Параллельное программирование

ИИКС ИБ
Б19-505
Голигузов Алексей

Содержание

1 Исследование

Реализация системы шифрования, основанной на идее, предложенной в статье.

2 Алгоритм

Базовая часть алгоритма состоит из следующих частей:

1. Генерация последовательности по сигнатуре реккуренты; в качестве базы реккуренты берется набор из n единиц(1), где n - длина сигнатуры
2. Генерация базовых подстрок(блоков) по сигнатуре
3. Обновление множества базовых подстрок за счет полной параллельной проекции их на строку, длиной, равной размеру блока
4. Генерация карты представления чисел

Шифрование:

1. Каждый элемент во входном тексте представляеся в виде суммы элементов(жадным алгоритмом) карты
2. Данное представление упаковывается в конечное представление - строку, длиной, равной размеру блока

Дешифрование:

1. Входной текст делится на куски, длиной в размер блока
2. Каждый кусок скалярно перемножается с посчитанной рекурентой, что дает значение исходного символа

Методика распараллеливания заключается в том, что мы можем обрабатывать каждый символ(при шифровании) и каждый блок(при дешифровании) независимо. Также некоторые внутренние операции также могут быть выполнены параллельно(скалярное перемножение, упаковка и другие).

3 Аппаратные характеристики тестового стенда

```
Distributor ID: Ubuntu
Description: Ubuntu 20.04.3 LTS
Release: 20.04
Codename: focal
```

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          8
On-line CPU(s) list:             0-7
Thread(s) per core:              2
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           142
Model name:                      Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz
Stepping:                        12
CPU MHz:                         3301.553
CPU max MHz:                     4900.0000
CPU min MHz:                     400.0000
BogoMIPS:                        4599.93
Virtualization:                  VT-x
L1d cache:                       128 KiB
L1i cache:                       128 KiB
L2 cache:                        1 MiB
L3 cache:                        8 MiB
NUMA node0 CPU(s):               0-7
Vulnerability Itlb multihit:     KVM: Mitigation: Split huge pages
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Enhanced IBRS, IBPB conditional, RSB filling
Vulnerability Srbds:             Mitigation; TSX disabled
Vulnerability Tsx async abort:   Not affected
Flags:                           fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush
```

```
              total        used        free      shared  buff/cache   available
Mem:           15Gi       2.3Gi       9.3Gi       423Mi       3.9Gi        12Gi
Swap:          29Gi          0B        29Gi
```

