Homework - Threads

George Bosilca
bosilca@eecs.utk.edu
A POSIX barrier implementation
What is a barrier?

• Barriers are a synchronization primitive, which trigger once a certain number of participants reached the synchronization point
• The POSIX standard propose a simple interface

```c
int pthread_barrier_init ( pthread_barrier_t *barrier,
                          const pthread_barrierattr_t *attr,
                          unsigned count);

int pthread_barrier_destroy ( pthread_barrier_t *barrier);

int pthread_barrier_wait ( pthread_barrier_t *barrier);
```
What is a barrier?

- We need
  - A counter: to count how many threads are waiting
  - A condition to make them wait until we release the barrier
A possible implementation

```c
int barrier_wait ( barrier_t *barrier) {
    pthread_mutex_lock( &(barrier->mutex) );
    barrier->curcount++;
    if( barrier->curcount == barrier->count) {
        barrier->curcount = 0;
        pthread_cond_broadcast( &(barrier->cond) );
        pthread_mutex_unlock( &(barrier->mutex) );
        return 1;
    }
    pthread_cond_wait( &(barrier->cond),
                       &(barrier->mutex) );
    pthread_mutex_unlock( &(barrier->mutex) );
    return 0;
}
```
Barrier analysis

• Is the previous implementation correct? Please explain.
• If no provide a correct implementation.
Threaded DGEMM
Threaded DGEMM

• Take the best performing routing from the Homework 2 and implement a threaded version
• The goal is to optimize this operation on a multi-core environment.
• Analyze the behavior of your function when the number of threads varies.