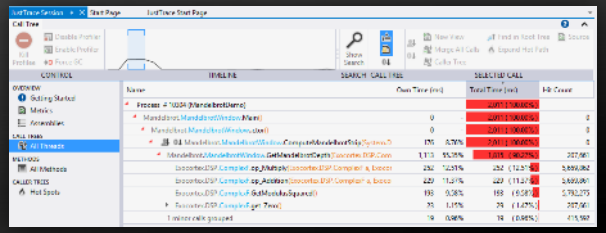
Telerik Academy



Code Tuning and Optimization

When and How to Improve Code Performance?

Telerik Software Academy
Learning & Development
http://academy.telerik.com

Telerik Academy

Table of Contents

- Computer Performance
- Code Tuning
- JustTrace
- C# Optimizations

Telerik Academy

Computer Performance

Telerik Academy

What is Performance?

Computer performance is characterized by the amount of useful work accomplished by a computer system compared to the time and resources used.

- An aspect of software quality that is important in human–computer interactions
- Resources:
  - en.wikipedia.org/wiki/Computer_performance
  - C#: http://www.dotnetperls.com/optimization

Good Computer Performance

- Good computer performance:
  - Short response time for a given piece of work
  - High throughput (rate of processing work)
  - Low utilization of computing resource(s)
  - High availability of the computing system or application
  - Fast (or highly compact) data compression and decompression
  - High bandwidth / short data transmission time

Actual vs. Perceived Performance

- Example: *"Vista's file copy performance is noticeably worse than Windows XP"* – false:
  - Vista uses algorithm that perform better in most cases
  - Explorer waits 12 seconds before providing a copy duration estimate, which certainly provides no sense of smooth progress
  - The copy dialog is not dismissed until the write-behind thread has committed the data to disk, which means the copy is slowest at the end

Is Performance Really a Priority?

- Performance improvements can reduce readability and complexity

> "Premature optimization is the root of all evil."
>
> Donald Knuth

> "More computing sins are committed in the name of efficiency (without necessarily achieving it) than for any other single reason – including blind stupidity."
>
> W. A. Wulf

Telerik Academy

How to Improve Performance?

- Software requirements
  - Software cost vs. performance
- System design
  - Performance-oriented architecture
  - Resource-reducing goals for individual subsystems, features, and classes
- Class and method design
  - Data structures and algorithms

How to Improve Performance? (2)

- External Interactions
  - Operating system
  - External devices – storage, network, Internet
- Code Compilation / Code Execution
  - Compiler optimizations
- Hardware
  - Very often the cheapest way
- Code Tuning

Telerik Academy

Code Tuning Concepts

Telerik Academy

Introduction to Code Tuning

- What is code tuning / performance tuning?
  - Modifying the code to make it run more efficiently (faster)
  - Not the most effective / cheapest way to improve performance
  - Often the code quality is decreased to increase the performance
- The 80 / 20 principle
  - 20% of a program's methods consume 80% of its execution time

Systematic Tuning – Steps

- Systematic code tuning follows these steps:
  1. Assess the problem and establish numeric values that categorize acceptable behavior
  2. Measure the performance of the system before modification
  3. Identify the part of the system that is critical for improving the performance
     - This is called the bottleneck
  4. Modify that part of the system to remove the bottleneck

Systematic Tuning – Steps (2)

5. Measure the performance of the system after modification
6. If the modification makes the performance better, adopt it

   If the modification makes the performance worse, discard it

Code Tuning Myths

- "Reducing the lines of code in a high-level language improves the speed or size of the resulting machine code" → false!

```
for i = 1 to 10
   a[i] = i
end for
```

VS.

```
a[1] = 1
a[2] = 2
a[3] = 3
a[4] = 4
a[5] = 5
a[6] = 6
a[7] = 7
a[8] = 8
a[9] = 9
a[10] = 10
```

Code Tuning Myths (2)

- "A fast program is just as important as a correct one" → false!
  - The software should work correctly!

Code Tuning Myths (3)

- "Certain operations are probably faster or smaller than others" → false!
  - E.g. "add" is faster than "multiply"
  - Always measure performance!

- "You should optimize as you go" → false!
  - It is hard to identify bottlenecks before a program is completely working
  - Focus on optimization detracts from other program objectives
  - Performance tuning breaks code quality!

When to Tune the Code?

