Tips for optimizing C#/.NET programs [closed]

Asked 14 years, 9 months ago Modified 9 years, 3 months ago

Viewed 40k times

79

As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, visit the help center for guidance.

Closed 12 years ago.

It seems like optimization is a lost art these days. Wasn't there a time when all programmers squeezed every ounce of efficiency from their code? Often doing so while walking five miles in the snow?

In the spirit of bringing back a lost art, what are some tips that you know of for simple (or perhaps complex) changes to optimize C#/.NET code? Since it's such a broad thing that depends on what one is trying to accomplish it'd help to provide context with your tip. For instance:

- When concatenating many strings together use `StringBuilder` instead. See link at the bottom for caveats on this.
- Use `string.Compare` to compare two strings instead of doing something like `string1.ToLower() == string2.ToLower()`

The general consensus so far seems to be measuring is key. This kind of misses the point: measuring doesn't tell you what's wrong, or what to do about it if you run into a bottleneck. I ran into the string concatenation bottleneck once and had no idea what to do about it, so these tips are useful.

My point for even posting this is to have a place for common bottlenecks and how they can be avoided before even running into them. It's not even necessarily about plug and play code that anyone should blindly follow, but more about gaining an understanding that performance should be thought about, at least somewhat, and that there's some common pitfalls to look out for.

I can see though that it might be useful to also know why a tip is useful and where it should be applied. For the `StringBuilder` tip I found the help I did long ago at here on Jon Skeet's site.

c# **.net** **optimization**

Share

edited Sep 5, 2015 at 10:13

Improve this question

Follow

community wiki
3 revs, 2 users 80%
Bob

10 It's also important to walk to line between optimization and readability. – Ryan Elkins Mar 18, 2010 at 22:09

3 The "handful of strings"; the *number* isn't the issue - it is whether they are in a single composite concatenation statement, or multiple statements. – Marc Gravell Mar 18, 2010 at 22:22

2 StringBuilder is often slower than the + operator. The C# compiler automatically translates repeated + into the appropriate overload(s) of String.Concat. – Richard Berg Mar 18, 2010 at 22:33

2 You're going to have to have a tough time fighting the CLR while it's runtime optimizing IL and you tried to do the same at compile time - tug of war. In the good old days you optimized the instructions for the machine and the machine dumbly ran them. – John K Mar 19, 2010 at 1:31

11 Answers

Sorted by: Highest score (default)

105

> It seems like optimization is a lost art these days.

There was once a day when manufacture of, say, microscopes was practiced as an art. The optical principles were poorly understood. There was no

standarization of parts. The tubes and gears and lenses had to be made by hand, by highly skilled workers.

These days microscopes are produced as an engineering discipline. The underlying principles of physics are extremely well understood, off-the-shelf parts are widely available, and microscope-building engineers can make informed choices as to how to best optimize their instrument to the tasks it is designed to perform.

That performance analysis is a "lost art" is a very, very good thing. That art was practiced *as an art*. Optimization should be approached for what it is: an *engineering problem* solvable through careful application of solid engineering principles.

I have been asked dozens of times over the years for my list of "tips and tricks" that people can use to optimize their vbscript / their jscript / their active server pages / their VB / their C# code. I always resist this. **Emphasizing "tips and tricks" is exactly the wrong way to approach performance.** That way leads to code which is hard to understand, hard to reason about, hard to maintain, that is typically not noticably faster than the corresponding straightforward code.

The right way to approach performance is to approach it as an engineering problem like any other problem:

- Set meaningful, measurable, customer-focused goals.

- Build test suites to test your performance against these goals under realistic but controlled and repeatable conditions.
- If those suites show that you are not meeting your goals, use tools such as profilers to figure out why.
- Optimize the heck out of what the profiler identifies as the worst-performing subsystem. Keep profiling on every change so that you clearly understand the performance impact of each.
- Repeat until one of three things happens (1) you meet your goals and ship the software, (2) you revise your goals downwards to something you can achieve, or (3) your project is cancelled because you could not meet your goals.

