Lithe API Reference

Release 1.0

Kevin Klues, Andrew Waterman, Krste Asanović

February 08, 2013

CONTENTS

CHAPTER

ONE

WELCOME TO LITHE'S API REFERENCE

For the software industry to take advantage of multicore processors, we must allow programmers to arbitrarily compose parallel libraries without sacrificing performance. We believe that high-level task or thread abstractions built around a common global scheduler cannot provide this composition effectively. Instead, the operating system should expose unvirtualized processing resources that can be shared cooperatively between parallel libraries within an application.

Lithe is a system that standardizes and facilitates the exchange of these unvirtualized processing resources between libraries.

CHAPTER

TWO

API REFERENCE

2.1 Harts

To access the harts API, include the following header file:

```
#include <lithe/hart.h>
```

2.1.1 API Calls

```
bool in_hart_context();
int hart_id();
size_t num_harts();
size_t max_harts();
#define hart_set_tls_var(name, val)
#define hart_get_tls_var(name)
```

bool **in_hart_context**()

int **hart_id**()

size_t **num_harts**()

size_t **max_harts**()

#define **hart_set_tls_var**(name, val)

#define **hart_get_tls_var**(name)

2.2 Lithe C Schedulers

To access the Lithe C Scheduelers API, include the following header file:

```
#include <lithe/sched.h>
```

2.2.1 Types

```
struct lithe_sched;
typedef struct lithe_sched lithe_sched_t;
```

```
struct lithe_sched_funcs;
typedef struct struct lithe_sched_funcs lithe_sched_funcs_t;
```

struct **lithe_sched**

lithe_sched_t
Opaque scheduler type passed around between the lithe runtime and any user-level schedulers built on top of lithe.

struct **lithe_sched_funcs**

lithe_sched_funcs_t
A struct containing all of the callback functions that any user-level scheduler built on top of lithe must implement.

```
struct lithe_sched_funcs {
  int (*hart_request) (lithe_sched_t *__this, lithe_sched_t *child, int k);
  void (*hart_enter) (lithe_sched_t *__this);
  void (*hart_return) (lithe_sched_t *__this, lithe_sched_t *child);
  void (*child_enter) (lithe_sched_t *__this, lithe_sched_t *child);
  void (*child_exit) (lithe_sched_t *__this, lithe_sched_t *child);
  void (*context_block) (lithe_sched_t *__this, lithe_context_t *context);
  void (*context_unblock) (lithe_sched_t *__this, lithe_context_t *context);
  void (*context_yield) (lithe_sched_t *__this, lithe_context_t *context);
  void (*context_exit) (lithe_sched_t *__this, lithe_context_t *context);
};
```

int **lithe_sched_funcs_t.hart_request** (lithe_sched_t *__this*, lithe_sched_t **child*, int *k*)
Function ultimately responsible for granting hart requests from a child scheduler. This function is automatically called when a child invokes lithe_hart_request() from within it's current scheduler. Returns 0 on success, -1 on failure.

void **lithe_sched_funcs_t.hart_enter** (lithe_sched_t *__this*)
Entry point for hart granted to this scheduler by a call to lithe_hart_request().

void **lithe_sched_funcs_t.hart_return** (lithe_sched_t *__this*, lithe_sched_t **child*)
Entry point for harts given back to this scheduler by a call to lithe_hart_yield().

void **lithe_sched_funcs_t.child_enter** (lithe_sched_t *__this*, lithe_sched_t **child*)
Callback to inform that a child scheduler has entered on one of the current scheduler's harts.

void **lithe_sched_funcs_t.child_exit** (lithe_sched_t *__this*, lithe_sched_t **child*)
Callback to inform that a child scheduler has exited on one of the current scheduler's harts.

void **lithe_sched_funcs_t.context_block** (lithe_sched_t *__this*, lithe_context_t **context*)
Callback letting this scheduler know that the provided context has been blocked by some external component. It will inform the scheduler when it unblocks it via a call to lithe_context_unblock() which ultimately triggers the context_unblock() callback.

void **lithe_sched_funcs_t.context_unblock** (lithe_sched_t *__this*, lithe_context_t **context*)
Callback letting this scheduler know that the provided context has been unblocked by some external component and is now resumable.

void **lithe_sched_funcs_t.context_yield** (lithe_sched_t *__this*, lithe_context_t **context*)
Callback notifying a scheduler that a context has cooperatively yielded itself back to the scheduler.

void **lithe_sched_funcs_t.context_exit** (lithe_sched_t *__this*, lithe_context_t **context*)
Callback notifying a scheduler that a context has run past the end of it's start function and completed it's work. At this point it should either be reinitialized via a call to lithe_context_reinit() (and friends) or cleaned up via lithe_context_cleanup().