- Use a high-quality design
  - Make the program right
  - Make it modular and easily modifiable
  - When it's complete and correct, check the performance
- Consider compiler optimizations
- Measure, measure, measure
- Write clean code that's easy to maintain
- Write use unit tests before optimizing

When to Tune the Code (2)?

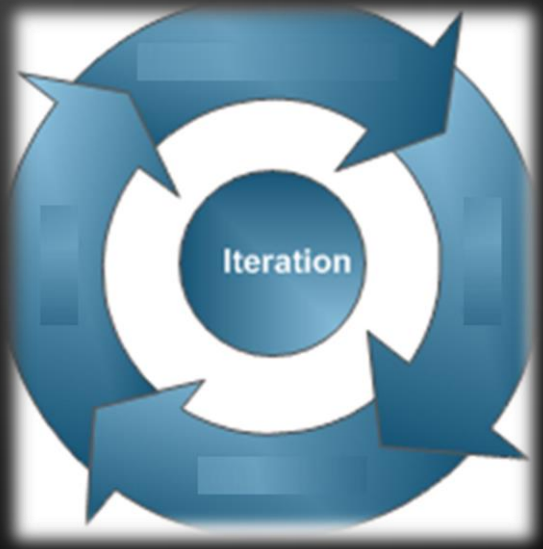
- Junior developers think that "selection sort is slow"? Is this correct?
  - Answer: depends!
  - Think how many elements you sort
  - Is "selection sort" slow for 20 or 50 elements?
  - Is it slow for 1,000,000 elements?
  - Shall we rewrite the sorting if we sort 20 elements?
- Conclusion: never optimize unless the piece of code is proven to be a bottleneck!

Measurement

- Measure to find bottlenecks
- Measurements need to be precise
- Measurements need to be repeatable

Optimize in Iterations

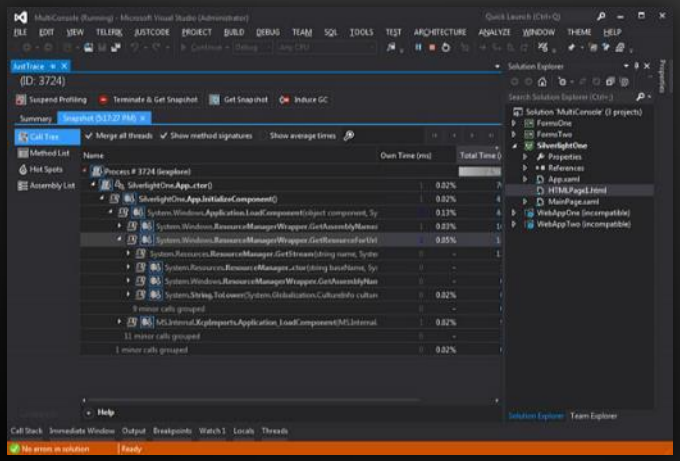
- Measure improvement after each optimization
- If optimization does not improve performance → revert it
- Stop testing when you know the answer

