GPU Acceleration of Special Purpose Integer Factoring Algorithms

Alexander Streit (Undergraduate), Dr. Don Spickler (Faculty Mentor)
NSF REU site EXERCISE (Award #1156509)

Introduction and Motivation

The RSA cryptosystem is in wide use today securing email, online credit card transactions and more. Following is the method:

- Choose two primes p and q.
- Compute $n = pq$.
- Compute $\varphi(n) = (p-1)(q-1)$, where φ is Euler's totient function.
- Choose e such that $1 < e < (n), \gcd(e, \varphi(n)) = 1$.
- The public key is e and n.
- Compute private exponent d as $d \equiv e^{-1}(mod\ (n))$.
- d is kept as the private key.
- Anyone can now encrypt a message represented by a number m by calculating the cyphertext $c = m^e (mod\ n)$.
- Only the possessor of the private key can decrypt by calculating $m = c^d\ (mod\ n)$.

The only known way to obtain the private exponent d mathematically, and thus break the encryption, is to find the factors p and q of n. Factorization is an exponentially complex computation, and therein lies the security of RSA because even with today's supercomputers it is virtually impossible to factor the 300-400 decimal digit numbers currently used for RSA key numbers in a reasonable amount of time.

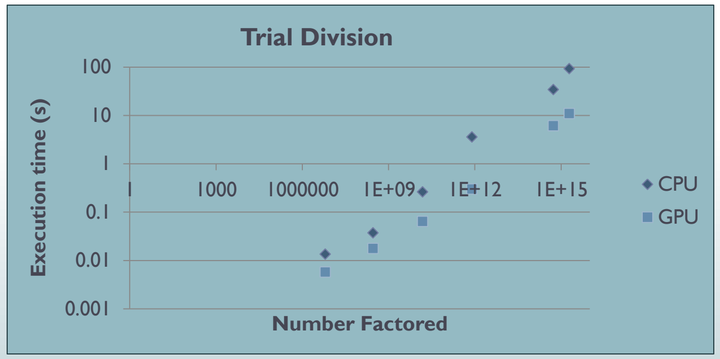
There are two main types of factoring algorithms in use today; special purpose, which are good at finding factors of particular form, and general purpose, which can be used to factor any integer regardless of the form of the factors. We implemented two special purpose algorithms for use on a standard CPU for benchmarking, and on a CUDA enabled GPU, where we took advantage of parallelization to accelerate the processes.

