What are the primary differences between 'gc' and 'gccgo'?

Asked 9 years ago Modified 3 years, 4 months ago Viewed 56k times

113

What are the primary differences between the two popular Go compilers, 'gc' and 'gccgo'? Build performance? Run-time performance? Command line options? Licensing?

I'm not looking for opinions on which is best, just a basic overview of their differences, so I can decide which is best for my needs.

go gccgo

Share Improve this question Follow

edited Sep 15, 2014 at 13:07

asked Sep 12, 2014 at 15:11
Jonathan Hall
75.6k ● 16 ● 146 ● 191

1 Answer

Sorted by: Highest score (default)

138

You can see more in "Setting up and using gccgo":

> gccgo, a compiler for the Go language. The gccgo compiler is a new frontend for GCC.
> Note that gccgo is not the gc compiler

As explained in "Gccgo in GCC 4.7.1" (July 2012)

> The Go language has always been defined by a spec, not an implementation. The Go team has written two different compilers that implement that spec: gc and gccgo.
>
> - Gc is the original compiler, and the go tool uses it by default.
> - Gccgo is a different implementation with a different focus
>
> Compared to gc, gccgo is slower to compile code but supports more powerful optimizations, so a CPU-bound program built by gccgo will usually run faster.

Also:

> - The gc compiler supports only the most popular processors: x86 (32-bit and 64-bit) and ARM.
> - Gccgo, however, supports all the processors that GCC supports.
>   Not all those processors have been thoroughly tested for gccgo, but many have, including x86 (32-bit and 64-bit), SPARC, MIPS, PowerPC and even Alpha.
>   Gccgo has also been tested on operating systems that the gc compiler does not support, notably Solaris.
>
> if you install the go command from a standard Go release, it already supports gccgo via the `-compiler` option: `go build -compiler gccgo myprog`.

In short: **gccgo: more optimization, more processors**.

However, as commented by OneOfOne (source), there is often a desynchronization between go supported by gccgo, and the latest go release:

> **gccgo only supports up to version go v1.2**, so if you need anything new in 1.3 / 1.4 (tip) gccgo cant be used. –
>
> **GCC release 4.9 will contain the Go 1.2 (not 1.3) version of gccgo**.
> The release schedules for the GCC and Go projects do not coincide, which means that 1.3 will be available in the development branch but that the next GCC release, 4.10, will likely have the Go 1.4 version of gccgo.

---

twotwotwo mentions in the comments the slide of Brad Fitzpatrick's presentation

> gccgo generates very good code
> ... but lacks escape analysis: kills performance with many small allocs + garbage
> ... GC isn't precise. Bad for 32-bit.

twotwotwo adds:

> Another slide mentions that non-gccgo ARM code generation is wonky.
> Assuming it's an interesting option for your project, probably compare binaries for your use case on your target architecture.

As peterSO comments, Go 1.5 now (Q3/Q4 2015) means:

> The compiler and runtime are now written entirely in Go (with a little assembler).
> **C is no longer involved in the implementation, and so the C compiler that was once necessary for building the distribution is gone**.

The "Go in Go" slide do mention:

> C is gone.
> Side note: gccgo is still going strong.

Berkant asks in the comments if `gccgo` is what `gc` was bootstrapped from.

Jörg W Mittag answers:

> No, `gccgo` appeared after `gc`.
>
> `gc` was originally written in C. It is based on Ken Thompson's C compiler from the Plan9 operating system, the successor to Unix, designed by the same people. `gc` was iteratively refactored to have more and more of itself written in Go.
>
> `gccgo` was started by Ian Lance Taylor, a GCC hacker not affiliated with the Go project.
>
> Note that the first fully self-hosted Go compiler was actually a proprietary commercial closed-source implementation for Windows whose name seems to have vanished from my brain the same way it did from the Internet. They claimed to have a self-hosted compiler written in Go, targeting Windows at a time where `gccgo` did not yet exist and `gc` was extremely painful to set up on Windows. (You basically had to set up a

> full Cygwin environment, patch the source code and compile from source.) The company seems to have folded, however, before they ever managed to market the product.