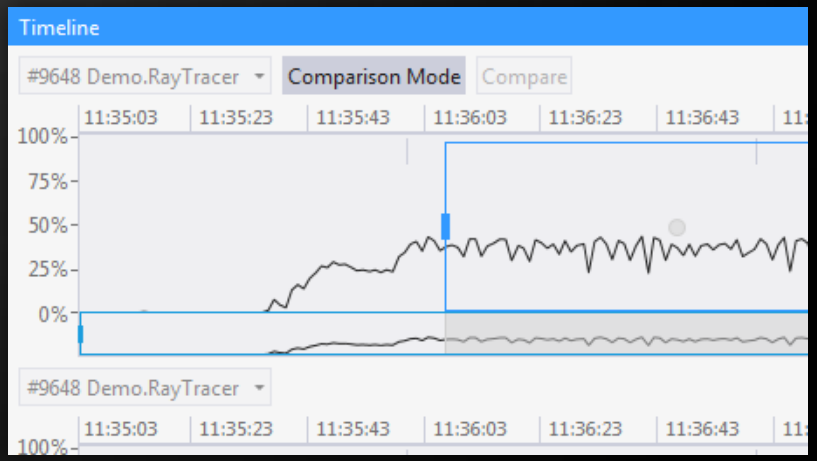


Telerik Academy

Telerik JustTrace

Resolve Performance Issues





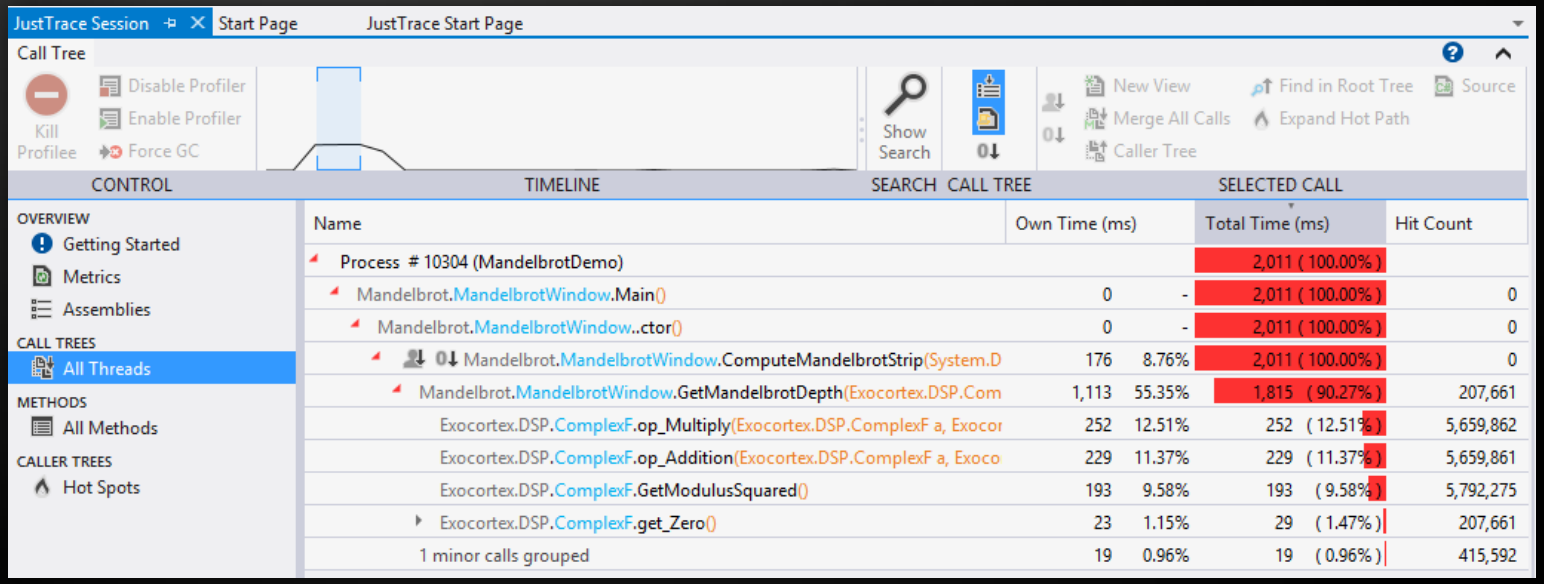
What is JustTrace?

- What is JustTrace?
  - A performance analysis tool
  - Also called profiler
  - Designed for code and memory profiling
  - Measures the frequency and duration of function calls
  - Traces the call stack
  - Collects information about memory usage

Telerik Academy

Demo: JustTrace

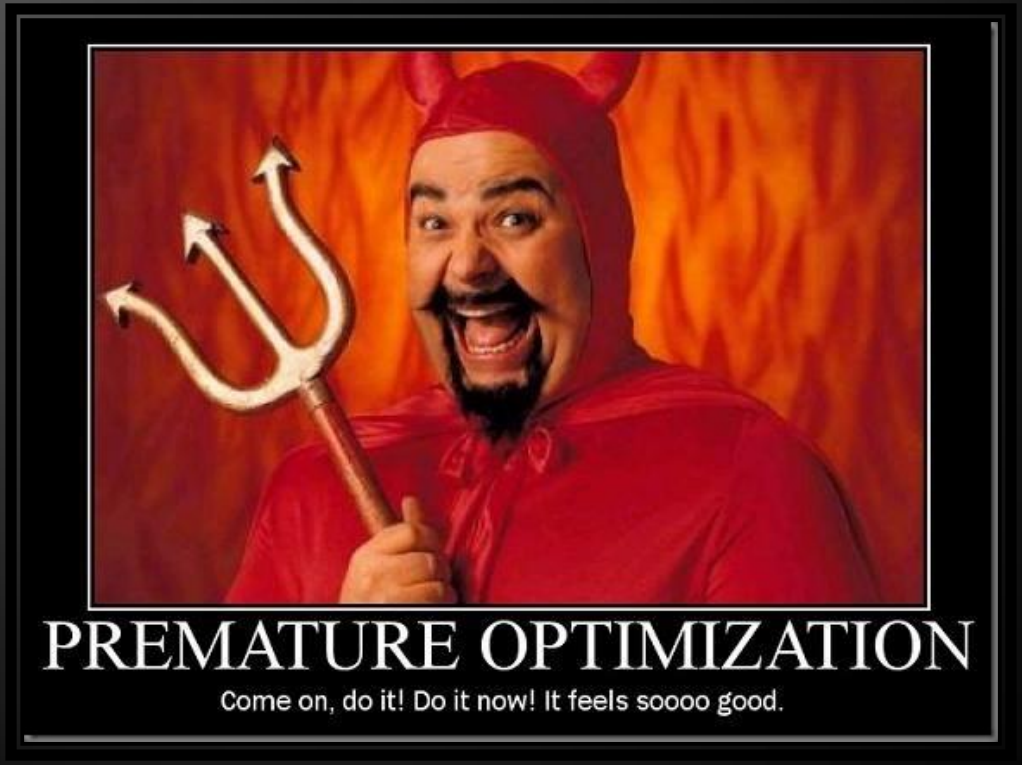
Profiling and Improving Performance of "Mandelbrot Fractal" application (Code-Tuning-and-Optimization-Demo.zip)



