2024-11-04

Video recording

Elias Naur: Support out of tree port.
https://github.com/golang/go/issues/46802#issuecomment-869084259

- Cherry Mui : We're trying to make the runtime more modular, but still a long want to Go
- Ian Lance Taylor : Can it be driven from the outside
- Dmitri Shuralyov : We don't have examples for out of tree runtime ports, but we have examples for out of tree compiler ports. It would be good to have strategies proposed from the outside, and how they want to maintain
- Michael Knyszek : Refactoring the runtime is a long-term goal. If some of that can happen from the outside that actually be really beneficial.

Elias Naur: Some embedded ARM processors run only on Thumb2. An in-tree Thumb port? Replace the existing arm32 port? https://embeddedgo.github.io/

- Michael Knyszek : Concerns on testing. Maintaining physical machines is very costly
- Elias: Some machines do support both arm32 and Thumb. This it should be testable on the existing ARM infrastructure.
- Keith Randall : The compiler is easy to add another GOARCH. All the assembly in runtime, math, crypto, etc. are the sticking points. Can we use the same assembly source and just change the assembler to generate Thumb?
- Elias: ARM has something called unified assembly where you just specify the assembly instructions and in thumb mode expanded to more than one instruction.

Eric Groose: Running smaller build machines and LUCI drives load very high. Specifically openbsd/ppc64 port

- Michael Knyszek : Are you exposing multiple bots to the system and it's running multiple tasks simultaneously? We can control how many builds/tasks we run at the same time. Will poke on the configuration on openbsd/ppc64 port
- Dmitri Shuralyov : If everything is configured right, you should only be having one of those happening at a time. But of course when a build is running it will look at the default number of CPUs you have and it will parallelize accordingly. If necessary maybe even add some support for reducing parallelism.
- Michael Knyszek : It does run nonstop, as we have so many repos and configurations. If some configurations are less important, maybe we can reduce that.

Rhys Hiltner: Trying to fix runtime mutex implementation. So many threads wake up and try to read the cache line, and end up with congestion collapse on many-core machines. Have a few CLs that solve this with different trade-offs. What trade-offs are acceptable?

- Michael Knyszek : Feel the pain. Had the same issue when working on the pacer. When we see a regression, try to get a concrete explanation, and see if it will occur in real programs. Some may be acceptible. On one of your CL, we see 16 vCPU builder win across the board, whereas 88 vCPU builder regress, which is the opposite of what we expect.
- Rhys: Tile38 does Gosched regularly. There don't seem to be benchmarks that are operating at a point of failure.
- Michael Knyszek : Benchmarking is hard. It is hard to find benchmarks working that way. People work around it. We always want to improve our benchmark suite. If you can find a reason that this doesn't behave like regular Go programs, that's a really good piece of information to know
- Rhys: The benchmarks measure only how fast can you go, not CPU cycles, or how much energy it uses
- Michael Knyszek : We are working on it
- Rhys: Scaling apps. People over reserve resources because they're afraid of falling over the edge of performance cliffs. The runtime mutex work is to make it safer to operate closer to the edge
- Michael Knyszek : We have users running on the edge, not for latency sensitive but throughput sensitive.
- Michael Knyszek : We can try tuning the parameters. Some parameters are based on old data.
- Rhys: Intel changed how long the PAUSE instruction sleeps. AMD is different, and ARM64's YIELD instruction is just no-op. We also don't have ARM64 performance builder.
- Michael Knyszek : Working on it.

Elias Naur: What the pacer does

- Michael Knyszek : Decide when the GC runs. We have a concurrent collector. The GC needs to decide when to start the concurrent collector so that it finishes is by the targets set by GOGC. It's kind of like a feedback loop mechanism estimating GC overheads and balancing a whole bunch of stuff to be able to decide where is the right place between the live heap and the target heap
- Elias: Try to look into the future

Elias Naur: Performance of iterators

- Cherry Mui : We've made a few improvements recently. There are probably still more room to do with more inlining and better handling closures.
- Elias: Nested iterator https://github.com/golang/go/issues/69411
- Cherry Mui : We don't handle method closures well. There is a phase ordering problem

2023-09-21

Video recording

Jon Forrest: Am I right that the Go build environment may build more stuff than is necessary? If Go produced artifacts for each .go file, could an equivalent Makefile do better?