This is the same as you'd solve any other engineering problem, like adding a feature -- set customer focused goals for the feature, track progress on making a solid implementation, fix problems as you find them through careful debugging analysis, keep iterating until you ship or fail. **Performance is a feature.**

Performance analysis on complex modern systems requires discipline and focus on solid engineering principles, not on a bag full of tricks that are narrowly applicable to trivial or unrealistic situations. I have never once solved a real-world performance problem through application of tips and tricks.

 edited Mar 18, 2010 at 23:06

Follow

community wiki
3 revs
Eric Lippert

1 Was going to write a similar screed, but yours is better. Bravo. – Richard Berg Mar 18, 2010 at 22:42

7 There's just some cases where there's a known better way to accomplish the same task while being less of a hog with resources. I don't buy that it's perfectly fine to program however you want as long as you meet some goal and it seems to work out ok. Or that it's for the best to program, *then* run a profiler, and *then* go back and change problem areas. What is wrong with one having a good idea of what it takes to optimize certain bits of code before they even start? – Bob Mar 18, 2010 at 22:55

5 @Bob: There's nothing wrong with being smart about using resources. Where things go wrong is when people (1) spend a lot of time (=money) on micro-optimizations that do not make a difference, (2) write programs which are *wrong*, and (3) write programs which are unclear. What you should be optimizing for is first, correctness. Second, good coding style. Third, performance. Once the code is correct and elegant, it will be much easier to make it performant. – Eric Lippert Mar 18, 2010 at 23:00

3 That's fine, but you'll notice I'm not saying one should not code for correctness first, or style second, or what have you. But, it's also true that sometimes (or maybe alot of times these days), programmers give no consideration to performance or optimization at all. Is just having 1 & 2 enough to make up for a total uncaring of 3? I can't see how it's a bad idea to pay some respects to optimization and to learn a thing or two about what it takes – Bob Mar 18, 2010 at 23:06

7 @Bob: I agree that some programmers do not care about performance. But I'm not following your point. A list of tips and tricks isn't going to suddenly turn them into people who care about performance. Supposing for the sake of argument that you *can* make people who are at present uninterested into people who are interested, a list of tips and tricks isn't going to help them attain good performance. You can apply tips and tricks to a body of code all day and never know if you're making any progress at all against your goals. You've got to have goals and measure your progress. – Eric Lippert Mar 18, 2010 at 23:10

45

Get a good profiler.

Don't bother even trying to optimize C# (really, any code) without a good profiler. It actually helps dramatically to have both a sampling and a tracing profiler on hand.

Without a good profiler, you're likely to create false optimizations, and, most importantly, optimize routines that aren't a performance problem in the first place.

The first three steps to profiling should always be 1) Measure, 2) measure, and then 3) measure....

Share Improve this answer Follow

answered Mar 18, 2010 at 22:09

community wiki
Reed Copsey

1 I would say, don't *measure*, *capture*. stackoverflow.com/questions/406760/… – Mike Dunlavey Mar 18, 2010 at 22:15

24 You forgot `4) measure` – Nifle Mar 18, 2010 at 22:17

1 @Nifle: If you're hunting elephants, do you need to measure them? – Mike Dunlavey Oct 12, 2010 at 13:45

1 @RobbieDee: See *Conrad Albrecht's answer*. – Mike Dunlavey Aug 7, 2015 at 13:06

1 @MikeDunlavey Sorry, I was just having a bit of fun with you, but thanks... :-) – Robbie Dee Aug 7, 2015 at 13:43

22

Optimization guidelines:

1. Don't do it unless you need to
2. Don't do it if it's cheaper to throw new hardware at the problem instead of a developer
3. Don't do it unless you can measure the changes in a production-equivalent environment
4. Don't do it unless you know how to use a CPU **and** a Memory profiler

5. Don't do it if it's going to make your code unreadable or unmaintainable

As processors continue to get faster the main bottleneck in most applications isn't CPU, it's bandwidth: bandwidth to off-chip memory, bandwidth to disk and bandwidth to net.

Start at the far end: use YSlow to see why your web site is slow for end-users, then move back and fix you database accesses to be not too wide (columns) and not too deep (rows).

In the very rare cases where it's worth doing anything to optimize CPU usage be careful that you aren't negatively impacting memory usage: I've seen 'optimizations' where developers have tried to use memory to cache results to save CPU cycles. The net effect was to reduce the available memory to cache pages and database results which made the application run far slower! (See rule about measuring.)

I've also seen cases where a 'dumb' un-optimized algorithm has beaten a 'clever' optimized algorithm. Never underestimate how good compiler-writers and chip-designers have become at turning 'inefficient' looping code into super efficient code that can run entirely in on-chip memory with pipelining. Your 'clever' tree-based algorithm with an unwrapped inner loop counting backwards that you thought was 'efficient' can be beaten simply because it failed to stay in on-chip memory during execution. (See rule about measuring.)