Telerik Academy

C# Code Optimizations

http://www.dotnetperls.com/optimization



Telerik Academy

Do We Need Optimizations?

- The C# language is fast (unlike Java)
  - A bit slower than C and C++
- Is it worthwhile to benchmark programming constructs?
  - We should forget about small optimizations
    - Say about 97% of the time: premature optimization is the root of all evil
  - At all levels of performance optimization
    - You should be taking measurements on the changes you make

Telerik Academy

Benchmarking with Stopwatch

- Stopwatch measures time elapsed
- Useful for micro-benchmarks optimization

```
using System.Diagnostics;

static void Main()
{
  Stopwatch stopwatch = new Stopwatch();
  stopwatch.Start();

    // Do something …

  stopwatch.Stop();
  Console.WriteLine("Time elapsed: {0}", stopwatch.Elapsed);
}
```

C# Optimization Tips

- Static fields are faster than instance fields
- Instance methods are always slower than static methods
  - To call an instance method, the instance reference must be resolved first
  - Static methods do not use an instance reference
- It is faster to minimize method arguments
  - Even use constants in the called methods instead of passing them arguments
  - This causes less stack memory operations

C# Optimization Tips (2)

- When you call a method in your C# program
  - The runtime allocates a separate memory region to store all local variable slots
  - This memory is allocated on the stack
  - Sometimes we can reuse the same variable
    - Well-known anti-pattern for quality code
- Constants are fast
  - Constants are not assigned a memory region, but are instead considered values
  - Injected directly into the instruction stream

C# Optimization Tips (3)

- The `switch` statement compiles in a different way than `if`-statements typically do
  - Some switches are faster than `if`-statements
- Using two-dimensional arrays is relatively slow
  - We can explicitly create a one-dimensional array and access it through arithmetic
  - The .NET Framework enables faster accesses to jagged arrays than to 2D arrays
  - Jagged arrays may cause slower garbage collections

C# Optimization Tips (4)

- `StringBuilder` can improve performance when appending strings
  - Using `char[]` may be the fastest way to build up a string
- If you can store your data in an array of bytes, this allows you to save memory
  - Smallest unit of addressable storage – byte
- Simple array `T[]` is always faster than `List<T>`
  - Using efficient data structures (e.g. `HashSet<T>` and `Dictionary<K,T>`) may speed-up the code

C# Optimization Tips (5)

- Use lazy evaluation (caching)

```
public int GetSize()
{
  if (this.size == null)
    size = CalculateSize();
  return this.size;
}
```

- for-loops are faster than foreach loops
  - foreach uses enumerator
- C# structs are slower (in most cases)
  - Structs are copied in their entirety on each function call or return value

C# Optimization Tips (6)

- Instead of testing each case using logic, you can translate it through a lookup table
  - It eliminates costly branches in your code
- It is more efficient to work with a char instead of a single-char string
- Use JustDecompile or any other decompilation tool to view the output IL code
- Use Debug → Windows → Disassembly in VS
- Don't do unnecessary optimizations!
- Measure after each change

