ISSN 2277-3061

Fast Effective Deterministic Primality Test Using CUDA/GPGPU

Abu Asaduzzaman[1)], Anindya Maiti[2)], Chok M. Yip[3)]
EECS Department, Wichita State University, Wichita, Kansas 67260-0083, USA
[1)] E-mail: Abu.Asaduzzaman@wichita.edu
[2)] E-mail: axmaiti@wichita.edu
[3)] E-mail: cxyip1@wichita.edu

ABSTRACT

There are great interests in understanding the manner by which the prime numbers are distributed throughout the integers. Prime numbers are being used in secret codes for more than 60 years now. Computer security authorities use extremely large prime numbers when they devise cryptographs, like RSA (short for Rivest, Shamir, and Adleman) algorithm, for protecting vital information that is transmitted between computers. There are many primality testing algorithms including mathematical models and computer programs. However, they are very time consuming when the given number n is very big or $n \to \infty$. In this paper, we propose a novel parallel computing model based on a deterministic algorithm using central processing unit (CPU) / general-purpose graphics processing unit (GPGPU) systems, which determines whether an input number is prime or composite much faster. We develop and implement the proposed algorithm using a system with an 8-core CPU and a 448-core GPGPU. Experimental results indicate that up to 94.35x speedup can be achieved for 21-digit decimal numbers.

Indexing terms/Keywords

CUDA architecture; deterministic algorithm; GPU computing; parallel computing; primality test;

Academic Discipline And Sub-Disciplines

Computer Science and Engineering; Cryptography; Mathematics;

SUBJECT CLASSIFICATION

High Performance Computing; Parallel Programming; Low Power System;

TYPE (METHOD/APPROACH)

Design; Experimental;

ISSN 2277-3061

1. INTRODUCTION

Since Euclid (the great Greek mathematician, around 300 BC), it has been known that the primes are infinite in number but the exact sequence of primes is not predictable. Prime numbers and computers have been linked since the 1950s. During the Cold War between the United States (U.S.) and Russia in the 1960s, the U.S. Central Intelligence Agency (CIA) and Russian KGB (short for Komitet Gosudarstvennoy Bezopasnosti, i.e., committee for state security) both tried to keep the other from learning defense secrets about missile, weapons, and other military secrets. Prime numbers played an important part in the secret spy codes that both countries used in relaying messages. In fact, prime numbers are still used in secret codes today. Hackers and other computer pirates try to steal information or break into private transactions. Computer security authorities use extremely large prime numbers when they devise cryptographs, like RSA algorithm, for protecting vital information that is transmitted between computers [1, 2]. The mathematicians who studied prime numbers hundreds of years ago used the knowledge from primes to develop new areas of mathematics, like number theory and knot theory, which are used today in computer and many scientific studies. Prime numbers are a key part of modern mathematics and have important uses in the era of computers. The introduction of parallel computing has potential to investigate primes vastly larger than previous generations were able to calculate, resulting in a search for ever-larger primes, so that some insight into these strange mathematical beasts might be gained.

A prime number is a natural number that has exactly two distinct natural number divisors: 1 and itself. The property of being a prime is called primality. Common curiosities are the properties that allow one to efficiently determine if a number is prime or composite. Verifying the primality of a given number *n* can be done by trial division. The simplest trial division method test whether n is a multiple of an integer *m* between 2 and √n. If *n* is a multiple of any of these integers between 2 and √n then it is a composite number; if it is not a multiple of any of these integers then it is prime. As this method requires up to √n trial divisions [3], it is only suitable for relatively small values of *n*. More sophisticated algorithms, which are much more efficient than trial division, have been devised to test the primality of large numbers. Modern primality test algorithms can be divided into two main classes: deterministic and probabilistic algorithms. Probabilistic algorithms may report a composite number as a prime, but certainly do not identify primes as composite numbers [4]; deterministic algorithms [5] on the other hand do not have the possibility of such erring.

The research on prime numbers thrusts the boundaries of our computing technology and in some cases advances our mathematical understanding. The largest prime yet discovered is truly huge - it has 17,425,170 digits! The largest prime was found as a part of the Great Internet Mersenne Prime Search in January 2013 [6] using computer technology. In concurrent computing systems, the application software is mostly run on distributed computing systems and the hardware which can run such tests includes CPUs and GPUs. In the last few years, GPUs have increasingly been used to accelerate numerical computation in nanotechnology research, consumer software, and the largest supercomputers in the world [7-9]. The high processing speed offered by the general purpose GPU cards through Application Programming Interfaces (APIs) like CUDA (short for Compute Unified Device Architecture, i.e., NVIDIA's parallel computing architecture) has been made available for non-graphical applications. Although GPGPUs are not as versatile as CPUs, they offer a huge amount of computation if the algorithm can be structured to take advantage of it.

