# Обработка исключений

# Актуальность

80% кода программ связаны с обработкой ошибок;
Код = время + деньги.

Механизм исключений позволяет:
- типизировать ошибку, дополнить ее сообщением;
- доставить ошибку до обработчика;
- использовать полиморфизм при обработке.

# Исключения снаружи и внутри

```
void thrower() {
    if (rand() % 2)
        throw Exception(99);
    // useful_action();
}

int main()
{
    try {
        thrower();
    }
    catch(Exception&) {
        //...
    }
}
```

```
6   thrower():
7           pushq   %rbx
8           subq    $32, %rsp
9           call    rand
10          testb   $1, %al
11          jne     .L5
12          addq    $32, %rsp
13          popq    %rbx
14          ret
15  .L5:
16          movl    $4, %ecx
17          call    __cxa_allocate_exception
18          movq    %rax, %rbx
19          movl    $99, %edx
20          movq    %rax, %rcx
21          call    Exception::Exception(int) [complete
22          leaq    Exception::~Exception() [complete o
23          leaq    typeinfo for Exception(%rip), %rdx
24          movq    %rbx, %rcx
25          call    __cxa_throw
26          nop
27  main:
28          subq    $40, %rsp
29          call    __main
30          call    thrower()
```

# Обработка ошибок без исключений

```
int foo(Error& err) {
    if (err.get()) return 0;
    if (rand() % 2) {
        err.set(1);
        return 0;
    }
    return 99; // <--- useful
}

int bar(Error& err) {
    if (err.get()) return 0;
    return foo(err); // <--- useful
}

int main() {
    Error err{ 0 };
    bar(err);

    if (!err.get()) { /* useful */ }
    else if (err.get() == 1) { }
}
```

```
4   foo(Error&):
5           pushq   %rbx
6           subq    $32, %rsp
7           movl    (%rcx), %eax
8           movq    %rcx, %rbx
9           testl   %eax, %eax
10          je      .L8
            L3: ..., L4: ...
17  .L8:
18          call    rand
19          movl    %eax, %edx
20          movl    $99, %eax
21          andl    $1, %edx
22          je      .L3
23          movl    $1, (%rbx)
24          jmp     .L4
25  bar(Error&):
26          movl    (%rcx), %eax
27          testl   %eax, %eax
28          je      .L11
29          xorl    %eax, %eax
30          ret
31  .L11:
32          jmp     foo(Error&)
33  main:
            ...
41          movq    %rbx, %rcx
42          call    bar(Error&)
```

Некоторые минусы:

- проверка на каждом уровне вызовов;
- повышается сложность кода, его становиться больше;
- нужно «думать» о возвращаемом значении.

# Предпосылки

В библиотеке Qt до 4.0 не использовались исключения, не используются они в проекте ЯндексБраузер.

… в докладе … г. идет речь о том, что механизм исключений влияет на оптимизацию кода.

Исключения нулевой стоимости.

# Задачи

1. Исследовать влияние исключений на оптимизацию кода компилятором;
2. Оценить издержки механизма исключений по сравнению с возвратом кода ошибки на x64 архитектуре;
3. Получить данные о влиянии исключений на параллелизм программы.