```
#define _OPENMP 201511
```

4 Программная конфигурация

Использованный язык программирования: C++
Компилятор: g++ (10.3.0)
Распараллеливание: OpenMP (201511)
Собиралось с флагами: -fopenmp -O3 -std=c++20

5 Эксперимент

Были проведены эксперименты для 5 различных размеров блока(16, 32, 64, 128, 256). Каждый эксперимент состоял из 10000 тестов, каждый тест для каждого кол-ва потоков(от 1 до 16). Внутри каждого теста сигнатура и входные данные генерировалиль случайно. В качестве сигнатуры реккуренты использовалась рандомно сгенерированная валидная сигнатура длины 2, величины коэффициентов - целые неотрицательные числа, не превышающие 9. Длина входного текста - фиксированная, 512 символов.

6 Графики

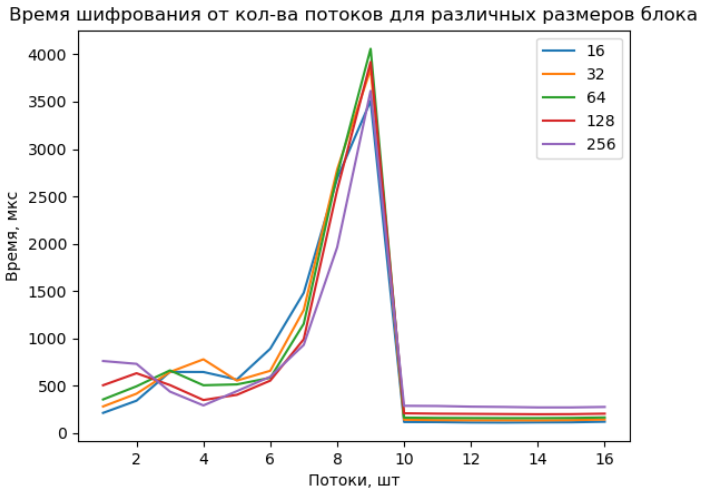

Рис. 1: Время

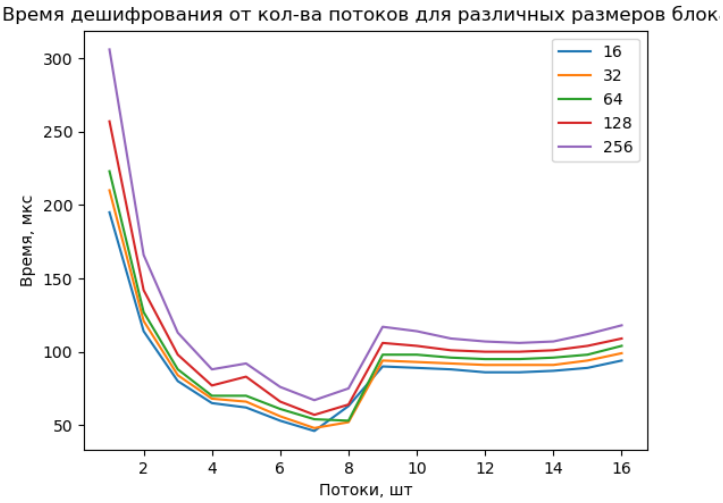

Рис. 2: Время

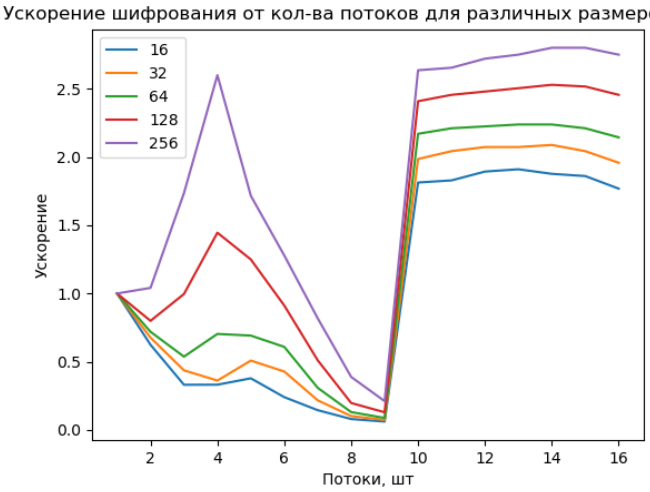

Рис. 3: Ускорение

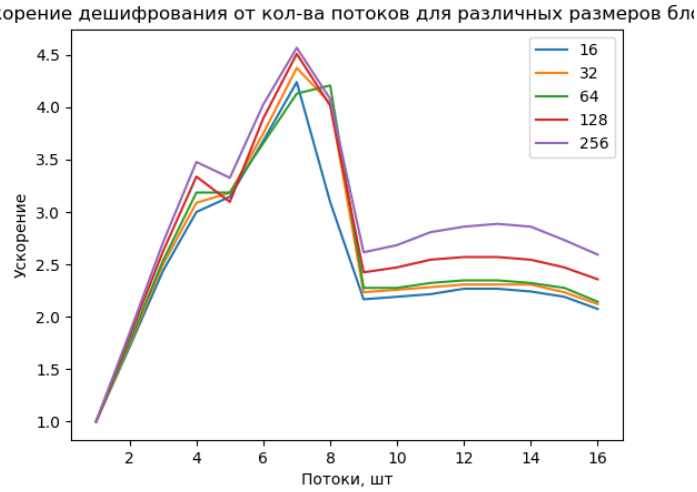

Рис. 4: Ускорение

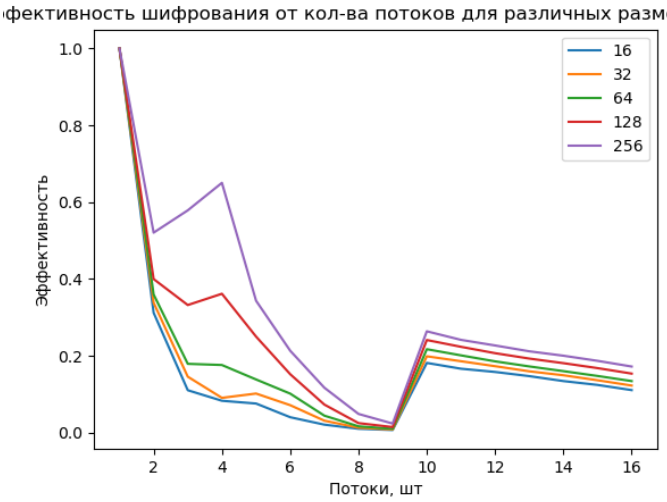

Рис. 5: Эффективность

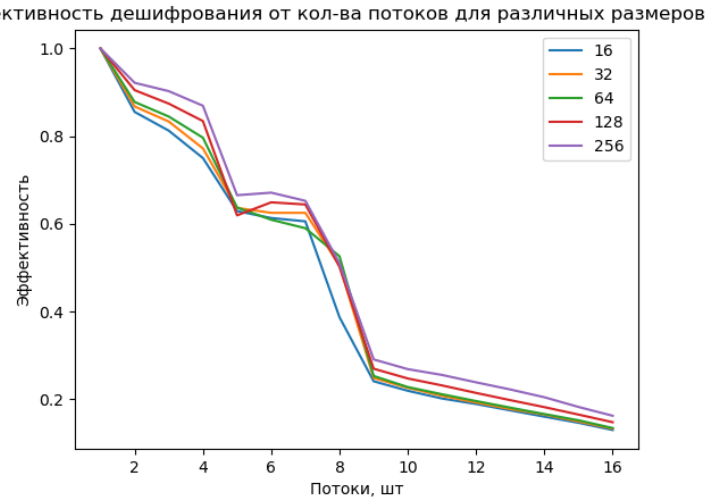

Рис. 6: Эффективность

7 Заключение

Получения реализация системы, в основе которой лежит идея из статьи. На основаниях графиков, можно сделать выводы, что распараллеленная версия дешифрования была ускорена в 2-3 раза по времени.

Для алгоритма шифрования, при кол-ве потоков > 9 имеем заметное ускорение в 2-3 раза, причем, чем больше размер блока, тем больше ускорение.

8 Приложение 1

