COMP4920 Senior Design Project II, Spring 2020
Advisor: Dr. Hüseyin Hışıl

FIWE: Factoring Integers with ECM Product Manual

Revision 2.0
24.05.2020

By:
Asena Durukan, Student ID: 16070001016
Aslı Altıparmak, Student ID: 15070001003
Elif Özbay, Student ID: Ç15070002011
Hasan Ozan Soğukpınar, Student ID: 17070001047
Nuri Furkan Pala, Student ID: 15070006030

Revision History

Revision	Date	Explanation
1.0	13.04.2020	Initial Product Manual
2.0	24.05.2020	New functions are added GUI and website implementations are discarded Existing C functions are removed from CUDA functions in Section 2

Table of Contents

1 Introduction

This document aims to briefly explain the FIWE Software in terms of installation, setup, usage, and testing.
In Section 2, how the source code structured and organized, which specialized software tools are used in the development process and what kind of platforms the development process operated on are mentioned. Section 3, describes the testing procedures for software subsystems. Finally, how one can install the software for use, arrange it according to her/his needs, and operate it on a device, are mentioned in Section 4.
For additional information; the requirement specifications are defined in the RSD [1], the design specifications, the testing methods and their results are explained in the DSD [2].

2 FIWE Software Subsystem Implementation

2.1 Source Code and Executable Organization

In the final version of this project, all source codes are written in both C and CUDA language. Hence, source codes are divided into different files/parts.

2.1.1 C

main.c

- fiweBash(int $argc$, char $*argv[]$);

mplib.c
mplib.h

- #define W;
- #define fiweCopy(z, $startZ$, $endZ$, a, $startA$);
- void fiweRand(unsigned int *z, unsigned int l);
- void fiweModRand(unsigned int *z, unsigned int l, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul);
- void fiwePrint(unsigned int *a, unsigned int al, char *s);
- void fiweAdd(unsigned int *z, unsigned int *a, unsigned int al, unsigned int *b, unsigned int bl);
- void fiweModAdd(unsigned int *z, unsigned int *a, unsigned int al, unsigned int *b, unsigned int bl, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul);
- void fiweSub(unsigned int *z, unsigned int *a, unsigned int al, unsigned int *b, unsigned int bl);
- void fiweModSub(unsigned int *z, unsigned int *a, unsigned int al, unsigned int *b, unsigned int bl, unsigned int *n, unsigned int nl);
- void fiweMul(unsigned int *z, unsigned int *a, unsigned int al, unsigned int *b, unsigned int bl);
- void fiweModMul(unsigned int *z, unsigned int *a, unsigned int al, unsigned int *b, unsigned int bl, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul);
- void fiweBarretReduction(unsigned int *z, unsigned int *m, unsigned int ml, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul);
- void fiweBinaryGCD(unsigned int *d, unsigned int *a, unsigned int al, unsigned int *n, unsigned int nl);

- void fiweInvert(unsigned int *z, unsigned int *a, unsigned int al, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul);
- void fiweIsEqual(unsigned int *z, unsigned int *a, unsigned int *b, unsigned int l);
- void fiweIsEqualUi(unsigned int *z, unsigned int *a, unsigned int al, unsigned int *b);
- void fiweGetMu(unsigned int *z, unsigned int *n, unsigned int nl);
- void fiweGetA24(unsigned int *$A24$, unsigned int *A, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul, int *$flag$);

safegcd.c
safegcd.h

- #define fiweSafegcdUFVGC(u, f, v, g, c, h, l);
- void fiweSafegcdPrint(FILE *file, char *str, long *a, long len);
- void fiweSafegcdMul(long *zn, long u, long *fn, long v, long *gn, long len);
- void fiweSafegcdDivSteps(long n, long t, long *δ, long f, long g, long *uu, long *vv, long *qq, long *rr);
- void fiweSafegcd(long *z, long *f, long *g, long *U, long *V, long *Q, long *R, long *$precomp$, long len)