This paper presents a potential deterministic algorithm using a parallel computing machine, which determines whether an input large number is prime or composite very fast. Like other modern primality test algorithms, the proposed algorithm also faces usage limitations on a tangible computer in reasonable finite time. This limitation arise when the given number *n* is very big or $n \to \infty$. So, a distributed computational framework should be setup to test the primality of large numbers using the presented deterministic algorithm.

The rest of the paper is organized as follows: Section 2 summarizes some related selected published articles. The proposed parallel solution for deterministic primality test using CUDA/GPGPU technology is presented in Section 3. Experimental details are described in Section 4. Experimental results are discussed in Section 5. Finally, this work is concluded in Section 6.

2. LITERATURE SURVEY

As attackers are getting smarter, the need for larger prime numbers for the encryption systems is also persistent. A number of work has been done to address the prime number and primality test issues; some published articles, closely related to this work, are discussed in this section.

According to Fermat's Theorem, if any (odd) number *n* is prime, then for any *a* we have $a^{n-1} = 1 \pmod{n}$ [10]. This suggests that the Fermat test for a prime is: pick a random $a \in \{1, ..., n-1\}$ and see if $a^{n-1} = 1 \pmod{n}$. If not, then *n* must be composite. Another simplest primality test is as follows: given an input number *n*, check whether any integer *m* from 2 to *n - 1* evenly divides *n* (the division leaves no remainder). If *n* is divisible by any *m,* then *n* is composite; otherwise it is prime. We can further eliminate testing divisors greater than √n. We can also eliminate all the even numbers greater than 2, since if an even number can divide *n*, so can 2. The algorithm can be improved further by observing that all primes are of the form $6k \pm 1$, with the exception of 2 and 3. This is because all integers can be expressed as $(6k + i)$ for some integer *k* and for i = −1, 0, 1, 2, 3, or 4; 2 divides $(6k + 0)$, $(6k + 2)$, $(6k + 4)$; and 3 divides $(6k + 3)$. So, a more efficient method is to test if *n* is divisible by 2 or 3, then to check through all the numbers of form $6k \pm 1$. This is 3x as fast as testing all *m*.

Software that can test the primality of numbers is used in a number of online distributed computing projects, including the Great Internet Mersenne Prime Search (GIMPS) [6] and PrimeGrid [11]. These projects attempt to discover prime numbers not only for the sake of curiosity but also to satisfy certain hypotheses that, while lacking a mathematical proof, make some claim that can be searched exhaustively. The "Seventeen or bust" project [12] is an example of a group

attempting to prove a hypothesis by exhaustive search. This project is attempting to solve the Sierpinski problem - to find the smallest Sierpinski number. A number *k* is a Sierpinski number if k x 2^N +1 is not prime for all *N*; in 1967 Sierpinski and Selfridge conjectured that 78,557 was the smallest such number. Seventeen or Bust is searching for primes of the form k x 2^N + 1; there are very few *k*'s left that are smaller than 78,557.

The software LLR (short for Lucas-Lehmer-Riesel) was developed to test the primality of large integers. It utilizes various algorithms to determine the primality of many different types of number. Numbers of the form $N = k \times 2^n - 1$ can be tested by the LLR algorithm [13] which is the code path this project aims to accelerate. Factorization is more computationally demanding and as such these efficient prime tests do not provide a 'short cut' to obtain prime factors more quickly than integer factorization is able to provide.

It can therefore be seen that the core of the algorithm is calculation of the sequence $\{u_i\}$. Normally, integer multiplication is extremely fast on modern CPUs and is not a performance issue; however, the numbers being tested by this algorithm go far beyond the limits of even 64-bit floats. Naive or 'long' multiplication, has a time complexity of $O(n^2)$ (for multiplying two numbers of length *n*) which would take far too long to execute. Therefore, a faster method is required.

Modern primality test algorithms can be divided into two main classes: probabilistic and deterministic algorithms [14]. Probabilistic algorithms may report a composite number as a prime, but certainly do not identify primes as composite numbers. Deterministic algorithms, on the other hand, do not have the possibility of such erring. While deterministic primality tests take more computing power, probabilistic tests need fairly less computing power.

Instead of covering all primes, Thall et al focus on Mersenne primality test using the LLR test [15, 16]. Modern GPUs, with their massively parallel and shared-memory architectures, are quintessence platforms for the tests. Using Fast Fourier transforms to complete a p-bit multiplication, a Mersenne number can be tested in $O(pN \log N)$ arithmetic operations for an N-word radix representation.

Worley et al discuss how simple trial division with the integers from 2 to √n is a classic method, but in O(√n) [17]. The article proposes a variant of Fermat's Little Theorem to find strong probable prime (SPRP). Since the SPRP test may sometime fail to detect a composite, testing needs to be repeated with multiple independent SPRP tests, so that a number repetitively passed is more likely a prime. A CUDA implementation of the procedure is made by the author and the results indicate that the primality test takes significant less time on a GPU than on a CPU. The GPGPU demonstrates up to 257x speedup over MATLAB while solving the Laplace's equation with an error tolerance of 0.0001 [7].

