Compress safepoint tables (not only for Wasm)

Attention - this doc is public and shared with the world!

Contact: clemensb@google.com
Contributors: Clemens Backes
Tracking Bug: https://crbug.com/v8/12401
Status: Inception | Draft | **Accepted** | Done

LGTMs needed

Name	Write (not) LGTM in this row
ahaas@	
jkummerow@	
you?	

Abstract

Safepoint information can get huge; we should try to minimize the encoded size.

Objective

V8 stores safepoint information for every call in generated code. This information can be huge (25-30% of generated machine code). We should try to reduce the encoded size of the safepoint tables to safe memory.

Value proposition to V8/Chromium

Reducing resource consumption. We would considerably reduce the memory needed to store compiled (WebAssembly) code.

Background

V8 uses safepoint information, to (primarily) encode which stack slots hold references (or "tagged values") in contrast to raw pointers or numbers. The garbage collector uses this information when "visiting" a frame that executes compiled code. Via the PC offset within the associated code object it finds the correct slot in the safepoint table. Each safepoint table entry contains:

1) the PC (4 bytes)
2) a deoptimization index (4 bytes)
3) a trampoline PC (4 bytes)
4) the bit vector for tagged stack slots (variable size)

The bit vector tells the GC which slots in the stack frame of that function holds a "tagged value", i.e. Smi or reference into the managed heap. The garbage collector then visits those slots, and makes sure to rewrite the references in case the referenced object moves.

Each entry currently needs 12 bytes for pc, deopt index, and trampoline pc, plus additional bytes for the bit vector.
WebAssembly code does not use the deopt index and the trampoline pc fields, and the bit vector contains mostly zeros. References into the heap were only introduced with post-MVP features like WasmGC or exception handling.

There is already some logic to "collapse" the safepoint table if all entries are identical (except for the PC), see previous tracking bug. To further safe size, this was later extended to trim trailing zeros from all entries, see other tracking bug.
The recent example of a big partner page that uses exception handling still generates huge amounts of safepoint table data though. **From a 63MB Wasm binary Liftoff produces 241MB of machine code, including 61MB of safepoint table data. TurboFan produces 228MB of machine code, including 66MB of safepoint table data.**

Design

There are multiple things that could be improved to reduce that size:

1) Collapse identical entries [yes]

If all entries were zero (i.e. no references into the heap), the entries would already be collapsed into a single entry. This does not happen if occasionally there is an exception reference on the stack. But looking at real-world code, most succeeding entries are still identical, so we could collapse them. This would require a change to the lookup logic to not look for a precise PC, but for the last entry that is not greater than the PC.

2) Remove unneeded fields [yes]

As WebAssembly does not use the deopt index and the trampoline PC (yet), we should not emit them for every single entry. Instead, the header could encode if those fields are present or not. The "fixed size" (the part without the bit vector) would not be a static constant any more, which should not cause major pain.
This optimization would not be specific for WebAssembly, but for any code which does not have any depot index and trampoline PC.

3) Use variable-length encoding [maybe]

Depending on how much safepoint table size remains after applying the previous optimizations, we could compress the safepoint table further by using only the needed number of bytes per field, i.e. if no PC offset is outside the [0, 65535] range, 2 bytes instead of 4 would suffice. The decoding logic in SafepointTable::GetEntry would become slightly more complex.

4) Use delta-encoding [maybe not]

To compress even stronger we could delta-encode the table, i.e. only store differences from the last entry. As a result, we would lose random access within the safepoint table. The current lookup in SafepointTable::FindEntry uses a linear search, so this would not be a limitation for now, but might make later optimizations to the lookup more difficult.
Also, encoding and decoding would obviously become more complex.
As the benefit of this is not expected to be large, this idea will not be implemented for now.

Security considerations

Wrong safepoint information can easily lead to stack corruption or heap corruption, but our existing gc-stress infrastructure should be able to find such issues.

Test plan

The feature will be tested by the existing tests.

Rollout considerations

This will just be a series of commits on the main branch.

Results

[added on Dec 20, 2021]
internal link: go/photoshop-code-size-2021
The following changes were implemented:
2) Remove unneeded fields
→ from 66MB to 30MB for TurboFan on x64 (60 to 24MB for Liftoff)
1) Collapse identical entries
→ from 30MB to 3.5MB for TurboFan on x64 (24 to 3.5MB for Liftoff)
3) Use variable-length encoding
→ another ~25% saving for safepoint tables

Safepoint table size **reduced by ~95%**.
Safepoint tables are stored as part of "code".
Code size: TurboFan reduced by 24% (x64) - 28% (arm), Liftoff by 18% (x64) - 24% (arm)