Мельников Лев СКБ 161

Детали реализации

Проект доступен в репозитории https://github.com/markgrin/factorization . Выбранные алгоритмы были реализованы на C++ с использованием библиотеки GMP и её оберкти для C++ GMPXX. Так как большинство алгоритмов вычисляют только один делитель числа, был реализован мета алгоритм последовательного применения факторизации:

```
std::vector<mpz_class> apply_method(
    std::function<mpz_class(mpz_class)> method,
    mpz_class number) {
    std::vector<mpz_class> result;
    std::vector<mpz_class> remaining;
    remaining.push_back(number);
    while (!remaining.empty()) {
        auto number = remaining.back();
        remaining.pop_back();
        auto divisor = method(number);
        if (number == divisor) {
            result.push_back(number);
            continue;
        }
        number /= divisor;
        remaining.push_back(number);
        remaining.push_back(divisor);
    }
    return result;
}
```

Полный перебор

Метод полного перебора никакого интереса не представляет

```
std::vector<mpz_class> full (mpz_class number) {
    mpz_class border = sqrt(number) + 1;
    mpz_class check = 0;
    std::vector<mpz_class> divisors;
    while (check <= border) {
```

```
        check = check + 1;
        if (!mpz_probab_prime_p(check.get_mpz_t(), 50)) {
            continue;
        }
        if (number % check == 0) {
            divisors.push_back(check);
            number = number / check;
            check -= 1; // Retry
        }
    }
    if (number != 1) {
        divisors.push_back(number);
    }
    return divisors;
}
```

Метод Ферма

В методе Ферма без начальной проверки на четность, алгоритм циклится. Помимо этого - ничего интересного.

```
mpz_class find_one (mpz_class number) {
    if (mpz_probab_prime_p(number.get_mpz_t(), 50) > 0) {
        return number;
    }
    if (number % 2 == 0) {
        return 2;
    }
    mpz_class left = sqrt(number);
    mpz_class full = left * left;
    if (full == number) {
        return left;
    } else if (full < number) {
        left += 1;
    }
    mpz_class x = left;
    mpz_class v = x * x - number;
    mpz_class k = 0;
    mpz_class additor = 1;
    while(true) {
        mpz_class check = sqrt(v);
```

```
        if (check * check == v) {
            return x + check;
        }
        x = x + 1;
        k += 1;
        // (x + 1)^2 = x^2 + 2x + 1
        // (x + 2)^2 = x^2 + 4x + 4
        // (x + 3)^2 = x^2 + 6x + 9
        v = v + left * 2 + k * 2 - 1;
        //v = x * x - number;
    }
    return number;
}
```

Метод Полларда-Флойда

В методе Полларда-Флойда оказалось, что некоторые числа не могут быть факторизованы с помощью данного многочлена при любых начальных значениях. Например x^2+1 не факторизует некоторые степени 5, например 25. Поэтому было написано несколько функций x^2+1, x^3+x^2+2, x^4+1. Если одна не срабатывает, используется следующая.

```
mpz_class find_one_polflo (mpz_class number) {
    if (mpz_probab_prime_p(number.get_mpz_t(), 50) > 0) {
        return number;
    }
    std::function<mpz_class(mpz_class)> fncs[3] =
    {func_1, func_2,func_3};

    for (auto fnc : fncs) {
        mpz_class x = 1;
        mpz_class z = x;
        mpz_class y = x;
        mpz_class p = 1;
        while(p <= 1) {
            x = fnc(x) % number;
            y = fnc(z) % number;
            z = fnc(y) % number;
            mpz_class diff = z - x;
            diff = (diff >= 0 ? diff : diff*-1);
```

```
            p = gcd(number, diff % number);
        }
        if (p != number) {
            return p;
        }
    }
    return number;
}
```