As most primality test algorithms are sequential in nature, they should take more execution time as the problem size increases. Multithreaded parallel algorithms can be applied for multicore/manycore computing systems so that the primality test for large numbers can be performed in a short period of time.

3. PROPOSED ALGORITHM

In this work, we propose an algorithm to perform the primality test in constant time. We use a multicore/manycore CPU/GPGPU system to execute the proposed multithreaded parallel algorithm. Let's refer the system as M-Machine with 4 tuples as depicted below using Equation (1).

$$M = \langle n, Q, \delta, f \rangle \quad \text{..... Equation (1)}$$

Where, *n* is the input number whose primality is to be tested,

Q is the set of $\sqrt{n} - 1$ count of cores (CUDA threads in our experiment) numbered as $\{Q_i ; i = 2 \text{ to } \sqrt{n}\}$,

δ is the generalized algorithm to be executed in each of the $\sqrt{n} - 1$ processors in parallel, and

f is the flag used to mark n as composite by making $f \leftarrow 0$. It is initialized as $f \leftarrow 1$.

The pseudo code of the proposed algorithm is shown in Figure 1(a), where Step 1 is executed concurrently in parallel in $\sqrt{n} - 1$ different processors.

Flag $f \leftarrow 1$.

Input integer $n > 1$.

Step 1. For every Q_i, if $\{n \bmod i \neq 0\}$, make $f \leftarrow 0$. [$i = 2,3,4,\ldots \lfloor\sqrt{n}\rfloor$; ***executed in parallel***]

Step 2. If $\{f = 1\}, n$ is Prime

Else n is composite

Fig 1(a): Proposed Algorithm (pseudo code)

The parallel flow-of-control of the M-Machine in Equation (1) is illustrated in Figure 1(b). Here, each GPGPU processing core is responsible to execute only one CUDA thread.

ISSN 2277-3061

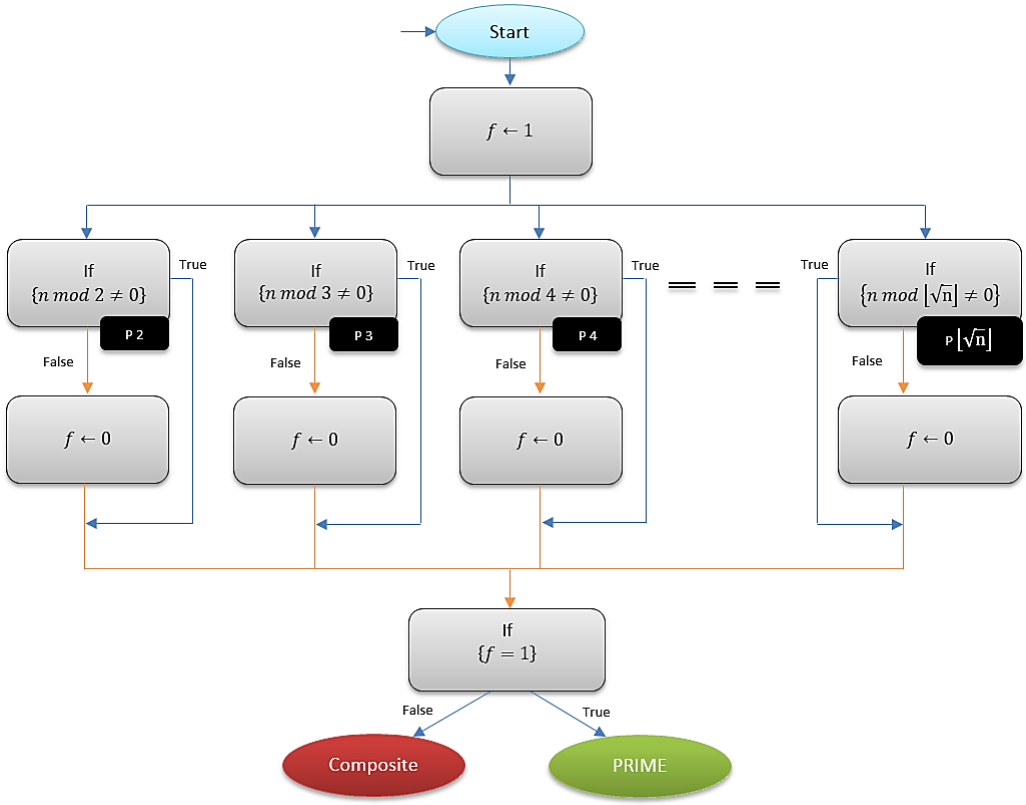


Fig 1(b): The parallel flow of control in M-Machine

Primality tests can be performed in constant time using the proposed algorithm in a multicore system with $\sqrt{n} - 1$ processing units. A generalized algorithm which is to be executed in every processing core of the M-Machine is illustrated in Figure 1(b).

