import multiprocessing as mp

Synchronous：依据输入顺序进行计算

Asynchronous：不必依据输入顺序进行计算

**Synchronous**

Pool.map(func, iterable=[input1,input2,…])

#input\_i不能是iterable

Pool.starmap(func, iterable=[input1,input2,…])

#input\_i也是iterable，所以如果是单个参数，需要用[[input1],[input2],…]

Pool.apply(func,args=单次input的参数)

#apply是作用于工作单元上的函数，map和starmap则是作用于主机上的函数

**Asynchronous**

Pool.map\_async(func, iterable=[input1,input2,…])

Pool.starmap\_async(func, iterable=[input1,input2,…])

Pool.apply\_async(func, args=单次input的参数)

#async类函数返回的是一个multiprocessing.pool.MapResult类，需要用.get()方法来获取计算结果

pool.apply()和pool. apply\_async()在Windows和Mac M1操作系统

**Multiprocess单机多进程**

**Vanilla Approach**

处理器们每次从task list中取一个进行处理，直到task list清空

**Vanilla Approach模板**

pool = mp.Pool(processes=使用处理器数量)

# processes=mp.cpu\_count() 使用全部CPU

res=pool.并行计算函数(func,按格式输入的inputs) # **Synchronous方法**

res=pool.并行计算函数(func,按格式输入的inputs).get() # **Asynchronous方法**

pool.close()

**Splitting Approach**

将task list拆分为数个sub task list，分别传给处理器，每个处理器只从自己的sub task list中取任务直至其清空

**Splitting Approach模板**

pool = mp.Pool(processes=使用处理器数量)

# processes=mp.cpu\_count() 使用全部CPU

inputs=np.array\_split(按格式输入的task\_list,mp.cpu\_count())

#使用np将task list拆分为数个sub list

res=pool.并行计算函数(func, inputs) # **Synchronous方法**

res=pool.并行计算函数(func, inputs).get() # **Asynchronous方法**

pool.close()