Basic Pollard Rho Algorithm Implementation On CUDA Device

Martin Beránek

Faculty of Information Technology – Czech Technical University in Prague

May 5, 2017

1 Introduction

Factorisation problem of a huge number resolved into massive parallel solutions. Large number of algorithms are currently state-of-art and are continuously developed into better forms. This short article is focused on implementation of Pollard-Rho algorithm on CUDA device. In first part there is a definition of algorithm. Next the article focuses on options of parallelism on CUDA device. Results are measured in multiple instances and compared in graphs.

2 Definition of the algorithm

The ρ algorithm (named after the shape of curves symbolising two functions trying to reach themselves in projective space) is based on finding cycle. In t random numbers of $x_1, x_2, \ldots, x_t$ in range $[1, n]$ will contain repetition with probability of $P > 0.5$ if $t > 1.777n^{\frac{1}{2}}$

The ρ algorithm uses $g(x)$ for modulo n as a generator for pseudo-random sequence. In this article function $x^2 + k$; $k \in \mathbb{N}$. We are assuming that $n = p \cdot q$ that also mean $p < \sqrt{n}$ and $q < \sqrt{n}$. Algorithm actually generates sequence of $x_1 = g(2)$, $x_2 = g(g(2))$, ... in two separated sequences running in same time. One sequence is generated as $x_1 = g(x_0) \mod n$ and second as $x_1 = g(g(x_0)) \mod n$. Since we know that $p < \sqrt{n}$ the faster sequence is likely to cycle faster then the sequence generated just in in application of a g function. The repetition of the $\mod p$ will be detected with $|x_k - x_m|$ which will divide p without residue [1].

```
x=2; y=2; d=1
while d is 1:
    x = g(x)
    y = g(g(y))
    d = gcd(|x - y|, n)
    if d = n:
        return failure
    else:
        return d
```

Figure 1: Sequential pseudo-code of algorithm

While implementation in Python is basically just a copy of pseudo-code, in C there is plenty struggle with fix size integer size. Another problem will be with transformation of libraries like GNU MP, that's why the basic focus on sequential solution was on basic mathematical operation. For the basic adding and subtracting there was no bigger effort than just creating basic implementation with carry bits. Much more time was spent on operation like division and modulo.

2.1 Division

First idea on how to divide numbers was to use school *Long division* algorithm which proved to be little bit expensive but was behaving in the same manners for every input numbers.

```
Q = 0
R = 0
for i = n - 1 .. 0 do
    R = R << 1
    R(0) = N(i)
    if R >= D then
        R = R - D
        Q(i) = 1
```

Figure 2: Long division [2]

Another much faster division was to use *Knuth's division*

3 CUDA Solution

3.1 First solution with explicit barrier

3.2 Second solution with independent runners

4 Conclusion

References

[1] Wikipedia. Pollard's Rho algorithm. 2016. Available from: https://en.wikipedia.org/wiki/Pollard's_rho_algorithm

[2] Wikipedia. Division algorithm. 2016. Available from: https://en.wikipedia.org/wiki/Division_algorithm