Share Improve this answer Follow

edited Mar 18, 2010 at 22:55

community wiki
2 revs
Ian Mercer

11 Similarly, don't get obsessed with big-O analysis. The O(nm) naive string search algorithm is, for common business cases, thousands of times faster than the O(n+m) algorithms that preprocess the search strings looking for patterns. Naive string search matching the first character often compiles down to a single machine instruction which is blazingly fast on modern processors that make heavy use of optimistic memory caches. – Eric Lippert Mar 18, 2010 at 23:04

16

When working with ORMs be aware of N+1 Selects.

```
List<Order> _orders = _repository.GetOrders(DateTime.N
foreach(var order in _orders)
{
    Print(order.Customer.Name);
}
```

If the customers are not eagerly loaded this could result in several round trips to the database.

Share Improve this answer Follow

answered Mar 18, 2010 at 22:21

community wiki
Aaron

13

- Don't use magic numbers, use enumerations
- Don't hard-code values
- Use generics where possible since it's typesafe & avoids boxing & unboxing
- Use an error handler where it's absolutely needed
- Dispose, dispose, dispose. CLR wound't know how to close your database connections, so close them after use and dispose of unmanaged resources
- Use common-sense!

Share Improve this answer

Follow

edited Sep 5, 2015 at 9:59

community wiki
2 revs, 2 users 67%
SoftwareGeek

15 As much as I agree that they're good things to do, the first two things here have no impact on performance - just maintainability... – Reed Copsey Mar 18, 2010 at 22:14

true but it's still an optimized code. – SoftwareGeek Mar 18, 2010 at 22:18

4 Additionally, the 3rd (boxing) is rarely a genuine pinch-point; it is exaggerated as an issue; as are exceptions - not *usually* a problem. – Marc Gravell Mar 18, 2010 at 22:18

1 "but it's still an optimized code" - that is a big claim; the only thing there I would expect to be a significant issue is

"dispose"; and that is more likely to surface as exceptions (out of handles, etc), not performance degradation.
– Marc Gravell Mar 18, 2010 at 22:19

1 Actually, the finalizer pattern is quite bad if optimization is your goal. Objects with finalizers are automatically promoted to Gen-1 (or worse). Furthermore, forcing your finalizer code to run on the GC thread usually isn't optimal if there's anything remotely expensive on that Todo list. Bottom line: it's a feature aimed at convenience & correctness, not one intended for raw speed. Details: msdn.microsoft.com/en-us/magazine/bb985010.aspx – Richard Berg Mar 18, 2010 at 22:30

11

OK, I have got to throw in my favorite: If the task is long enough for human interaction, use a manual break in the debugger.

Vs. a profiler, this gives you a call stack and variable values you can use to really understand what's going on.

Do this 10-20 times and you get a good idea of what optimization might really make a difference.

Share Improve this answer Follow

edited Sep 5, 2015 at 9:55

community wiki
2 revs, 2 users 80%
Conrad Albrecht

2 ++ Amen. I've been doing that since before profilers existed. & your program DrawMusic looks awesome! – Mike Dunlavey

Mar 20, 2010 at 16:14

6 This is essentially what profilers do, except they do it better than you in about a thousand different ways (faster, more often, more accurate, etc). They also do give call-stacks. This is the poor-man's (and the old-man-who-is-afraid-to-learn-new-things) solution. – BlueRaja - Danny Pflughoeft May 1, 2013 at 9:07

1 @BlueRaja-DannyPflughoeft: They deceive you. They tell you with great precision that there's nothing much to be done. The difference between this method and profilers is that in this method you can see things to speed up that cannot be teased out from simple statistics. Instead they take 1000s of samples when the information that can lead you to the problem is evident in the first 10 if you can actually see the raw samples. I'm sure you've seen *this post*.
– Mike Dunlavey Jul 4, 2013 at 14:04

@BlueRaja-DannyPflughoeft: Look at results. What's the biggest speedup ratio you ever got using a profiler?
– Mike Dunlavey Jul 4, 2013 at 19:08

3 @BlueRaja-DannyPflughoeft: I'm sure you wouldn't, and when you get to my age you will run into people like yourself. But let's leave that aside. *Here's some source code* If you can speed it up by 3 orders of magnitude, without looking at how I did it, using any other method, you will have bragging rights :) – Mike Dunlavey Jul 4, 2013 at 23:02

10

If you identify a method as a bottleneck, but *you don't know what to do about it*, you are essentially stuck.

So I'll list a few things. All of these things are *not silver bullets* and you *will still have to profile* your code. I'm just

making suggestions for things you *could* do and can sometimes help. Especially the first three are important.