2.3 Lithe C++ Schedulers

To access the Lithe C++ Schedueles API, include the following header file:

```
#include <lithe/sched.hh>
```

2.3.1 Namespaces

```
namespace lithe;
```

2.3.2 Classes

```
class lithe::Scheduler : public lithe_sched_t {
 protected:
  virtual void hart_enter() = 0;

  virtual int hart_request(lithe_sched_t *child, int k);
  virtual void hart_return(lithe_sched_t *child);
  virtual void child_enter(lithe_sched_t *child);
  virtual void child_exit(lithe_sched_t *child);
  virtual void context_block(lithe_context_t *context);
  virtual void context_unblock(lithe_context_t *context);
  virtual void context_yield(lithe_context_t *context);
  virtual void context_exit(lithe_context_t *context);

 public:
  Scheduler();
  virtual ~Scheduler();
};
```

class `Scheduler` : **public** *lithe_sched_t*
    `lithe_sched_t`

void `Scheduler::`**`hart_enter`**`()`

int `Scheduler::`**`hart_request`**`(`lithe_sched_t* *child*, int *k*`)`

void `Scheduler::`**`hart_return`**`(`lithe_sched_t* *child*`)`

void `Scheduler::`**`child_enter`**`(`lithe_sched_t* *child*`)`

void `Scheduler::`**`child_exit`**`(`lithe_sched_t* *child*`)`

void `Scheduler::`**`context_block`**`(`lithe_context_t* *context*`)`

void `Scheduler::`**`context_unblock`**`(`lithe_context_t* *context*`)`

void `Scheduler::`**`context_yield`**`(`lithe_context_t* *context*`)`

void `Scheduler::`**`context_exit`**`(`lithe_context_t* *context*`)`

`Scheduler::`**`Scheduler`**`()`

`Scheduler::`**`~Scheduler`**`()`

2.4 Lithe Contexts

To access the Lithe contexts API, include the following header file:

```
#include <lithe/context.h>
```

2.4.1 Types

```
struct lithe_context_stack;
typedef lithe_context_stack lithe_context_stack_t;

struct lithe_context;
typedef struct lithe_context lithe_context_t;

DECLARE_TYPED_DEQUE(lithe_context, lithe_context_t *);
```

struct **lithe_context_stack**
lithe_context_stack_t
Struct to maintain lithe context stacks

```
struct lithe_context_stack {
  void *bottom;
  ssize_t size;
};
```

lithe_context_stack_t.bottom
Stack bottom for a lithe context.

lithe_context_stack_t.size
Stack size for a lithe context.

struct **lithe_context**
lithe_context_t
Basic lithe context structure. All derived scheduler contexts MUST have this as their first field so that they can be cast properly within the lithe scheduler.

```
struct lithe_context {
  lithe_context_stack_t stack;
  ... // Other "private" fields
};
```

lithe_context_t.stack
The stack associated with this lithe context. This is the only "public" field exposed through the lithe_context_t API. It should be set before calling `lithe_context_init()` on the context.

struct **lithe_context_deque**
A lithe context "deque" type so that lithe contexts can be enqueued and dequeued into list by components external to the lithe runtime. For example, the supplied lithe mutex, barrier and condvar implementations use this construct to hold blocked contexts and resume them later.

2.5 Lithe Runtime

To access the Lithe runtime API, include the following header file:

```
#include <lithe/lithe.h>
```

2.5.1 API Calls

```
int lithe_sched_enter(lithe_sched_t *child)
int lithe_sched_exit()
lithe_sched_t *lithe_sched_current()

int lithe_hart_request(int k)
void lithe_hart_grant(lithe_sched_t *child, void (*unlock_func) (void *), void *lock)
void lithe_hart_yield()

void lithe_context_init(lithe_context_t *context, void (*func) (void *), void *arg)
void lithe_context_reinit(lithe_context_t *context, void (*func) (void *), void *arg)
void lithe_context_recycle(lithe_context_t *context, void (*func) (void *), void *arg)
void lithe_context_reassociate(lithe_context_t *context, lithe_sched_t *sched)
void lithe_context_cleanup(lithe_context_t *context)
lithe_context_t *lithe_context_self()
int lithe_context_run(lithe_context_t *context)
int lithe_context_block(void (*func) (lithe_context_t *, void *), void *arg)
int lithe_context_unblock(lithe_context_t *context)
void lithe_context_yield()
void lithe_context_exit()
```

void **lithe_sched_enter** (lithe_sched_t *child)
: Passes control to a new child scheduler. The 'funcs' field of the scheduler must already be set up properly or the call will fail. It hands the current hart over to this scheduler and then returns. To exit the child, a subsequent call to lithe_sched_exit() is needed. Only at this point will control be passed back to the parent scheduler.