Fundamentally, as $n \to \infty$, it becomes impossible to accommodate $\sqrt{n} - 1$ processing cores on a GPGPU card using today's technology. So, for finite number of processing cores, the machine and the algorithm are modified accordingly, as shown in Equation (2).

$M^{CUDA} = <n, Q', \delta, f>$ ..... Equation (2)

Where, *n* is an input number whose primality is to be tested, *Q'* is the set of *x* threads numbered as $\{Q'_i ; i = 2 \text{ to } (x + 1)\}$, δ is the generalized algorithm to be executed in each of the x threads in parallel, and f is the flag used to mark *n* as composite by making $f \leftarrow 0$. It is initialized as $f \leftarrow 1$.

The improved proposed algorithm is shown in Figure 2(a). This modified algorithm allows execution of more than one CUDA thread in a GPGPU processing core as illustrated in Figure 2(b).

Flag $f \leftarrow 1$.

Input integer $n > 1$.

---

1. For integer $a \leftarrow 0$ to $a < (\lceil \frac{\sqrt{n}}{x} \rceil)$

   For every Q'_i, if $\{n \bmod ((a \times x) + i) \neq 0\}$, make $f \leftarrow 0$.

   $[i = 2,3,4,\ldots (x + 1); \textbf{executed in parallel}]$

2. If $\{f = 1\}, n$ is Prime

   Else n is composite

Fig 2(a): Improved Proposed Algorithm (pseudo code)