# 1. Исследовать влияние исключений на оптимизацию кода компилятором

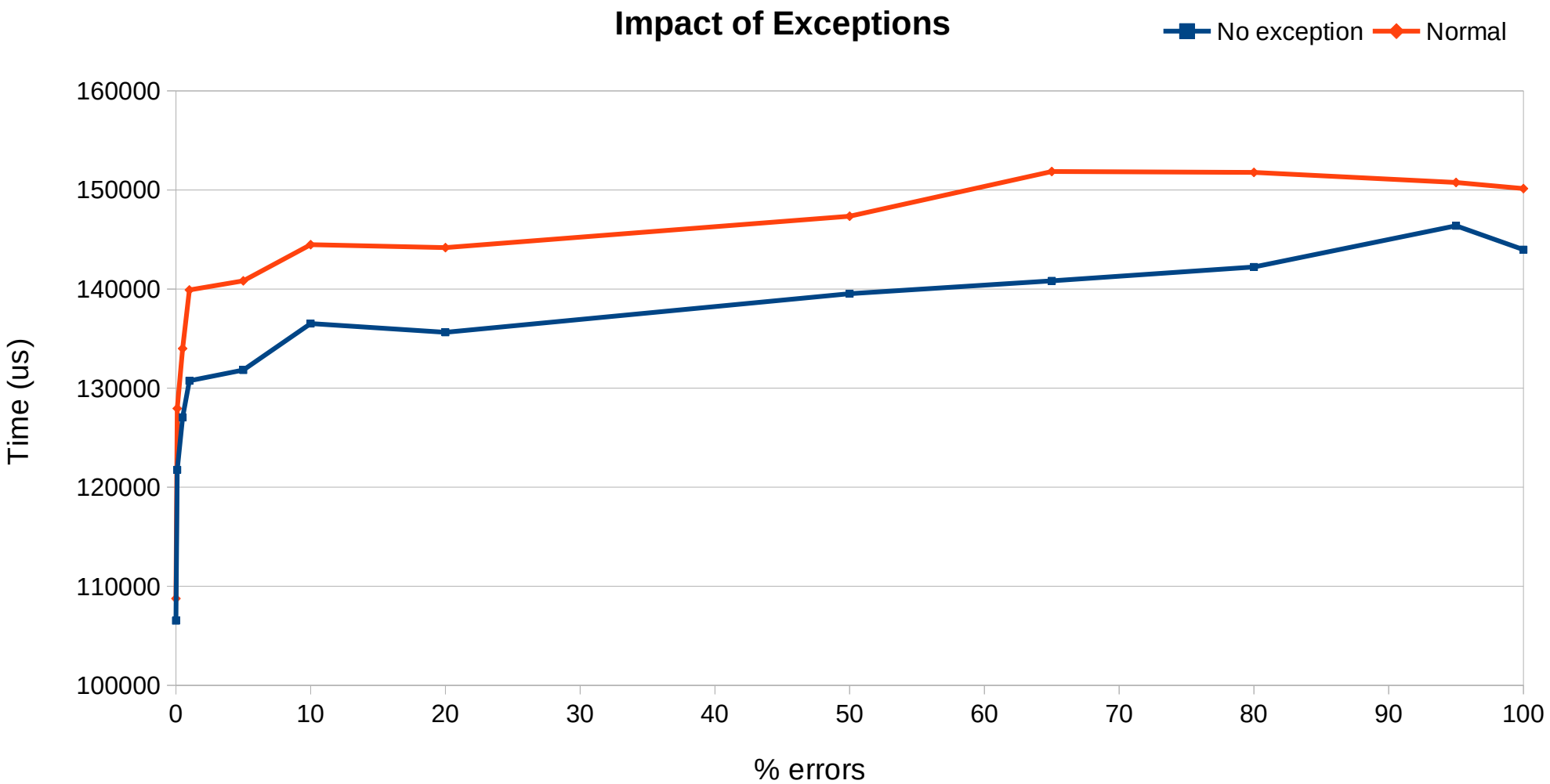
Пример — программа решения квадратных уравнений. Ошибкой считается случай, когда a=b=c=0.

Написано 2 версии программы — с возвратом кода ошибки и с исключениями. На вход поданы такие данные, что ошибка никогда не произойдет.

