Limiting Behaviour Of Multi-Threaded Prime Number Sieves

B. Lowe

Introduction

This research studies multi-threaded prime sieve algorithms implemented in the CUDA parallel processing architecture. We'll analyse time complexity for these algorithms, then implement, benchmark and compare multi-threaded variations of Divisibility Test and Atkins Sieve algorithms to the benchmark Sieve of Eratosthenes. Both these algorithms have a pair of nested loops making them naturally parallelizable. We'll see that parallelizing the inner loops will result in $O(n \log(n)) \rightarrow O(n)$ time complexity for divisibility testing[3] and $O(n) \rightarrow O(\sqrt{n})$ for the Sieve of Atkin's[4].

Parallel Divisibility Testing

To parallelize Divisibility Testing, we shall dispatch sqrt(n) threads for each integer we test. This is given in algorithm (1.1). Since algorithm (1.1) loops once over n indices the algorithm has complexity O(n).

```
(1.1)
Initialize prime[n] = true
For integer values of x, 0 <= i < n:
    Dispatch sqrt(n) kernels (1.2)
(1.2)
j = thread Id
if j | i:
    prime[i] = false
```

Parallel Atkin's Sieve

Sieve of Atkin contains two loops which we must compute sequentially. First takes O(n) operations to initialize an array of length n sequentially so this too can be done in parallel using algorithm (1.1).

We can then compute all the binary quadratic forms in one kernel applied to sqrt(n) sieves over a sqrt(n)-long loop. This is done in algorithm (2.3). Each thread evaluates the quadratic form for x with its Id in algorithm (2.4).Finally, algorithm (2.5) then removes all the squares of primes numbers that are not removed by the quadratic form sieve.

The time complexity of this algorithm is O(sqrt(n)) since each of the loops used are repeated sqrt(n) times. The result of this algorithm is an array of marked primes stored on the GPU. This means to access the array it must first be moved to the CPU side which is done in O(n) time. A second disadvantage of the algorithm is its use of sqrt(n) threads which means that for smaller GPUs computing sufficient prime numbers is done in O(n) time.

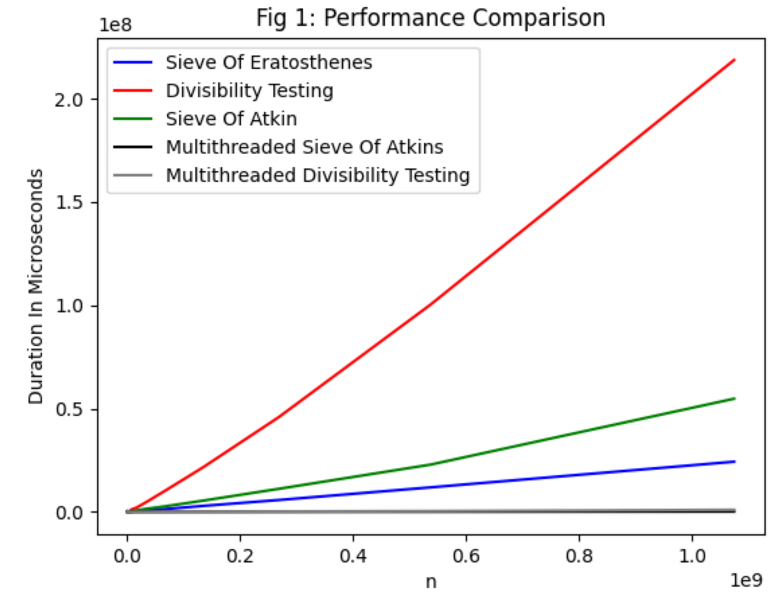
```
(2.1)
for integer values of x, 0 <= x < sqrt(n):
    Dispatch sqrt(n) kernels (2.2)
(2.2)
y = thread Id
Initialize prime[x * y] = false
(2.3)
for integer values of x, 0 <= x < sqrt(n):
    dispatch sqrt(n) kernels (2.4)
(2.4)
y = thread Id
z = 4x^2 +y^2
if (z <= n) and ((z mod 12 = 1) or (z mod 12 = 5)):
    prime[z] = not prime[z]
z = 3x^2 + y^2
if (z <= n) and (z mod 12 = 7):
    prime[z] = not prime[z]
z = 3x^2 – y^2
if (z <= n) and ((z mod 12 = 11) and  (x > y)):
    prime[z] = not prime[z]
(2.5)
for integer values of x, 0 <= x < sqrt(n):
    dispatch sqrt(n) kernels (2.6)
(2.6)
y = thread Id
If (y not in {0, 1}) and (x * x * y <= n):
    prime[x * x * y] = false
```

Benchmarking

Evaluating implementations of the algorithms discussed using an AMD Ryzen 5 2600 and Nvidia GeForce GTX 1060 gives figure 1. This only demonstrates that the parallel implementations are the fastest.

