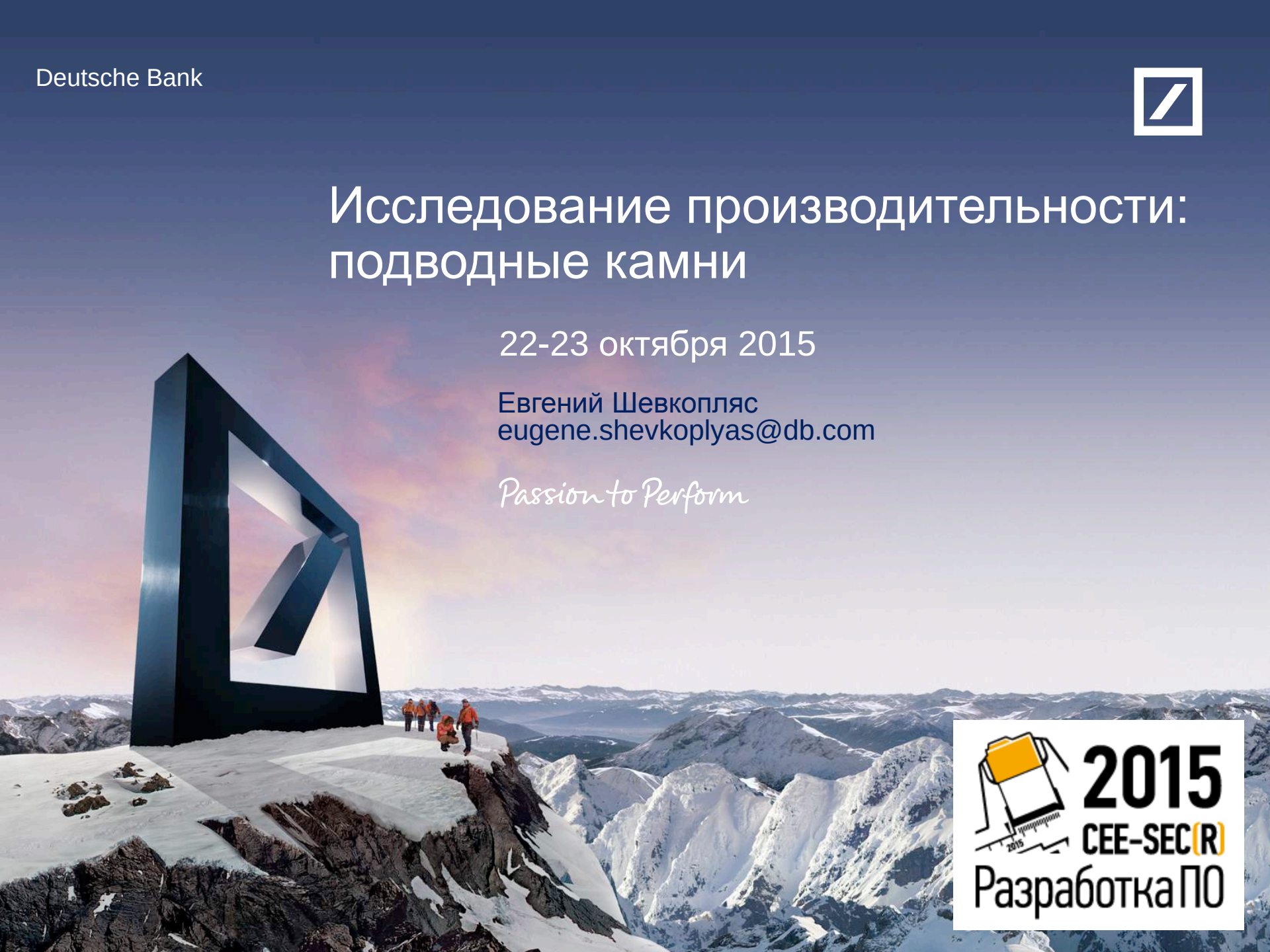
Deutsche Bank



Исследование производительности: подводные камни

22-23 октября 2015

Евгений Шевкопляс
eugene.shevkoplyas@db.com

Passion to Perform

2015 CEE-SEC(R) Разработка ПО

Самая надёжная проверка производительности алгоритма – тест!

В теории нет разницы
между теорией и практикой.
А на практике есть.

Йоги Берра

Обычный тест на производительность.

```
struct Event
{
    char data[1024];
    char * extra = nullptr;
};

template <typename EventT>
void set_event(EventT & event, const char * msg)
{
    size_t size = strlen(msg) + 1;
    void * data = event.data;
    if (size > sizeof(event.data)) {
        event.extra = (char*)malloc(size);
        data = event.extra;
    }
    memcpy(data, msg, size);
}
```

```
int main(int argc, char ** argv)
{
    size_t N = 1000000;
    std::string msg(100, 'a');
    Event * event = new Event();

    CHECKPOINT_START();
    for (size_t i = 0; i < N; ++i) {
        set_event(*event, msg.c_str());
    }
    CHECKPOINT("set event");

    PRINT_CHECKPOINT_SUMMARY() ;
    return 0;
}
```

Время: 15.3 msec

Как делать замеры времени?

