Threading in WaveScalar Assembly

Of Coarse it’s Fine

Random Notes

- The .sim files aren’t working for everyone.

Relevant Directories & Files

- ws_workloads/src/include/ws
  - threads.h
  - mutex.h
  - barrier.h
  - tid_acquire.h
  - types.h
- ws_workloads/src/lib/mutex
- ws_workloads/src/lib/threads

Thread Methods

- tid_t thread_create_pc (thread_fn fnptr, wsint64 a1, wsint64 a2, wsint64 a3);
- void thread_detach (tid_t tid);
- wsint64 thread_join (tid_t tid);
- tid_t thread_get_tid (void);

- void initialize_tid_acquire (void);
- void deinitialize_tid_acquire (void);

Spawning Threads - Setup

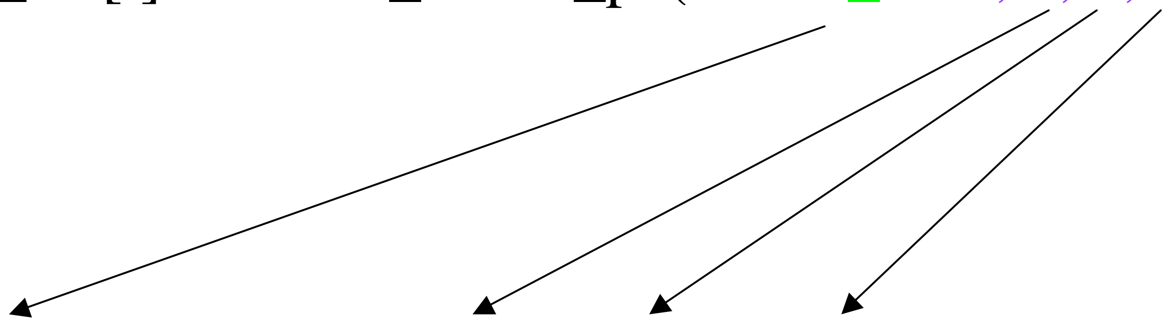
Call initialize and deinitialize tid acquire to get automatic generation of thread IDs.

```
#include <ws/tid_acquire.h>
#include <ws/threads.h>

int main()  {
        initialize_tid_acquire ( );
                //Thread code here
        deinitialize_tid_acquire ( );
}
```

Spawning Threads

```
for(i = 0; i < NUM_SPAWNED; i++) {
    thread_ids[i] = thread_create_pc(thread_todo, 0, 0, 0);
  }
```



```
wsint64 thread_todo(int tid, int a1, int a2, int a3) {
  DOPRINTF(("Inside thread %d\n", tid));
  return tid;
}
```

tid is implicit. It is sent automatically by the assembly macros

Detaching Threads

If you don’t need the threads to join back up to the spawning thread:

```
for(i = 0; i < NUM_SPAWNED; i++) {
    thread_detach( thread_ids[i] );
  }
```

We can write this more cleanly as:

```
for(i = 0; i < NUM_SPAWNED; i++) {
    thread_detach( thread_create_pc ( thread_todo, 0, 0, 0 ) );
  }
```

Joining Threads

```
wsint64 sum = 0;
for(i = 0; i < NUM_SPAWNED; i++) {
  val = thread_join ( thread_ids[i] );
  sum += val;
  DOPRINTF(("Joined w/ thread %d.  Sum is %d\n", val, sum));
 }
```

Join terminates the thread, and returns the 64-bit value defined by the function called in thread_create_pc()

```
thread_ids[i] = thread_create_pc(thread_todo, 0, 0, 0);
…
wsint64 thread_todo(int tid, int a1, int a2, int a3) {…}
```

Barriers

Barriers hold threads until all have arrived, then lets them all go.

```
wsint64 theBarrier = barrier_create(NUM_SPAWNED+1);

for(i = 0; i < NUM_SPAWNED; i++) {
        thread_detach(thread_create_pc(thread_todo_b, theBarrier, 0, 0));
      }

thread_todo_b ( 0, theBarrier, 0, 0 );
DOPRINTF( ("all finished\n") );
barrier_destroy(theBarrier);
```

Barrier Methods

- barrier_id barrier_create (wsint64 max_count);
- void barrier_wait (barrier_id id);
- void barrier_destroy (barrier_id id);
- void barrier_reset (barrier_id id, wsint64 new_max_count);

Barrier_reset releases all of the threads it's holding. Use it with great care (or don't use it at all. Just use a second barrier).

Thread function with Barriers

```
wsint64 thread_todo_b (int tid, int barrier, int a2, int a3)
{
    wsint64  value;

    DOPRINTF(("thread  %d started\n", tid));
    do_phase1( );
    barrier_wait(barrier);
    DOPRINTF(("thread %d finished phase 1\n", tid));
    value = do_phase2( );
    barrier_wait(barrier);
    DOPRINTF(("thread %d finished phase 2\n", tid));

    return value;
}
```

Only matters for thread_join()

Mutex Methods

- mutex_id mutex_create ();
- mutex_token mutex_acquire (mutex_id id);
- void mutex_release (mutex_id id, mutex_token token);

Mutexes

```
struct LockedInt {
  int data;
  mutex_id lock;
};
```

Can use a struct, or you can use mutex_id independently

```
Struct LockedInt intVal
intVal.lock = mutex_create();
mutex_release(intVal.lock, -1);
…
mutex_acquire(intVal[0].lock);
intVal.data ++;  //Do work on protected data
mutex_release(intVal.lock, -1);
```

The thread that creates the mutex automatically acquires the mutex. You must release it before anyone else can acquire it.

Value passed will be returned on next call to mutex_acquire()

Putting it all together

```
//
// Testing ability to spawn threads
//

#include <ws/tid_acquire.h>
#include <ws/threads.h>
#include <ws/barrier.h>

#define NUM_SPAWNED 3

int thread_todo(int tid, int barrier, int a2, int a3)
{
    DOPRINTF(("thread  %d started\n", tid));
    barrier_wait(barrier);
    DOPRINTF(("thread %d finished\n", tid));
    return 0;
}
```

```
int main(int argc, char *argv[]) {
    int theBarrier;
    int i;
    initialize_tid_acquire();

    theBarrier = barrier_create(NUM_SPAWNED+1

    for(i = 0; i < NUM_SPAWNED; i++) {
      thread_detach(thread_create_pc(thread_todo,
theBarrier, 0, 0));
    }

    thread_todo(0, theBarrier, 0,0);

    DOPRINTF(("all finished\n"));

    barrier_destroy(theBarrier);

    deinitialize_tid_acquire();
    return 0;
}
```

More Examples

- Simple examples
  - Regressions/
  - Regressions/libs/threads
  - Regressions/libs/mutex
- Coarse-Grained Threading
  - workloads/fir - Finite Impulse Response
  - workloads/lcs - coarse-grained longest common substring
- Combined fine & coarse-grained threading
  - workloads/fir-ufine
  - workloads/lcs-newfine