To test our theoretical values for the limiting behaviour of these algorithms we instead use the log-log plot in figure 2. This shows that the divisibility testing algorithm, as expected, grows linearly. However, the Parallel Sieve of Atkin is slower than theory would suggest. This is likely due a combination of factors. Sqrt(N) exceeds the thread count of the GPU resulting in a O(n) phase. Further, the benchmark includes the time to read the prime number array from the GPU in linear time. resulting in linear behaviour for N > 2^24.

Fig 1: Performance Comparison



Future Improvements

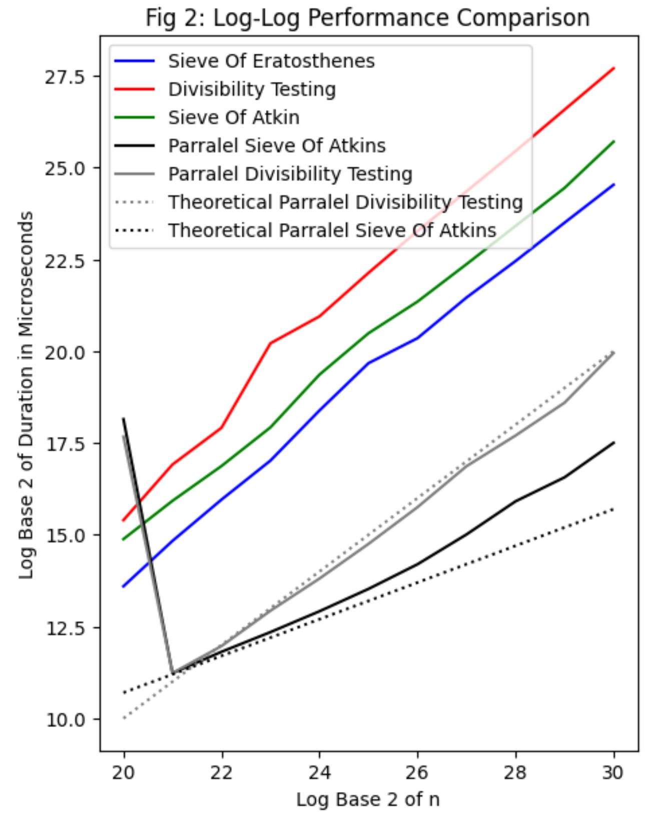
This benchmark only tests up to N = 2^30 due to restrictions from hitting the maximum memory available. Since this benchmark is interested in the limiting behaviour of these algorithms; segmentation and bit arrays were not used as this only changes the constant time. However, these methods are expected to provide significant increases in performance.

z is evaluated three times per thread with a total of 11 multiplications. Utilising new GPU technology by calling Tensor cores to do this operation in one matrix multiplication step rather than evaluating three quadratic forms with CUDA cores may provide further improvements in performance. However, current GPUs have greater number of CUDA cores than Tensor cores which will limit the performance improvements until the number of Tensor cores in available GPUs increases.

In (2.6) every thread checks if x is marked prime. This is extra work that could be eliminated by moving an array of primes < sqrt(n) from the GPU then check the primality of x once on the CPU which may provide performance improvement.

To conclude, we've found that utilising the GPU can provide significant performance improvements over series sieving methods however the sieves are restricted by current GPU technology due to high thread utilisation. However, with ever increasing number of cores in modern GPUs this limitation is likely to become less relevant.

Fig 2: Log-Log Performance Comparison



Implementation can be found at:
https://github.com/BenLowe2003/MultiThreadedPrimeNumberSieves

References

[1] A.O.L. Atkin, D.J. Bernstein. (2004) Prime sieves using binary quadratic forms. Math. Comp. 73, 1023-1030.

[2] O'Neill, Melissa E. (2008) The Genuine Sieve of Eratosthenes. Journal of Functional Programming, published online by Cambridge University Press doi:10.1017/S0956796808007004

[3] Jonathan Sorenson. (1990) An Introduction to Prime Number Sieves. Computer Sciences Technical Report #909, Department of Computer Sciences University of Wisconsin-Madison

[4] D. J. Bernstein. (2004) Primegen. Available at: www.cr.yp.to