— Time stamp counter

```
uint64_t rdtsc()
{
    uint32_t a,d;
    asm volatile("rdtsc" : "=a" (a), "=d" (d));
    return ((uint64_t)a) | (((uint64_t)d)<<32;
}
```

Время: 7 nsec (24)

— Смотрим на распределение времён (не на среднее)

Избегаем переиспользования памяти

```
int main(int argc, char ** argv)
{
    std::string msg(100, 'a');
    size_t N = 1000000;

    CHECKPOINT_START();
    Event * event = new Event();
    CHECKPOINT("allocate");
    for (size_t i = 0; i < N; ++i) {
        set_event(*event, msg.c_str());
    }
    CHECKPOINT("set to one event");
    std::vector<Event> events(N);
    CHECKPOINT("allocate");
    for (size_t i = 0; i < N; ++i) {
        set_event(events[i], msg.c_str());
    }
    CHECKPOINT("set to different events");

    PRINT_CHECKPOINT_SUMMARY() ;
    return 0;
}
```

Избегаем переиспользования памяти

```
int main(int argc, char ** argv)
{
    std::string msg(100, 'a');
    size_t N = 1000000;

    CHECKPOINT_START();
    Event * event = new Event();
    CHECKPOINT("allocate");
    for (size_t i = 0; i < N; ++i) {
        set_event(*event, msg.c_str());
    }
    CHECKPOINT("set to one event");
    std::vector<Event> events(N);
    CHECKPOINT("allocate");
    for (size_t i = 0; i < N; ++i) {
        set_event(events[i], msg.c_str());
    }
    CHECKPOINT("set to different events");

    PRINT_CHECKPOINT_SUMMARY() ;
    return 0;
}
```

allocate: 0.301 us
set to one event: **15.3** ms
allocate: 395 ms
set to different events: **69** ms

Callgrind statistics

Один объект:

```
      Ir              Dr          Dw     I1mr     D1mr D1mw ILmr   DLmr DLmw
       .               .           .        .        .    .    .      .    .  void set_event(EventT & event, const char * msg)
       .               .           .        .        .    .    .      .    .  {
4,000,004 1,000,002 1,000,002      1        2       1    1       2    1     size_t size = strlen(msg) + 1;
1,000,000           .           .        .        .    .    .      .    .     void * data = event.data;
2,000,000           .           .        .        .    .    .      .    .     if (size > sizeof(event.data)) {
       .            .           .        .        .    .    .      .    .        event.extra = (char*)malloc(size);
       .            .           .        .        .    .    .      .    .        data = event.extra;
       .            .           .        .        .    .    .      .    .     }
5,000,004  1,000,002   1,000,002   1        1       0    1    1       .     memcpy(data, msg, size);

70,000,000 16,000,00015,000,000  5        0       0    5    .       .      => ???:memcpy (1000000x)
.       .       .   .   .   .   .   .   . }
```

```
> valgrind --tool=callgrind --simulate-cache=yes <app>

> callgrind_annotate --auto=yes ./callgrind.out.<pid>
```

Callgrind statistics

Один объект:

```
      Ir             Dr          Dw     I1mr    D1mr D1mw ILmr   DLmr DLmw
       .              .           .       .       .    .    .      .    .  void set_event(EventT & event, const char * msg)
       .              .           .       .       .    .    .      .    .  {
4,000,004 1,000,002 1,000,002      1       2       1    1      2    1     size_t size = strlen(msg) + 1;
1,000,000          .           .       .       .    .    .      .    .     void * data = event.data;
2,000,000          .           .       .       .    .    .      .    .     if (size > sizeof(event.data)) {
       .           .           .       .       .    .    .      .    .        event.extra = (char*)malloc(size);
       .           .           .       .       .    .    .      .    .        data = event.extra;
       .           .           .       .       .    .    .      .    .     }
5,000,004  1,000,002   1,000,002      1       1    0    1      1    .     memcpy(data, msg, size);

70,000,000 16,000,00015,000,000  5       0       0    5      .    .      => ???:memcpy (1000000x)
.       .       .   .   .   .   .   .   . }
```

Много объектов:

```
      Ir             Dr          Dw     I1mr    D1mr D1mw ILmr   DLmr DLmw

70,000,000 16,000,00015,000,000  5       0   2,500,000  5      0   2,500,000  => ???:memcpy (1000000x)
```