- Ian Lance Taylor: All of the files in a package are built together.

Evan Phoenix: I'm helping maintain the WASI/WASM support in the compiler. We're planning to do a 32-bit port for the WASM backend. Pretty much every language compiles to 32-bit words, so for interop it's a necessary evil. I wanted to confirm that there aren't discussions to deprecate 32-bit pathways.

- Austin: There hasn't been any talk of deprecating all 32-bit platforms. We sometimes talk about specific ports.
- Austin: I thought WASM was already 32-bit?
- Evan: Originally we thought WASM would move toward 64-bit, but in practice everything has stayed 32-bit. Most runtimes support both, but JS interop expects to work with 32-bit heap pointers.

Evan: How to support WASM export? Rather than just "main" as an entrypoint, there would be arbitrary entrypoints. How to fit goroutines into that? TinyGo doesn't let you start goroutines from an exported function, but that's really hard to work with.

- Austin: I would expect this to be similar to C calling Go on a non-Go thread. We just conjure up a goroutine.
- Evan: It's harder because we don't have threads. We could support this once WASM threads are a thing.
- Austin: Short of having threads, you might just need to keep reinvoking the Go driver.
- Evan: It's easy enough to tell people that no more goroutines are running when main returns from the driver. It's a much weirder mental model to say "after you've called this exported function, you have to keep invoking the scheduler or your goroutines are all stuck."
- [Discussion of WASM threads vs fibers.]

Jon: Ian, do you have write access to /usr/local/go?

- Ian: There's something broken with buildmode shared writing to the local Go tree.
- [Discussion of philosophy of static linking vs dynamic linking.]

Fredrik Strömberg: I would like to run Go on a very small RV32I with 128KB of memory. I was thinking I'd need to port to rv32, and strip basically everything from the runtime (goroutines, GC).

- Michael Knyszek: Anything is possible. Historically, we've been reluctant to support a "no OS" GOOS because those environments look so different and the value of an OS is that it abstracts much of that away. It's certainly possible, like the Tomago people have shown. You can make it work. We've also been reluctant to upstream this sort of support.

The fact that TinyGo exists gives us a strong reason not to because there *is* an alternative.
- Fredrik: The GOOS=none discussions all seem to focus on having no OS. What about a GOOS=abi0?
- Austin: note that abi0 is a calling convention, not the set of services.
- Fredrik: Sorry, I was thinking a Linux API but with an ABI0 calling convention.
- Michael: I see, so basically GOOS=linux, but you have to provide the implementations of all of the syscalls.
- Michael: Wild idea: GOOS=package path, where we sub in that package for the runtime's syscall6.

2023-07-20

Video recording

Mark Diener: Question about bootstrapping failure:
cmd/vet: ambiguous import: found package cmd/vet in multiple modules:
cmd (/Users/sdev/godev/gitrepo/src/cmd/vet)
(/Users/sdev/godev/gobin/src/cmd/vet)

Rhys Hiltner: lock contention in a few parts of the runtime
- https://go.dev/issue/61426
- GOMAXPROCS=96
- Seems to be metastable: the fast path is very fast, but if we're running out of work, everything starts contending on the lock and things get very slow.
- Michael Knyszek: The trace shows *a lot* of idle workers. Those tend to exacerbate any scalability issues. You're right that if the pacer is doing its job, the background workers shouldn't be contending on the credit pool. I would love to just delete them, but there are reasons we can't just delete them. A lot of the time I've noticed idle workers are doing useless work. Maybe they make the GC finish a little earlier, but you don't actually need this much parallelism to run the GC, so you just end up with a lot of burned CPU cycles.
- Austin Clements: We can delete them.
- Michael Knyszek: There's one case where we need them. But if we fixed the issue with fractional workers, we wouldn't need them anymore.
- Michael Knyszek: This might be easy to fix. We could have a try lock around the assist queue.
- [Discussion of work.full contention]
- Austin Clements: work.full has a high write rate and a very high read rate, so it collapses pretty badly.
- Michael Knyszek: I think we currently collapse on work.full much faster than we'll collapse on a trylock.
- Rhys: The other scalability problem is that each size class has a lock, but this particular program only uses a few of them. (https://go.dev/issue/61428)

- Michael Knyszek: Those are the sweep queues. That used to be a locked linked list, which was much worse. Now it's a mostly lock-free data structure. That still plateaus at ?12? Cores, but I wasn't able to get it to collapse up to 48.
- Keith Randall: Part of the problem is that pages are small and you're going to this for each new page. You could take more than one page at a time.
- Rhys: Does this have to be so global? Could we use a tree? Pull a batch of pages from a global root into a level shared by just a few Ps.