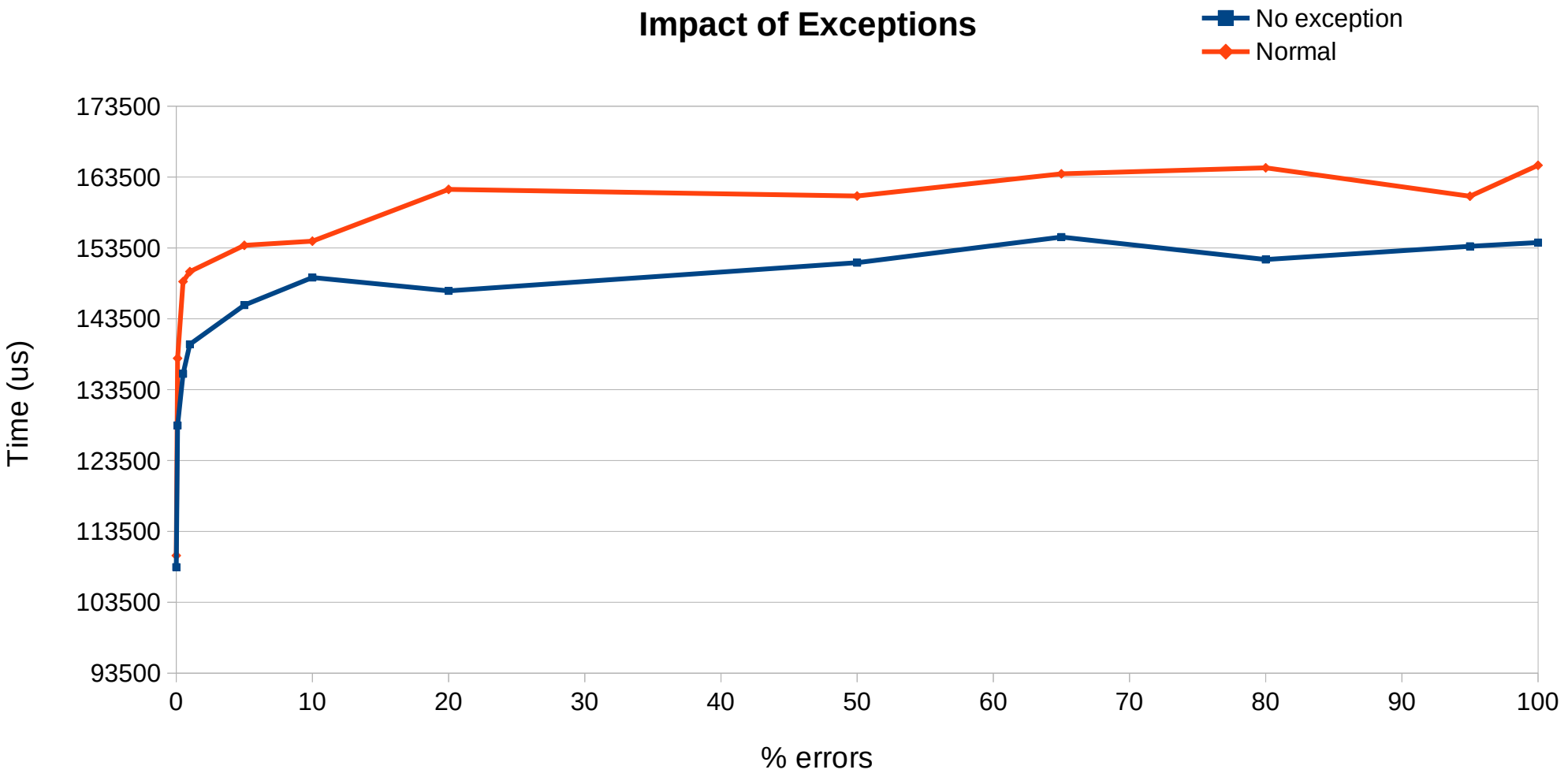
2. Оценить издержки механизма исключений по сравнению с возвратом кода ошибки на x64 архитектуре

На вход программе подаются данные, приводящие к ошибкам с различной фиксированной вероятностью.

# MinGW 11.2.0


Impact of Exceptions
No exception
Normal
160000
150000
140000
130000
120000
110000
100000
Time (us)
0
10
20
30
40
50
60
70
80
90
100
% errors