Метод Брента

Для метода Брента была использована аналогичная оптимизация:

```
mpz_class find_one_brent (mpz_class number) {
    if (mpz_probab_prime_p(number.get_mpz_t(), 50) > 0) {
        return number;
    }
    std::function<mpz_class(mpz_class)> fncs[3] = {
        func_1, func_2, func_3};

    for (auto fnc : fncs) {
        mpz_class x = 1;
        mpz_class z = 1;
        mpz_class p = 1;
        mpz_class k = 0;
        mpz_class two_power = 1;
        mpz_class tries = 1000 * 1000 * 10;
        while(p <= 1 && tries) {
            tries = tries - 1;
            k = k + 1;
            if (two_power == k) {
                z = x;
                two_power = two_power * 2;
            }
            x = fnc(x) % number;
            mpz_class diff = z - x;
            diff = (diff >= 0 ? diff : diff*-1);
            p = gcd(number, diff % number);
        }
        if (p != number) {
            return p;
```

```
        }
    }
    return number;
}
```

Метод Полларда

В качестве k было взято произведение всех простых чисел меньших 1000. Сначала возведение в степерь производилось в цикле, что занимало очень большое время. Потом была найдена быстрая функция из gmp mpz_powm, которая ускорила алгоритм.

```
mpz_class find_one (mpz_class number) {
    if (mpz_probab_prime_p(number.get_mpz_t(), 50) > 0) {
        return number;
    }
    std::vector<mpz_class> primes;
    mpz_class B = 1000;
    for (mpz_class i = 2; i < B; i++) {
        if (mpz_probab_prime_p(i.get_mpz_t(), 50) > 0) {
            primes.push_back(i);
        }
    }
    mpz_class k = 1;
    for (auto const & prime : primes) {
        k = k * prime;
    }

    std::size_t counter = 0;
    std::size_t tries = 1000 * 1000;
    while (counter < tries && counter < primes.size()) {
        mpz_class a = primes[counter];
        mpz_class p = gcd(a, number);
        if (p > 1) {
            return p;
        }
        mpz_class powered = 1;
        mpz_powm(powered.get_mpz_t(), a.get_mpz_t(),
            k.get_mpz_t(), number.get_mpz_t());
        powered = (powered + number - 1) % number;
```

```
        p = gcd(number, powered);
        if (p > 1 && p < number) {
            return p;
        }
        counter = counter + 1;
    }
    return number;
}
```

Многопоточный метода Брента

Предлагается модернизация метода Брента - искать делитель параллельно по разным многочленам:

```
void find_one_brent (mpz_class number,
std::function<mpz_class(mpz_class)> func,
std::atomic_char& stopper,
mpz_class& result) {
    char stop = 0;
    if (mpz_probab_prime_p(number.get_mpz_t(), 50) > 0) {
        if (stopper.compare_exchange_weak(stop, 1)) {
            result = number;
        }
        return ;
    }

    mpz_class x = 1;
    mpz_class z = 1;
    mpz_class p = 1;
    mpz_class k = 0;
    mpz_class two_power = 1;
    mpz_class tries = 1000 * 1000 * 1;
    while(p <= 1 && tries) {
        if (stopper.load()) {
            return ;
        }
        tries = tries - 1;
        k = k + 1;
        if (two_power == k) {
            z = x;
            two_power = two_power * 2;
```

