

Re: Idea for ECMA/C# Standard – compile time hash for performance

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2005-04/msg01939.html>

- *From:* WXS <WXS@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Thu, 7 Apr 2005 17:53:01 -0700
-

I agree with you the chance of a compiler change is slim, but if it wasn't it would be the best performing option available, with the easiest maintenance and least effort for us at least :)

Yes this problem is specific to the business and types of data, plus architecture in use. Realtime programming with constant datastreams most people aren't used to except for maybe people involved in low level networking layers, hardware, some signal processing, and the financial industry (probably plenty of others I left out, but still probably a very small subset of developers dealing with this.). We've actually been through all of those types of ideas and the best performing solution in this scenario but the ugliest code was is an if statement checking the high access fields and then delegating to the standard hash for fields accessed less frequently.

I'm sure there are some new ideas out there, we've come up with a lot here but none would perform better than either the if statement which is ugly and less efficient when very large, or the array lookup which would require the compiler change. (Unless we just plain changed to direct member access but we usually have to relate those members back to field id's... but if we got out of that completely we would just have a lot of very specific objects (we were trying to implement a generic object that can optimize specific fields and fallback for unknown fields. (We are writing generic data objects but need to be optimized for specific types of data because the hash is too much overhead for our high access rate.)

We actually test are system at data rates under stress up to 20,000 or more messages per second and each message could be one of these objects with up to 100+ fields in a hash. Some messages may only have as few as 10 fields stored in the hash but have disparate field id's (the hash code that could be any value). So allocating an array to the max that enum value could be which could be very high for 20,000 messages per second would obviously be out (as if the range were just 1000 allocating an array of references for of the size 1000 for each message would mean $1000 * 20,000 = 10,000,000$ object references * 4 bytes on 32bit intel which would be 40MB). The other indexing schemes would require a binary search, linear walk, or possibly skip list implementations which are all slower than the if statement method I mentioned (but a bit cleaner I must say), and obviously slower than the array lookup since there

Re: Idea for ECMA/C# Standard – compile time hash for performance

is both a search and lookup in the other methods (Methods#2) you mentioned.

We've got a pretty good handle on some exotic datastructures and we've tried quite a few including a look-aside type addressing as you mention, priority heap trees, skip lists, binary trees, red-black trees, custom hashes, priority queues, etc. When it comes down to it the fastest is no lookup at all which means non-generic specific code, the next best is hard coded checks like if's for small numbers of items (like jump tables, or ordered jump tables), and depending on the number of items probably a direct memory lookup like an array lookup of known size of each element. Anything more complex will likely be slower, or if not, non-generic. The other idea we tossed around was a hardware accelerator for certain memory accesses, like a private cache we could control that could not be impacted by other threads or processes on the machine with some special hardware functionality, don't think we'd have any better luck at getting that implemented than the compiler change though :(

"Bruce Wood" wrote:

>> As I mentioned there a probably plenty of otherways to improve
> performance, but none as simple or as straightforward as providing this
> compiler option
>
> I don't mean to be confrontational, but I see nothing simple or
> straightforward about having Microsoft propose changes to a language
> standard to ECMA, have it approved, and then have their army of
> programmers implement a change to the C# language, while all the while
> ensuring that his doesn't break any code that's already out there...
> doesn't seem simple to me. :)
>
>> and since we see this problem come up all the time
>
> Perhaps in your business. I've never had this problem before and never
> been anywhere else where they had it. That's why I'm fishing around
> trying to cook up another solution for you, because I think that this
> is a very specific problem to your domain, and your odds of having it
> fixed by an addition to the language definition are about nil. Not that
> I don't want you to succeed... I just think that it's highly unlikely.
> :)
>
>> Since in the course of one second we could have as much as 13,000
> messages each requiring some of the fields if we did what you had
> earlier suggested and just burned memory in one second we would eat up
> easily 1.3MB and that is not even the worst case scenario.
>

Re: Idea for ECMA/C# Standard – compile time hash for performance

- > You've completely lost me here. How can the number of messages arriving
- > chew up more and more memory in a mapping that is fixed at compile
- > time? The only memory lost by using a sparse array would be one pointer
- > (either 4 or 8 bytes, depending upon whether you're running 32-bit or
- > 64-bit architecture) for each enum value that doesn't map to anything.
- > That loss is fixed, and not affected by any messages arriving. It's
- > just a different way of performing the mapping that your current hash
- > table gives you, unless I'm missing something. (Highly likely.)
- >
- > My first two choices for an alternate data structure would be:
- >
- > 1. Replace the hash table by an array of pointers, and use the enum to
- > index directly into the array. As I pointed out, this is just the same
- > thing as your hash table, except that it wastes either 4 or 8 bytes for
- > each value that either has no equivalent enum or has an enum but that
- > enum doesn't map to anything (but then the latter would have overhead
- > in your compiler-change scenario as well). This is the classic
- > speed-for-memory tradeoff.
- >
- > 2. Replace the hash table by an array of byte or short values
- > (depending upon the maximum number of enums that will map at any one
- > time). Look up the enum, get a value which is an index into another
- > array. This will cut the memory waste in half (or in quarter), but at
- > the cost of limiting the number of enums that can map. This is an
- > unlikely solution unless you're running a 64-bit architecture and you
- > know that very few enums will be mapped at any one time.
- >
- >
- .

• *Follow-Ups:*

- ◆ *Re: Idea for ECMA/C# Standard – compile time hash for performance*
◇ From: Bruce Wood

• *References:*

- ◆ *Idea for ECMA/C# Standard – compile time hash for performance*
◇ From: wxs
- ◆ *Re: Idea for ECMA/C# Standard – compile time hash for performance*
◇ From: Bruce Wood
- ◆ *Re: Idea for ECMA/C# Standard – compile time hash for performance*
◇ From: WXS
- ◆ *Re: Idea for ECMA/C# Standard – compile time hash for performance*
◇ From: WXS
- ◆ *Re: Idea for ECMA/C# Standard – compile time hash for performance*
◇ From: Bruce Wood
- ◆ *Re: Idea for ECMA/C# Standard – compile time hash for performance*
◇ From: WXS
- ◆ *Re: Idea for ECMA/C# Standard – compile time hash for performance*
◇ From: Bruce Wood

Re: Idea for ECMA/C# Standard – compile time hash for performance

- ◆ [Re: Idea for ECMA/C# Standard – compile time hash for performance](#)
 - ◇ From: WXS
- ◆ [Re: Idea for ECMA/C# Standard – compile time hash for performance](#)
 - ◇ From: Bruce Wood

- Prev by Date: [DeviceIoControl from C# ?](#)
- Next by Date: [any solution for my problem?](#)
- Previous by thread: [Re: Idea for ECMA/C# Standard – compile time hash for performance](#)
- Next by thread: [Re: Idea for ECMA/C# Standard – compile time hash for performance](#)
- Index(es):
 - ◆ [Date](#)
 - ◆ [Thread](#)