Which is the Fastest?

```
string str = new string('a', 5000000);

int count = 0;
for (int i = 0; i < str.Length; i++)
  if (str[i] == 'a')
    count++;
```

```
int count = 0;
int len = str.Length;
for (int i = 0; i < len; i++)
  if (str[i] == 'a')
    count++;
```

```
int count = 0;
foreach (var ch in str)
  if (ch == 'a')
    count++;
```

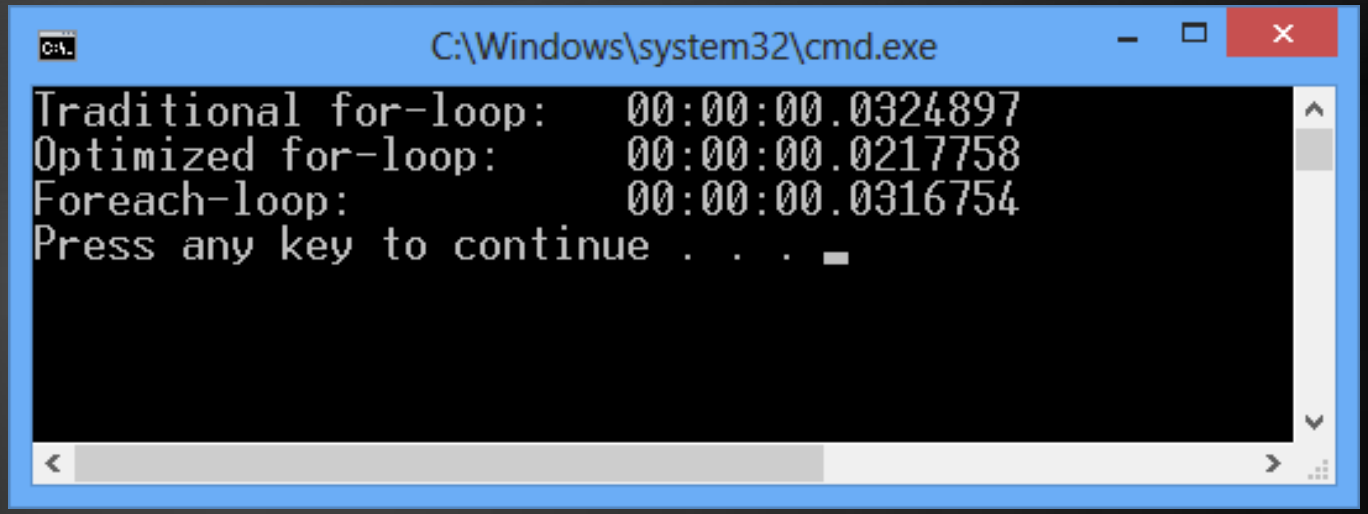
Telerik Academy

Optimization Tips – Inline Code

- Manual or compiler optimization that replaces a method call with the method body
  - You can manually paste a method body into its call spot or let the compiler to decide
- Typically, inlined code improves performance in micro-benchmarks
  - … but makes the code hard to maintain!
- In .NET 4.5 you can force code inlining:

```
[MethodImpl(MethodImplOptions.AggressiveInlining)]
void SomeMethod() { … }
```

Telerik Academy



Measuring Performance in C#

Live Demo

Telerik Academy

Code Tuning and Optimizations

http://academy.telerik.com

Homework

1. You are given a C# application (Code-Tuning-and-Optimization-Homework.zip) which displays an animated 3D model of the Solar system.
   - Use a profiler to find the places in its source code which cause significant performance degradation (bottlenecks).
     - Provide a screenshot of the profiler's result and indicate the place in the source code where the bottleneck resides (name of the file, line of code).
   - Make a quick fix in the source code in order to significantly improve the performance. Test the code after the fix for correctness + performance.

Homework (2)

2. Write a program to compare the performance of add, subtract, increment, multiply, divide for `int`, `long`, `float`, `double` and `decimal` values.
3. Write a program to compare the performance of square root, natural logarithm, sinus for `float`, `double` and `decimal` values.
4. * Write a program to compare the performance of insertion sort, selection sort, quicksort for `int`, `double` and `string` values. Check also the following cases: random values, sorted values, values sorted in reversed order.

Telerik Academy

Free Trainings @ Telerik Academy

- C# Programming @ Telerik Academy
  - csharpfundamentals.telerik.com

- Telerik Software Academy
  - academy.telerik.com

- Telerik Academy @ Facebook
  - facebook.com/TelerikAcademy

- Telerik Software Academy Forums
  - forums.academy.telerik.com