```
#include <vector>
#include <string>
#include <iostream>
#include <string_view>
#include <algorithm>
#include <cassert>
#include <map>
#include <iomanip>
#include <set>
#include <iterator>
#include <random>
#include <chrono>
#include <omp.h>

inline static constexpr size_t BASE_TESTS_NUM{ 10 };
inline static constexpr size_t BASE_BLOCK_SIZE_MIN{ 10 };
inline static constexpr size_t BASE_BLOCK_SIZE_MAX{ 16 };
inline static constexpr size_t BASE_SIGN_LENGTH{ 2 };
inline static constexpr size_t BASE_INPUT_LENGTH{ 128 };
inline static constexpr size_t THREADS_NUM_MAX{ 16 };

#ifndef TESTS_NUM
#define TESTS_NUM BASE_TESTS_NUM
#endif /* !TESTS_NUM */

#ifdef BLOCK_SIZE
#ifndef BLOCK_SIZE_MIN
#define BLOCK_SIZE_MIN BLOCK_SIZE
#endif /* !BLOCK_SIZE_MIN */

#ifndef BLOCK_SIZE_MAX
#define BLOCK_SIZE_MAX BLOCK_SIZE
#endif /* !BLOCK_SIZE_MAX */
#endif /* BLOCK_SIZE */