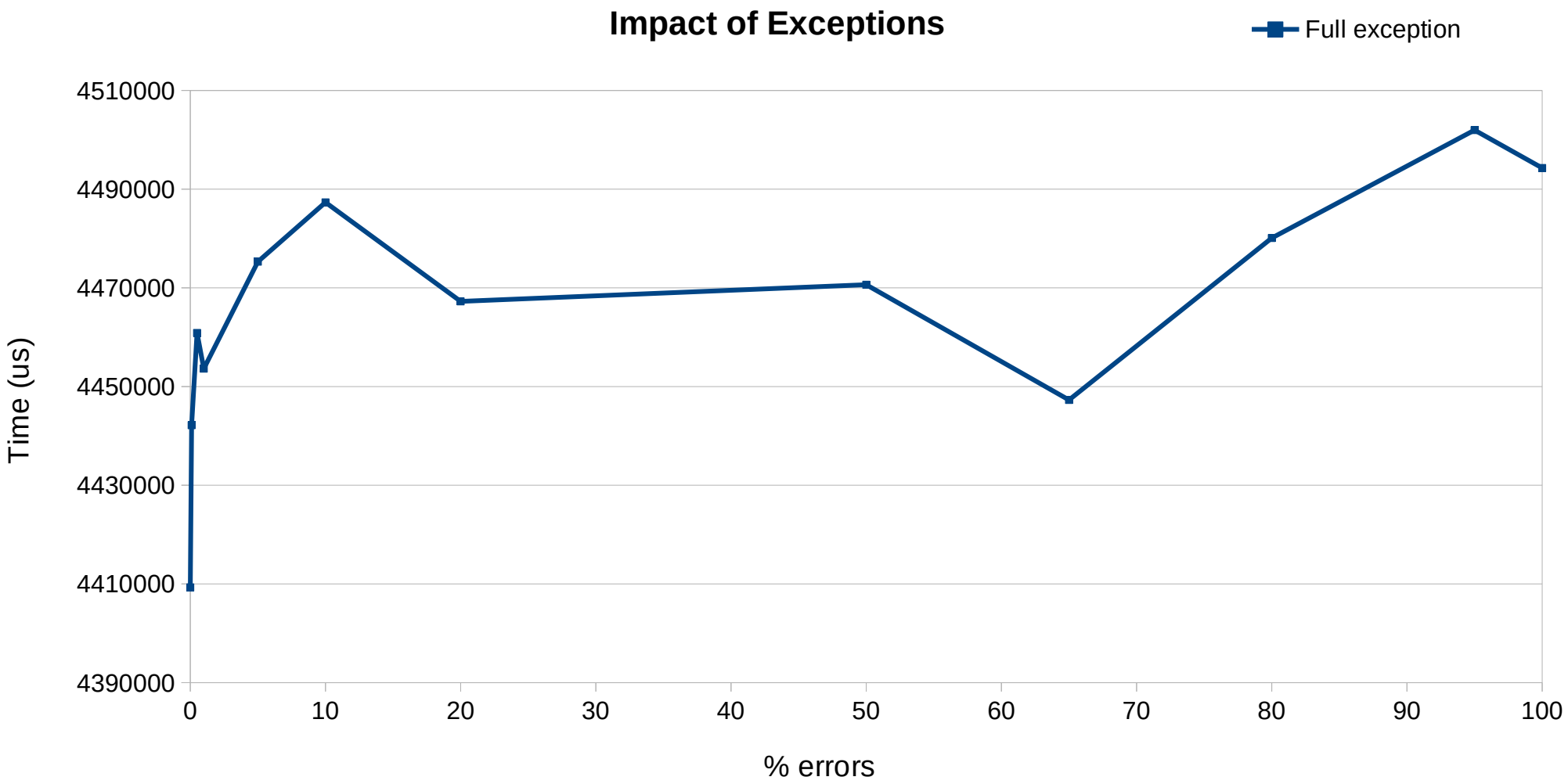
# MSVC 17


Impact of Exceptions
No exception
Normal
173500
163500
153500
143500
133500
123500
113500
103500
93500
Time (us)
0
10
20
30
40
50
60
70
80
90
100
% errors

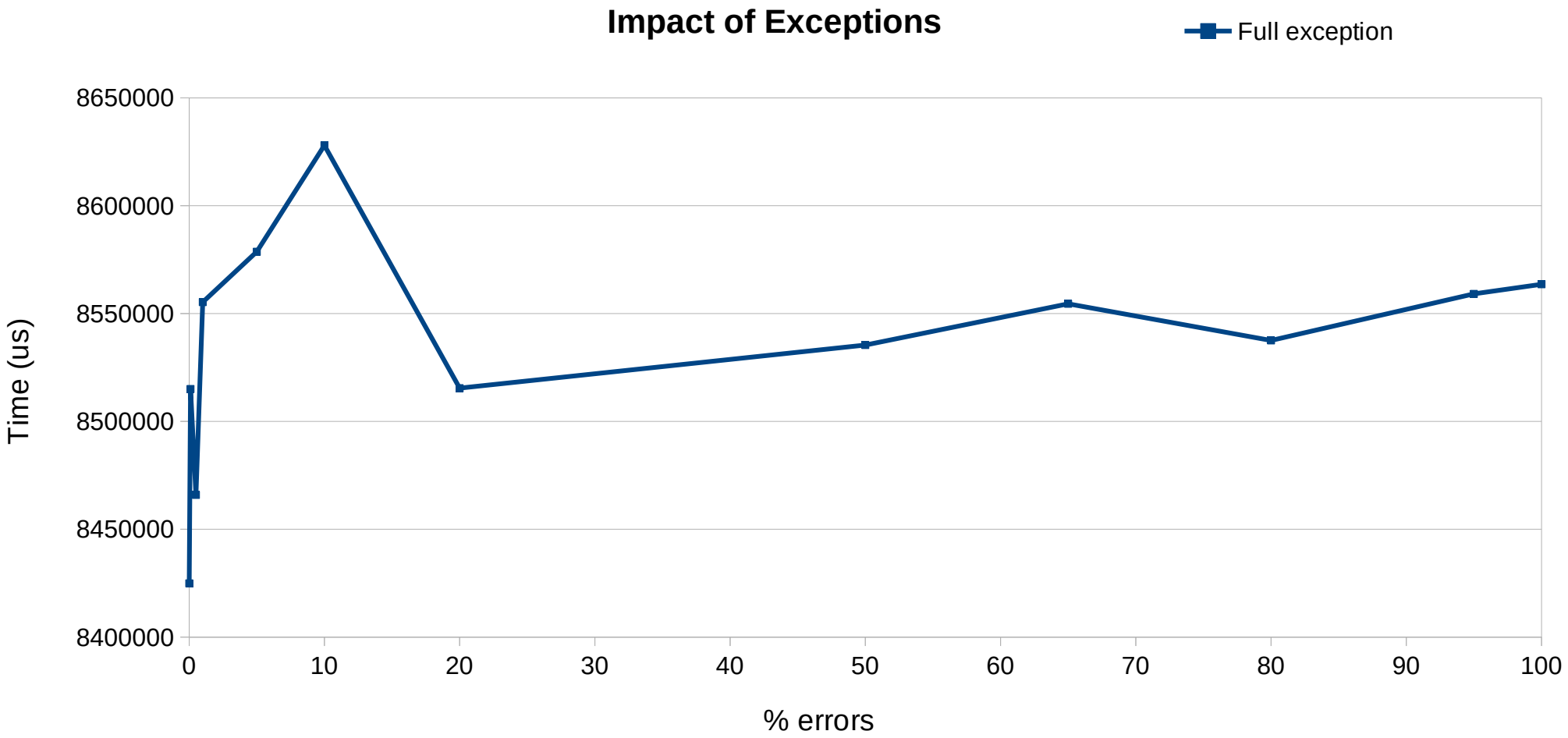
2. Оценить издержки механизма исключений по сравнению с возвратом кода ошибки на x64 архитектуре