Hector Chu did release a Windows port of Go in Nov. 2009.
And the `go-lang.cat-v.org/os-ports` page mentions **Joe/Joseph Poirier** initial work as well. In this page:

> Any chance that someone in the know could request that one of the guys (**Alex Brainman** - Hector Chu - Joseph Poirier) involved in producing the Windows port could make a wiki entry detailing their build environment?

Add to that (in Writing Web Apps in Go) **!光京 (Wei Guangjing)**.

Share Improve this answer Follow

edited Jun 1, 2020 at 13:02

answered Sep 12, 2014 at 15:14
VonC
1.3m ●530 ●4436 ●5275

4 golang.org/doc/go1.3#gccgo And more importantly gccgo only supports up to version go v1.2, so if you need anything new in 1.3 / 1.4 (tip) gccgo cant be used. – OneOfOne Sep 12, 2014 at 15:40

1 @OneOfOne good point, I have included your comment in the answer for more visibility. – VonC Sep 12, 2014 at 17:11

1 @twotwotwo good find. I have included it in the answer for more visibility. – VonC Sep 12, 2014 at 18:56

3 I read that gccgo implements goroutines by giving each goroutine a dedicated thread (as opposed to multiplexing many goroutines onto one thread). If this is still true then that can be a major difference to some people. – DragonFax Feb 23, 2015 at 21:16

2 @LeoGallucci The general idea in this answer stands with Go 1.11 ("gccgo: more optimization, more processors."). You can see a more recent illustration in stackoverflow.com/a/46970176/6309. – VonC Feb 14, 2019 at 10:23

golang

What are the Differences Between Gc and Gccgo?

5 months ago • by Komal Batool Batool

Go is a popular programming language that offers high performance and easy concurrency and when it comes to compiling Go code, there are multiple options available. Two of the most frequently utilized compilers for Go are **'gc' and 'gccgo'**. While both aim to compile Go code, they have some fundamental differences in their design, implementation, and performance.

In this article, we will explore the key differences between **'gc' and 'gccgo'**, and how they can impact your Go development process.

What is 'gc'?

The '**gc**' is a compiler that is used to translate the Go programming language into machine code that computers can understand. Due to how quickly and effectively it compiles code, it is very well-known among developers. The incremental compilation feature of '**gc**' means that it can recompile only the changed parts of code, making the process faster and more efficient. Another great feature of '**gc**' is its garbage collector, which helps manage memory allocation and deallocation.

What is 'gccgo'?

The '**gccgo**' is another Go compiler, but it's different from '**gc**'. It's part of a tool called the GNU Compiler Collection (GCC). People might choose to use '**gccgo**' when they already have GCC installed. '**gccgo**' does the same job as '**gc**' by taking the Go code and turning it into machine code.

Difference Between 'gc' and 'gccgo'?

When it comes to Go compilers, **'gccgo' and 'gc**' have some significant differences that developers should be aware of. How they manage waste collection is one of the biggest differences. GCCGO makes use of a moderate garbage collector. which may not free all unused memory. In some circumstances, this can result in higher memory consumption and worse performance. Additionally, '**gccgo**' may generate less optimized code than '**gc**', leading to slower performance in some scenarios.

Another notable difference between ‘**gc**’ **and** ‘**gccgo** ’ is the level of support for new language features. The ‘**gc** ’ is typically more up-to-date with the latest Go language features and improvements, while ‘**gccgo** ’ may lag in some cases. This can be important for developers who want to leverage the newest language features in their projects.

The ‘**gc** ’ compiler only supports x86 (32-bit and 64-bit) as well as ARM, the two most common processors, whereas ‘**gccgo** ’ supports any processor that GCC supports. Some of those processors, such as x86 (32- and 64-bit), SPARC, MIPS, PowerPC, as well as Alpha, have undergone thorough testing for ‘**gccgo** ’, though not all of them have.

Final Thoughts

It’ s crucial to take your unique demands and surroundings into account while choosing between ‘**gc** ’ and ‘**gccgo** ’. The ‘**gc** ’ is the recommended choice for most developers, as it’ s reliable, efficient, and well-supported. However, ‘**gccgo** ’ may be a better option if you are already using the GCC toolchain or need specific features. When deciding, it is important to consider factors such as performance, garbage collection, and language feature support. Go is a strong and effective programming language that is excellent for contemporary software development, regardless of the compiler you select.