PSPrimeQ - version 1, 10.07.2009
by Jarosław Wróblewski

jwr@math.uni.wroc.pl
http://www.math.uni.wroc.pl/∼jwr/AP26/GPU

This document is an attempt to produce pseudoprimality testing algorithm suitable for implementing on GPU, i.e. using only 32-bit operations.

p - unsigned integer with slightly less than 64 significant bits. This number is being tested for base 2 pseudoprimality:

$$2^p \equiv 2 (\mathrm{mod}\, p)$$

z - number of leading zeros of p when represented as 64-bit unsigned int. Hence p has $64-z$ significant bits:

$$2^{64-z-1} \leqslant p < 2^{64-z}$$

R - reciprocal of p with 128 significant bits, rounded down:

$$R = \left\lfloor \frac{2^{191-z}}{p} \right\rfloor$$

R is represented as 128-bit unsigned int.
$r = 1/p$ can be computed using recursive formula:

$$r = \frac{1-e}{p}$$

$$r_{new} = 2r - pr^2$$

Then

$$r_{new} = \frac{1-e^2}{p}$$

Note that the above recursion assumes r to be a real number close to $1/p$.

A - a number with at most $127-z$ significant bits. Our short term goal is to compute $A (\mathrm{mod}\, p)$.
As - number A shifted left by $z+1$ bits.

f - 64-bit product of two 128-bit integers As and R. We do not care about 128 highest bits of the 256-bit product, so in principle we could multiply those as 128-bit integers, where the overflow bits are being discarded. Also we do not care about the 64 lowest bit, but we do care about some precision of the lowest bit we keep. So perhaps one could compute 80 bits (ignore computations on the lowest 48 bits) and discard the 16 lowest bits from the result.

out - 64-bit product of two 64-bit integers f and p, where we want to keep only the 64 highest bits. It should be enought to perform computations on the highest 80 bits and then discard 16 lowest bits from the result.

We now have

$$A \pmod p = out + 1$$

provided errors aren't too large - this has to be traced carefully.

Also we assume that A is not divisible by p, but that is always the case if A arises from pseudoprimality testing and p is not a power of 2.