SIT315 M2.S2P: Decomposition, and Multithreading with *pthreads* Part 1

Codey Funston | S222250824
No Group

Activity 1 – Decomposition Techniques

1. **Sliding Puzzle**: To solve a sliding puzzle problem, you could either use dynamic programming with recursive function calls as the problem size reduces, or you could use an exploring approach akin to graph searching problems.

2. **Frequency of String Occurrence**: Using input data decomposition you could assign each thread a section of the Wikipedia page, then sum the outputs of each thread after they are all joined. For example, you can get each thread to start reading from their own starting line number and space out the line numbers as required among the threads.

3. **Binary Search**: In the binary search examples, we have seen so far in computer science a parallel solution wouldn't be very helpful since the search size gets small very quickly as it halves. However, with very large data sets these searches can be delegated to multiple threads looking through the array subsections. This would be an example of recursive decomposition together with input data decomposition.

Activity 2 – Parallel Vector Addition

Parallelisation Roadmap

Since the individual entries of the vectors are different locations in memory they can be accessed concurrently and thus modified (to a random integer) concurrently. Also, due to the nature of vector addition where the sum of two vectors is the sum of each of their entry's, addition can be performed in parallel. In my implementation of *VectorAdd.cpp* I had threads request chunks of the data to be processed so shared indices had to be locked and unlocked to prevent a race conditions. Aside from these no other safety features are necessary unless you count thread joining.

The vectors are simply decomposed into segments that are split between threads evenly, the splitting is done in-place with indexing.

Sequential Procedure	Parallel Procedure
1. Call *ranadomVector()* with *v1*. 2. Call *randomVector()* with *v2*. 3. For each entry in resultant vector, set value to corresponding *v1 + v2*.	1. Assign half of total threads to filling *v1* and the rest to filling *v2* with **vectorFill()*. 2. Assign all threads for adding sections of *v1* and *v2* to *v3* with **vectorAdd()*.

Partition Size Analysis

With the threads in my program continuously requesting extra chunks of data until the data is completely processed means that there is an extra overhead as a result of the locking of the shared vector index. This is less noticeable as the partition/chunk size approaches smaller values and this can be seen with the following output of the program, running with larger numbers of partitions:

*Note: My machine has 11 cores.

```
codey@codey-mac ps2 % ls
VectorAdd.cpp            partitions-20
partitions-100           partitions-40
partitions-1000          partitions-NUM_CORES
codey@codey-mac ps2 % ./partitions-NUM_CORES

Concurrent Runtime (µs):      1495846 microseconds
Parallelized Runtime (µs):    237305 microseconds
Parallel Speed Increase:      6.3

codey@codey-mac ps2 % ./partitions-20

Concurrent Runtime (µs):      1491689 microseconds
Parallelized Runtime (µs):    196620 microseconds
Parallel Speed Increase:      7.6

codey@codey-mac ps2 % ./partitions-40

Concurrent Runtime (µs):      1488607 microseconds
Parallelized Runtime (µs):    200729 microseconds
Parallel Speed Increase:      7.4

codey@codey-mac ps2 % ./partitions-100

Concurrent Runtime (µs):      1481950 microseconds
Parallelized Runtime (µs):    198943 microseconds
Parallel Speed Increase:      7.4

codey@codey-mac ps2 % ./partitions-1000

Concurrent Runtime (µs):      1490151 microseconds
Parallelized Runtime (µs):    197285 microseconds
Parallel Speed Increase:      7.6

codey@codey-mac ps2 %
```

SERIAL MONITOR   TERMINAL   zsh - ps2

```
codey@codey-mac ps2 % ./partitions-2 ./partitions-4 ./partitions-6

Sequential Runtime (µs):      1524006 microseconds
Parallelized Runtime (µs):    416067 microseconds
Parallel Speed Increase:      3.7

codey@codey-mac ps2 % ./partitions-2; ./partitions-4; ./partitions-6

Sequential Runtime (µs):      1507888 microseconds
Parallelized Runtime (µs):    416673 microseconds
Parallel Speed Increase:      3.6


Sequential Runtime (µs):      1477722 microseconds
Parallelized Runtime (µs):    244095 microseconds
Parallel Speed Increase:      6.1


Sequential Runtime (µs):      1474029 microseconds
Parallelized Runtime (µs):    276320 microseconds
Parallel Speed Increase:      5.3

codey@codey-mac ps2 %
```