Q3

a. The reason that we choose to use these lightweight threads is that OS threads have relatively expensive context switching and synchronization mechanisms. Therefore, dynamic scheduling and user-level thread (ULT) tasklet models were proposed to deal with the required levels of parallelism, offering more efficient context switching and synchronization operations.

   With ULT, The number of hierarchical levels exposed by the different threading library may vary and it depends on the number of execution units or concepts that each library exposes. Moreover, thanks to the stackless tasklet and flexible scheduler, these user-level threads may offer more excellent performance with less overhead than os-level threads.

b. We use Qthreads here

```
#define N 100
#define LEN 100

int a[LEN];
int b[LEN];
int c[LEN];

void vector_addition (i) {
    int start = get_start(i);
    int end = get_end(i);
    for (int i = start, i < end; i++) {
        c[i] = a[i] + b[i];
    }
}

int main(int argc , char * argv []) {
    qthread_initialize();

    for (int i=0; i<N; i++)
        qthread_fork(vector_addition, i);
    qthread_yield();

    for (int i=0; i<N; i++)
        qthread_readFF();

    qthread_finalize();
}
```

c. Here is the BLAS-1 function kernel code:

```
for (int i = 0; i < N; i++) {
    v[i] = v[i] * a;
}
```

So, the subprogram of the function is to multiple the elements in the array by a constant a. In the paper, the experiment is designed to create N threads with each updating 1 array element to test the overhead of threads creation and joining.

References:

1. A Review of Lightweight Thread Approaches for High Performance Computing, Castelló et al. NOTE that the link is offered with the assignment.