**Analysis and optimization of prime number detection in parallel with CUDA**

JD Kilgallin, Eric Fernandes

Computer Science Department

University of Akron

A prime number is a positive integer with exactly two divisors – the unit ‘1’ and the integer itself – with other numbers termed “composite”. This definition excludes the number 1 itself, as well as every even number except 2, every multiple of 3 except 3 itself, and so on. It can be easily proven that there are infinitely many prime numbers, though there is no known straightforward procedure for finding primes in a given range. Identification of large prime numbers is an important task, however, with critical applications in cryptography and other fields. The precise complexity of identifying prime numbers is not known but is believed to be solvable in polynomial time.

The techniques employed in the search for prime numbers have been incrementally improved throughout history. Techniques from ancient Greece remain applicable today, such as the Sieve of Eratosthenes. This is essentially an extension of the definition above, where one simply lists the integers, then crosses out the multiples of 2, then of 3 since it is the next number that is not crossed out, then of 5 as the next number remaining, and so on. Any number remaining must be prime. Such millennia-old techniques underlie newer approaches, which generally further rely on modern number-theoretic results and probabilistic methods. For example, modular arithmetic can be used to rule out large classes of integers immediately. That is, all prime numbers must be of the form "6k +/- 1" for some k, as numbers congruent to 0, 2, 3, or 4 mod 6 are divisible by either 2 or 3. Further, they must all be of the form “30k + r”, with r in the set “1, 7, 11, 13, 17, 19, 23, 29” as elements from any other congruence class must be divisible by 2, 3, or 5. A more rigorous test can be devised by considering the congruence class mod 2\*3\*5\*7=210, and so on. Considering the possible congruence classes with respect to the product of the first k primes can quickly rule out large numbers of candidates as composite. This is, however, essentially nothing but a modern phrasing of the Sieve of Eratosthenes.

Other tests rely on probabilistic methods, such as Fermat's test or Miller-Rabin primality testing which can be used to further eliminate most composite numbers quickly. To conclusively prove that a number is prime, however, a basic brute force technique of testing divisibility by smaller prime numbers can be used. The set of primes to be tested as divisors of a candidate prime 'n' encompasses the range [2, sqrt(n)], as if k > sqrt(n) divides n then n/k < sqrt(n) also divides n.

At many stages of this process, the search for prime numbers can be parallelized effectively. First, for a given range [u,v] to search with p processors, processor 'i' can examine integers of the form 6k +/- 1 in the range [(v-u)/(p) \* i, (v-u)/(p) \* (i + 1)) and apply a probabilistic algorithm to generate candidate primes in this range. A final candidate list, then, can be split between processors for confirmation. For a single candidate, too, the process of testing divisibility by a set of integers can be split by divisor.

The CUDA programming environment, which offers C++ extensions for parallel processing on an NVIDIA graphics card, is suitable for exploring an implementation of parallel primaility testing. Ideally, such a program would work as follows:

Input: A range u, v, of integers in which to search for primes

Return: A set P of primes in the range [u, v]

Begin:

1. In parallel on all processors, generate a set of candidate primes of the form 6k +/- 1, that furthermore are not divisible by any known “small” primes.
2. Again in parallel, for each remaining candidate employ a heuristic test such as Fermat’s test or the Miller-Rabin test to rule out composites.
3. Again in parallel, for a given remaining candidate prime, parallelize the test for divisibility against all additional primes in the range [2, sqrt(n)]

In this project, we have implemented and examined a variety of primality testing algorithms and work-division strategies for effective parallel detection of primes. For each approach, we have computed properties such as the speedup and efficiency of the technique for various test cases. We verified correctness against published lists of known primes.