Toward a Multi-GPU Implementation of the Modular Integer GCD Algorithm: Extended Abstract

KENNETH WEBER  
University of Mount Union  
Department of Computer Science  
Alliance, OH, USA  
weberk@mountunion.edu

JUSTIN A. BREW  
University of Mount Union  
Alumnus, Department of Computer Science  
Alliance, OH, USA  
jbrew5662@gmail.com

ABSTRACT
The modular integer greatest common divisor (GCD) algorithm [9] holds promise to provide superior performance to sequential algorithms on extremely large input. In order to demonstrate the efficacy of the algorithm, an implementation on a system with multiple Graphics Processing Units (GPUs) is proposed, based on a single-GPU implementation described herein. The implementation’s performance is analyzed to predict the size of input needed to demonstrate superior performance when compared to one popular sequential implementation of the integer GCD.

CCS CONCEPTS
• Theory of computation → Massively parallel algorithms;

KEYWORDS
GPU, Integer GCD

ACM Reference Format:

1 INTRODUCTION
Euclid’s algorithm to compute the greatest common divisor (GCD) of two integers is one of the oldest algorithms known [7, sect. 4.5.2]. His algorithm describes a process that is inherently sequential, as are most algorithms typically used to compute the GCD, including those currently used by the the GNU Multiprecision Arithmetic Library (GMP) [6]. The modular integer GCD algorithm [9] is unique in that it employs a modular representation for the integer inputs and intermediate results in order to provide a way to parallelize the task. What follows describes an implementation [1] of the modular algorithm on a single NVIDIA graphics processing unit (GPU) [5] that could be used as a foundation for a multinode implementation providing superior performance on very large input values.

2 ALGORITHM OVERVIEW
The implementation of the modular algorithm described here uses the variant of the original given in Figure 1. It estimates the number of moduli that will be needed and checks to see whether there will be enough moduli to finish the computation at each iteration of the reduction loop. It also incorporates corrections to two errors in Steps MGCD3.2 and MGCD4 of the original algorithm [9].

Input: Positive integers $U$ and $V$, with $U \geq V$  
Output: gcd($U$, $V$)

Constants: $L = \text{integer} \geq 2$  
$M = \text{set of primes in the range } (2^{L-1}, 2^{L})$  
$C_L = 1.6 - 0.015 \cdot L$

1 $N_u \leftarrow \lfloor \log_2 U \rfloor + 1$, $N_v \leftarrow \lfloor \log_2 V \rfloor + 1$
2 $N_Q \leftarrow \lceil C_L \cdot N_u / \log_{10} N_u \rceil$
3 if $N_Q > ||M||$ then return fail
4 $Q \leftarrow \text{the set of } N_Q \text{ largest elements of } M$
5 forall $q \in Q$ do
6 $\lfloor u_q, v_q \rfloor \leftarrow [U \mod q, V \mod q]$
7 $t_q \leftarrow$ if $v_q = 0$ then $\infty$ else $u_q / v_q \mod q$
8 end
9 $[p, b] \leftarrow \{ \text{element } q \text{ of } Q \text{ for which } |t_q| \text{ is minimal, } t_q \}$
10 repeat // Reduction loop
11 $Q \leftarrow Q - \{ p \}$
12 forall $q \in Q$ do
13 $\lfloor u_q, v_q \rfloor \leftarrow \lfloor v_q, (u_q - b \cdot v_q) / p \mod q \rfloor$
14 $t_q \leftarrow$ if $v_q = 0$ then $\infty$ else $u_q / v_q \mod q$
15 end
16 $N_Q \leftarrow N_Q - 1$, $[N_u, N_v] \leftarrow [N_v, N_u - L + \lfloor \log_2 b \rfloor]$  
17 if $N_Q (L - 2) \leq N_u$ then // Can’t recover $G$
18 return fail
19 $[p, b] \leftarrow \{ \text{element } q \text{ of } Q \text{ for which } |t_q| \text{ is minimal, } t_q \}$
20 until $b = c$
21 $k \leftarrow 1$, $G \leftarrow 0$
22 $[p, g] \leftarrow \{ \text{element } q \text{ of } Q \text{ with priority to } u_q \neq 0, u_q \}$
23 repeat // Recover mixed-radix representation
24 $Q \leftarrow Q - \{ p_k \}$
25 forall $q \in Q$ do
26 $u_q \leftarrow (u_q - g_k) / p_k \mod q$
27 $k \leftarrow k + 1$
28 $[p_k, g_k] \leftarrow \{ \text{element } q \text{ of } Q \text{ with priority to } u_q \neq 0, u_q \}$
29 until $g_k = 0$
30 for $i = k - 1$ downto 0 do
31 $G \leftarrow g_i + p_i \cdot G$
32 return $|G|$ // Return standard representation

Figure 1: Modular algorithm, as implemented
A least-squares fit of the first 10 execution times for the Tesla V100 GPU, the most powerful device (and having the most recent architecture) of those included in the graph, is also provided. Any system activity that may be included in the measurements. Compute the values; care was taken to reduce, as much as possible, the remainder when dividing a 64-bit value by a 32-bit modulus via integer operations [9]. The GCD algorithms used by GMP for large input are essentially $O(N^{1+\log N})$ for a fairly large value of $\epsilon$ [6, section 15.3.3], so—given enough processing elements—a multinode implementation of the modular GCD algorithm should be faster than the GMP implementation for very large input.

It can be seen from Figure 2 that the execution times exhibit linear behavior up to the point at which the device becomes saturated by the number of threads it must support, which appears at input sizes of 160 Kibit for the Tesla V100. Using the linear least-squares fit displayed in the figure, we predict that the modular algorithm will be faster than the GMP implementation for inputs of over 250 million bits, at which value GMP GCD took 121 seconds for one pair of 250 million bit inputs, and our extrapolation projects a running time of around 112 seconds on a hypothetical multinode system with enough GPUs of the same type as the projection is based on, and with fast enough communication between GPUs. With 68 million usable 32-bit moduli, a multi-GPU implementation should be able to handle input sizes of up to 529 million bits, based on the formula for $N_Q$ from line 2 of Figure 1. Although this projection encourages further investigation, only an actual multinode implementation of the modular algorithm will allow a definitive assessment of its efficacy.

5 CONCLUSION

Given enough processing elements, the modular GCD algorithm should exhibit linear behavior [9]. The GCD algorithms used by GMP for large input are essentially $O(N^{1+\log N})$ for a fairly large value of $\epsilon$ [6, section 15.3.3], so—given enough processing elements—a multinode implementation of the modular GCD algorithm should be faster than the GMP implementation for very large input.

It can be seen from Figure 2 that the execution times exhibit linear behavior up to the point at which the device becomes saturated by the number of threads it must support, which appears at input sizes of 160 Kibit for the Tesla V100. Using the linear least-squares fit displayed in the figure, we predict that the modular algorithm will be faster than the GMP implementation for inputs of over 250 million bits, at which value GMP GCD took 121 seconds for one pair of 250 million bit inputs, and our extrapolation projects a running time of around 112 seconds on a hypothetical multinode system with enough GPUs of the same type as the projection is based on, and with fast enough communication between GPUs. With 68 million usable 32-bit moduli, a multi-GPU implementation should be able to handle input sizes of up to 529 million bits, based on the formula for $N_Q$ from line 2 of Figure 1. Although this projection encourages further investigation, only an actual multinode implementation of the modular algorithm will allow a definitive assessment of its efficacy.