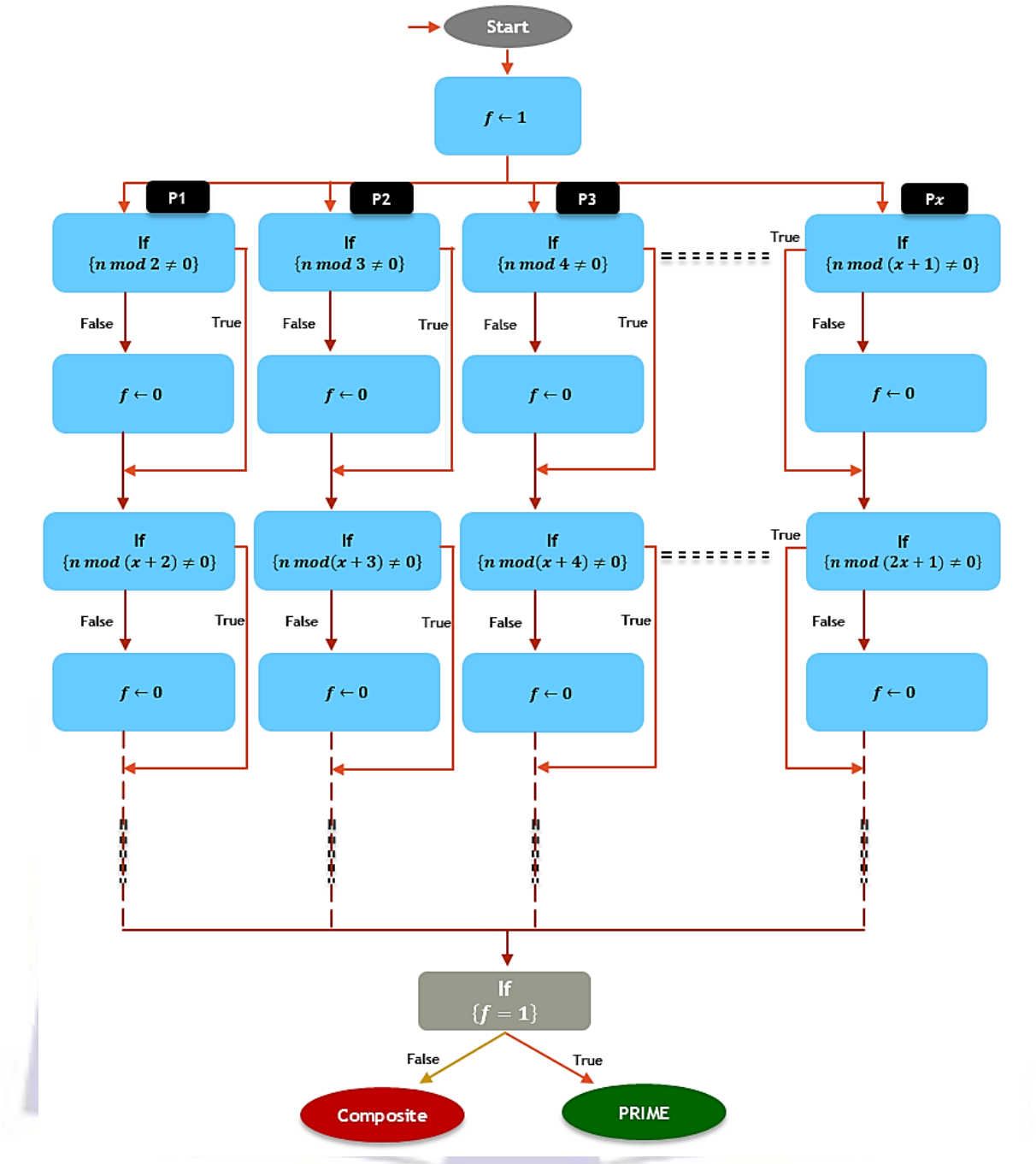


Fig 2(b): The parallel flow of control in the modified M-Machine

According to the improved algorithm, primality tests can be performed in $O(\log_x \sqrt{n})$ time. So, the time required for a primality test is inversely proportional to the number of threads/cores. The time to test the primality of a number *n* can be presented using Equation (3).

$T_n \propto 1/x$ ..... Equation (3)

In order to compare the proposed algorithm illustrated in Figure 2(b) with the sequential execution, let's say, *n* = 16769023 and *x* = 1024.

Therefore, $\sqrt{n} \approx 4094$

And, time required due to the improved algorithm is: $T_{16769023} = \left\{\left\lceil \frac{\sqrt{n}}{x} \right\rceil\right\} \approx 4 \; time \; units$

In contrast, sequentially execution would have taken about 4094 time units.

4. EXPERIMENTAL DETAILS

In this section, we discuss the experimental details to evaluate the proposed CUDA/GPGPU assisted deterministic primality test algorithm. Experimental setup including related hardware, related software, assumptions, and speedup are presented in the following subsections.

4.1 Related Hardware

The significant hardware components used in this experiment are: an Intel Xeon CPU and a NVIDIA Tesla GPGPU card. Some important CPU and GPGPU parameters are listed in Table 1. The CPU has a total of 8 cores and the GPGPU has a total of 448 processing cores. The size of the shared memory on each GPGPU streaming multiprocessor (SMP) (i.e., for each CUDA block) is 32 KB.

Table 1. Important CPU and GPGPU parameters

CPU	GPGPU
• Processor: Intel Xeon E5506	• Type: NVIDIA Tesla C2075
• Cores: 2 x Quad-Core	• Cores: 14 x 32 Cores
• Threads: 2 x 4	• RAM: 6GB GDDR5
• Clock Speed: 2.13 GHz	• RAM Speed: 1.5 GHz
• RAM: 8GB DDR3	• RAM Bandwidth: 144 GB/sec
• Max. Memory Bandwidth: 19.2 GB/sec	• Shared Memory per Block: 32 KB
• Power: 80 Watt	• Power: 255 Watt
• OS: Linux (Debian)	• CUDA: Version 5.5

4.2 Related Software

We use Linux Debian 7.0 operating system and GNU Compiler Collection (GCC) version 4.6.3. We configure CUDA 5.5 following the instructions provided in NVIDIA Developer Zone (URL: https://developer.nvidia.com/cuda-downloads). NVIDIA Driver version 319.37, supplied with cuda-toolkit installer, is used. The CUDA architecture includes several components designed strictly for GPGPU computing and it is aimed to alleviate many of the limitations that prevented previous graphics processors from being legitimately useful for general-purpose computation. The CUDA Architecture includes a unified shader pipeline, allowing each and every arithmetic-logic unit (ALU) on the chip to be marshaled by a program intending to perform general-purpose computations [14]. Because NVIDIA intends this new family of graphics processors to be used for general purpose computing, these ALUs are built to comply with IEEE requirements for single-precision floating-point arithmetic and are designed to use an instruction set tailored for general computation rather than specifically for graphics. Furthermore, the execution units on the GPGPU are allowed arbitrary read and write access to memory as well as access to a software-managed cache known as shared memory. All of these features of the CUDA Architecture are added in order to create a GPGPU that would excel at computation in addition to performing well at traditional graphics tasks. In this experiment, the software is compiled with no optimization.

4.3 Assumptions

Following important assumptions are made:

- It is assumed that the appropriate host system's kernel supports and the utilities for the GPGPU card are fully functional to ensure the best computation performance of the CPU/GPGPU system.
- It is assumed that 1024 CUDA threads per CUDA blocks should provide the best performance.
- CUDA with and without shared memory are implemented. It is assumed that the amount of shared memory per 32-core in a SMP equals the maximum amount of shared memory per block.

4.4 Speedup

In this work, we analyze the performance improvement of the proposed CUDA/GPGPU-based primality test algorithm using speedup as shown in Equation (4):

$$S_P = \frac{T_S}{T_P} \quad \text{..... Equation (4)}$$

where S_P is the speedup, T_S is the best sequential time, and T_P is the parallel run time. It should be noted that T_S represents the time due to the CPU/C solution, and T_P represents the time due to the CUDA/GPGPU/C solution.