void **lithe_sched_exit** ()
: Exits the current scheduler, returning control to its parent. Must be paired with a previous call to lithe_sched_enter().

lithe_sched_t ***lithe_sched_current** ()
: Return a pointer to the current scheduler. I.e. the pointer passed in when the scheduler was entered.

int **lithe_hart_request** (int *k*)
: Request a specified number of harts from the parent. Note that the parent is free to make a request using the calling hart to their own parent if necessary. These harts will trickle in over time as they are granted to the requesting scheduler. Returns 0 on success and -1 on error.

void **lithe_hart_grant** (lithe_sched_t **child*, void (*unlock_func) (void **)*, void **lock*)
: Grant the current hart to another scheduler. Triggered by a previous call to lithe_hart_request() by a child scheduler. This function never returns. The 'unlock_func()' and its corresponding 'lock' parameter are passed in by a parent scheduler so that the lithe runtime can synchronize references to the child scheduler in the rare case that the child scheduler may currently be in the process of exiting. For example, the unlock function passed in should be the same one used to add/remove the child scheduler from a list in the parent scheduler.

void **lithe_hart_yield** ()
: Yield current hart to parent scheduler. This function should never return.

void **lithe_context_init** (lithe_context_t **context*, void (*func) (void **)*, void **arg*)
: Initialize the proper lithe internal state for an existing context. The context parameter MUST already contain a valid stack pointer and stack size. This function MUST be paired with a call to lithe_context_cleanup() in order to properly cleanup the lithe internal state once the context is no longer usable. This fucntion MUST only be called exactly ONCE for each newly created lithe context. To reinitialize a context with a new start function

without calling lithe_context_cleanup() use lithe_context_reinit() or lithe_context_recycle() as described below. Once the context is restarted it will run from the entry point specified.

void **lithe_context_reinit** (lithe_context_t **context*, void (*func) (void **)*, void **arg*)
Used to reinitialize the lithe internal state for a context already initialized via lithe_context_init(). Normally each call to lithe_context_init() must be paired with a call to lithe_context_cleanup() before the context can be reused for anything. The lithe_context_reinit() function allows you to reinitialize this context with a new start function, context id, and tls region (assuming tls support is enabled in the underlying parlib library). It can be called any number of times before pairing it with lithe_context_cleanup(). Once the context is restarted it will run from the entry point specified.

void **lithe_context_recycle** (lithe_context_t **context*, void (*func) (void **)*, void **arg*)
Similar to lithe_context_reinit() except that the context id and tls region are preserved. Useful when tls and context id should be preserved across lithe scheduler invocations.

void **lithe_context_reassociate** (lithe_context_t **context*, lithe_sched_t **sched*)
Reassociate a lithe context with a new scheduler. Leaving the rest of it alone. Useful when a context is parked on a barrier instead of exited and then reused accross scheduler invocations.

void **lithe_context_cleanup** (lithe_context_t **context*)
Cleanup an existing context. This context should NOT currently be running on any hart, though this is not enforced by the lithe runtime. Once called, lithe_context_reinit() can no longer be called on this context. It is now safe to either call lithe_context_init() to reinitialize this context from scratch, or simply free any memory associated with the context.

lithe_context_t ***lithe_context_self** ()
Returns a pointer to the currently executing context.

int **lithe_context_run** (lithe_context_t **context*)
Run the specified context. MUST only be run from hart context. Upon completion, the context is yielded, and must be either retasked for other use via lithe_context_reinit() (and friends) or cleaned up via a call to lithe_context_cleanup(). This function never returns on success and returns -1 on error.

void **lithe_context_block** (void (*func) (lithe_context_t *, void **)*, void **arg*)
Invoke the specified function with the current context and block that context. This function is useful for blocking a context and managing when to unblock it by some component other than the scheduler associated with this context. The scheduler will recieve a callback notifying it that this context has blocked and should not be run. The scheduler will receive another callback later to notify it when this context has unblocked and can once again be resumed.

void **lithe_context_unblock** (lithe_context_t **context*)
Notifies the current scheduler that the specified context is now resumable. This chould only be called on contexts previously blocked via a call to lithe_context_block().

void **lithe_context_yield** ()
Cooperatively yield the current context to the current scheduler. The scheduler receives a callback notifiying it that the context has yielded and can decide from there when to resume it.

void **lithe_context_exit** ()
Stop execution of the current context immediately. The scheduler receives a callback notifiying it that the context has exited and can decide from there what to do.