Quim Muntal: I did SEH unwinding in Go 1.21. For Go 1.22, I want to extend that to exceptions thrown in cgo functions. Exceptions first go to the vectored handler, and then up the stack through SEH. Right now, Go always registers a VEH handler, which always just crashes the application if it's not handled by Go. That means if C code throws a structured exception, even if it's set up to handle it, Go will immediately crash the program in the VEH handler. It should only crash if it makes it all the way to the cgo boundary. Prototype.

Austin: We're rewriting the inliner. Expect an issue and high-level doc soon.

Austin: Thinking about moving runtime -> internal/runtime. Will break things that make assumptions about runtime symbols, including delve and things processing tracebacks.
- Alessandro Arzilli: I would appreciate being cc'd on the CL so I know when I have to un-break delve
- Austin: Definitely
- Austin: The big move won't happen until 1.23 at this point, since it definitely has to be early-in-cycle. But I plan to land a bunch of cleanups in preparation before then.

Austin: We're thinking about a standard interning API, like a std version of go4.org/intern, which is transitively used by 0.1% of modules and depends on GC internals.
- Michael: You can intern any comparable value and you get back a token. Two tokens are == if and only if their keys are ==. A token is one word and comparison is only == on that word. If the token becomes unreachable, we can delete that entry from the map (without violating any of these properties!)
- Rhys: This is used in Tailscale's netip package, which is now in std and archived. Does that mean this isn't a problem any more?
    - Michael checked after the meeting and we learned that not only is it in std, we basically have a copy of go4.org/intern in internal/intern that none of us on the C&R team knew about!

2023-05-18

Video recording

Mark Diener: https://github.com/golang/go/issues/60078 Proposal for shrinking uncertainty of for loops. What is the formal process?

- https://github.com/golang/proposal#readme
- Mark: What counts as a small or large change?
- Austin: The main question is the impact on the ecosystem.
- Mark: Is this specific change going to affect debuggers?
- Alessandro Arzilli: I don't expect any impact.

Jon Forrest: Dynamic linking Go programs. Given the speed of the Go toolchain, it's not a critical issue. But when I tried to do this, it didn't work. Issue 47788. When you try to do a go build as a normal user, it tries to write std to a place a normal user can't write. Example: Ansible creates small Python programs it sends to another host. Imagine doing something like that with Go.

- Austin: I would expect something like that to use plugin mode.

Ryan Berger: Vectorization! It seems like the ARM folks are interested in vectorization in Go.

- Keith Randall: One of the latest SIMD proposals: https://github.com/golang/go/issues/60149
- Austin: IMO, auto-vectorization is a poor fit for Go for many reasons. Fine if you're not trying to write vector code, but not good if you're trying to write vector code. Secret handshake between programmer and compiler. Performance cliffs.
- Ryan: Java has had a lot of those problems. But it also has a Vector API.
- Austin: We also lack expertise in high-performance numerical vector code within the team. E.g., two ends of the spectrum: platform-dependent or independent? We can't agree in part because we don't have that much experience.
- Michael Knyszek: Looking at discussions in other languages, it's hard to watch. Rust has had low-level vector intrinsics for a while. They want to add a general API, but that's been in discussion for a very long time. People who really care about performance are just going to use the low-level instructions anyway. "What about all of these instructions? Variable-length vectors?" It gets so complicated so quickly. I'm not sure it makes sense to do anything beyond exposing all instructions as intrinsics, or asking everyone to write Go assembly.
    - https://github.com/rust-lang/rust/issues/86656
    - portable SIMD is the keyword
- Austin: My current thinking (which is not worth much) is that we should provide per-architecture operations, but on a common type system.
- Robert Griesemer: One could imagine adding operations on slices.
- Austin: That only gets you access to uninteresting operations.
- Austin: The type system question is messy. The natural expression of "Add" is to take two slices and return a slice. But you really want to pass those by value, and are you going to heap-allocate the result? Are they the same length?
- Robert: In practice, numerical code builds on top of existing highly optimized building blocks. Maybe we need to just have those building blocks.
- Austin: That's kind of gonum.
- David Chase: Aliasing. You could compile a function two ways.
- Ryan: LLVM does that, and checks for pointer equality to call the right one.

