Sum of Degrees Squared

Emil Lynegaard

November 23, 2017

1 Sum of Derees Squared

1.1 Implement

See attached code `sum_degrees_squared.go`.

The algorithm is implemented using an array of size n, which keeps track of each vertex' degree. The sum is computing by squaring each degree and summing them up.

1.2 Question to discuss

1.2.1

A final scan over the array is inevitable due to the nature of squaring numbers. The difference between incrementing a small degree vs. ig degree will have an impact on the output, meaning that we cannot compute any of the degrees before reading the last line.

1.2.2

If we are accessing our array of degrees in linear order as we parse the input, we will have natural cache efficiency with minimal cache misses. On the other hand if we are constantly incrementing degrees of arbitrary vertices, we can end up with a total of $m * 2$ cache misses.

1.2.3

This phenomenon is explained by the I/O-model and the quantification is mentioned in **??**

1.2.4

For our problem, the most interesting factor is the size of `n`. For graphs with small n and large `m`, the majority of the time will be spent parsing the input as we will only ever have to do `n` squares in the end.

To compare the platforms we would therefore use a size of `n` causing the total time to be atleast 2˜seconds to eliminate noise in the form of mostly file I/O and OS noise.

Factors that may be impactful for our experiment include:

- Size of m - increases file I/O time
- Size of n - increases summation time
- Input ordering - impacts cache efficieny

If we wanted to measure how the potentially different CPU's performed in terms of caching, we could run various graph instances on both platforms and count the cache misses as our metric. If we instead wanted measure performance in the form of time, we could run the algorithm with sufficiently large values for n and random m on both platforms measuring wall clock time as our metric. With the current implementation this would give us a good indication of single-core performance, as the algorithm is ran sequentially.