5. RESULTS AND DISCUSSION

In this section, we discuss some experimental results obtained by implementing the proposed deterministic primality test algorithm as explained using Equation (2) and Figure 2(b).

To test the proposed primality test algorithm, we consider decimal numbers up to 21 digits (18,446,744,073,709,551,615 to be specific). Initially, for smaller (less than 16 digits) numbers, the time required to execute three different implementations (CPU/C, CUDA/C without shared memory, and CUDA/C with shared memory) are almost the same. As the target number start getting larger (i.e., the number of digits increases from 16), the CPU/C implementation takes more time to complete the execution. The times required due to CUDA/C implementations with and without shared memory also increase. However, time due to CUDA/C with shared memory is much smaller than that due to CUDA/C without shared memory. This is because the GPGPU shared memory holds data closer to the processing core and help reduce average (GPGPU) memory latency.

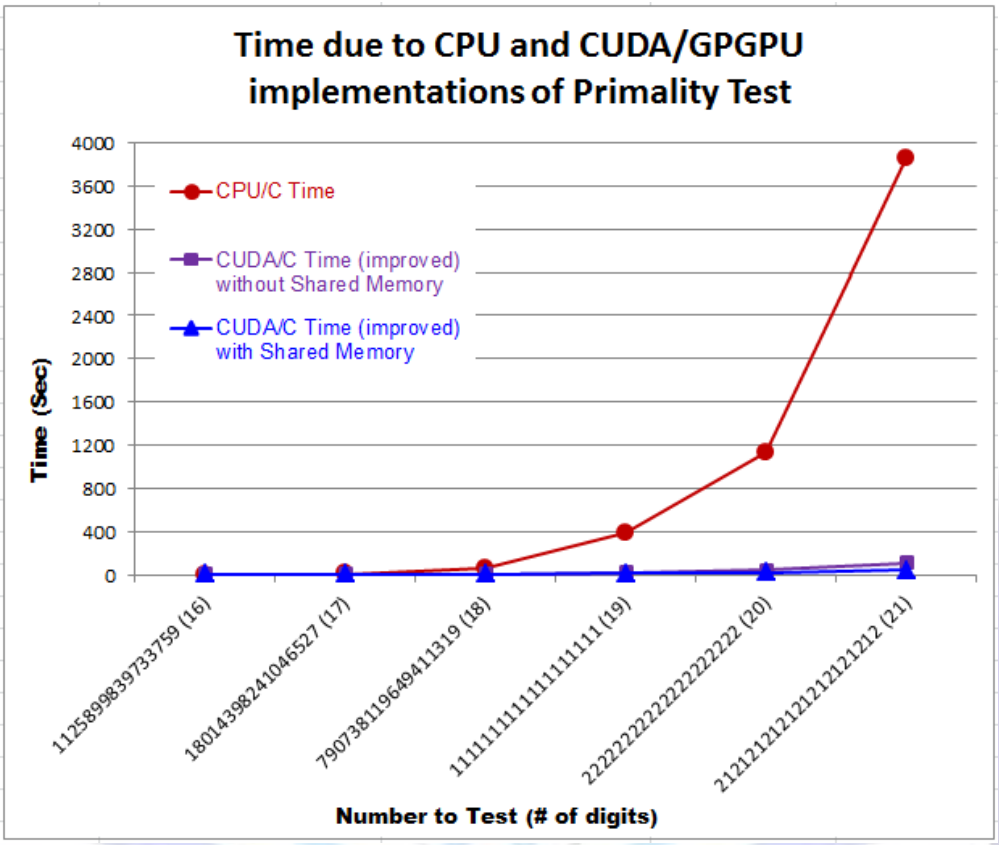


Fig 3: Execution time due to CPU/C and GPGPU/C implementations

We calculate the speedup due to CUDA/GPGPU implementations over CPU-only implementation using Equation (4). The speedups due to CUDA/C with and without shared memory are illustrated in Figure 4. For smaller (less than 16 digits) numbers, the speedup is less than 1.0 (due to the CPU/GPU programming overhead). However, the speedup increases as the target number start getting larger (i.e., the number of digits increases from 16). It is observed that the speedup due to CUDA/C without shared memory is same or smaller than that due to CUDA/C with shared memory. It is important to note that for large numbers the speedup due to CUDA/C with shared memory is significant, but the speedup due to CUDA/C without shared memory does not increase much as the problem size increases.

We were unable to test numbers larger than 18,446,744,073,709,551,615 because of long long data type constrains in C. This can be evaded by using compiler-specific extensions allowing custom data types, such as Big Integer Library [18]. With Big Integer Library, one can do arithmetic on integers of size limited only by computer's memory.