- David: My other question is these small types being used in ML. uint4 and such. Are we going to have to add these interesting scalar types just for interaction with these kernels?

… The discussion floated to custom scalar types like decimal and big int and how annoying and error-prone they are to use …

- Robert: math/big is error-prone because it does in-place updates. math/big operations used to just return new values, but that was incredibly slow.
- Austin: You could make values immutable and build up a tree and come up with an optimal memory reuse plan when you need to evaluate it. Or you could get more help from the compiler. Or if we could do the fancier up-stack escape analysis that we've been thinking about for a long time, maybe it could just return a new slice and we could handle that efficiently.
- Robert: The "plan" approach is interesting, but the tree could get really big and I think you'd still want help from the programmer.
- Austin: What about the escape analysis-based approach? I'm thinking something like FORTH stacks where we decouple the code and data stacks and let them get a bit out of sync. In general, you have a goroutine-local heap you can bump-pointer allocate into. You can do a local tracing GC on that since it's all within the goroutine. Or, if you can somehow prove (I haven't worked out the details) that a function returns without returning any of these local allocation, it can just unwind the bump pointer on return.
- Robert: I think it would still be less efficient because you wouldn't be reusing memory as efficiently as in-place operations.
- Austin: Maybe. Definitely during a single operation the footprint would be a little larger, but overall we'd be able to reuse memory quite efficiently, so I don't think the working set of most operations would be much bigger than the in-place version, and would probably still fit in L1.

2023-02-23

Video recording

Austin Clements: Fun with rewriting stack traces and going down related rabbit holes.

David Chase: Experimenting with stronger alignment of strings, interfaces, and slices so we can use double-word loads and stores.

Austin: We've been thinking about how to better share definitions like constants and types between the runtime and compiler.

- Keith Randall: We've also been looking at generating the compiler builtins directly.

Quim Muntal: I have CLs to drop implicit NOFRAME in amd64. This might break external third-party users and expect NOFRAME to be implicit. Today, if a function is NOSPLIT and has 0 stack, it's implicitly NOFRAME and doesn't get a frame pointer.

- Austin: I think it's only if you're making SP-relative references outside your frame.
- Michael Pratt: It looks like your amd64 patch was already merged. At some point in the next few weeks, the internal Google Go team will test everything at tip and we'll probably see if it breaks anything then.
- Michael Knyszek: If we're worried about breaking things and this is something we want to do long term, we could do this simultaneously with a carrot like giving people stable regabi. I know that doesn't help with the actual use case for these, which is to get unwinding. In other places we're also trying to move toward frame pointer unwinding. We're also going to run into problems with user assembly code smashing the frame pointer. Maybe this is one way to find out.
- Keith Randall: I had a CL a long time ago that would make the compiler access all arguments off of FP instead of SP. If assembly smashed the FP, it would return with a bad FP and the caller would quickly fail. That quickly found a lot of issues. I could revive that.
- Austin: Could vet check for assembly code that's smashing the FP?
- Keith: I was seeing assembly code smashing the FP it saved on its own frame. That's harder to check for.

Daniel Marti: Let's have a reminder set on the calendar invite.

Quim: I was trying to call some Windows assembly functions from Go, but they expect a 16 byte aligned stack and some other things. It would be useful to have an ABIGCC next to ABI0 and ABIInternal that takes care of saving registers and aligning the stack and fetching the TLS. Right now I have to write assembly for each architecture to glue the Go and GCC ABIs.