Исключения — альтернативный способ вернуть результат из функции. Позволяет вернуть значения любого типа.

# MinGW 11.2.0


Impact of Exceptions
Full exception
4510000
4490000
4470000
4450000
4430000
4410000
4390000
Time (us)
0
10
20
30
40
50
60
70
80
90
100
% errors

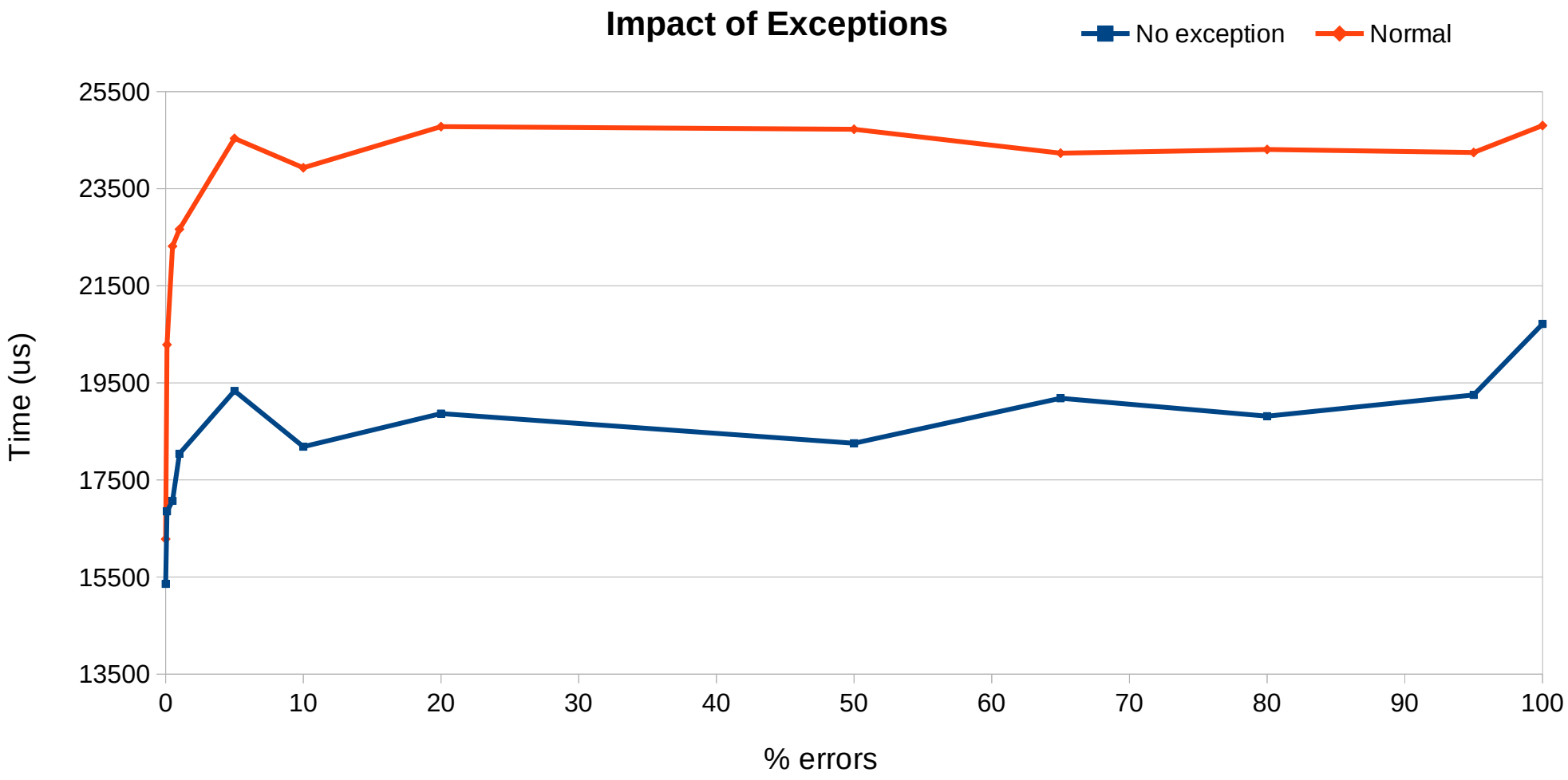
