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

up vote33down vote**7**
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.

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.

> 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**.