- Quim: Example is time code, which we have implemented in assembly for amd64 and 386, but not arm64.
- Austin: You also need to do a stack switch if we don't control the stack size of the target code. We have the libcall mechanism, which is for lightweight calls to the system ABI.
- Keith: I'm not sure if that exists on Windows right now.
- Austin: stdcall on Windows.
- Quim: Those require a G, which we don't always have. Example usleep2
- Austin: usleep2 requires the caller to already be on the OS stack. You could imagine parallel functions to stdcall that place more onus on the caller. It's unfortunate that stdcall requires a G because it's not for interesting reasons.
- Austin: I've been wondering for a while if the M should be in the thread-local register instead of the G. We always have an M.
- Quim: That's not necessarily true, like if you're coming into Go without any TLS.
- Austin: In general, if your choices are to make the runtime deal with more weird situations or make there be fewer weird situations, do the latter.

Quim: I have a WER proposal for if processes crash.

- Austin: IIRC the objection was that it phones home to Microsoft.
- Quim: It's opt-in in the sense that you have to set GOTRACEBACK=crash. And there are system-wide ways to opt out.

2023-01-19

Video recording

Daniel Marti: I've seen people have been working on binary sizes. I was looking at linker deadcode, and it's all-or-nothing if there's any reflection.

- Matthew Dempsky: I imagine the linker could see what triggered the reflection.
- Daniel: I was only hoping for package-level. Function is even better.
- Keith Randall: We could know that all MethodByName calls take a constant string and do better.
- Austin There's still a cliff
- Keith: It's still better
- Daniel: If I have a lot of dependencies, I mostly want to know what's using MethodByName so I can look into dropping or patching it.
- Austin: -ldflags=-c I think

Rhys Hiltner: I'm interested in support for big machines. GOMAXPROCS of 36 48 96. Some problems I've run into is lots of Ps so lots of run queues so finding work is slow. The latency between when a goroutine becomes runnable and starts running is longer on large machines. Some Ps are idle for a long time but get checked every time. The other crack is lock contention on channels. Are we using unusually large machines or doing something strange?

- Michael Pratt: I've investigated high GOMAXPROCS. There are a lot of things that are linear with GOMAXPROCS. I saw things getting bad at 256 Ps, but measurable at 96. I did some work to improve this. Long term, we've brainstormed getting rid of Ps/GOMAXPROCS or some more scalable system. 256 is probably on the upper end of what people normally run. We do want to improve that. Machines are only getting more cores, so it becomes more pressing every year.
- Michael Knyszek : I'm surprised you're not seeing issues with *other* locks because they aren't "scalable" locks. E.g., the allocator flattens out at ~10 cores. There are also cache scalability issues and many small costs you probably wouldn't even see in traces. We have thought about making the default runtime lock be scalable. That might help with channels, too.
- Austin Clements : Scalable lock soap box
- Keith Randall : WRT channels, we've had non-blocking channel designs in the past.
- rhys@twitch.tv : The problem in this case is the channel is completely full or completely empty, and they get spread out across all of the Ps, and every P needs this one lock and it can't run any other goroutines because it doesn't know about the lock.
- Ori Bernstein: I'm curious why that didn't go in
- Austin Clements : Fine-grained locking will get you about the same result as non-blocking algorithms with a lot less pain
- rhys@twitch.tv : What does it mean to get rid of GOMAXPROCS?
- Michael Knyszek : Basically GOMAXPROCS=infinity and the runtime tries to figure out the natural level of parallelism. It tries scheduling things and if it can schedule more, use

more parallelism. If things stop scheduling, dial back. Strawman: Use an idle priority thread.
- Austin Clements: Also separating resource pool of Ps from semaphore behavior.

Jacob Moody: Ori and I are trying to make things easier for Go on Plan 9, especially on ARM64. It seems like there needs to be some “spring cleaning” before we add another Plan 9 architecture.
- Michael Pratt: I can think of two big issues. Builders: more builders or more reliable builders. I’ve seen proposed getting Plan 9 running on a Cloud.
- Ori: GCE for amd64 and 386 is easy. For ARM64 (on GCE) it’s more of a pain, though I don’t think it would be a giant task.
- Jacob: AFAIK, all the existing builders haven’t been updated in 10+ years. Perhaps it’s time we set up some 9front runners to run what we have now.
- Michael Pratt: Generally, it’s up to the secondary port maintainers what versions of the OS are best to support and test. The other issue with Plan 9 is that the builds are just very flaky. There are a lot of failing tests. Everyone would be happier if they were fixed or skipped on Plan 9. You can’t look at the dashboard and tell if new commits are causing problems.
- Jacob: I don’t think I’ve had too many inconsistent tests failing lately.
- Ori: I’ve seen some flaky things.
- Austin Clements: Before the new port policy, debugging was a pain. That’s probably less now. Also things that are missing from Plan 9 that other OSes have which can get in the way of the runtime. Virtual memory management examples.
- Jacob: If there are issues with the Plan 9 port, it would be nice if there was a way to loop us in.
- Ori: It’s been the 9legacy group doing most of the Go support lately, so we’re not in the group.

