Benchmarking C++
From video games to algorithmic trading

Alexander Radchenko
Quiz. How long it takes to run?

- 3.5GHz Xeon at CentOS 7

```cpp
std::string MakeFizzBuzz(uint64_t number)
{
    std::string output;
    if (number % 3 == 0)
    {
        output += "Fizz";
    }
    if (number % 5 == 0)
    {
        output += "Buzz";
    }
    if (output.empty())
    {
        output = std::to_string(number);
    }
    return output;
}
```

- Write your name
- Write your guess as a single number
- Write time units clearly
- Answers will be collected in the next 5 minutes
Outline

• Performance challenges in games
• How games tackle performance
• Performance challenges in trading
• How trading tackles performance
• Lightweight tracing use case
My background

• Game development for 15 years
• 3D graphics programming and optimisation
• Shipped 8 titles on various platforms
  – PS2, PS3, Xbox 360, Wii, iOS, Android, PC
• 3 years @ Optiver
  – Low latency trading systems
• Performance matters in both domains
Why performance matters?

• Slow running game is no fun to play
  – Guess what’s the second most common complaint about any PC game?
• Slow trading system is not making money
  – In fact, it might lose your money
Games

- Soft real-time systems
- Performance is important
- Normally run at 30 frames per second
- Consistent CPU/GPU load
- Occasional spikes
- Throughput is the king
Game loop

• Performance as a currency
  – Graphics
  – Animations
  – Physics

![Game loop diagram]
Performance challenges in games

- PC and Mobiles
  - Fragmented HW
- Game consoles
  - Fixed HW 😊
  - They are cheap for a reason 😞
  - Proprietary tools and devkits
Performance challenges in games

If you got pedal with your PS3 devkit you can use it for Vim:

alevchuk/vim-clutch
A hardware pedal for improved text editing in Vim. Contribute to vim-clutch development by creating an account on GitHub.
github.com
How games tackle performance

- Reference game levels
- Custom profilers
  - Whole game session
  - Single frame
World of Tanks

• Online MMO shooter
• Fragmented platform
• Wide range of HW
  – Old laptops
  – High-end desktops
  – Everything in between
Replays

- Record incoming network traffic
- Initially created to repro bugs
- Very useful tool for performance testing
- At some point released to the public
Replays: problems

• Protocol upgrades
• Game map changes may invalidate replay
• Security

Vulnerability Within Replay Files

A couple of months ago I was rooting around within the WoT replay files and their format. I discovered that they way they stored data within certain packets in the replays made it extremely easy to get code execution. After a couple of days working at reliable execution I came upon a reliable way to take any replay file and inject code to execute. This happens very quickly after opening the infected replay file with no way to prevent it once WoT begins reading from the replay.
Regression testing and replays

- Avoiding performance degradation
- Categorize HW: low, medium, high
- Run replays on a fixed set of HW
- 2s / 5s window averaged frame rate
Trading

• Low latency request processing systems
• Performance is a currency
  – Everyone will identify big opportunities
  – Race to the exchange
  – Winner takes all
Trading

- Most of the time system is idle
- Bursts on big events
- Latency is the king
  - Speed to take profitable trades
  - Speed to adjust our own orders
Trading

• Dedicated high end Linux HW
• Speedlab environment to test performance
• Lightweight tracing in speedlab and production
• Using time series DB to store captured data
  – Easy data retrieval for given time range
  – Historical data analysis
Money loop
Performance challenges in trading

• Cache!
Cache

• Generally L3 is shared across all cores
• Pick your neighbours wisely
• HT threads share L1.
  – This is one of the reasons why we disable HT
• You want all your data to be in cache!
• Cache warming techniques
  – Keep running
  – Keep touching memory
How trading measures latency

TRADING STACK

INFORMATION

EXECUTION

STRATEGY

EXCHANGE

Hardware timestamps

Software timestamps
Using timestamps

• Latency histograms
  – simulated environment
  – production
• Detecting outliers
• Drilling down specific events
Lightweight tracing

• How light it is?
  – HW timestamp cost is a few nanoseconds
  – SW timestamp is higher, still very cheap
• Very useful for understanding performance profile
• Visualizing and recognizing patterns
Low Latency Fizzbuzz

- [https://github.com/phejet/benchmarkingcpp_games_trading](https://github.com/phejet/benchmarkingcpp_games_trading)
- C++ server which reads input data
- Outputs Fizz, Buzz, FizzBuzz or just a number
- How to make it fast ?
- Measure first !!!
Fizzbuzz