montgomery.c
montgomery.h

- struct fiweMontgCurve;
- struct fiweProPoint;
- void fiweProCurvePoint(unsigned int *d, fiweMontgCurve *c, fiweProPoint *p, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul, int *$flag$);
- void fiweProAdd(fiweProPoint *p, fiweProPoint *$p1$, fiweProPoint *$p2$, fiweProPoint *pd, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul);
- void fiweProDbl(fiweProPoint *p, fiweProPoint *$p1$, unsigned int *$A24$, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul);
- void fiweProLadder(fiweProPoint *p, fiweProPoint *$p1$, unsigned int *$A24$, unsigned int k, unsigned int *n, unsigned int nl, unsigned int *mu, unsigned int mul);

ecm.c
ecm.h

- void fiwECM(unsigned int *d, unsigned int *n, unsigned int nl);

test.c
test.h

- void fiweProCurvePointGMPTest(int THRESHOLD);
- void fiweProAddGMPTest(int THRESHOLD);
- void fiweProAddMagmaTest(int THRESHOLD);
- void fiweProDblMagmaTest(int THRESHOLD);
- void fiweProLadderGMPTest(int THRESHOLD);
- void fiweProLadderMagmaTest(int THRESHOLD);
- void fiwECMGMPTest(int THRESHOLD);
- void fiweSafegcdMulPyTest(int THRESHOLD);

2.1.2 CUDA

All functions listed in C section are also implemented in CUDA programming language. The additional functions and macros for CUDA are listed below.
main.cu
montgomery.cu
montgomery.h
mplib.cu
mplib.h

- #define __sub_cc(r,a,b) ASM ("sub.cc.u32")
- #define __subc_cc(r,a,b) ASM ("subc.cc.u32")
- #define __addcy(carry) ASM ("addc.u32")
- #define __addcy2(carry) ASM ("addc.cc.u32")
- #define __subcy(carry) ASM ("subc.u32")
- #define __mul_lo(r,a,b) ASM("mul.lo.u32")
- #define __mul_hi(r,a,b) ASM("mul.hi.u32")
- __device__ float fiweGPUGenerate (curandState* globalState);

test.cu
test.h
ecm.cu
ecm.h

2.2 Software Development Tools

In this section, various software tools, compilers, and development environments which are used in this project are described.

2.2.1 Compilers

GCC (7.5.0 - 2019)
At first, GNU Compiler Collections (GCC) is developed by Richard Stallman for his GNU project. Nowadays, it is used to compile major languages such as C, C++ and many others. In this project, the functions are written in C language before they are implemented on CUDA. Therefore, GCC is used as a compiler of C programming language.
NVCC - (CUDA 10.0 2015)
NVCC stands for NVIDIA CUDA Compiler which is developed by NVIDIA in order to compile CUDA codes. Since CUDA runs both on the GPU and CPU, the code must be split with using a compiler. NVCC seperates the GPU and CPU codes, and then it compiles device codes and sends host codes to a C compiler such as GCC.

2.2.2 Development Environments

Nsight (9.2 - 2018)
NVIDIA Nsight is a CUDA development environment for numerous platforms of GPU computing. In this project, Nsight, integrated with Eclipse IDE on Linux operating system, is used. In addition, Nsight provides a debugger on GPU which makes parallel coding much easier.

Visual Studio Code (1.44.0 - 2015)
Visual Studio Code is a code editor for many languages such as C, C++ and Python. It is also a cross-platform editor which means it is available for Windows, macOS and Linux. As mentioned, before the CUDA implementation of the project, all functions are written in C programming language. This step is done on Visual Studio Code.

2.2.3 Testing

GMP (6.2.0 - 2020)
GMP is a multi-precision arithmetic library written in C programming language. It allows fast computation with the numbers that are bigger than the word size of personal computers. It is designed to be fast by optimizing the code according to the hardware architectures, implementing well-designed algorithms, using assembly for efficiency and so on [3]. Since GMP is fast and easy to implement on C codes, it is used for testing purposes throughout the project.

Magma Calculator (2.25-4 - 2020)
Magma is a programming language designed for the examination of algebraic, geometric and combinatorial structures. The syntax of Magma is similar to other well-known programming languages [4]. Due to the complexity of the elliptic curve and multi-precision arithmetic operations, Magma calculator is used in this project. It allows easy computation for testing purposes [2].