rhys@twitch.tv: There’s an app I support that serves latency-sensitive requests. 10–20ms. To serve the requests, it needs a bunch of data in memory that it periodically reloads. Live heap is 9GB. When it reloads it loads 1GB at a time, which comes as a surprise to the pacer, which is used to serving the low-latency requests. That causes the pacer to run a GC cycle with an expensive assist load because it suddenly came up short 1GB. That slows down all goroutines that allocate, including all the request serving goroutines. That causes tail latency. We had scaled the cluster up but that’s expensive. We added runtime.GC calls to just the right place, but that’s tricky. Instead of trying to find just the right places and keep them up to do, we built a “user space pacer” that tracks the heap size and tries to figure out when the runtime will start the cycle and requests the cycle to start ahead of that. There’s now no assist load. We spend more CPU on GC, but we can use the rest of the CPU time in an interactive way. Two goroutines: one monitoring GC metrics every 100ms and triggering the other that calls runtime.GC.
- Michael Knyszek: The next release or so will have a heap-marked metric, which would make this a little easier.

- Michael Knyszek : It's probably too specific to give the pacer and API to say "something is going to happen soon." The pacer can't predict the future, but *you* have information you want to give to the GC. Even just a "hedge for the next few cycles."
- rhys@twitch.tv : We now have two ways to tell the GC not to worry, but even now the pacer schedules right up to the brink.
- Michael Knyszek : We're on the same page. Even random statistical noise can cause the pacer to go over. If the pacer were just hedging a little bit, I feel like a lot of these problems would be reduced.
- rhys@twitch.tv : The other number that would be useful is when the runtime is going to *start* the mark.
- Michael Knyszek : We could expose the trigger. I've been reluctant to. The trigger 10 years from now might not be useful.
- Austin Clements : The trigger logic changes over time and it's more complicated these days than "is the heap over X bytes?"
- Michael Knyszek : It's probably not a bad idea to just expose more.
- rhys@twitch.tv : I've only seen one app that really needed it.

2022-12-19

Video recording

Austin Clements: I'm the tech lead on the compiler and runtime team at Google. That's why I'm leading this meeting and sent out the invite. [intros of people who work on the C&R team at Google]

Brad Fitzpatrick: Windows is neglected (proxying on behalf of others)
- Quim Muntal: I work at Microsoft, working (somewhat) on getting Windows support.

Matt Layher: arenas? What are people doing with it? Works at PlanetScale on Vitesse.
- Michael Knyszek, it's behind a GOEXPERIMENT. It provides a benefit, mostly used for serializing and deserializing protos. It is memory safe, but a little bit icky from the POV of someone who works on a GC. Specific feedback would be great. Still haven't committed to anything.
- Austin Clements: Arenas bleed out of API boundaries, but they also give the runtime a lot more information about programmer intent. A plan/goal is to work on Go's RAM efficiency, both footprint and bandwidth to RAM. Hope to find optimizations that will make arenas less necessary.

Thepudds: A question about primary versus secondary platforms, in terms of relative performance. Atomic reads in particular. If a proposed change would make amd64 and "modern ARM" [arm64?], but would slow down s390 / zOS, how does the team consider the tradeoff?

- Austin Clements: We'd like to make amd64 and arm64 fast. We'd like to not get into a situation where we'd have to make that tradeoff.
- Thepudds: context is the grow work for maps, and the atomics for that [maybe re the rough proposals on "grow/evacuate in the read path"?]
- Austin Clements: Kinda depends on the numbers. For instance, we wouldn't want to accept a 0.5x performance penalty hit to s390. Maybe we'd keep the old implementation as a special case for some platforms.
- Thepudds: I'm working on SwissTables.
- David Chase: Does modern ARM mean ARM64 or 32 and 64 but with modern atomics?
- Brad F: many people with Raspberry Pi run 32-bit userspace+apps on them.

