# Faster Modular Exponentiation using Double Precision Floating Point Arithmetic on the GPU

Niall Emmart*, Fangyu Zheng†‡, and Charles Weems*

*College of Information and Compute Sciences, University of Massachusetts, Amherst, MA 01003, USA
†State Key Laboratory of Information Security, Institute of Information Engineering,
Chinese Academy of Sciences, Beijing, China
‡Data Assurance and Communication Security Research Center, Chinese Academy of Sciences, Beijing, China

***Abstract*—This paper presents a new approach to integer multiple precision (MP) modular exponentiation, using double-precision floating point (DPF) operations, that is suitable for GPU implementation. We show speedups ranging from 20% to 34% over the best prior GPU times for sizes corresponding to common RSA cryptographic operations (2048 to 4096 bits). Three techniques are described. First, by adding $2^{104}$ to the high half of the product, and $2^{52}$ to the low half, we set the implicit leading 1 in the DPF mantissa so that the full 52 explicit bits are available for each half of the 104-bit products of samples. Second, the DPF values are cast bitwise to 64-bit integers for adding the column sums to get the MP result. Normally the cast would require masking off the exponents, but because they are constant, we can include them in the column sums and correct just once for their total. Third, by initializing the column sums with the appropriate negative value to compensate for the exponent sums, no corrective subtraction is needed. Our implementation on an NVIDIA GTX Titan Black GPU achieves between 132.5K and 161.9K modular exponentiations per second of size 1024 bits, with latencies ranging from 21.7 ms to 17.8 ms, making it practical for online RSA applications. Proportional results are shown for 1536 and 2048 bits. The implementation is so efficient that its maximum sustained performance is actually bounded by the thermal limit of the GPU.**

In this paper we examine a new approach to multiple precision (MP) modular exponentiation on the GPU using double precision floating point (DPF) arithmetic. There have been a number of papers that have used floating point arithmetic as the basis for modular multiplication and exponentiation algorithms. The idea is to leverage the high speed floating point units on the GPU to perform integer arithmetic. Moss, Page and Smart [1] use a mixed radix approach with a vector of co-prime moduli, where each modulus is 12-bits in length and compute $a_i \cdot b_i$ fmod $m_i$ using single precision floating point (SPF) arithmetic. Fleissner [2] implements 192-bit modular exponentiation with six 24-bit values. The 24-bit values are further sampled into bytes and the computations on the bytes are done using SPF arithmetic. In [3] Bernstein et al. implement a 280-bit modular multiplication based on a traditional fixed radix number system (FRNS) with 28 limbs of 10-bit samples (each sample is a 10-bit integer stored in a SPF) and a three-full-products approach to Montgomery multiplication [4]. They use a sophisticated scheme to distribute each multiplication across 28 threads in a warp. But in essence, the column sums of a multiple precision product (see Figure 1) can be thought of as dot-products, $\sum_{i,j} a_i \cdot b_j$ where $A$ and $B$ are the MP values to be multiplied and $i + j$ equals the column number. The $a_i$ and $b_i$ samples and all of the computations are done using SPF arithmetic. But we note that $28 \cdot (2^{10} - 1) \cdot (2^{10} - 1)$ can exceed $2^{24}$ which can result in rounding errors. To work around this problem, Bernstein et al. compute two partial sums of at most 14 terms, which guarantees there won't be any rounding, and then perform the final sum using integer arithmetic. Bernstein et al. have carefully crafted a highly optimized implementation, but unfortunately, 10-bit samples are too small to be efficient. In their subsequent paper [5], Bernstein et al. perform all computations using integer arithmetic, which proved to be much faster. Zheng et al. in [6] (and two follow-on papers [7] and [8]) use floating point arithmetic to implement modular multiplication, modular exponentiation, and RSA. In [6] they use an FRNS with 23 bits per limb and Montgomery's multiple-precision modular reduction, which

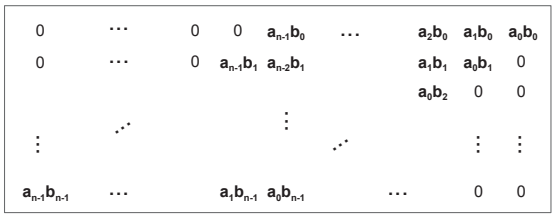


Fig. 1. Product terms to be summed for an $n$-sample by $n$-sample multiply.

This material is based upon work supported by the National Science Foundation (NSF) under award No. CCF-1525754 and the National Key R&D Program of China under award No. 2017YFB0802103.

978-1-5386-2613-9/18/$31.00 ©2018 IEEE