Assignment 1

Aman Rojjha 2019111018

Performance Tools and Analyzers

To benchmark and test the performance of code, several analyzers have been used and the optimizations applied.

Perf

Perf is a performance analysis tool (also known as *perf_tools*) which gives access to Performance Monitoring Unit of CPU. It can instrument CPU performance counters, tracepoints, kprobes and uprobes (for dynamic tracing).

Example interface commands:

```
perf stat
perf record
perf report
perf annotate
perf top
perf bench
```

Cache Grind

This tool is a cache profile and branch-prediction profiler. It basicall simulates how a program interacts with the machine's cache by running it (in sandbox environment) simulating caching and then measures the cache usage patters of the given program. Given the difference in CPU and memory speeds, cache play a vital role in the performance of a program and are one of the main reasons of bottleneck in several ones. Using this tool we can analyse the usage patterns and try to overcome them (by standard and non-standard ways).

Gprof

Gprof is a performance profiler tool. It gives statistics about the program by inserting time-measuring macros at the start and end of each function call thereby generating a data file and dumping it in human readable form.

clock_gettime

The `clock_gettime` function gets the time stored in clock and populates it's `timespec` structure.
It is basically used to time the various parts in a program. {Checkout link for example}

Question 1 - Efficient Matrix Multiplication

I first pondered over whether to use recursive multiplication in this case, but given that `n < 1000` I infered that it wouldn't a huge optimization over other methods so I resorted to:

1. Using *Dynamic Programming* to find the optimal order of matrix multiplication which result in least number of total multiplications.
2. *Loop unrolling* for better cache-optimization.
3. Definition of variables once rather than each loop.
4. Use of *register keyword* for frequently used variables.
5. Reducing the number of memory accesses by storing frequent used values in seperate register variables.
6. Defining a 1-D array rather than 2-D array (to just get an edge in time).

Question 2 - Floyd Warshall

1. Dynamic array allocation way slower than static (takes about `4x` time) but storing it as a linear array and mapping it manually gave a performance boost.
2. Bitwise `min` taking way more time than simple integer function min. (when used for each iteration of the algorithm)
3. *Loop unfolding* increased the performance of the algorithm by about `5.5x` bring down the time for test-case 29 from `52.5 sec` for the base program to about `8.9 sec` on my machine.
4. Using `register` keyword for frequently used variables increase about 2% performance.
5. Initializing all the loop variables once increased about 1% performance (for smaller testcases).

Machine Specifications

1. **Model** - Acer Nitro An515-52
2. **Processor** - Intel i5 8300H (2.30 GHz Base Clock)
3. **Memory** - 8 GB Ram
4. **SSD** - Samsung 970 Evo Plus 1 TB.