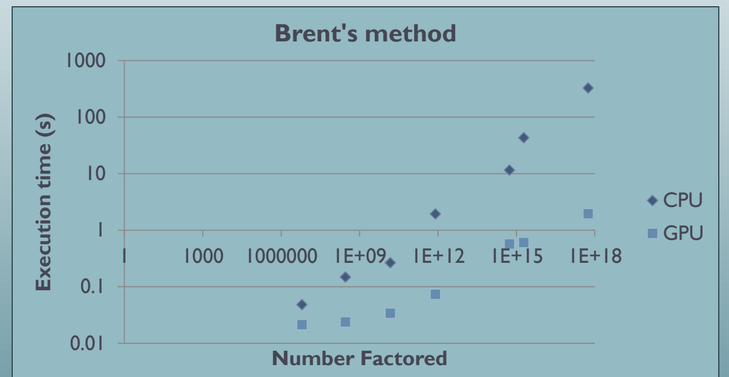
Trial Division Method

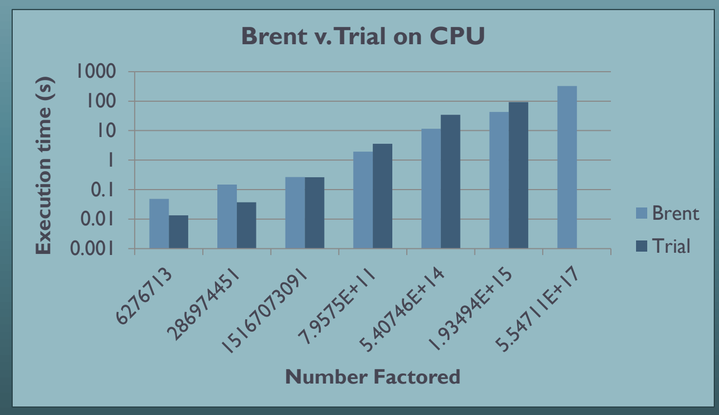
The most straightforward way to find the factorization of a number is to compute a list of the primes up to the square root of the number and try dividing by each of them. This will always yield a factor, but it is very slow for large factors. To parallelize it you simply perform each division in parallel using the GPU.

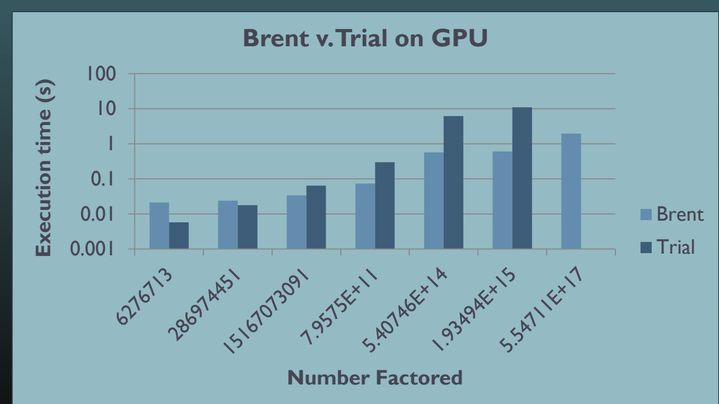
Brent's Method

This variant of Pollard's rho method uses a pseudorandom sequence to search for factors with a higher probability of finding one faster than trial division. The evaluation of each sequence is not readily parallelizable, but since not all sequences generate a non-trivial factor, it can be accelerated by testing many different sequences in parallel.









Brent's Method Pseudocode

```
(1)  Set x = x0 Mod N;
       y = x0^2 + a Mod N and k = 1.
(2) if (gcd(x-y; N) > 1) then
(3)   return (gcd(x-y; N))
(4) end if
(5) while (1) do
(6)   Set x = y.
(7)   for (j = 1 to k) do
(8)     Set y = y^2 + a Mod N.
(9)   end for
(10)  for (j = 1 to k) do
(11)    Set y = y^2 + a Mod N.
(12)    if (gcd(x-y; N) > 1) then
(13)      return (gcd(x-y; N))
(14)    end if
(15)  end for
(16)  Set k = 2 k.
(17) end while
```

Results

After much programming and even more debugging, we have obtained some factorization times that show the GPU as the clear winner in factoring speed for the two algorithms, when compared with the serial version of the same algorithm. Also note that the speed-up is much greater for Brent's method than for Trial. This is because Brent's method is less data-intensive and more compute-intensive, and thus better suited for the GPU. In the lower two graphs we compare the two algorithms on the CPU and GPU. For each system, the Trial division is faster for small numbers and then Brent's method becomes faster for larger ones.

Conclusions

Based on the results we can see that at least these particular algorithms benefit from parallelization on the GPU. The times we have obtained may not, however, be optimal and in future work, the program could be optimized to handle larger numbers as well. Also, in the future general purpose algorithms should be explored.

References

- J. Buchmann, V. Müller, *Algorithms for Factoring Integers*, 2005
- C. Archer, *GPU Integer Factorisation with the Quadratic Sieve*, University of Bath, 2010
- K. Zhao, *Implementation of Multiple-precision Modular Multiplication on GPU*
- P. Leslie Jensen, *Integer Factorization*, University of Copenhagen, 2005
- NVIDIA CUDA C Programming Guide, v.4.2, 2012
- CUDA C BEST PRACTICES GUIDE, v.4.1, 2012
- http://stackoverflow.com/questions/497685/how-do-you-get-around-the-maximum-cuda-run-time
- http://www.rsa.com/rsalabs/node.asp?id=2190