#ifndef BLOCK_SIZE_MIN
#define BLOCK_SIZE_MIN BASE_BLOCK_SIZE_MIN
#endif /* !BLOCK_SIZE_MIN */

#ifndef BLOCK_SIZE_MAX
#define BLOCK_SIZE_MAX BASE_BLOCK_SIZE_MAX
#endif /* !BLOCK_SIZE_MAX */

#ifndef SIGN_LENGTH
#define SIGN_LENGTH BASE_SIGN_LENGTH
#endif /* !SIGN_LENGTH */

#ifndef INPUT_LENGTH
#define INPUT_LENGTH BASE_INPUT_LENGTH
#endif /* !INPUT_LENGTH */

namespace
{
        inline static constexpr auto MAX_SIZE_TO_PRINT{ 10u };
```

```
template<typename _T>
void dbg(const std::vector<_T>& data, const std::string_view& name)
{
        std::cout << "\n[" << name << "] {size=" << std::size(data) << "}\n";

        if (std::size(data) <= MAX_SIZE_TO_PRINT)
                std::copy(std::cbegin(data), std::cend(data), std::ostream_iterator<_T>(st
        else
        {
                std::copy(std::cbegin(data), std::cbegin(data) + MAX_SIZE_TO_PRINT - 1, st
                std::cout << "...\n";
        }

        std::cout << '\n';
}

template<typename _T>
void dbg(const std::map<_T, const std::string&>& data, const std::string_view& name)
{
        std::cout << "\n[" << name << "] {size=" << std::size(data) << "}\n";

        if (std::size(data) <= MAX_SIZE_TO_PRINT)
                for (const auto& [key, value] : data)
                        std::cout << key << ' ' << value << '\n';
        else
        {
                auto it = std::begin(data);
                for (size_t i{ 1 }; i < MAX_SIZE_TO_PRINT; ++i)
                {
                        std::cout << it->first << ' ' << it->second << '\n';
                        ++it;
                }

                std::cout << "...\n";
        }

        std::cout << '\n';
}

template<typename _T>
void dbg(const _T& value, const std::string_view& name)
{
        std::cout << "\n[" << name << "]\n" << value << '\n';
}

void dbg(const std::map<uint8_t, std::set<std::string>>& data, const std::string_view& nam
{
        std::cout << "\n[" << name << "] {size=" << std::size(data) << "}\n";
        for (const auto& [key, value] : data)
        {
                std::cout << "\t[len=" << static_cast<size_t>(key) << "]{size = " << std::
                if (std::size(value) <= MAX_SIZE_TO_PRINT)
                        for (const auto& str : value)
                                std::cout << "\t\t" << str << '\n';
                else
```

```
                                for (auto it{ std::begin(value) }; ; ++it)
                                {
                                        std::cout << "\t\t" << *it << '\n';

                                        if (std::distance(std::begin(value), it) >= MAX_SIZE_TO_PI
                                        {
                                                std::cout << "\t\t...\n";
                                                break;
                                        }
                                }
                        }

                        std::cout << '\n';
                }
        } // anonymous namespace

        namespace global
        {
                std::vector<std::vector<int>> encrypted_time(THREADS_NUM_MAX);
                std::vector<std::vector<int>> decrypted_time(THREADS_NUM_MAX);
        } // namespace global

        template<typename _Func>
        auto profile(_Func&& func, std::vector<int>& time)
        {
            auto t1 = std::chrono::high_resolution_clock::now();
            auto&& result = func();
            auto t2 = std::chrono::high_resolution_clock::now();

                time.push_back(std::chrono::duration_cast<std::chrono::microseconds>(t2 - t1).count());

                return result;
        }

        auto gen_S(const std::string_view& signature)
        {
                size_t A{};
        #pragma omp parallel for reduction(+: A)
                for (int i = 0; i < std::size(signature); ++i)
                        A += signature.at(i) - '0';

                std::vector<std::string> res;
                for (int i = 0; i < std::size(signature); ++i)
                        for (size_t j{}; j < signature.at(i) - '0'; ++j)
                                res.push_back(std::string(signature.substr(0, i)) + std::to_string(j));

                return std::pair{ res, A };
        }

        auto update_S(const std::vector<std::string>& S, const size_t block_size)
        {
                std::vector<std::string> new_s(S);
                for (const auto& s1 : S)
                        if (s1 != "0")
                                for (auto i{ 1LLU }; i + s1.length() <= block_size; ++i)
                                        new_s.push_back(s1 + std::string(i, '0'));
```