Python (3.8.1 - 2019)
Python is a high level programming language with having an exceptionally fast edit-test-debug cycle. Since it has easy to build data structures and dynamic semantics, it is one of the best platforms for testing purposes. Thus, it is used for the testing of GCD functions in this project.

2.2.4 Version Control

Git (2.17.1 - 2018)
Git is an open-source version control system that allows project members to trace the development process of a system. Repositories are created in this project for both the software development and documentation purposes. Those repositories are hosted on GitHub which is explained below.

2.2.5 Web Services

GitHub
GitHub is the most common hosting service, which released in February 2008, for developers to share their source codes. It also provides interfaces for computer and mobile users. GitHub is used by all project members for sharing their source codes to synchronize the project.

Dropbox
Dropbox is another hosting service which founded in 2007. It provides many features such as cloud storage, file synchronization and so on. In this project, it is used for file sharing across the project members.

2.3 Hardware and Software System Platform

2.3.1 NVIDIA GPU

Within the scope of this project, a GPU was required to execute the ECM algorithm in parallel. NVIDIA GTX 1050 Ti is used to met this need. It has 768 CUDA Cores which are necessary to be used in order to run the algorithm on SIMD architecture. Furthermore, NVIDIA GTX 1050 Ti supports many technologies such as CUDA programming language. Thus, it provides the usage of CUDA language to parallelize the ECM algorithm on the graphic card.

2.3.2 INTEL CPU

Considering that CUDA language works both on CPU and GPU, a CPU was also needed. For instance, a lot of functions which are C programming language related, can only be used on CPU. In this project, Intel Core i7-7700HQ model is used.

2.3.3 Ubuntu 18.04

Ubuntu is an open source complete Linux operating system based on Debian which is appropriate for both desktop and server use. When compared with other operating systems, Ubuntu is significantly more flexible to run the generated codes on GPU. For this reason, Ubuntu is used as an operating system.

3 FIWE Software Subsystem Testing

This section describes the testing procedures for the software system. It includes the step-by-step explanations of the test cases, their outcomes and the total time spent for each test case.
For all functions in the montgomery and ECM libraries, test cases are generated by the help of GMP and Magma. For test cases using GMP, after the test case is run and completed, it printouts the number of successes and fails. For test cases using Magma, test case printouts the calculations to a file in Magma syntax, then from Magma, this file is read and number of successes & fails are written into another file. All tests are applied to C functions. The crucial ones are tested in CUDA also.
The generated test cases are explained below.

- Montgomery Curve and Projective Point Generation Function GMP Test
  1. Generates a random curve and a projective point using the function
  2. Checks if the coefficients of the curve and the point satisfies the curve equation using GMP
- Projective Addition Function GMP Test
  1. Generates two random points
  2. Computes the addition by implementing the algebraic calculations using GMP
  3. Compares the result of the function with the GMP result
- Projective Addition Function Magma Test
  1. Generates two random points
  2. Computes the addition by implementing the algebraic calculations using Magma
  3. Compares the result of the function with the Magma result
- Projective Doubling Function Magma Test
  1. Generates a random point
  2. Computes the double by implementing the algebraic calculations using Magma
  3. Compares the result of the function with the Magma result
- Projective Ladder Function GMP Test
  1. Generates a curve c and a projective point $p1$ on it
  2. Generates random multipliers k an l
  3. Multiplies the point with first k and then l such that $p3 = l * (k * p1)$
  4. Multiplies the point with first l and then k such that $p5 = k * (l * p1)$

5. Calculates $p3 \rightarrow X * p5 \rightarrow Z$ using GMP
6. Calculates $p5 \rightarrow X * p3 \rightarrow Z$ using GMP
7. Compares the results of step 5 and 6 using GMP

