Bounds Checking Elimination

Alexandru Moșoi
linked from #14808
1.April.2016

Introduction

Bounds Checking Elimination (or BCE) is a general term for removing redundant bound checking. Normally a go program will panic when a slice or a string is accessed outside of its bounds. There are two types of bound checking: for indexing (a[i]) and for slicing (a[i:j]). The go compiler inserts these bounds checks at every access, but in most cases they are not needed and are redundant based on the context.

Bound checking is important because it provides a defense against buffer overflow attacks and catches a common programming error early. BCE is important because: it speeds up the code, makes the binary smaller. If binaries are slowed down by bound checks then developers will have an incentive to disable bound checking (using -gcflags=-B).

The target audience of this document are programmers focused on absolute performance. Generally, I tried to improve code examples I found in the standard lib and bug reports. If your favorite idiom is not handled properly please file a bug at https://github.com/golang/go/issues.

Some examples can be found in test/prove.go and in test/loopbce.go.

To see where the bound checks are introduced use -gcflags="-d=ssa/check_bce/debug=1"

```
% go build -gcflags="-d=ssa/check_bce/debug=1" boundscheck.go
# command-line-arguments
./boundscheck.go:17: Found IsInBounds
./boundscheck.go:22: Found IsSliceInBounds
./boundscheck.go:26: Found IsInBounds
./boundscheck.go:35: Found IsInBounds
./boundscheck.go:71: Found IsInBounds
```

NOT OFFICIAL RECOMMENDATION.

Which bounds checks are eliminated today

Duplicate checks

These happens when when the same index is repeatedly accessed. Removed by the “prove” pass. Examples:

```
var a []int
….
use a[i] // bounds checking inserted here
use a[i] // the bounds checking is a copy of the above, so it’s removed
use a[2*i+7]  // bounds checking inserted here
use a[2*i+7]  // removed, depending whether CSE can merge the values
```

```
var a []int
….
use a[:i]
use a[:i] // removed
```

```
var a []int
….
use a[i:]
use a[i:] // removed
```

Constant slice size with masked index

Removed during rewriting, “opt” and “late opt” passes.

```
var a[17]int
....
use a[i&5]   // 0 <= i&5 and i&5 <= 5 < 17 == len(a), bounds check removed
use a[i%5]  // not removed, i can be negative
```

Constant index

Removed by the “prove” pass. Special case.

```
var a[]int
....
if 5 < len(a) { use a[5] }   // 0 <= 5 and 5 < len(a), bounds check removed
```

Constant index and constant size

Removed during rewriting, “opt” and “late opt” passes.

```
var a[10]int
....
use a[5]   // 0 <= 5 and 5 < len(a) == 10, bounds check removed
```

Trivial bound checks

After https://go-review.googlesource.com/#/c/20306/

a[i:j] generates two bounds checks: for 0 <= i <= j and for 0 <= j <= cap(a). Sometimes we can remove one of them:

```
var a[]int
…
a[i:len(a)]  // 2nd bound check 0 <= len(a) <= cap(a) is removed
```

```
var a[]int
…
a[:len(a)>>1] // 1st bound check 0 <= 0 <= len(a)>>1 is removed because len(a)>>1 >= 0
```

```
var a[]int
…
a[:len(b)] // 1st bound check 0 <= 0 <= len(a) is removed
```

Induction variables based BCE

After https://go-review.googlesource.com/#/c/20517/ some bounds can be removed when the index is an iteration over the slice / string. See more examples in test/loopbce.go.

```
var a[]int
…
for i := range a {   // doesn’t work for strings
  use a[i]  // removed, i is an induction variable over a
  use a[i:]  // removed
  use a[:i]  // removed
}
```

```
var a []int
...
for i := 0; i < len(a); i++ {   // works for strings too
```

```
  use a[i]  // removed, i is an induction variable over a
  use a[i:]  // removed
  use a[:i]  // removed
}
```

```
var a[]int
…
for i := range a {   // doesn’t work for strings
  use a[:i+1] // removed, i is an induction variable over a
  use a[i+1:] // removed
}
```

Decreasing constant indexes

After https://go-review.googlesource.com/#/c/21008/.

```
var a[]int
…
_ = a[3] // one bound check
use a[1], a[2], a[3] // no bound checks

// or
a = a[:3:len(a)] // one bound check
use a[1], a[2], a[3] // no bound checks

// or
use a[3], a[2], a[1] // one bound check

// or
if len(a) < 3 {
    use a[0], a[1], a[3] // no bound checks
}
```

What is not removed

This is true with the current state of the compiler (commit 47c9e1).

what doesn’t work	**can be improved as**

var a []int
…
if len(a) < 128 { // not propagated to a[i]
  panic(“too small”)
	var a []int
	…
	if len(a) < 128 {
	  panic(“too small”)

}
for i := range a[:128] {
  use a[i] // not removed
}
	}
	c := a[:128:len(a)] // bounds check
	for i := range c {
	  use c[i] // removed after 20517
	}

var a []int
…
use a[0], a[1], a[2] // three bound checks
	var a[]int
	…
	_ = a[3] // early bounds check
	use a[0], a[1], a[3] // no bound checks
	// or
	a = a[:3:len(a)] // early bound check
	use a[0], a[1], a[3] // no bound checks
	// or
	use a[3], a[2], a[1] // one bound check

~~// fixed after 20654~~ ~~import "encoding/binary"~~ ~~var a []byte~~ ~~var u uint64~~ ~~….~~ ~~// 8 bounds checks, one for each byte~~ ~~binary.LittleEndian.PutUint64(a, u)~~	~~import "encoding/binary"~~ ~~var a []byte~~ ~~var u uint64~~ ~~….~~ ~~// 1 bounds check for slicing~~ ~~// PutUint64 gets inlined~~ ~~binary.LittleEndian.PutUint64(a[:8], u)~~

var a []int
var i int
…
if i < len(a) {
  use a[i] // not removed, i can be negative
}
if 0 <= i && i < len(a) {
  use a[i] // not removed, todo after 20306
}	

var a []int
…
if 0 <= i && i+2 < len(a) {
  use a[i+2] // i+2 might overflow int
}	

a := make([]int, 100) use a[0] // bound check if a escapes to heap	// bug: the compiler should handle this

// iterating in reverse
var a []int
…
for i := len(a)-1; i >= 0; i-- {
  use a[i] // one bound check
  use a[:i] // one bound check
  use a[i:] // one bound check
}	

Errata

21.March.2016 - Constant slice length assertion should be done using `a = a[:256:len(a)]` or `_ = a[:256]` and not `a = a[:256]` as suggested initially. The latter version may extend the length of the array beyond the original length.

Questions

Please redirect your questions to golang-dev@.