Abstract

The Quadratic Sieve is used to increase the efficiency of factoring semi-primes that are usually used in cryptography. The algorithm for the sieve has many steps that can be parallelized. Due to this, efficiency can be increased if these steps are computed on the GPU.

Goals

- Learn CUDA
- Implement the Quadratic Sieve on the CPU
- Implement the Quadratic Sieve on the GPU
- Implement the Quadratic Sieve as a combination of CPU and GPU

Conclusions

Parallelizing the Quadratic Sieve on the GPU became faster as the digit count increased. Initially, the parallelized Brute Force on the CPU was faster since it takes time to transfer data to the GPU.

Results

Brute Force did jumps on time on even numbers when both factors had high digit counts. The Quadratic Sieve started slower because it had to generate test values then compare them. As the digits count increased the Quadratic Sieve became quicker compared to Brute Force.

Future Work

- Generate a library for the GPUPrec code
- Continue to improve the efficiency of the Quadratic Sieve on the GPU
- Other factoring methods such as Elliptic Curve and other sieving methods

References

- GPUPrec Source Code and Documentation.
- MPIR Source Code and Documentation.
- CUDA by Example: An Introduction to General-Purpose GPU Programming (29 July 2010) by Jason Sanders, Edward Kandrot
Algorithm
• \( x^2 \equiv y^2 \pmod{n} \)
• \( x \equiv y \pmod{n} \)
• \( \text{GCD}(x - y, n) = p \) so \( p \mid n \)

Procedure
• \( x^2 \equiv z \pmod{n} \) with \( z = p_1^{\alpha_1}p_2^{\alpha_2}p_3^{\alpha_3} \ldots p_k^{\alpha_k} \) such that \( p_1, p_2, p_3, \ldots, p_k \leq \text{Upper Prime Bound} \)
• \( \text{Floor}(\sqrt{n^i} + j) = x \)
• \( i \) is moderate in size
• \( j \) is small in size (1 \( \leq j \leq 100 \))