# MSVC 17


Impact of Exceptions
Full exception
Time (us)
8650000
8600000
8550000
8500000
8450000
8400000
0
10
20
30
40
50
60
70
80
90
100
% errors

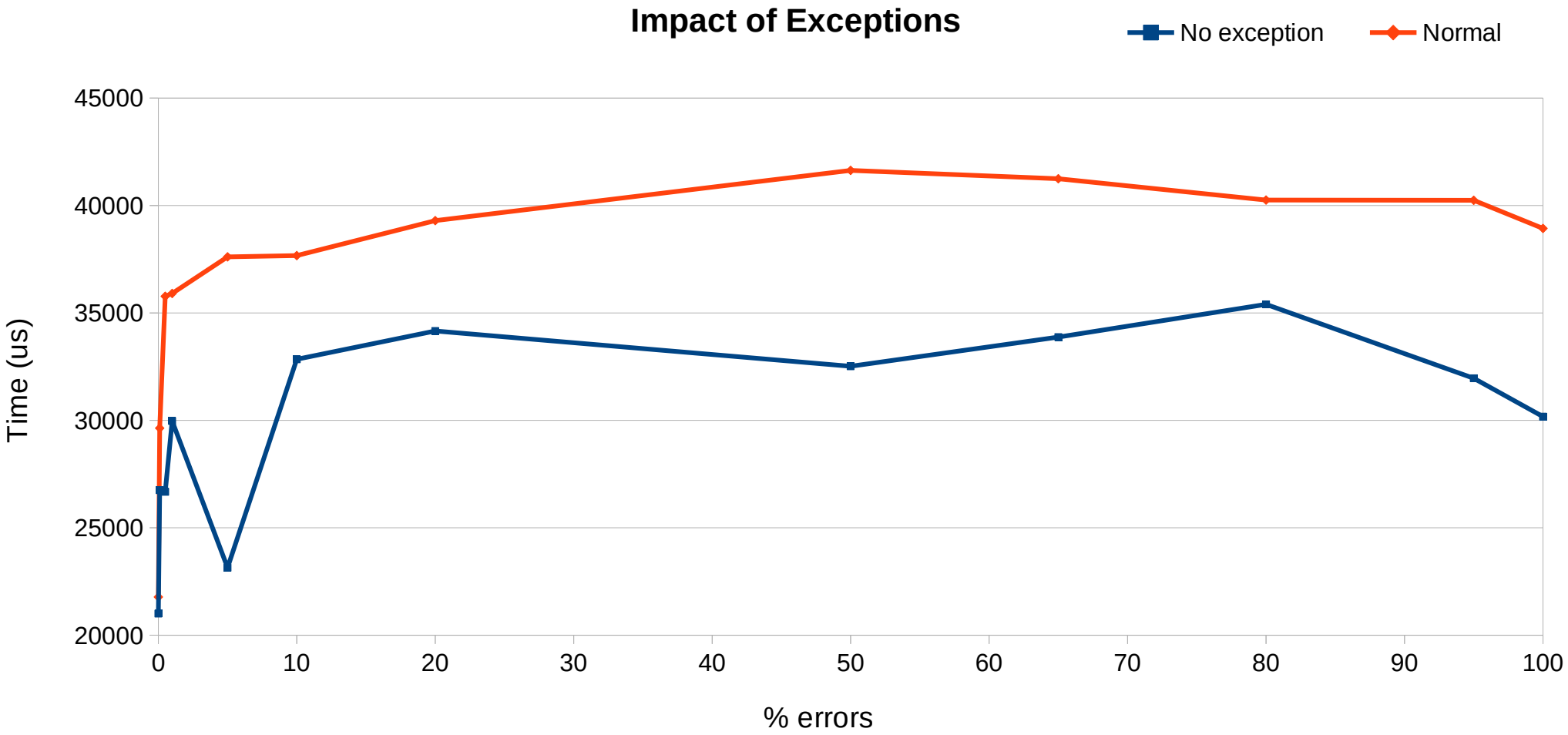
# 3. Получить данные о влиянии исключений на параллелизм программы

Каждая из трех версий программ распараллелена средствами OpenMP.