- Projective Ladder Function Magma Test
  1. Generates a curve c and a projective point $p1$ on it
  2. Generates random multipliers k an l
  3. Multiplies the point with first k and then l such that $p3 = l * (k * p1)$
  4. Multiplies the point with first l and then k such that $p5 = k * (l * p1)$
  5. Calculates $p3 \rightarrow X * p5 \rightarrow Z$ using Magma
  6. Calculates $p5 \rightarrow X * p3 \rightarrow Z$ using Magma
  7. Compares the results of step 5 and 6 using Magma
- fiweGCDMul Function Python3 Test
  1. Generates three 64 bits, two multi-precision numbers
  2. Computes $u \times f + v \times g + c$
  3. Compares the result of the function with the Python3 result
- fiwECM Function GMP Test
  1. Generates a random composite number
  2. Calculates a factor of the number using the function
  3. Checks whether the factor found actually divides the composite number

Table 1: Test Case Statistics

Function Name	Is Tested?	Size of Inputs	# of Trial	# of Success	# of Fail	Time Spent in C (sec)
proCurvePointGMP	Yes	1-100	10000	10000	0	15
proAddGMP	Yes	1-100	10000	10000	0	12
proAddMagma	Yes	1-100	10000	10000	0	45
proDblMagma	Yes	1-100	10000	10000	0	33
proLadderGMP	Yes	1-100	10000	10000	0	27
proLadderMagma	Yes	1-100	10000	10000	0	143
safegcdPyMul	Yes	1-100	10000	10000	0	37
ecmGmpTest	Yes	1-10	1000	1000	0	34

As seen in Table 1, all the functions are tested and validated which verifies that the software works as planned.

4 FIWE Installation, Configuration and Operation

There are installation and operation processes for the FIWE project as all qualified software projects. Users can use FIWE in two different ways which are listed below.

- Source code as a library
- Bash script on Unix-like terminal

Installation is needed to use the Bash script.

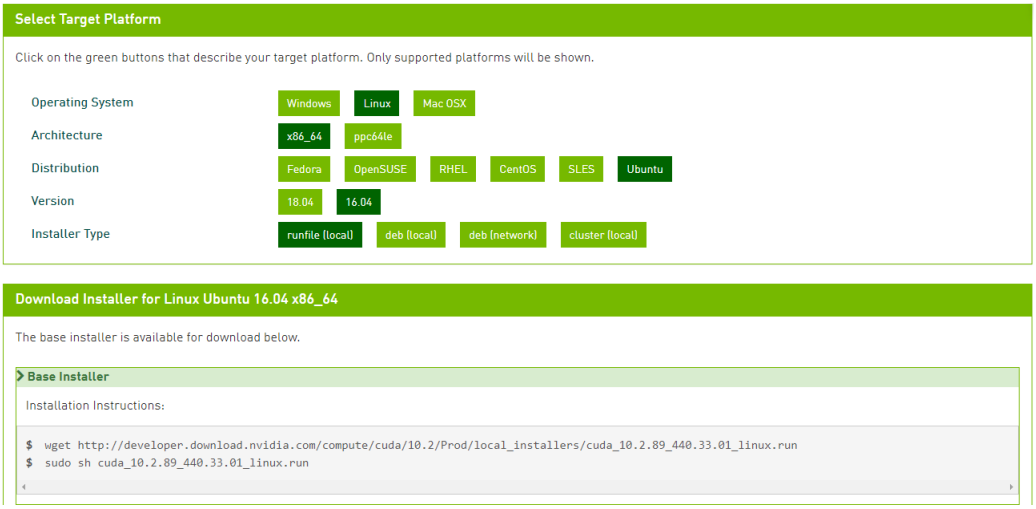
4.1 Installation

There is an executable file to setup the Bash script. This executable file is programmed for Unix-like operating systems. When the executable is run, fiwecm program gets callable from any directory.

System Requirements
The requirements for FIWE to work are listed below.

- CUDA-capable GPU
- Linux Environment
- GCC compiler
- NVIDIA CUDA Toolkit
  - The user can easily install the Toolkit from the NVIDIA website (`https://developer.nvidia.com/cuda-10.0-download-archive`) as shown in Figure 1. The CUDA version used in this project is 10.0.