Although the algorithm is quite efficient for primality tests when a large number of processing units are available, there is a scope for colossal performance improvements [19]. A good way to speed up is to pre-compute and store a list of all primes up to a certain bound. Then, before testing n for primality with a serious method, n can first be checked for divisibility by any prime from the list. If it is divisible by any of those numbers then it is composite, any further tests can be skipped. Such a list can be computed with the Sieve of Eratosthenes [20] or by an algorithm that tests each incremental n against all known primes $< \sqrt{n}$.

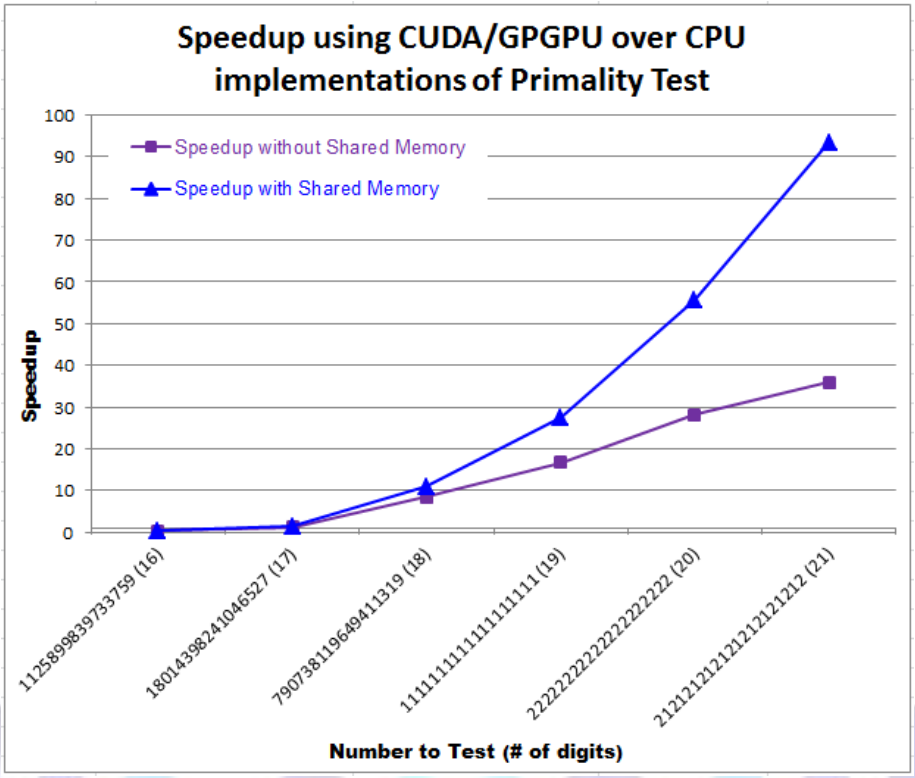


Fig. 4. Speedup due to GPGPU/C implementations over CPU/C implementation

6. CONCLUSION

Number theory in general and the study of prime numbers in particular are drawing more attentions than ever before from many fields (including computer science and statistics) outside of its own domain (Mathematics). Since 1970s, prime numbers are being used as the basis for the creation of public key cryptography algorithms. Several public-key cryptography algorithms, such as RSA (512-bit primes) and the Diffie–Hellman (1024-bit primes) key exchange, are based on large prime numbers. Prime numbers are also used for hash tables and pseudorandom number generators. Many primality testing algorithms, including mathematical models and computer programs, have been developed. However, the computer programs are very time consuming when the given number *n* is very large. Recently, multithreaded parallel programming using CUDA/GPGPU has increasingly been used to speed up the numerical computations.

In this work, we introduce a CUDA-accelerated novel parallel computing model based on a deterministic algorithm for CPU/GPGPU systems to perform primality tests much faster. We implement the proposed algorithm on a system with a 8-core CPU and 448-core GPGPU system using with and without GPGPU shared memory. We run the proposed algorithm to complete primality test on numbers up to 21-digit long. Experimental results direct that CUDA/GPGPU implementations execute much faster comparing with the CPU-only implementation (as shown in Figure 3). Results also suggest that GPGPU shared memory implementation is the best solution and it may help achieve speedup up to 94.35x for 21-digit numbers (see Figure 4). Using CUDA/GPGPU implementations not only saves time, but also should shrink power consumption and thus should help reduce heat dissipation.

Due to the difficulty of writing some portions of the code into CUDA, some specific portion of data processing needs to be computed on host CPU; therefore, the performance may severely be limited by memory transfers between CPU and GPGPU. In the near future, GPGPU is expected to support more features such as dynamic parallelism and recursive kernels, which will result in full use of primality test code implemented in CUDA.