Большие и маленькие сообщения

```
int main(int argc, char ** argv)
{
    std::string msg(50, 'a');
    size_t N = 1000000;

    CHECKPOINT_START();
    Event * event = new Event();
    CHECKPOINT("allocate");
    for (size_t i = 0; i < N; ++i) {
        set_event(*event, msg.c_str());
    }
    CHECKPOINT("set to one event");
    std::vector<Event> events(N);
    CHECKPOINT("allocate");
    for (size_t i = 0; i < N; ++i) {
        set_event(events[i], msg.c_str());
    }
    CHECKPOINT("set to different events");

    PRINT_CHECKPOINT_SUMMARY() ;
    return 0;
}
```

Message Size: **100**
set to one event: **15.3** ms
set to different events: **69** ms

Большие и маленькие сообщения

```
int main(int argc, char ** argv)
{
    std::string msg(50, 'a');
    size_t N = 1000000;

    CHECKPOINT_START();
    Event * event = new Event();
    CHECKPOINT("allocate");
    for (size_t i = 0; i < N; ++i) {
        set_event(*event, msg.c_str());
    }
    CHECKPOINT("set to one event");
    std::vector<Event> events(N);
    CHECKPOINT("allocate");
    for (size_t i = 0; i < N; ++i) {
        set_event(events[i], msg.c_str());
    }
    CHECKPOINT("set to different events");

    PRINT_CHECKPOINT_SUMMARY() ;
    return 0;
}
```

Message Size: **100**
set to one event: **15.3** ms
set to different events: **69** ms

Message Size: **50**
set to one event: **11.6** ms
set to different events: **66.4** ms

Little-event vs Big-event

```
struct LittleEvent
{
    char data[1024];
    char * extra = nullptr;
};
```

```
struct BigEvent
{
    char data[10240];
    char * extra = nullptr;
};
```

1. Little Event лучше для маленьких сообщений

```
std::vector<LittleEvent> little_events(N);
CHECKPOINT("little events allocate");
for (size_t i = 0; i < N; ++i) {
    set_event(little_events[i], message.c_str());
}
CHECKPOINT("little event");

std::vector<BigEvent> big_events(N);
CHECKPOINT("big events allocate");
for (size_t i = 0; i < N; ++i) {
    set_event(big_events[i], message.c_str());
}
CHECKPOINT("big event");
```

little events allocate: 389 ms
little event: **68** ms

big events allocate: 3.55 sec
big event: **100** ms

2. Big Event лучше для больших сообщений

```
std::string big_msg(5000, 'b');

for (size_t i = 0; i < N; ++i) {
    set_event(big_events[i], big_msg.c_str());
}
CHECKPOINT("big event");

for (size_t i = 0; i < N; ++i) {
    set_event(little_events[i], big_msg.c_str());
}
CHECKPOINT("little event");
```

big event: **977** ms

little event: **2.33 sec**

3. Little Event лучше и для больших сообщений!

```
std::string big_msg(5000, 'b');

for (size_t i = 0; i < N; ++i) {
    set_event(big_events[i], big_msg.c_str());
}
CHECKPOINT("big event");

for (size_t i = 0; i < N; ++i) {
    set_event(little_events[i], big_msg.c_str());
    if (i > 1000) {
        free(little_events[I - 1000].extra);
    }
}
CHECKPOINT("little event && free");
```

big event: **977** ms

little event: **2.33 sec**

little event && free: **537** ms

4. Сторонние операции между итерациями

```
void clear_cache(char * buf, int size, uint64_t ticks)
{
    snprintf(buf, size, "echo %lu > /dev/null", ticks);
    system(buf);
    for(int x=0; x<size; x+=size/64) {
        buf[x] = buf[size-x]+1 + ticks;
    }
}

int main(int argc, char ** argv)
{
    const int size = 20*1024*1024; // 20MB
    char *buf = (char *)malloc(size);
    // … init
    size_t N = 20000; uint64_t ticks = 0;
    for (size_t i = 0; i < N; ++i) {
        uint64_t t1 = RDTSC::get_tsc();
        set_event(events[i],  big_msg.c_str());
        ticks += RDTSC::get_tsc() – t1;
        clear_cache(buf, size, ticks);
    }
    print(“big event:" << msec(ticks) << " ms“);
```

big event: **72** ms

little event: **86** ms

Выигрыш: **14** ms

5. Общее время

```
size_t N = 20000;
CHECKPOINT_START();

std::vector<BigEvent> events(N);
CHECKPOINT("allocate");

uint64_t ticks = 0;
for (size_t i = 0; i < N; ++i) {
    uint64_t t1 = RDTSC::get_tsc();
    set_event(events[i],  big_msg.c_str());
    ticks += RDTSC::get_tsc() – t1;
    clear_cache(buf, size, ticks);
}
CHECKPOINT(“big event total");

print(“big event:" << msec(ticks) << " ms“);
```

