```
import asyncio
import threading

#asyncio 是一个消息循环 我们从asyncio模块中获取一个eventloop
#然后把要协程扔到Eventloop中执行 实现异步

@asyncio.coroutine #把一个函数标记为协程
def hello():
    print('hello begin!', threading.current_thread())
    r = yield from asyncio.sleep(2)
    print('hello end!', threading.current_thread())

# yield from 调用另外一个函数
# hello() 首先会打印出hello world 然后yield from 语法可以
# 方便的调用另外的一个函数 由于 asyncio.sleep也是一个协程
# 所以线程不会等待 asyncio.sleep 而是终止执行并执行下一个循环
# 当asyncio.sleep()返回时 线程可以从 yield from 拿到返回值 此处是none
# 然后继续执行下一行语句
# asyncio.sleep 看成是一个耗时1秒的IO操作 在此期间 主线程并未等待
# 而是去执行 eventloop中其他的协程了 因此可以实现并发执行

import time
beg = time.time()
loop = asyncio.get_event_loop()
tasks = [hello(), hello()]
loop.run_until_complete(asyncio.wait(tasks))
loop.close()
end = time.time()
print(end - beg)
```