Project Proposal
Prime Factorization

Hazel Flaming
haf2136

PFP Fall 2025
Stephen Edwards, Maxwell Levatich

Summary

Prime factorization is a core concept in the foundations of mathematics, cryptography, hashing, randomness, and other fields, involving the deconstruction of large numbers into two or more prime factors (self-explanatory). In this project, I seek to explore the sequential and parallel optimization of several algorithms dedicated to solving this problem.

Motivation

This problem (from afar) presents several advantages with respect to this specific project. Firstly, solving this problem is not IO bound, as it simply requires a single input and otherwise requires no input from the filesystem and all logic can be handled internally. Secondly, this problem has several "knobs" that can be adjusted with respect to testing code, the most obvious being the size of the number, along with additional examples being the number of total factors, the proximity of those factors, and so on, such that if the algorithm takes too long or too short to present interesting comparisons, no complicated overhaul or project changes are required and those "knobs" can be adjusted easily by selecting different test cases. The final advantage is the existence of multiple algorithms, some very simple like trial division, and some very complicated like General Number Field Sieve (GNFS), which allows the project to proceed iteratively by first improving each algorithm serially and then parallelizing it, and then proceeding onto more complex algorithms. Additionally, the existence of multiple algorithms allows intelligent selection of algorithms on the basis of theoretical parallelizability.

Input Data

Input data is easy– we can simply generate random numbers of increasing size and factorization complexity (i.e. by multiplying together n random prime numbers), or sourced from https://en.wikipedia.org/wiki/RSA_Factoring_Challenge if any of the algorithms get efficient enough to solve challenges this hard.

High-Level Algorithm Descriptions

(1) Trial division works by testing (naively) all numbers up to the $\sqrt{p}$ to see if they are a factor of p. If a number a is a factor of p, then it is run recursively on $\frac{p}{a}$ until $\frac{p}{a}$ itself is prime.

(2) Fermat's Factorization Method works by expressing p as $x^2 - y^2$, where the factors of p then would be $x - y$ and $x + y$. This works starting with $x = \sqrt{p}$, and iteratively checking if $x^2 - p$ is a perfect square.

(3) Quadratic Sieve works by first finding *smooth numbers* close to $\sqrt{p}$ via the formula $Q(x) = (x + \sqrt{p})^2 - p$, which are numbers which factor into small primes set by some small prime bound B. Then, it combines sufficiently smooth numbers into a single perfect

square by using smooth numbers which can be written purely as products of small primes (below B) with *even exponents* (i.e. $2^4 + 5^6$), guaranteeing a perfect square. Then, using the formula $gcd(X - Y, p)$, where $X = x + \sqrt{p}$ and $Y = \sqrt{Q(x)}$, where x corresponds to the specific smooth number, a non-trivial factor is found.

(4) Additional algorithms can be added if the previous scope is too small.

(The start of a) Parallel Implementation Plan

Starting at trial division, which checks the remainder when dividing the trial number p by increasing integers or primes, can be split up among the primes such that each thread handles a subset of possible divisors. This can be done naively (splitting into evenly sized groups) or intelligently by using some heuristic to estimate the probability of finding a factor within a chunk and splitting into equal probability chunks. Fermat's factorization method (see source 1) can be trivially parallelized by assigning threads to check whether $x^2 - N$ is a perfect square for $x = N + c' + ck, k \in N$, where c is the number of threads and c' is the current thread number. Moving onto more complex algorithms, for the quadratic sieve algorithm (see source 1) which expands upon Fermat's method has similar condition checking which can be done independently between threads (finding smooth numbers), and then some linear algebra work which will be less parallelizable, however, because the sieving likely dominates the linear algebra, parallelizing it should prove beneficial. Because of the simplicity of much of the parallelism discussed here, techniques which favor many small computations, like accelerate, strategies, and REPA would work better over the par monad which comes with more overhead and favors fewer longer computations.

References

https://medium.com/nerd-for-tech/heres-how-quadratic-sieve-factorization-works-1c878bc94f81