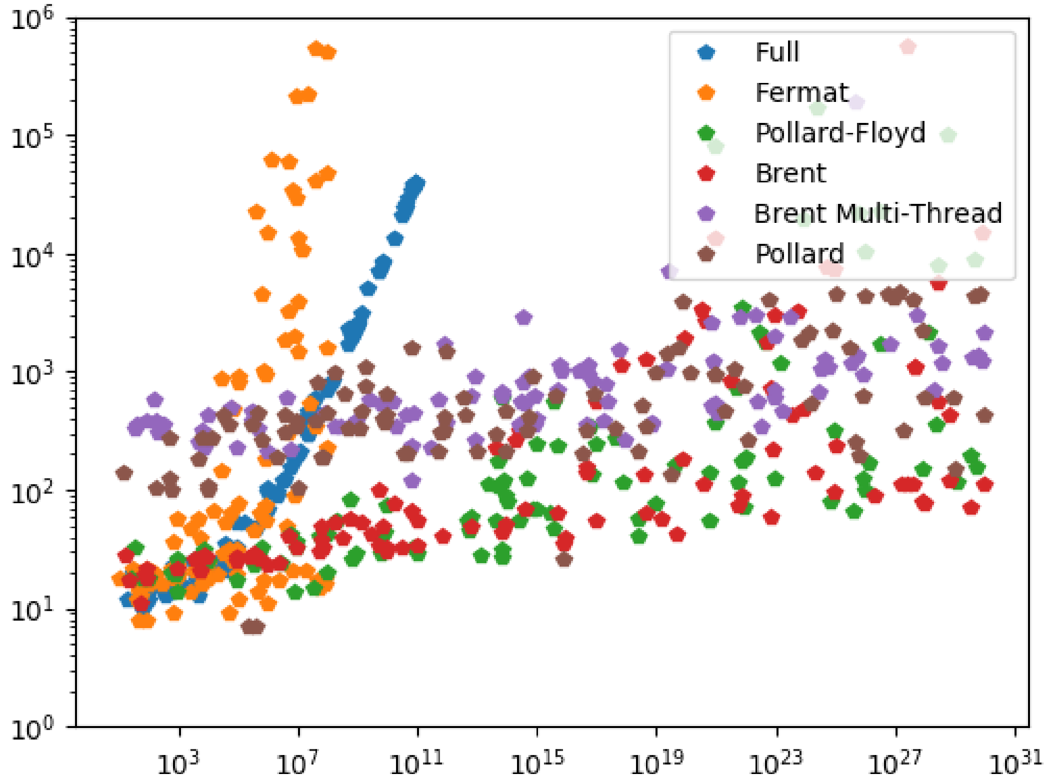
```
        }
        x = func(x) % number;
        mpz_class diff = z - x;
        diff = (diff >= 0 ? diff : diff*-1);
        p = gcd(number, diff % number);
    }
    if (number == p) {
        return ;
    }
    if (stopper.compare_exchange_weak(stop, 1)) {
        result = p;
    }
}
```

Запускается с помощью обертки:

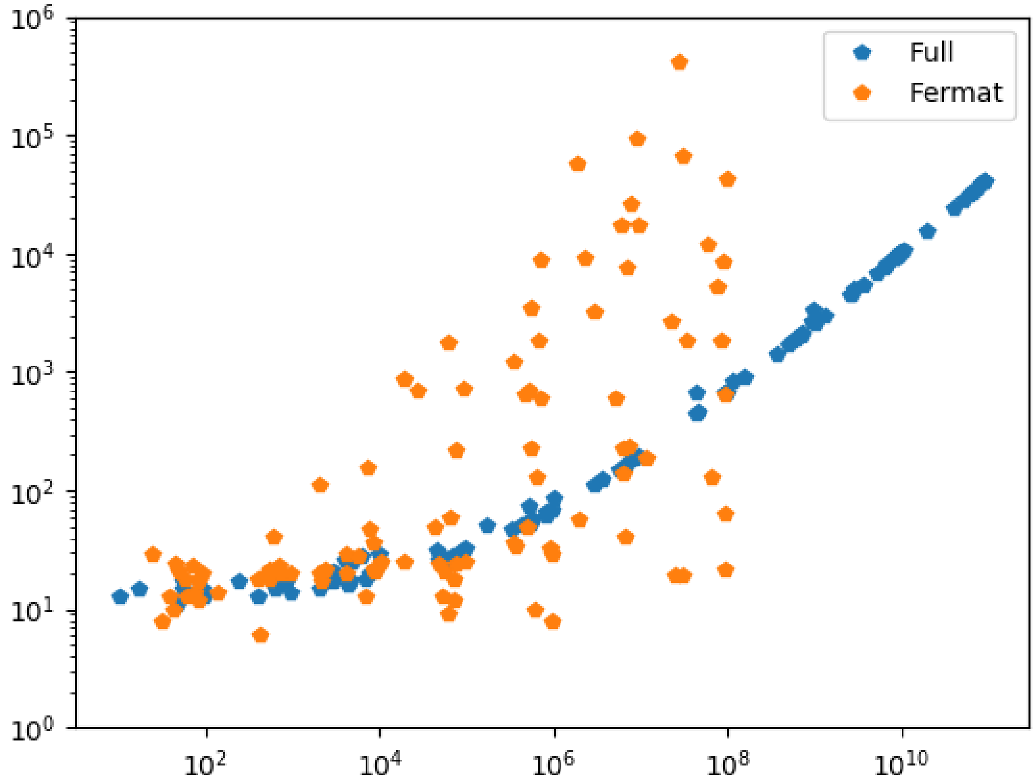
```
mpz_class find_one_mt(mpz_class number) {
    mpz_class result = 0;
    std::atomic_char stopper;
    stopper.exchange(0);
    std::vector<std::thread> threads;
    for (auto& func : functions) {
        if (threads.size() >= std::thread::hardware_concurrency() ||
        stopper.load()) {
            break ;
        }
        threads.emplace_back([&number, &result , &stopper , &func](){
            find_one_brent(number, func , stopper , result );
        });
    }
    for (auto& thread : threads) {
        thread.join();
    }
    if (!result) {
        result = number;
    }
    return result;
```