As an extension to this work, we plan to apply the proposed algorithm to complete prinality test on number larger than 21 digits. We also plan to implement the proposed algorithm on GPGPU cards with more than 448 cores using GPGPU shared memory and GPGPU texture memory.

REFERENCES

[1] Rivest, R., Shamir, A., and Adleman, L., 1977. RSA Algorithm. http://en.wikipedia.org/wiki/RSA_%28algorithm%29 (accessed on 12/15/2013).

[2] Adleman, L.M., Rivest, R.L., and Shamir, A. 1983. U.S. Patent 4,405,829: Cryptographic communications system and method. Granted to MIT in 1983. http://www.google.com/patents/US4405829 (accessed on 12/15/2013).

[3] Apostol, T.M. 1976. Introduction to Analytic Number Theory. In Springer (New York).

[4] Rabin, M.O. 1980. Probabilistic algorithm for testing primality. In J. Number Theory, Vol. 12, pp. 128–138.

[5] Cormen, T.H., Leiserson, C.E., Rivest, R.L., and Stein, C. 2001. Section 31.8: Primality testing - Introduction to Algorithms. MIT Press and McGraw–Hill (2nd Ed.), pp. 887–896.

[6] Cooper , C. 2013. 48th Known Mersenne Prime Discovered. http://www.mersenne.org/various/57885161.htm (accessed on 12/15/2013).

[7] Asaduzzaman, A., Yip, C.M., Asmatulu, R., and Rahman, M. 2013. CUDA/C Based 'Green' Technology for Very Fast Analysis of Nanocomposite Properties. In SAMPE Tech 2013 Conference, Wichita, KS, Oct. 2013.

[8] Asaduzzaman, A., Yip, C.M., Kumar, S., and Asmatulu, R. 2013. Fast Effective Computer Simulation of Lightning Strike Protection on Nanocomposites. In IEEE SoutheastCon Conference 2013, Jacksonville, FL, 2013.

[9] TOP500 Supercomputer Sites. 2013. http://www.top500.org/lists/2013/06/ (accessed on 12/15/2013).

[10] Lynn, B. 2013. Number Theory - Primality Tests, Stanford Crypto Group. http://crypto.stanford.edu/~blynn/ (accessed on 12/15/2013).

[11] PrimeGrid. http://www.primegrid.com/ (accessed on 12/15/2013).

[12] Seventeen or Bust: Distributed Computing. http://www.seventeenorbust.com/

[13] Riesel, H. 1969. Lucasian Criteria for the Primality of N = h x 2n – 1. In Mathematics of Computation (American Mathematical Society), Vol. 23, No. 108, pp. 869-875.

[14] Sanders, J. and Kandrot, E. 2011. CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley (1st Ed.).

[15] Thall, A. 2011. Fast Mersenne prime testing on the GPU. In Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units, ACM.

[16] McBain, D. 2013. Accelerated Primality Testing Using GPUs. Thesis, MSc in HPC, the University of Edinburgh.

[17] Worley, S. 2009. Optimization of primality testing methods by GPU evolutionary search. In GPUs for Genetic and Evolutionary Computation (2009).

[18] McCutchen, M. 2013. C++ Big Integer Library. https://mattmccutchen.net/bigint/index.html (accessed on 12/15/2013).

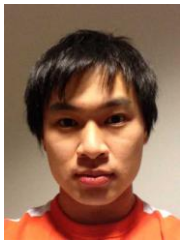
[19] Agrawal, M., Kayal, N., and Saxena, N. 2004. PRIMES is in P. In Annals of Mathematics (2nd Series), Vol. 160, No. 2, pp. 781-793.

[20] O'Neill, M.E. 2008. The Genuine Sieve of Eratosthenes. In Journal of Functional Programming, pp. 10-11, DOI: 10.1017/S0956796808007004.

Author' biography with Photo

Abu Asaduzzaman received the Ph.D. and M.S. degrees, both in computer engineering, from Florida Atlantic University, USA. He is an Assistant Professor in the department of Electrical Engineering and Computer Science (EECS) at Wichita State University (WSU), USA. His research interests include computer architecture, embedded systems, and parallel programming for high-performance low-power computing. He has authored 60+ articles in these areas. He is a member of IEEE, ASEE, PKP, TBP, etc. He served as NSF panel reviewers and currently serving as IEEE TPC/IPC members.

Anindya Maiti is currently working on his M.S. degree program in Electrical Engineering from the EECS department at WSU, USA. He received the Bachelor of Technology degree in Computer Science from Vellore Institute of Technology, India, in 2012. His research interests include parallel programming and security & privacy in Computing.



Chok M. Yip is currently working on his M.S. degree program in Computer Networking from the EECS department at WSU, USA. He received the Bachelor of Science degree in Computer Engineering from WSU in 2011. His research interests include high performance computing, vehicular ad hoc network (VANET), and software defined network (SDN). He has published severel articles in these areas. He is a student member of IEEE.