2.6 Lithe Default Scheduler Callbacks

To access the Lithe default scheduler callbacks API, include the following header file:

```
#include <lithe/defaults.h>
```

2.6.1 API Calls

```
lithe_context_t* __lithe_context_create_default(bool stack);
void __lithe_context_destroy_default(lithe_context_t *context, bool stack);

int __hart_request_default(lithe_sched_t *__this, lithe_sched_t *child, int k);
void __hart_entry_default(lithe_sched_t *__this);
void __hart_return_default(lithe_sched_t *__this, lithe_sched_t *child);
void __child_enter_default(lithe_sched_t *__this, lithe_sched_t *child);
void __child_exit_default(lithe_sched_t *__this, lithe_sched_t *child);
void __context_block_default(lithe_sched_t *__this, lithe_context_t *context);
void __context_unblock_default(lithe_sched_t *__this, lithe_context_t *context);
void __context_yield_default(lithe_sched_t *__this, lithe_context_t *context);
void __context_exit_default(lithe_sched_t *__this, lithe_context_t *context);
```

lithe_context_t* **__lithe_context_create_default** (bool *stack*)

void **__lithe_context_destroy_default** (lithe_context_t **context*, bool *stack*)

int **__hart_request_default** (lithe_sched_t **__this*, lithe_sched_t **child*, int *k*)

void **__hart_entry_default** (lithe_sched_t **__this*)

void **__hart_return_default** (lithe_sched_t **__this*, lithe_sched_t **child*)

void **__child_enter_default** (lithe_sched_t **__this*, lithe_sched_t **child*)

void **__child_exit_default** (lithe_sched_t **__this*, lithe_sched_t **child*)

void **__context_block_default** (lithe_sched_t **__this*, lithe_context_t **context*)

void **__context_unblock_default** (lithe_sched_t **__this*, lithe_context_t **context*)

void **__context_yield_default** (lithe_sched_t **__this*, lithe_context_t **context*)

void **__context_exit_default** (lithe_sched_t **__this*, lithe_context_t **context*)

2.7 Lithe Mutexes

To access the Lithe mutex API, include the following header file:

```
#include <lithe/mutex.h>
```

2.7.1 Constants

```
enum {
  LITHE_MUTEX_NORMAL,
  LITHE_MUTEX_RECURSIVE,
  NUM_LITHE_MUTEX_TYPES,
};
#define LITHE_MUTEX_DEFAULT

#define LITHE_MUTEX_INITIALIZER
```

LITHE_MUTEX_NORMAL

LITHE_MUTEX_RECURSIVE

NUM_LITHE_MUTEX_TYPES

LITHE_MUTEX_DEFAULT

LITHE_MUTEX_INITIALIZER

2.7.2 Types

```
struct lithe_mutxattr;
typedef struct lithe_mutexattr lithe_mutexattr_t;

struct lithe_mutex;
typedef struct lithe_mutex lithe_mutex_t;
```

struct **lithe_mutexattr**
lithe_mutexattr_t

struct **lithe_mutex**
lithe_mutex_t

2.7.3 API Calls

```
int lithe_mutexattr_init(lithe_mutexattr_t *attr);
int lithe_mutexattr_settype(lithe_mutexattr_t *attr, int type);
int lithe_mutexattr_gettype(lithe_mutexattr_t *attr, int *type);

int lithe_mutex_init(lithe_mutex_t *mutex, lithe_mutexattr_t *attr);
int lithe_mutex_trylock(lithe_mutex_t *mutex);
int lithe_mutex_lock(lithe_mutex_t *mutex);
int lithe_mutex_unlock(lithe_mutex_t *mutex);
```

int **lithe_mutexattr_init** (lithe_mutexattr_t **attr*)

int **lithe_mutexattr_settype** (lithe_mutexattr_t **attr*, int *type*)

int **lithe_mutexattr_gettype** (lithe_mutexattr_t **attr*, int **type*)

int **lithe_mutex_init** (lithe_mutex_t **mutex*, lithe_mutexattr_t **attr*)
Initialize a lithe mutex.

int **lithe_mutex_trylock** (lithe_mutex_t **mutex*)
Try and lock a lithe mutex.

int **lithe_mutex_lock** (lithe_mutex_t **mutex*)
Lock a lithe mutex.

int **lithe_mutex_unlock** (lithe_mutex_t **mutex*)
Unlock a lithe mutex.

