Advanced Scientific Programming in Python Summer School, Day 3

Exercises

Instructor: Eilif Muller

NB: The lecture notes are available in the svn under day1.

Exercise 1 – Races and Deadlocks

Goal: Get comfortable with interactive multiprocessing. Get a sober second look at races and deadlocks, and fix them. Killing deadlocks.

a) Working interactively, as I showed in the lecture (process functions are in mythread.py, reload(mythreads) after changes), observe that mp_race.py has variable output due to a race. Use locks to fix it.

b) Observe that mp_deadlock.py sometimes deadlocks. Kill the deadlocked process by finding its pid at a terminal:

$ ps ax | grep python

$ kill <pid>

The pid is the # on the left of the offending process (guess), ex: 23507

Why does this script deadlock? Fix it.

Exercise 2 – Parallel Random Numbers: SMP or IPython?

Goal: Become comfortable with multiprocessing shared memory, ipython, and understand performance differences between the two.

a) Using examples in day1/examples/matmul, specifically mp_matmul_shared.py and ipython_matmul.py determine the mean of enough random numbers to keep several processes busy for a few seconds, by generating locally, summing, and gathering the sum. Write both a SMP (multiprocessing) and an Ipython implementation.

b) Profile each implementation for a few problem sizes, compared to a local implementation.

c) Now gather the random numbers at the master and take the mean. (You might have to reduce the quantity of random numbers so that they fit in memory on the master). Can Ipython keep up with SMP shared memory?

Exercise 3 – Need for Speed

Goal: Find the fastest use of your local 4 cores

a) Look in SVN under examples/matmul. Compare performance of the mpi_matmul.py

with mp_matmul_shared.py and mp_matmul_local_opt.py (uses Pool.map rather than shared memory).

Exercise 4 – Parallel “Monte-Carlo” Agent

Goal: Steps towards a simple agent which uses parallel computing

a) Realize and evaluate random Agents paths on slaves, collect on the master, sort … take next move in winner, keep a population of winners, and proceed to realize and evaluate a few more.