Результаты

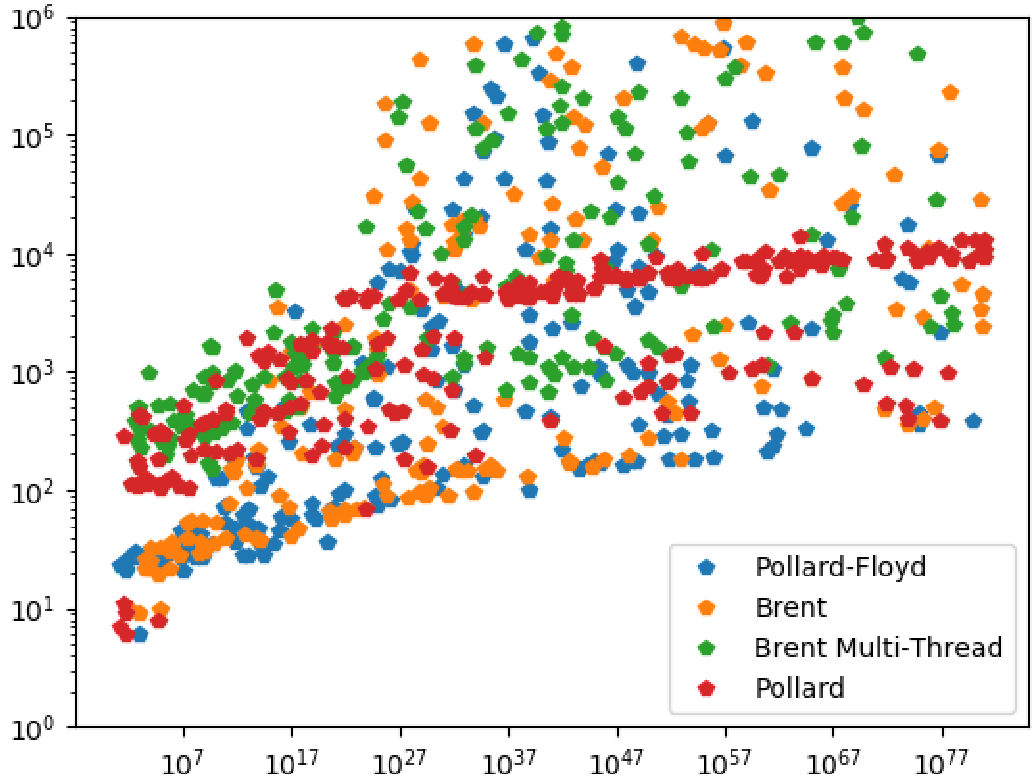
Как видно из общего графика, полный перебор и метод Ферма лучше рассматривать отдельно.



Полный перебор приблизительно равен по скорости методу Ферма. Максимальные значения чисел для факторизации 10^{10}.



Остальные методы могут факторизовать числа до 10^{77}. Явного лидера - нет.



Многопоточный метод Брента уступает однопоточному. Возможно его ускорение за счет создания потоков до факторизации.