# MinGW 11.2.0


Impact of Exceptions
No exception
Normal
25500
23500
21500
19500
17500
15500
13500
Time (us)
0
10
20
30
40
50
60
70
80
90
100
% errors

# MSVC 17


Impact of Exceptions
No exception
Normal
45000
40000
35000
30000
25000
20000
Time (us)
0
10
20
30
40
50
60
70
80
90
100
% errors

# Выводы

Современные реализации C++ сокращают накладные расходы. Использование исключений составляет несколько процентов xxx%, и это по сравнению с отсутствием обработки ошибок.

*Написание кода с кодами возврата ошибок и тестами тоже не бесплатно.*

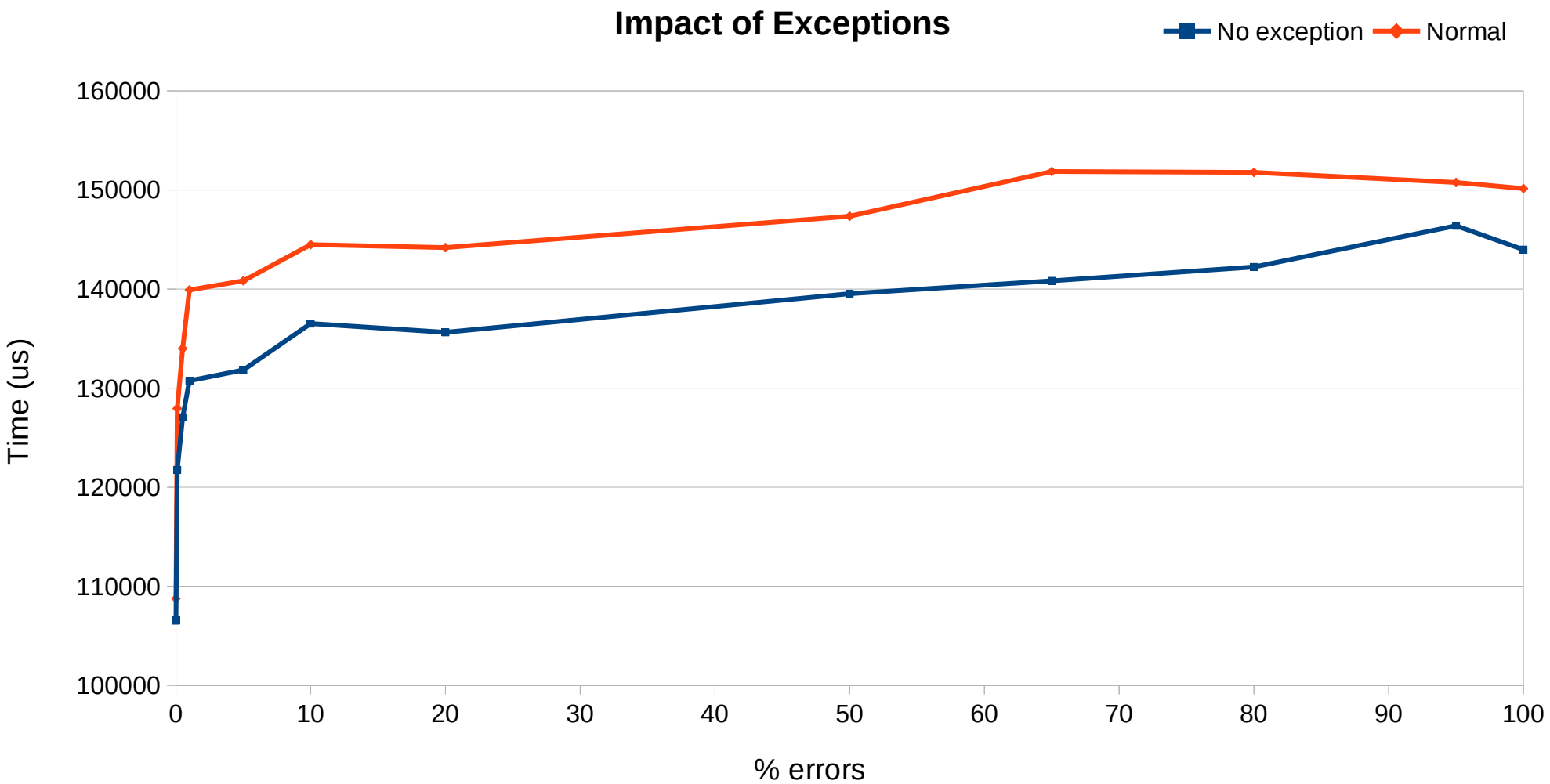
Как правило, обработка исключений обходится очень дешево, если вы не генерируете исключение.

