```
# http://eli.thegreenplace.net/2012/01/16/python-parallelizing-cpu-bound-tasks-with-multiprocessing/

import math
import threading
from factorize import factorize_naive

def serial_factorizer(nums):
    return {n: factorize_naive(n) for n in nums}

def worker(nums, outdict):
    """ The worker function, invoked in a thread. 'nums' is a
        list of numbers to factor. The results are placed in
        outdict.
    """
    for n in nums:
        outdict[n] = factorize_naive(n)

def threaded_factorizer(nums, nthreads):
    # Each thread will get 'chunksize' nums and its own output dict
    chunksize = int(math.ceil(len(nums) / float(nthreads)))
    threads = []
    outs = [{} for i in range(nthreads)]

    for i in range(nthreads):
        # Create each thread, passing it its chunk of numbers to factor
        # and output dict.
        t = threading.Thread(
                target=worker,
                args=(nums[chunksize * i:chunksize * (i + 1)],
                      outs[i]))
        threads.append(t)
        t.start()

    # Wait for all threads to finish
    for t in threads:
        t.join()

    # Merge all partial output dicts into a single dict and return it
    return {k: v for out_d in outs for k, v in out_d.iteritems()}

if __name__ == '__main__':

    import sys
    args = sys.argv[1:]
    c = len(args)

    if c > 0: nums = range(int(args[0]))
    else:     nums = range(100)
    if c > 1: threads = int(args[1])
    else:     threads = 5

    for k,v in threaded_factorizer(nums, threads).items():
        print k,v
```