structure that already has all the arguments associated with the function

Returns

A long, long integer which can be used to refer to this task (e.g. for cancellation)

1.1.2.13 Quark * QUARK_New (int *num_threads*)

Called by the master thread. Allocate and initialize the scheduler data stuctures and spawn worker threads. Used when this scheduler is to do all the thread management.

Parameters

← ***num_threads*** Number of threads to be used (1 master and rest compute workers). If num_threads < 1, first try environment variable QUARK_NUM_THREADS or use use num_threads = number of cores

Returns

Pointer to the QUARK data structure.

1.1.2.14 int QUARK_Sequence_Cancel (Quark * *quark*, Quark_Sequence * *sequence*)

Can be called by any thread. Cancels all the remaining tasks in a sequence using QUARK_Cancel_Task and changes the state so that future tasks belonging to that sequence are ignored.

Parameters

↔ ***quark*** Pointer to the scheduler data structure

↔ ***sequence*** Pointer to a sequence data structure

Returns

0 (QUARK_SUCCESS) on success
-1 (QUARK_ERR) on failure

1.1.2.15 Quark_Sequence * QUARK_Sequence_Create (Quark * *quark*)

Called by the control program. Creates a new sequence data structure and returns it. This can be used to put a sequence of tasks into a group and cancel that group if an error condition occurs.

Parameters

← ***out*** quark Pointer to the scheduler data structure

Returns

Pointer to the newly created sequence structure.

1.1.2.16 Quark_Sequence * QUARK_Sequence_Destroy (Quark * *quark*, Quark_Sequence * *sequence*)

Called by the control program. Cancels all the remaining tasks in a sequence using QUARK_Cancel_Task and deletes the sequence data structure.

Parameters

↔ ***quark*** Pointer to the scheduler data structure

↔ ***sequence*** Pointer to a sequence data structure

Returns

A NULL pointer; which can be used to reset the sequence structure

1.1.2.17 int QUARK_Sequence_Wait (Quark * *quark*, Quark_Sequence * *sequence*)

Called by the control program. Returns when all the tasks in a sequence have completed.

Parameters

↔ ***quark*** Pointer to the scheduler data structure

↔ ***sequence*** Pointer to a sequence structure

Returns

0 on success
-1 on failure

1.1.2.18 Quark * QUARK_Setup (int *num_threads*)

Called by the master thread. This routine does not do thread management, so it can be used with a larger libarary. Allocate and initialize the scheduler data stuctures for the master and num_threads worker threads.

Parameters

← ***num_threads*** Number of threads to be used (1 master and rest compute workers).

Returns

Pointer to the QUARK scheduler data structure.

1.1.2.19 intptr_t QUARK_Task_Flag_Get (Quark * *quark*, int *flag*)

Get the value of various task level flags. Each returned value can be either an integer or a pointer (intptr type).

Select from one of the flags: TASK_PRIORITY : an integer (0-MAX_INT) TASK_LOCK_TO_THREAD : an integer for the thread number TASK_LOCK_TO_THREAD_MASK : a pointer to a bitmask where task can run TASK_LABEL : a string pointer (NULL terminated) for the label TASK_COLOR : a string pointer (NULL terminated) for the color. TASK_SEQUENCE : pointer to a Quark_Sequence structure THREAD_SET_TO_MANUAL_SCHEDULING: boolean integer {0,1} setting thread to manual (1) or automatic (0) scheduling

Parameters

← ***flag*** One of the flags shown above.

Returns

Intptr type giving the value of the flag; -9 on error

1.1.2.20 Quark_Task_Flags * QUARK_Task_Flag_Set (Quark_Task_Flags * *task_flags*, int *flag*, intptr_t *val*)

Set various task level flags. This flag data structure is then provided when the task is created/inserted. Each flag can take a value which is either an integer or a pointer.

Select from one of the flags: TASK_PRIORITY : an integer (0-MAX_INT) TASK_LOCK_TO_THREAD : an integer for the thread number TASK_LOCK_TO_THREAD_MASK : a pointer to a bitmask where task can run TASK_LABEL : a string pointer (NULL terminated) for the label TASK_COLOR : a string pointer (NULL terminated) for the color. TASK_SEQUENCE : takes pointer to a Quark_Sequence structure THREAD_SET_TO_MANUAL_SCHEDULING: boolean integer {0,1} setting thread to manual (1) or automatic (0) scheduling

Parameters

↔ ***flags*** Pointer to a Quark_Task_Flags structure

← ***flag*** One of the flags listed above

← ***val*** A integer or a pointer value for the flag (uses the intptr_t)

Returns

Pointer to the updated Quark_Task_Flags structure

1.1.2.21 Task * QUARK_Task_Init (Quark * *quark*, void(*)(Quark *) *function*, Quark_Task_Flags * *task_flags*)

Called by the master thread. This is used in argument packing, to create an initial task data structure. Arguments can be packed into this structure, and it can be submitted later.