```
        return new_s;
}

template<typename _T>
void gen_seq(const std::string_view& signature, std::vector<_T>& seq, const size_t n)
{
        seq.reserve(std::size(seq) + n);
        for (int i = 0; i < n; ++i)
        {
                _T u{};
        #pragma omp parallel for reduction(+: u)
                for (int j = 0; j < std::size(signature); ++j)
                        u += static_cast<_T>(signature.at(j) - '0') * seq.at(std::size(seq) - 1 -

                seq.push_back(u);
        }
}

template<typename _T>
auto calc(const std::string& str, const std::vector<_T>& seq)
{
        _T sum{};
#pragma omp parallel for reduction(+: sum)
        for (int j = 0; j < str.length(); ++j)
                sum += static_cast<_T>(str.at(j) - '0') * seq.at(str.length() - 1 - j);

        return sum;
}

template<typename _T>
auto gen_V(const std::vector<_T>& seq, const std::vector<std::string>& S)
{
        std::map<_T, const std::string&> V;
        for (const auto& s : S)
                if (s != "0")
                {
                        auto value = calc(s, seq);
                        if (auto&& [_, suc] = V.try_emplace(value, s); !suc)
                                ;//std::cerr << "Error: value " << value << " exists with string \

                }

        return V;
}

template<typename _T>
std::vector<_T> find_repr(const std::map<_T, const std::string&>& V, _T value)
{
        if (V.contains(value))
                return { value };

        for (_T i = value - 1; i > 0; --i)
        {
                if (i < 0)
                        break;
```

```
                if (!V.contains(i))
                        continue;

                auto res = find_repr(V, static_cast<_T>(value - i));
                if (std::size(res) != 0)
                {
                        res.push_back(i);

                        return res;
                }
        }

        return {};
}

template<typename _T>
auto pack(const std::vector<_T>& repr, const std::map<_T, const std::string&>& V, const _T c, cons
{
        if (!std::size(repr))
        {
                std::cerr << "Error: find representation of '" << c << "'\n";
                return std::string(block_size, '0');
        }

        auto tmp = V.at(repr.front());
        auto res = std::string(block_size - tmp.length(), '0') + std::string(tmp);
        for (auto i{ 1LLU }; i < std::size(repr); ++i)
        {
                tmp = V.at(repr.at(i));
                auto tmp_str = std::string(block_size - tmp.length(), '0') + std::string(tmp);

        //#pragma omp parallel for
                for (int j = 0; j < tmp_str.length(); ++j)
                        res.at(j) = (tmp_str.at(j) != '0') ? tmp_str.at(j) : res.at(j);
        }

        return res;
}

template<typename _T>
auto encrypt(const std::string& str, const std::map<_T, const std::string&>& V, const size_t block
{
        std::string result(str.length() * block_size, '0');

#pragma omp parallel for
        for (int i = 0; i < str.length(); ++i)
        {
                auto&& packed = pack(find_repr(V, static_cast<_T>(str.at(i))), V, static_cast<_T>(
                for (size_t j{}; j < block_size; ++j)
                        result.at(i * block_size + j) = packed.at(j);
        }

        return result;
}

template<typename _T>
```

```
auto decrypt(const std::string& str, const std::map<_T, const std::string&>& V, const std::vector<
{
        assert(!(str.length() % block_size));

        std::string result(str.length() / block_size, '~');