Ben Hoyt: General question on benchmarks. Run them on new releases. Go 1.20 was slower. But Go 1.20's PGO made it faster than 1.19 and was really easy to use. How do I dive into the regression? What is PGO _doing_?

- Austin Clements: At Google, we monitor the performance of Go too, and we didn't notice a slowdown. Can you share the benchmarks/data? PGO for Go 1.20 does inlining, relatively simple tweaks to our heuristics. Hope for more in Go 1.21.
- Ben Hoyt: What's the best way to poke at where the problem might be?
- Austin Clements: Bisecting would be best, maybe it was a single commit. Otherwise, a differential profile. "go tool pprof -base" will tell you what changed.
- David Chase: Are there benchmarks for applications, or are they microbenchmarks? And are they go-gettable, are they on GitHub and are they easy to install? I collect benchmarks, though my collection is mainly focused on the compiler.
- Michael Pratt: Please give us feedback on our choice of benchmarks [is this right?]
- Michael Knyszek: Looks like we don't have benchmarks for goawk. Look at links in the meeting chat. [From MK, https://github.com/google/pprof/blob/main/doc/README.md#comparing-profiles and https://perf.golang.org/dashboard/?benchmark=all, from Austin Clements, https://perf.golang.org/dashboard/]

Quim Muntal: Working on windbg support for Go. The main limitation right now is windbg doesn't know how to unwind Go stacks. Prototype

- Austin Clements: What does windbg need from Go?
- QM: need PDB information (function names, variable names, types, locations, like DWARF. Encoded as "CodeView", PDB is the container) Also unwinding.
- AC: we include DWARF on windows because that enables gdb. Seems wasteful to include both (but windbg is what Windows users expect).

Cuong Manh Le: PGO is in the tree already, do we have any concrete plans for landing it? Do we have any concrete plan for moving more compiler frontend to the SSA phase?

- Austin Clements: PGO is already landed in 1.20, though we consider it a preview. Shouldn't break your programs, since it currently only affects inlining. We plan to

remove limitations in 1.21 – it defaults to off in 1.20, for example. New default should be "auto", will look for a profile in "the" main package. Other reason for "preview"is it has bad effects on build scalability because (currently) each of N packages reads the N-sized profile information.

More compiler front-end into SSA; we have intentions, not so much plans.

- David Chase: There are complicated phase ordering issues. E.g., it would be really valuable to move inlining to SSA, but we do escape analysis after inlining so that would also have to move, and that breaks the complete parallelism of SSA currently. This makes it hard to approach it incrementally, so we need to wait for a "boring" release where it doesn't conflict with other big changes.
- Robert: Matthew Dempsky definitely has interest in doing inlining in SSA instead of in IR. Switching to unified IR has created opportunities to do more in the current organization, and we're looking into moving escape analysis earlier, partially.

Brad Fitzpatrick: Hitting x/sys access-cpu-flags problems on arm64. Could we just ask the Go runtime for this information? CL 458256?

Quim Muntal: Issue about using MSVC for cgo instead of Ming. It was difficult to infer the Go types for the C types using MSVC. I've found that we can generate 99% of the types from PDB info. Would something like that be accepted? For me, at least, it's a pain to not be able to use MSVC.

- David: We'd need to be able to test it.
- Quim: I read the PDB and map it to the DWARF types, so most of the tooling doesn't change.
- Austin: Are there other blockers to using MSVC?
- Quim: We're able to link using MSVC. This seems to be the last blocker, but I can't be sure.
- Than: We'd need to do testing of the race detector.

Jon Forrest: Question about "All files in a package must be in one directory". Is it a big deal to remove that restriction so packages can have subdirectories?

- Ian Lance Taylor: It's kind of a big issue. It's really a go tool issue, and that's all defined around packages being defined by directories. It's an opinionated choice Go has made and it would be difficult to change.
- Jon: As part of my efforts to learn Go and looking at applications written in other languages, I see applications that have clear package structures, but can't map to Go.
- Ian: You're right that the structure of a program in another language will not translate to Go.