Figure 1: Cuda Download

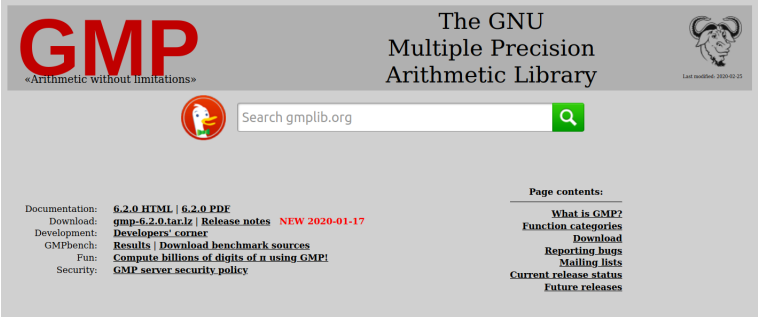


- GMP library
  - The user should install and include the GMP library to run the test cases. Since one of the operations which requires division was out of scope for this project, it was handled by GMP. For this reason, the user must download the GMP library from the official website `https://gmplib.org/` shown in Figure 2.

4.2 Configuration

A CUDA code can be compiled using NVCC. Therefore, the user must switch from GCC to NVCC with using the following terminal command.
$ sudo prime-select nvidia

Figure 2: GMP Download



4.3 Operation

Since there are two different ways to use this software, operations for each is different. Simply, user enters the number to be factorized, and the factors get printed out. In case of any unexpected behavior which the software cannot handle, error messages are displayed.

4.3.1 Source Code

The source code of FIWE is written in the C programming language. So, the ECM function and other auxiliary functions can be called by including the library as below.

```
#include <fiwe.h>
```

After including this library, the ECM function gets callable. There are three parameters of this function which are; the pointer for the factor found, the pointer of the number to be factorized, the size of the number in base 2^W where W is the word size of the architecture. The multi-precision numbers are represented as *unsigned int* arrays, so the parameter types are; *unsigned int **, *unsigned int ** and *unsigned int* respectively. The first parameter for the factor acts as an output for the function. An example of the usage of the ECM function is given below.

```
unsigned int number[3];
unsigned int factor[3];
number = {<digit1>, <digit2>, <digit3>};
ecm(factor, number, 3);
```

To factorize multiple multi-precision numbers simultaneously, the numbers should be given as a flat array. A flat array is constructed by adding multiple numbers sequentially into an array. An example of factorizing three 3-digit numbers is given below.

```
unsigned int numbers[9];
unsigned int factors[9];
numbers = {<num1digit1>, <num1digit2>, <num1digit3>, <num2digit1>, <num2digit2>, <num2digit3>, <
  num3digit1>, <num3digit2>, <num3digit3>};
ecm(factors, numbers, 3);
```

To compile the code on CPU, user must go to the ECM folder and use the compile command below.
$ gcc main.c -o main.o ecm.c montgomery.c mplib.c test.c -lgmp
To compile the code on GPU, user must go to the CUDA folder and use the compile command below.

$ nvcc main.cu -o main.o ecm.cu montgomery.cu mplib.cu test.cu -lgmp

4.3.2 Bash Script

This software can be run from the Bash terminal in any directory. The usage of the script with all options are given in Table 2.

Table 2: Commands & Explanations

Command	Explanation
fiwecm - -help	Shows the manual
fiwecm <number>	Printouts a factor of the number
fiwecm -f <in.txt>	Printouts the factors of the numbers inside "in.txt"
fiwecm <number> -o <out.txt>	Writes a factor of the number to "out.txt"
fiwecm -f <in.txt> -o <out.txt>	Writes the factors of the numbers inside "in.txt" to "out.txt"

References

[1] A. Durukan, A. Altıparmak, E. Özbay, H. O. Soğukpınar, and N. F. Pala. Requirement specification document, 2019.

[2] A. Durukan, A. Altıparmak, E. Özbay, H. O. Soğukpınar, and N. F. Pala. Desing specification document, 2020.

[3] T. Granlund and G. D. Team. *GNU MP 6.0 Multiple Precision Arithmetic Library.* Samurai Media Limited, London, GBR, 2015.

[4] C. P. John J. Cannon. *An Introduction to Algebraic Programming with Magma.* 2001.