Parameters

↔ ***quark*** The scheduler's main data structure.

← ***function*** The function (task) to be executed by the scheduler

← ***task_flags*** Flags to specify task behavior

1.1.2.22 void QUARK_Task_Pack_Arg (Quark * *quark*, Task * *task*, int *arg_size*, void * *arg_ptr*, int *arg_flags*)

Called by the master thread. This is used in argument packing, to pack/add arguments to a task data structure.

Parameters

↔ ***quark*** The scheduler's main data structure.

↔ ***task*** The task data struture to hold the arguments

← ***arg_size*** Size of the argument in bytes (0 cannot be used here)

← ***arg_ptr*** Pointer to data or argument

← ***arg_flags*** Flags indicating argument usage and various decorators INPUT, OUTPUT, INOUT, VALUE, NODEP, SCRATCH LOCALITY, ACCUMULATOR, GATHERV TASK_COLOR, TASK_LABEL (special decorators for VALUE) e.g., arg_flags INPUT | LOCALITY | ACCUMULATOR e.g., arg_flags VALUE | TASK_COLOR

1.1.2.23 int QUARK_Thread_Rank (Quark * *quark*)

Return the rank of a thread.

Parameters

← ***quark*** The scheduler's main data structure.

Returns

The rank of the calling thread

1.1.2.24 void QUARK_Waitall (Quark * *quark*)

Called by the master thread. Wait for all the tasks to be completed, then return. The worker tasks will also exit from their work loop at this time.

Parameters

↔ ***quark*** The scheduler's main data structure.

1.1.2.25 void QUARK_Worker_Loop (Quark * *quark*, int *thread_rank*)

This function is called by a thread when it wants to start working. This is used in a system that does its own thread management, so each worker thread in that system must call this routine to get the worker to participate in computation.

Parameters

↔ ***quark*** The main data structure.

← ***thread_rank*** The rank of the thread.

1.2 QUARK: Unsupported functions

Functions

- unsigned long long QUARK_Execute_Task (Quark *quark, void(*function)(Quark *), Quark_Task_Flags *task_flags,...)

1.2.1 Detailed Description

These functions are used by internal QUARK and PLASMA developers to obtain very specific behavior, but are unsupported and may have unexpected results.

1.2.2 Function Documentation

1.2.2.1 unsigned long long QUARK_Execute_Task (Quark * *quark*, void(*)(Quark *) *function*, Quark_Task_Flags * *task_flags*, ...)

Run this task in the current thread, at once, without scheduling. This is an unsupported function that can be used by developers for testing.

Parameters

↔ ***quark*** The scheduler's main data structure.

← ***function*** The function (task) to be executed by the scheduler

← ***task_flags*** Flags to specify task behavior

← **...** Triplets of the form, ending with 0 for arg_size. arg_size, arg_ptr, arg_flags where arg_size: int: Size of the argument in bytes (0 cannot be used here) arg_ptr: pointer: Pointer to data or argument arg_flags: int: Flags indicating argument usage and various decorators INPUT, OUTPUT, INOUT, VALUE, NODEP, SCRATCH LOCALITY, ACCUMULATOR, GATHERV TASK_COLOR, TASK_LABEL (special decorators for VALUE) e.g., arg_flags INPUT | LOCALITY | ACCUMULATOR e.g., arg_flags VALUE | TASK_COLOR

Returns

Error value 0 since the task is run at once and there is no need for a task handle.

1.3 QUARK: Depreciated Functions

Functions

- int QUARK_Get_Priority (Quark *quark)
- char * QUARK_Get_Task_Label (Quark *quark)

1.3.1 Detailed Description

These functions have been depreciated and will be removed in a future release.

1.3.2 Function Documentation

1.3.2.1 int QUARK_Get_Priority (Quark * *quark*)

For the current thread, in the current task being executed, return the task's priority value. This is the value provided when the task was Task_Inserted.

Parameters

↔ ***quark*** Pointer to the scheduler data structure

Returns

priority of the task

1.3.2.2 char * QUARK_Get_Task_Label (Quark * *quark*)

For the current thread, in the current task being executed, return the task label. This is the value that was optionally provided when the task was Task_Inserted.

Parameters

↔ ***quark*** Pointer to the scheduler data structure

Returns

Pointer to null-terminated label string
NULL if there is no label

Chapter 2

Data Structure Documentation

2.1 quark_task_flags_s Struct Reference

Data Fields

- int **task_priority**
- int **task_lock_to_thread**
- char * **task_color**
- char * **task_label**
- void * **task_sequence**
- int **task_thread_count**
- int **thread_set_to_manual_scheduling**
- unsigned char * **task_lock_to_thread_mask**

Index