Performance Profiling

Profiling with gprof

Aim: identify functions that are computational 'hotspots'
Function-level reports: accumulative runtime, number of calls, call graph

```
# First, compile your program with the gprof instrumentation
$ gcc -pg my_file.c -o my_executable

# Run your instrumented program (this will generate a "gmon.out" file)
$ ./my_executable

# Analyze the results
$ gprof my_executable
```

- Profile regularly (twice per day): the profile will change during the week.
- Profile using a real test-case and on your target hardware.
- Keep the test short (minimal iterations) but long enough so that the initialisation/finalisation is no more than 10% of the total runtime

Measuring performance

Aim: monitor progress of parallelization and optimization work

1. Add timing calls to your code.
2. Check for correctness first (the Debug test). Check EVERY time you plan to benchmark a new version
3. Measure the runtime for 3 to 7 Benchmark runs (the more the better, but this can be time consuming).
   a. Record each runtime on 1, 2, 4, 8.. Etc. threads/cores
   b. Record the time for the sequential part of your code.
   c. Calculate Speedup, Efficiency and serial fraction, α
   d. ONLY RECORD THE RUNTIME OF THE SERIAL CODE REGIONS, α, WHEN RUNNING ON 1 THREAD/CORE

Speedup, S, for problem size N on P processes/threads/cores

$$S(N, P) = \frac{T(N,1)}{T(N,P)}$$

Parallel efficiency, E, for problem size N on P processes/threads/cores

$$E(N, P) = \frac{S(N,P)}{P} = \frac{T(N,1)}{(P*T(N,P))}$$

For runtime T, using problem size N for P processes (*Amdahl's Law*)

$$T = \alpha\, T(N,1) + \frac{(1-\alpha)T(N,1)}{P}$$

Theoretical performance limit:
Maximum Speedup = 1/α
(where parallel portion asymptotically approaches zero runtime)