#pragma omp parallel for
        for (int i = 0; i < str.length(); i += block_size)
        {
                if (auto pos = str.find_first_not_of('0', i); pos != std::string::npos)
                        result.at(i / block_size) = static_cast<char>(calc(str.substr(pos, block_s
        }

        return result;
}

auto gen_sign(const size_t length)
{
    static auto& numbers = "0123456789";

    thread_local static std::mt19937 rg{ std::random_device{}() };
        thread_local static std::uniform_int_distribution<std::string::size_type> extreme_pick(1,
    thread_local static std::uniform_int_distribution<std::string::size_type> pick(0, sizeof(numb

    std::string s;
    s.reserve(length);
    for (size_t i{}; i < length; ++i)
        if (!i || i + 1 == length)
                        s += numbers[extreme_pick(rg)];
                else
                        s += numbers[pick(rg)];

    return s;
}

auto gen_plaintext(const size_t length)
{
        static auto& chars = "0123456789"
                "abcdefghijklmnopqrstuvwxyz"
                "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                "!@#$%^&*()-=_+";

    thread_local static std::mt19937 rg{ std::random_device{}() };
    thread_local static std::uniform_int_distribution<std::string::size_type> pick(0, sizeof(chars

    std::string s;
    s.reserve(length);

    for (size_t i{}; i < length; ++i)
                s += chars[pick(rg)];

    return s;
}

auto main() -> signed
{
```

```
std::ios::sync_with_stdio(false);

std::cout << "input length:" << INPUT_LENGTH
        << " sign length:" << SIGN_LENGTH;

for (size_t block_size{ BLOCK_SIZE_MIN }; block_size <= BLOCK_SIZE_MAX; block_size *= 2)
{
        std::cout << "\nblock size:" << block_size;

        for (size_t i{}; i < TESTS_NUM; ++i)
        {
                auto&& sign = gen_sign(SIGN_LENGTH);
                // dbg(sign, "Signature");

                std::vector<size_t> U(sign.length(), 1);
                gen_seq(sign, U, block_size);
                // dbg(U, "U");

                auto&& [S, A] = gen_S(sign);
                // dbg(S, "S base");

                S = std::move(update_S(S, block_size));
                // dbg(S, "S updated");

                auto V = gen_V(U, S);
                // dbg(V, "V");

                auto&& plaintext = gen_plaintext(INPUT_LENGTH);
                // dbg(plaintext, "Plaintext");

                for (auto threads_num{ 1LLU }; threads_num <= THREADS_NUM_MAX; ++threads_nu
                {
                        omp_set_num_threads(threads_num);

                        auto encrypted = profile([&]() -> std::string { return encrypt(pla
                        // dbg(encrypted, "Encrypted");

                        auto decrypted = profile([&]() -> std::string { return decrypt(enc
                        // dbg(decrypted, "Decrypted");

                        // dbg(decrypted == plaintext, "Equals");

                        if (decrypted != plaintext)
                        {
                                dbg(sign, "Signature");
                                dbg(U, "U");
                                dbg(S, "S updated");
                                dbg(V, "V");
                                dbg(plaintext, "Plaintext");
                                dbg(encrypted, "Encrypted");
                                dbg(decrypted, "Decrypted");
                                std::exit(1);
                        }
                }
        }
```

```
            auto print_result = [](std::vector<std::vector<int>> data)
            {
                    std::vector<int> avgs;
                    for (size_t i{}; i < THREADS_NUM_MAX; ++i)
                    {
                            int avg{};
                            for (size_t j{}; j < TESTS_NUM; ++j)
                            {
                                    avg += data.at(i).at(j);
                                    //std::cout << data[j][i] << ',';
                            }

                            avg /= TESTS_NUM;

                            //std::cout << avg << '\n';

                            avgs.push_back(avg);
                    }

                    std::ranges::copy(avgs, std::ostream_iterator<int>(std::cout, ","));
                    std::cout << std::endl;
            };

            std::cout << "\nenc: ";
            print_result(global::encrypted_time);

            std::cout << "dec: ";
            print_result(global::decrypted_time);

            global::encrypted_time = std::vector<std::vector<int>>(THREADS_NUM_MAX);
            global::decrypted_time = std::vector<std::vector<int>>(THREADS_NUM_MAX);
    }
    return 0;
}
```