big event: **72** ms

little event: **86** ms

Выигрыш: **14** ms

5. Общее время

```
size_t N = 20000;
CHECKPOINT_START();

std::vector<BigEvent> events(N);
CHECKPOINT("allocate");

uint64_t ticks = 0;
for (size_t i = 0; i < N; ++i) {
    uint64_t t1 = RDTSC::get_tsc();
    set_event(events[i],  big_msg.c_str());
    ticks += RDTSC::get_tsc() – t1;
    clear_cache(buf, size, ticks);
}
CHECKPOINT(“big event total");

print(“big event:" << msec(ticks) << " ms“);
```

big event: **72** ms

little event: **86** ms

Выигрыш: **14 ms**

big event total: **97.2 sec**

little event total: **33.1 sec**

Проигрыш: **1 минута**

Если вы думаете, что понимаете квантовую механику, значит вы её не понимаете

Ричард Фейнман

Строковое представление дробной части числа

value = 12 300 factor = 1 000 000

“0.0123”

Строковое представление дробной части числа

```
void convert(char * buffer, size_t & pos, uint64_t value, uint64_t factor)
{
    buffer[pos++] = '0';
    uint64_t fraction = value % factor;
    if (fraction > 0) {
       buffer[pos++] = '.';
       do {
          unsigned index = 0;
          if (factor >= 100) { factor /= 100; index = (fraction / factor) * 2; fraction %= factor; }
          else if (factor == 10) { index = fraction * 20;  fraction = 0; }
          else { index = fraction * 2; fraction = 0; }

         if (fraction > 0) {
             *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
             pos += 2;
          } else if (digits[index + 1] != '0') {
             *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
             pos += 2;
          } else {
             buffer[pos++] = digits[index];
          } } while (fraction > 0);
   }
}
```

Тест на производительность конвертации

```
char buffer[64];
size_t pos = 0;

CHECKPOINT_START();

for (uint64_t i = 0; i < 1000000; ++i) {
    pos = 0;
    convert(buffer, pos, i, 1000000);
}
CHECKPOINT("fraction");
```

fraction: **10.1** ms

Строковое представление дробной части числа

```
void convert(char * buffer, size_t & pos, uint64_t value, uint64_t factor)
{
    buffer[pos++] = '0';
    uint64_t fraction = value % factor;
    if (fraction > 0) {
       buffer[pos++] = '.';
       do {
          unsigned index = 0;
          if (factor >= 100) { factor /= 100; index = (fraction / factor) * 2; fraction %= factor; }
          else if (factor == 10) { index = fraction * 20;  fraction = 0; }
          else { index = fraction * 2; fraction = 0; }

         if (fraction > 0) {
             *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
             pos += 2;
          } else if (digits[index + 1] != '0') {
             *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
             pos += 2;
          } else {
             buffer[pos++] = digits[index];
          } } while (fraction > 0);
    }
}
```

Две одинаковые реализации

```
if (fraction > 0) {
    *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
    pos += 2;
 } else if (digits[index + 1] != '0') {
    *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
    pos += 2;
 } else {
    buffer[pos++] = digits[index];
 }
```

Две одинаковые реализации

```
if (fraction > 0) {
   *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
   pos += 2;
 } else if (digits[index + 1] != '0') {
   *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
   pos += 2;
 } else {
   buffer[pos++] = digits[index];
 }
```

```
 if ((fraction > 0) || (digits[index + 1] != '0')) {
    *(int16_t *)&buffer[pos] = *(int16_t*)&digits[index];
     pos += 2;
 } else {
    buffer[pos++] = digits[index];
 }
```

Две одинаковые реализации

```
if (fraction > 0) {
   *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
   pos += 2;
 } else if (digits[index + 1] != '0') {
   *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
   pos += 2;
 } else {
   buffer[pos++] = digits[index];
 }
```

fraction: **10.1** ms

```
 if ((fraction > 0) || (digits[index + 1] != '0')) {
    *(int16_t *)&buffer[pos] = *(int16_t*)&digits[index];
     pos += 2;
 } else {
    buffer[pos++] = digits[index];
 }
```

Две одинаковые реализации

```
if (fraction > 0) {
   *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
   pos += 2;
 } else if (digits[index + 1] != '0') {
   *(int16_t *)&buffer[pos] = *(int16_t *)&digits[index];
   pos += 2;
 } else {
   buffer[pos++] = digits[index];
 }
```

fraction: **10.1** ms

```
 if ((fraction > 0) || (digits[index + 1] != '0')) {
    *(int16_t *)&buffer[pos] = *(int16_t*)&digits[index];
     pos += 2;
  } else {
    buffer[pos++] = digits[index];
  }
```

fraction: **30.7** ms

Спасибо!

Q&A

eugene.shevkoplyas@db.com