• How long do you think it takes run this code?
• 3.5GHz Xeon at CentOS 7

```cpp
std::string MakeFizzBuzz(uint64_t number)
{
    std::string output;
    if (number % 3 == 0)
    {
        output += "Fizz";
    }
    if (number % 5 == 0)
    {
        output += "Buzz";
    }
    if (output.empty())
    {
        output = std::to_string(number);
    }
    return output;
}
```
## Quiz results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM_FizzBuzz</td>
<td>98 ns</td>
</tr>
</tbody>
</table>
Request processing

```cpp
void Application::HandleRequest(const char* recvBuf, size_t recvSize, IConnection* conn)
{
    TimeCapture timeCapture(mTimingLog); // <-- START PROCESSING TS
    uint64_t number = ReadInt<uint64_t>(recvBuf, recvSize);
    timeCapture.CaptureFinishParsingTSC(); // <-- FINISH Parsing TS

    std::string output = MakeFizzBuzz(number);
    timeCapture.CaptureFinishProcessingTSC(); // <-- FINISH PROCESSING TS

    conn->Send(output);
    timeCapture.CaptureFinishSendTSC(); // <-- FINISH SENDING TS

    timeCapture.SetRequest(number);
}
```
typedef std::chrono::time_point<std::chrono::high_resolution_clock> Timestamp;

struct TimeLogEntry
{
    uint64_t request = 0;
    Timestamp startTS;
    Timestamp finishParsingTS;
    Timestamp finishProcessingTS;
    Timestamp finishSendTS;
};

class TimeLog
{
    public:
    TimeLog();
    void DumpLog();

    private:
    const size_t MAX_TIME_LOG_SIZE = 1000000;
    friend class TimeCapture;
    std::vector<TimeLogEntry> mEntries;
};
Timing

```cpp
void CaptureFinishProcessingTSC()
{
    mEntry->finishProcessingTS = CaptureTS();
}

void CaptureFinishSendTSC()
{
    mEntry->finishSendTS = CaptureTS();
}

private:
    Timestamp CaptureTS()
    {
        return std::chrono::high_resolution_clock::now();
    }
```
Using Epoch

```cpp
unsigned long long ToEpoch(Timestamp ts)
{
    return std::chrono::duration_cast<std::chrono::nanoseconds>(
        ts.time_since_epoch()).count();
}
```
Timings output

Header: TimeLogEntry: request=uint64,startTS=timestamp,finishParsingTS=timestamp,fini
Data: TimeLogEntry: request=639788,startTS=148102154403323856,finishParsingTS=148102
Data: TimeLogEntry: request=275754,startTS=1481021544033248315,finishParsingTS=148102
Data: TimeLogEntry: request=736735,startTS=1481021544033250390,finishParsingTS=148102
Data: TimeLogEntry: request=892288,startTS=1481021544033250703,finishParsingTS=148102
Data: TimeLogEntry: request=422500,startTS=1481021544033251379,finishParsingTS=148102
Data: TimeLogEntry: request=219419,startTS=1481021544033251600,finishParsingTS=148102
Data: TimeLogEntry: request=27509,startTS=1481021544033251945,finishParsingTS=148102
Data: TimeLogEntry: request=650235,startTS=1481021544033252329,finishParsingTS=148102
Data: TimeLogEntry: request=221220,startTS=1481021544033252604,finishParsingTS=148102
Data: TimeLogEntry: request=809621,startTS=1481021544033252836,finishParsingTS=148102
Data: TimeLogEntry: request=806014,startTS=1481021544033253189,finishParsingTS=148102
Data: TimeLogEntry: request=340910,startTS=1481021544033253558,finishParsingTS=148102
Data: TimeLogEntry: request=957256,startTS=1481021544033253793,finishParsingTS=148102
Macro benchmark

def _run_benchmark(self):
    """Run process under test on simulation data and print timings""
    os.chdir(self.workspace_dir)
    cmd = "%s %s > output.txt" % (FULL_BINARY_PATH, SIMULATION_FILENAME)
    if IS_LINUX:
        cmd = 'taskset -c 1 ' + cmd
    os.system(cmd)
    print 'Parsing collected data'
    self._print_stats(self._parse_timings())

def test_bursts(self):
    # generate test data
    NUM_REQUESTS = 1000000

    file = ''
    for i in range(NUM_REQUESTS):
        file += "%d %d\n" % (random.randint(1000, 1000000), random.randint(0, 100))
    with open(os.path.join(self.workspace_dir, SIMULATION_FILENAME), 'w') as f:
        f.write(file)

    self._run_benchmark()
Quick feedback

- Time in nanoseconds

<table>
<thead>
<tr>
<th>Name</th>
<th>avg</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing</td>
<td>38</td>
<td>37</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>Processing</td>
<td>97</td>
<td>47</td>
<td>133</td>
<td>138</td>
</tr>
<tr>
<td>Send</td>
<td>42</td>
<td>37</td>
<td>41</td>
<td>43</td>
</tr>
<tr>
<td>Total</td>
<td>178</td>
<td>127</td>
<td>209</td>
<td>216</td>
</tr>
</tbody>
</table>
Jupyter notebooks

