OUTREF

GAME DEVELOPMENT AND .NET BLOG

ABOUT

IL2CPP optimisations

June 1, 2018 forwolk Uncategorized 0

At some point, you will encounter a situation in which even if you employ optimal algorithms and data structures, performance will not be sufficient for your task. You will still need to squeeze some more power from the CPU. Turning to micro-optimizations may be a solution in your case.

Unity exposes a way in which you can influence IL2CPP compiler to grant you additional performance.

There are three attributes which can be used to do this: **Null checks**, **Array bounds checks**, **Divide by zero checks**.

Now, according to CLR standard, calling any class method, even *non-virtual*, even if the class itself is *sealed*, requires a “*this != null*” check executed first. Otherwise, the function may start executing, and it will be harder to debug the issue if *this pointer* is actually – null. For this reason, emitted IL code employes “*callvirt*” instruction even for non-virtual methods. This instruction has a built-in “*this != null*” check. IL2CPP is trying to stay compatible with this approach, but you can turn off null checks generation by marking functions or classes with the special attribute. This can give you the additional performance boost you need. By the way, as structs cannot be null,

SEARCH …

RECENT POSTS

Memory alignment fun
Unity: Memory profiling
IL2CPP optimisations
Dictionary: Best practices

RECENT COMMENTS

ARCHIVES

July 2018
June 2018
May 2018

CATEGORIES

Uncategorized

META

Log in
Entries RSS
Comments RSS
WordPress.org

their functions are called by a “*call*” instruction without any checks. Thus if you are using structs, you will gain no benefit from this optimization.

Another check employed by the CLR is Array bounds check. It is meant to act when you are trying to access data out of Array/List range and fail early on with *IndexOutOfRange* exception. IL2CPP generates those checks as well, but the usage of the special attribute can force compiler not to emit them.

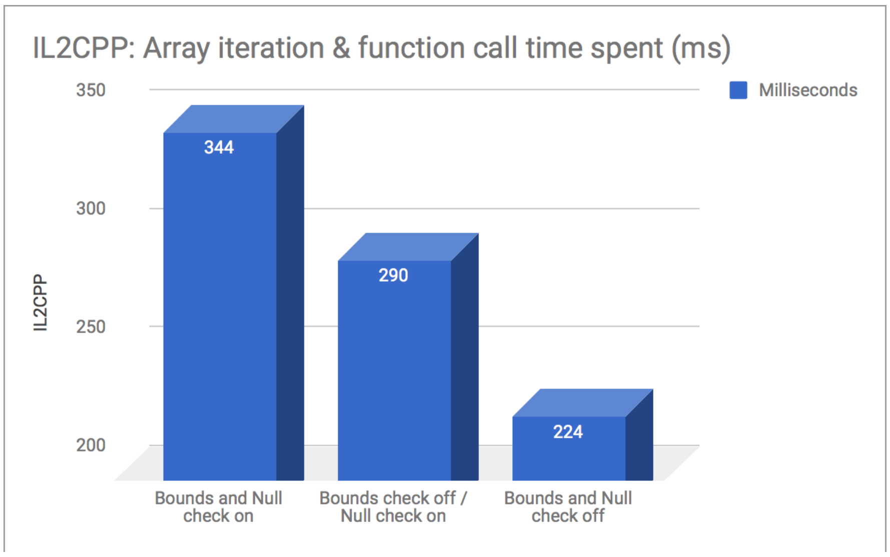
Divide by zero checks are automatically placed where integer division occurs. Turning them off may give you boost only in a very specific case, so we are going to skip them.

I’ve designed a benchmark to test how significant will be the speedup when disabling those attributes.

Benchmark iterates 10000 times over an Array with a size 10000, calling a function.

The benchmark was compiled with Unity 2018.1.0f2. Runtime version .Net 4x.
Api compatibility level .Net Standard 2.0.

Test hardware Samsung Galaxy S6, Android 7



It seems that running a loop with function call is almost 45% faster when both checks are off.
You can download the benchmark here. If you are going to run tests yourself, be careful, use only IL2CPP targets as Mono compilation is unaffected by those attributes.

« PREVIOUS
Dictionary: Best practices

NEXT »
Unity: Memory profiling