В некоторых реализациях это ничего не стоит. Все затраты возникают, когда вы генерируете исключение: то есть «нормальный код» работает быстрее, чем код, использующий коды возврата ошибок и тесты.

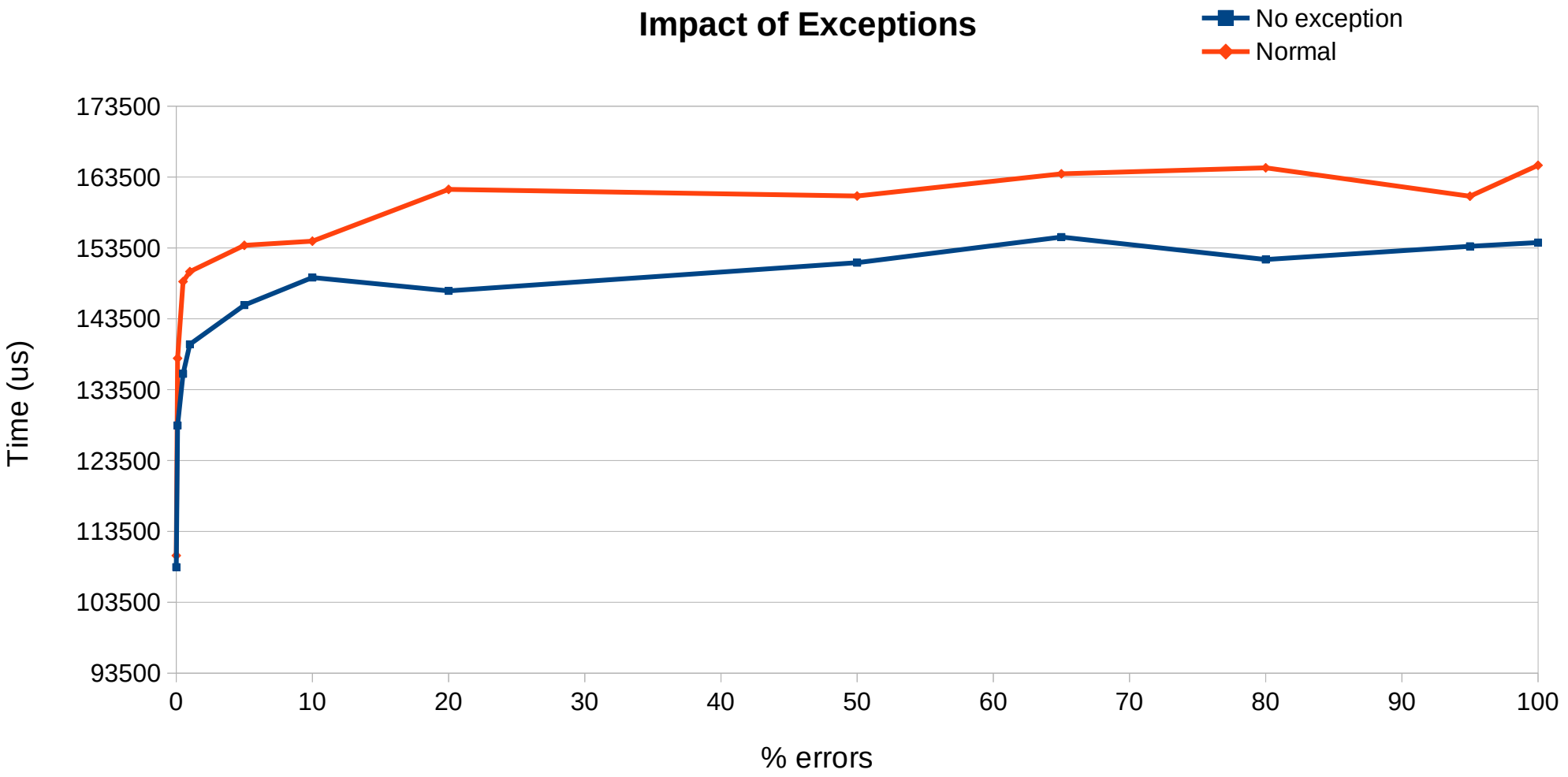
# Выводы

- Возвращать результат с помощью исключений точно не стоит;

# MinGW 11.2.0


Impact of Exceptions
No exception
Normal
160000
150000
140000
130000
120000
110000
100000
Time (us)
0
10
20
30
40
50
60
70
80
90
100
% errors

# MSVC 17


Impact of Exceptions
No exception
Normal
173500
163500
153500
143500
133500
123500
113500
103500
93500
Time (us)
0
10
20
30
40
50
60
70
80
90
100
% errors