2.8 Lithe Semaphores

To access the Lithe semaphores API, include the following header file:

```
#include <lithe/semaphore.h>
```

2.8.1 Constants

```
#define LITHE_SEM_INITIALIZER
```

LITHE_SEM_INITIALIZER

2.8.2 Types

```
struct lithe_sem;
typedef struct lithe_sem lithe_sem_t;
```

struct lithe_sem;
lithe_sem_t;

2.8.3 API Calls

```
int lithe_sem_init(lithe_sem_t *sem, int count);
int lithe_sem_wait(lithe_sem_t *sem);
int lithe_sem_post(lithe_sem_t *sem);
```

int **lithe_sem_init** (lithe_sem_t **sem*, int *count*)
Initialize a lithe semaphore.

int **lithe_sem_wait** (lithe_sem_t **sem*)
Wait on a lithe semaphore.

int **lithe_sem_post** (lithe_sem_t **sem*)
Post on a lithe semaphore.

2.9 Lithe Barriers

To access the Lithe barrier API, include the following header file:

```
#include <lithe/barrier.h>
```

2.9.1 Types

```
struct lithe_barrier;
typedef struct lithe_barrier lithe_barrier_t;
```

struct **lithe_barrier**
lithe_barrier_t

2.9.2 API Calls

```
void lithe_barrier_init(lithe_barrier_t *barrier, int nthreads);
void lithe_barrier_destroy(lithe_barrier_t *barrier);
void lithe_barrier_wait(lithe_barrier_t *barrier);
```

void **lithe_barrier_init** (lithe_barrier_t **barrier*, int *nthreads*)

void **lithe_barrier_destroy** (lithe_barrier_t **barrier*)

void **lithe_barrier_wait** (lithe_barrier_t **barrier*)

2.10 Lithe Condition Variables

To access the Lithe condition variable API, include the following header file:

```
#include <lithe/condvar.h>
```

2.10.1 Types

```
struct lithe_condvar;
typedef struct lithe_condvar  lithe_condvar_t;
```

struct **lithe_condvar**
lithe_condvar_t

2.10.2 API Calls

```
int lithe_condvar_init(lithe_condvar_t* c);
int lithe_condvar_wait(lithe_condvar_t* c, lithe_mutex_t* m);
int lithe_condvar_signal(lithe_condvar_t* c);
int lithe_condvar_broadcast(lithe_condvar_t* c);
```

int **lithe_condvar_init** (lithe_condvar_t* *c*)
Initialize a condition variable.

int **lithe_condvar_wait** (lithe_condvar_t* *c*, lithe_mutex_t* *m*)
Wait on a condition variable.

int **lithe_condvar_signal** (lithe_condvar_t* *c*)
Signal the next lithe context waiting on the condition variable.

int **lithe_condvar_broadcast** (lithe_condvar_t* *c*)
Broadcast a signal to all lithe contexts waiting on the condition variable.

2.11 Lithe Futexes

To access the Lithe futex API, include the following header file:

```
#include <lithe/futex.h>
```

2.11.1 Constants

```
#define FUTEX_WAIT
#define FUTEX_WAKE
```

FUTEX_WAIT

FUTEX_WAKE

2.11.2 API Calls

```
int futex(int *uaddr, int op, int val, const struct timespec *timeout,
          int *uaddr2, int val3);
```

int **futex**(int **uaddr*, int *op*, int *val*, const struct timespec **timeout*, int **uaddr2*, int *val3*)

CHAPTER

THREE

ABOUT

3.1 Mailing Lists

We have created a lithe-users Google group.
Feel free to join the list and post any questions you have there.

3.2 People

Current Contributors:

- Kevin Klues <klueska@cs.berkeley.edu>
- Andrew Waterman <waterman@cs.berkeley.edu>
- Krste Asanović <krste@cs.berkeley.edu>

Past Contributors:

- Heidi Pan
- Benjamin Hindman <benh@cs.berkeley.edu>
- Rimas Avizienis <rimas@cs.berkeley.edu>

INDEX

Symbols

F

H

I

L

M

N

S