- Open-source web application
- Create and share documents that contain
  - Live code
  - Equations
  - Visualizations
  - Narrative text
Jupyter notebook for in-depth analysis

```
In [7]: df.head()
```

```
<table>
<thead>
<tr>
<th>Parsing</th>
<th>Processing</th>
<th>Send</th>
<th>Total</th>
<th>finishParsingTS</th>
<th>finishProcessingTS</th>
<th>finishSendTS</th>
<th>request</th>
<th>startTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.177</td>
<td>11.333</td>
<td>3.711</td>
<td>15.221</td>
<td>2016-12-06</td>
<td>2016-12-06</td>
<td>639788</td>
<td>2016-12-06</td>
</tr>
<tr>
<td>1</td>
<td>0.047</td>
<td>1.678</td>
<td>0.233</td>
<td>1.958</td>
<td>2016-12-06</td>
<td>2016-12-06</td>
<td>25985</td>
<td>2016-12-06</td>
</tr>
<tr>
<td>2</td>
<td>0.050</td>
<td>0.084</td>
<td>0.048</td>
<td>0.182</td>
<td>2016-12-06</td>
<td>2016-12-06</td>
<td>275754</td>
<td>2016-12-06</td>
</tr>
<tr>
<td>3</td>
<td>0.038</td>
<td>0.354</td>
<td>0.054</td>
<td>0.446</td>
<td>2016-12-06</td>
<td>2016-12-06</td>
<td>223987</td>
<td>2016-12-06</td>
</tr>
<tr>
<td>4</td>
<td>0.038</td>
<td>0.057</td>
<td>0.045</td>
<td>0.140</td>
<td>2016-12-06</td>
<td>2016-12-06</td>
<td>736735</td>
<td>2016-12-06</td>
</tr>
</tbody>
</table>
```
**Histogram as text**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In [8]:</strong></td>
<td>df.Processing.describe(percentiles=[.25, .5, .75, 0.9, 0.99, 0.999])</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Out[8]:</strong></td>
<td>count</td>
<td>1000000.000000</td>
<td>mean</td>
<td>0.096735</td>
<td>std</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>min</td>
<td>0.025000</td>
<td>25%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>50%</td>
<td>0.133000</td>
<td>75%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>90%</td>
<td>0.142000</td>
<td>99%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>99.9%</td>
<td>0.218000</td>
<td>max</td>
</tr>
<tr>
<td></td>
<td>Name: Processing, dtype: float64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Looks big
Beware of outliers

In [17]: df.Processing.hist(grid=True, figsize=(20, 10), bins=50)
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x1b0ba6c0>
Discarding outliers

```
In [12]:
S = df.Processing
S = S[~((S - S.mean()).abs() > 3.5 * S.std())]
df.NORM = S
df.NORM.describe(percentiles=[.25, .5, .75, 0.9, 0.99, 0.999])
```

Out[12]:
```
count 999919.000000
mean  0.096387
std   0.045414
min   0.025000
25%   0.045000
50%   0.133000
75%   0.137000
90%   0.142000
99%   0.154000
99.9% 0.216000
max   0.338000
Name: Processing, dtype: float64
```
Distribution is strange…

<table>
<thead>
<tr>
<th>count</th>
<th>999919.000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean</td>
<td>0.096387</td>
</tr>
<tr>
<td>std</td>
<td>0.045414</td>
</tr>
<tr>
<td>min</td>
<td>0.025000</td>
</tr>
<tr>
<td>25%</td>
<td>0.045000</td>
</tr>
<tr>
<td>50%</td>
<td>0.133000</td>
</tr>
<tr>
<td>75%</td>
<td>0.137000</td>
</tr>
<tr>
<td>90%</td>
<td>0.142000</td>
</tr>
<tr>
<td>99%</td>
<td>0.154000</td>
</tr>
<tr>
<td>99.9%</td>
<td>0.216000</td>
</tr>
<tr>
<td>max</td>
<td>0.338000</td>
</tr>
</tbody>
</table>

