```
import asyncio

import multiprocessing as mp

import time

from functools import partial

from typing import Any, Dict, Union




class HighSpeedExecutor:

    def __init__(self, num_processes: int = None):

        """

        Initialize the executor with configurable number of processes.

        If num_processes is None, it uses CPU count.

        """

        self.num_processes = num_processes or mp.cpu_count()


    async def _worker(

        self,

        queue: asyncio.Queue,

        func: Any,

        *args: Any,

        **kwargs: Any,

    ):

        """Async worker that processes tasks from the queue"""

        while True:

            try:

                # Non-blocking get from queue
            await queue.get()

            await asyncio.get_event_loop().run_in_executor(

                None, partial(func, *args, **kwargs)

            )

            queue.task_done()

        except asyncio.CancelledError:

            break


async def _distribute_tasks(

    self, num_tasks: int, queue: asyncio.Queue

):

    """Distribute tasks across the queue"""

    for i in range(num_tasks):

        await queue.put(i)


async def execute_batch(

    self,

    func: Any,

    num_executions: int,

    *args: Any,

    **kwargs: Any,

) -> Dict[str, Union[int, float]]:

    """

    Execute the given function multiple times concurrently.


    Args:
```

```
    func: The function to execute
    num_executions: Number of times to execute the function
    *args, **kwargs: Arguments to pass to the function

Returns:
    A dictionary containing the number of executions, duration, and executions per second.
"""
queue = asyncio.Queue()

# Create worker tasks
workers = [
    asyncio.create_task(
        self._worker(queue, func, *args, **kwargs)
    )
    for _ in range(self.num_processes)
]

# Start timing
start_time = time.perf_counter()

# Distribute tasks
await self._distribute_tasks(num_executions, queue)

# Wait for all tasks to complete
await queue.join()
```

```
    # Cancel workers
    for worker in workers:
        worker.cancel()

    # Wait for all workers to finish
    await asyncio.gather(*workers, return_exceptions=True)

    end_time = time.perf_counter()
    duration = end_time - start_time

    return {
        "executions": num_executions,
        "duration": duration,
        "executions_per_second": num_executions / duration,
    }

def run(
    self,
    func: Any,
    num_executions: int,
    *args: Any,
    **kwargs: Any,
):
    return asyncio.run(
        self.execute_batch(func, num_executions, *args, **kwargs)
    )
```

```
# def example_function(x: int = 0) -> int:
#     """Example function to execute"""
#     return x * x


# async def main():
#     # Create executor with number of CPU cores
#     executor = HighSpeedExecutor()

#     # Execute the function 1000 times
#     result = await executor.execute_batch(
#         example_function, num_executions=1000, x=42
#     )

#     print(
#         f"Completed {result['executions']} executions in {result['duration']:.2f} seconds"
#     )
#     print(
#         f"Rate: {result['executions_per_second']:.2f} executions/second"
#     )


# if __name__ == "__main__":
#     # Run the async main function
#     asyncio.run(main())
```