SuperFastPython.com Cheat Sheet for multiprocessing.Pool

Why multiprocessing.Pool?

Execute ad hoc functions that perform CPU-bound tasks asynchronously in new child processes, such as compute tasks or mathematical operations.

Create, Configure, Use

Import

```
from multiprocessing import Pool
```

Create, default config

```
pool = Pool()
```

Config number of workers

```
pool = Pool(processes=8)
```

Config worker initializer function

```
pool = Pool(initializer=init,
initargs=(a1, a2))
```

Config max tasks per child worker

```
pool = Pool(maxtasksperchild=10)
```

Config multiprocessing context

```
ctx = get_context('spawn')
pool = Pool(context=ctx)
```

Close after tasks finish, prevent further tasks

```
pool.close()
```

Terminate, kill running tasks

```
pool.terminate()
```

Join, after close, wait for workers to stop

```
pool.join()
```

Context manager, terminate automatically

```
with Pool() as pool:
    # ...
```

Issue Tasks Synchronously

Issue tasks, block until complete.

Issue one task

```
value = pool.apply(task, (a1, a2))
```

Issue many tasks

```
for val in pool.map(task, items):
    # ...
```

Issue many tasks, lazy

```
for val in pool.imap(task, items):
    # ...
```

Issue many tasks, lazy, unordered results

```
for val in pool.imap_unordered(task,
items):
    # ...
```

Issue many tasks, multiple arguments

```
items = [(1, 2), (3, 4), (5, 6)]
for val in pool.starmap(task,
items):
    # ...
```

Issue Tasks Asynchronously

Issue tasks, return an AsyncResult immediately.

Issue one task

```
ar = pool.apply_async(tsk, (a1, a2))
```

Issue many tasks

```
ar = pool.map_async(task, items)
```

Issue many tasks, multiple arguments

```
items = [(1, 2), (3, 4), (5, 6)]
ar = pool.starmap_async(task, items)
```

Chunksize

Via all versions of map() functions.

Issue multiple tasks to each worker

```
for val in pool.map(task, items,
chunksize=5):
    # ...
```

Use AsyncResult (handles on async tasks)

Via apply_async(), map_async(), starmap_async()

Get result (blocking)

```
value = ar.get()
```

Get result with exception

```
try:
    value = ar.get()
except Exception as e:
    # ...
```

Get result with timeout

```
value = ar.get(timeout=5)
```

Wait for task to complete (blocking)

```
ar.wait()
```

Wait for task, with timeout

```
ar.wait(timeout=5)
```

Check if task is finished (not running)

```
if ar.ready():
    # ...
```

Check if task was successful (no exception)

```
if ar.successful():
    # ...
```

Async Callbacks

Via apply_async(), map_async(), starmap_async()

Add result callback, takes result as arg

```
ar = pool.apply_async(task,
callback=handler)
```

Add error callback, takes error as arg

```
ar = pool.apply_async(task,
error_callback=handler)
```