Not unimodal?
Bimodal distribution

In [18]: df.NORM.hist(grid=True, figsize=(20, 10), bins=50)
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1c4c6240>
Optiver profiler

• In-house tracing profiler
• Mark interesting parts of your code
  – Scope guards to capture entry/exit timestamps and function name
  – Single named events
• Nanosecond precision
• Multiple tools to view results
• Tarantula is the most interesting one
Tarantula

[path count = 2725883, steps = 9, total time = 660 ms (61.26%, 61.26% acc.)]
mean: 242 ns, min: 214 ns, max: 5.6 us, iqr: 5 ns
percentiles: 231 ns [235 ns, 238 ns, 240 ns] 251 ns

Delta

[0 ns 0% enter HandleRequest 11 ns 5% enter ReadInt 9 ns 4% enter atoi]
[35 ns 15% leave atoi 8 ns 3% leave ReadInt 9 ns 4% enter MakeFizzBuzz]
[12 ns 5% enter to_string 124 ns 51% leave to_string 8 ns 3% leave MakeFizzBuzz]
[25 ns 11% leave HandleRequest]

Delta

[0 ns 0% enter HandleRequest 11 ns 7% enter ReadInt 9 ns 6% enter atoi]
[36 ns 23% leave atoi 8 ns 5% leave ReadInt 9 ns 6% enter MakeFizzBuzz]
[59 ns 37% leave MakeFizzBuzz 26 ns 16% leave HandleRequest]
Two codepaths!

```cpp
std::string MakeFizzBuzz(uint64_t number) {
    std::string output;
    if (number % 3 == 0) {
        output += "Fizz";
    }
    if (number % 5 == 0) {
        output += "Buzz";
    }
    if (output.empty()) {
        output = std::to_string(number);
    }
    return output;
}
```

Non FizzBuzz code path
Optimisation

• FizzBuzz logic is the most expensive part of our request processing
• How can we make it faster?

```cpp
std::string MakeFizzBuzz(uint64_t number)
{
    std::string output;
    if (number % 3 == 0)
    {
        output += "Fizz";
    }
    if (number % 5 == 0)
    {
        output += "Buzz";
    }
    if (output.empty())
    {
        output = std::to_string(number);
    }
    return output;
}
```
Brute force approach

• Write custom function instead of using std::to_string
• Return result as const char* and use static buffer
Look at high level

```cpp
void Application::HandleRequest(const char* recvBuf, size_t recvSize, IConnection* conn)
{
    TimeCapture timeCapture(mTimingLog); // <-- START PROCESSING TS
    uint64_t number = ReadInt<uint64_t>(recvBuf, recvSize);
    timeCapture.CaptureFinishParsingTSC(); // <-- FINISH PARSING TS

    std::string output = MakeFizzBuzz(number);
    timeCapture.CaptureFinishProcessingTSC(); // <-- FINISH PROCESSING TS

    conn->Send(output);
    timeCapture.CaptureFinishSendTSC(); // <-- FINISH SENDING TS

    timeCapture.SetRequest(number);
}
```
Avoid int->string conversion

```cpp
const char* MakeFizzBuzz(const char* input) {
    int number = std::atoi(input);
    const bool isFizz = (number % 3) == 0;
    const bool isBuzz = (number % 5) == 0;
    if (isFizz && isBuzz)
    {
        return "FizzBuzz";
    }
    if (isFizz)
    {
        return "Fizz";
    }
    if (isBuzz)
    {
        return "Buzz";
    }
    return input;
}
```
## Measuring Optimised code

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time(ns)</th>
<th>CPU(ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM_FizzBuzz</td>
<td>92</td>
<td>98</td>
</tr>
<tr>
<td>BM_FizzBuzzAtoi</td>
<td>21</td>
<td>18</td>
</tr>
<tr>
<td>BM_FizzBuzzTakeStr</td>
<td>117</td>
<td>125</td>
</tr>
<tr>
<td>BM_FizzBuzzTakeStrOpt</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>
Closing

• It’s very hard to guess execution time by just looking at code
• Having a simple and reproducible way to measure performance is very important
• Visualising performance data helps to understand it
• Understanding is a necessary first step before optimization
• When optimizing code, always look at the high level picture
Questions?

• Alexander Radchenko
• phejet@gmail.com
• https://github.com/phejet/benchmarkingcpp_games_trading
• @phejet on Twitter

WE ARE HIRING!

VISIT OPTIVER.COM/JOB-OPPORTUNITIES FOR MORE INFORMATION