Sergi Aliaga

Northeastern University

Homework 2

Question 3

Read the paper by Castelló et al. that considers lightweight threads versus sticking with operating system threads. Then answer the following questions:

a) *Discuss some of the tradeoffs of using OpenMP alone versus adding the capabilities of light-weight threading libraries described in the paper.*

As described in the paper, Open MP is a high-level API to exploit hardware parallelism with easy-to-use pragma instructions to indicate the compiler which portion of the code can be executed in parallel. Then the compiler takes charge of dealing with how the parallelization is carried out. OpenMP allows for some user-level flexibility on some parameters, like the number of threads for a certain parallel task, but in general it is aimed at facilitating parallelization to the user by hiding flexibility. This might be beneficial for parallel tasks in systems with moderate number of cores. However, the usability of Open MP comes at the expense of relatively expensive overheads when dealing with common thread functions (creation, synchronization, etc..) and might be a problem when the number of available cores in exascale systems allow for massive concurrency, with hundreds of millions of threads.

It is in this context where Lightweight Thread (LWT) libraries are more useful. These libraries offer much more flexibility to the programmer when it comes to the parallel execution of a certain task. In addition they are optimized for reducing the overhead caused by conventional OS threading, like OpenMP. Some of them offer unconventional functionalities, like hierarchical levels of execution units, which might be especially useful when dealing with nested parallelism.

In summary, Open MP offers more usability to the programmer at the expense of larger overheads and lower flexibility, while LWT libraries offer even more flexibility than pthreads and reduced overheads of conventional OS threading at the cost of more complex implementations.

b) *Select one of the lightweight threading libraries discussed and provide an example of how you would use it to parallelize a simple vector addition kernel (adding two vectors of single-precision floating point numbers together). You do not need to provide code that compiles, just provide enough syntax to show you know how to use the library.*

An example parallel implementation of the proposed kernel with MassiveThreads would could look like the following (code included in Q3B.c, not to be compiled):

```c
//Library includes corresponsing to MassiveThreads and other
#define LENGTH 100
#define NTHREADS 4

float a[LENGTH], b[LENGTH], c[LENGTH];
myth_mutex_t mutex;
myth_thread_t threads[NTHREADS];

void adder(void *arg){
    int id = (int) arg;
    float sum[LENGTH/NTHREADS];
    for(int i = LENGTH/NTHREADS * id; i < LENGTH/NTHREADS * (id+1); i++){
        sum[i - LENGTH/NTHREADS * id] = a[i] + b[i];
    }
    myth_mutex_lock(&mutex);
    for(int i = LENGTH/NTHREADS * id; i < LENGTH/NTHREADS * (id+1); i++){
        c[i] = sum[i - LENGTH/NTHREADS * id];
    }
    myth_mutex_unlock(&mutex);

}

int main(int argc, char **argv){
    /* Initialize MassiveThreads */
    myth_init();

    /*Mutex creation*/
    myth_mutex_init(&mutex, NULL);

    /*ULT thread creation*/
    for(int i = 0; i < NTHREADS; i++){
        threads[i] = myth_create(&adder, (void *) i);
    }

    /*ULTs join*/
    for(int i = 0; i < NTHREADS; i++){
        myth_join(thread[i], NULL);
    }

    /*Destroy mutex*/
    myth_mutex_destroy(&mutex)

    /* Finalize MassiveThreads */
    myth_fini();
}
```

Figure 1: Implementation of parallel vector addition with MassiveThreads

c) *The paper evaluates the performance of BLAS-1 functions. Discuss what is included in the BLAS-1 subprograms.*

BLAS stands for Basic Linear Algebra Subprograms and, as its name indicates, is a specification that prescribes a set of low-level routines for performing common linear algebra operations such as vector addiction, scalar multiplication, dot products and other. BLAS-1 refers to the level 1 BLAS functions, which consist of all the routines described in the original presentation of BLAS (1979). This routines include only vector operations on strided arrays: dot products, vector norms, a generalized vector addition ("axpy") and several others.