- Try solving the problem using just (or: mainly) low-level types or arrays of them.
- Problems are often small - using a smart but complex algorithm does not always make you win, especially if the less-smart algorithm can be expressed in code that only uses (arrays of) low level types. Take for example InsertionSort vs MergeSort for n<=100 or Tarjan's Dominator finding algorithm vs using bitvectors to naively solve the data-flow form of the problem for n<=100. (the 100 is of course just to give you some idea - **profile**!)
- Consider writing a special case that can be solved using just low-level types (often problem instances of size < 64), even if you have to keep the other code around for larger problem instances.
- Learn bitwise arithmetic to help you with the two ideas above.
- BitArray can be your friend, compared to Dictionary, or worse, List. But beware that the implementation is not optimal; You can write a faster version yourself. Instead of testing that your arguments are out of range etc., you can often structure your algorithm so that the index can not go out of range anyway - but you can not remove the check from the standard BitArray and *it is not free*.

- As an example of what you can do with just arrays of low level types, the BitMatrix is a rather powerful structure that can be implemented as *just an array of ulongs* and you can even traverse it using an ulong as "front" because you can take the lowest order bit in constant time (compared with the Queue in Breadth First Search - but obviously the order is different and depends on the *index* of the items rather than purely the order in which you find them).
- Division and modulo are really slow unless the right hand side is a constant.
- Floating point math is **not** in general slower than integer math anymore (not "something you can do", but "something you can skip doing")
- Branching is *not free*. If you can avoid it using a simple arithmetic (anything but division or modulo) you can sometimes gain some performance. Moving a branch to outside a loop is almost always a good idea.

edited Sep 5, 2015 at 10:11

community wiki
2 revs, 2 users 77%
harold

Some good stuff there which helped me greatly - thanks!
– Robbie Dee Aug 7, 2015 at 9:19

8

People have funny ideas about what actually matters. Stack Overflow is full of questions about, for example, is `++i` more "performant" than `i++`. Here's an example of real performance tuning, and it's basically the same procedure for any language. If code is simply written a certain way "because it's faster", that's guessing.

Sure, you don't purposely write stupid code, but if guessing worked, there would be no need for profilers and profiling techniques.

Share Improve this answer Follow

edited May 23, 2017 at 10:31

community wiki
4 revs, 2 users 67%
Mike Dunlavey

6

The truth is that there is no such thing as the perfect optimised code. You can, however, optimise for a specific *portion* of code, on a known system (or set of systems) on a known CPU type (and count), a known platform (Microsoft? Mono?), a known framework / BCL version, a known CLI version, a known compiler version (bugs, specification changes, tweaks), a known amount of total and available memory, a known assembly origin (GAC? disk? remote?), with known background system activity from other processes.

In the real world, use a profiler, and look at the important bits; usually the obvious things are anything involving I/O, anything involving threading (again, this changes hugely between versions), and anything involving loops and lookups, but you might be surprised at what "obviously bad" code isn't actually a problem, and what "obviously good" code is a huge culprit.

Share Improve this answer Follow

edited Sep 5, 2015 at 9:58

community wiki
2 revs, 2 users 64%
Peter Mortensen

5

Tell the compiler *what* to do, not *how* to do it. As an example, `foreach (var item in list)` is better than `for (int i = 0; i < list.Count; i++)` and `m = list.Max(i => i.value);` is better than `list.Sort(i => i.value); m = list[list.Count - 1];`.

By telling the system what you want to do it can figure out the best way to do it. LINQ is good because its results aren't computed until you need them. If you only ever use the first result, it doesn't have to compute the rest.

Ultimately (and this applies to all programming) minimize loops and minimize what you do in loops. Even more important is to minimize the number of loops inside your loops. What's the difference between an O(n) algorithm

and an O(n^2) algorithm? The O(n^2) algorithm has a loop inside of a loop.

Share Improve this answer Follow

answered Mar 18, 2010 at 22:40

community wiki
Gabe

ironacly LINQ adds extra sausage, and one should wonder if a solution without it exists. – Peter Mar 31, 2016 at 9:27

2

I don't really try to optimize my code but at times I will go through and use something like reflector to put my programs back to source. It is interesting to then compare what I wrong with what the reflector will output. Sometimes I find that what I did in a more complicated form was simplified. May not optimize things but helps me to see simpler solutions to problems.

Share Improve this answer Follow

answered Mar 19, 2010 at 3:20

community wiki
Aaron Havens