TangoLLVM: An LLVM Backend for the Go compiler

Tianxiao Gu, Rui Feng, Fengkai Liu, Nian Sun

ByteDance

Contents

- Motivation: Why We Optimize Golang at ByteDance
- Background: The standard Go compiler
- TangoLLVM: LLVM as a Backend
- Challenges: Support Garbage Collection in LLVM
- Evaluation

Why We Need LLVM for Golang?

- Golang: easy-to-learn, fast compilation and small binary
  - not designed for high performance

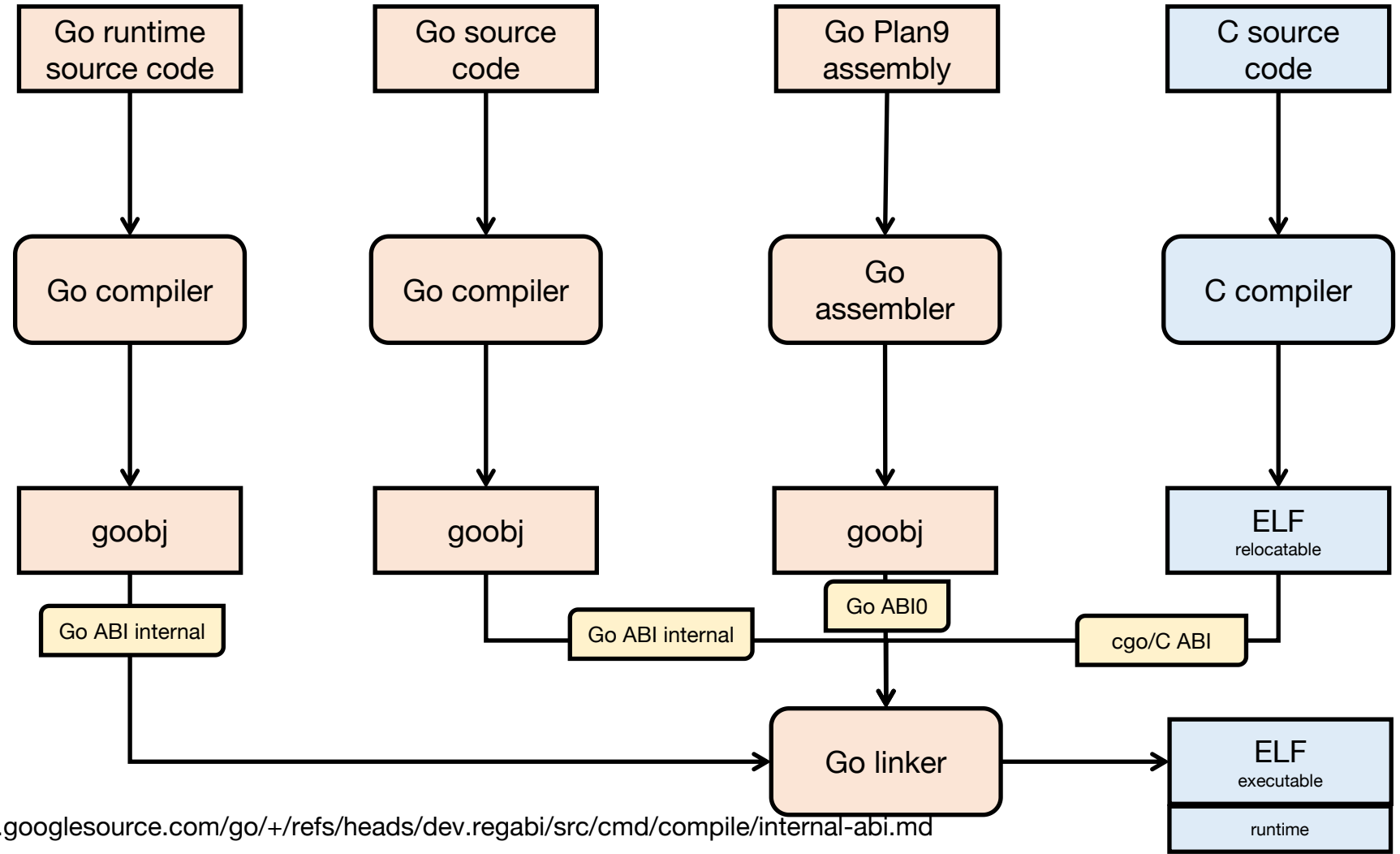
- Golang at ByteDance
  - the most widely used programming language for backend services
  - consuming a huge number of CPU cores daily

Existing LLVM-based Approaches

- TinyGo
  - for embedded environment
  - ***non-standard runtime and SDK***
  - https://github.com/tinygo-org/tinygo

- gollvm
  - not actively maintained
  - ***non-standard runtime and SDK***
    - cannot build almost all ByteDance internal applications
  - https://go.googlesource.com/gollvm/

The Standard Go Toolchain



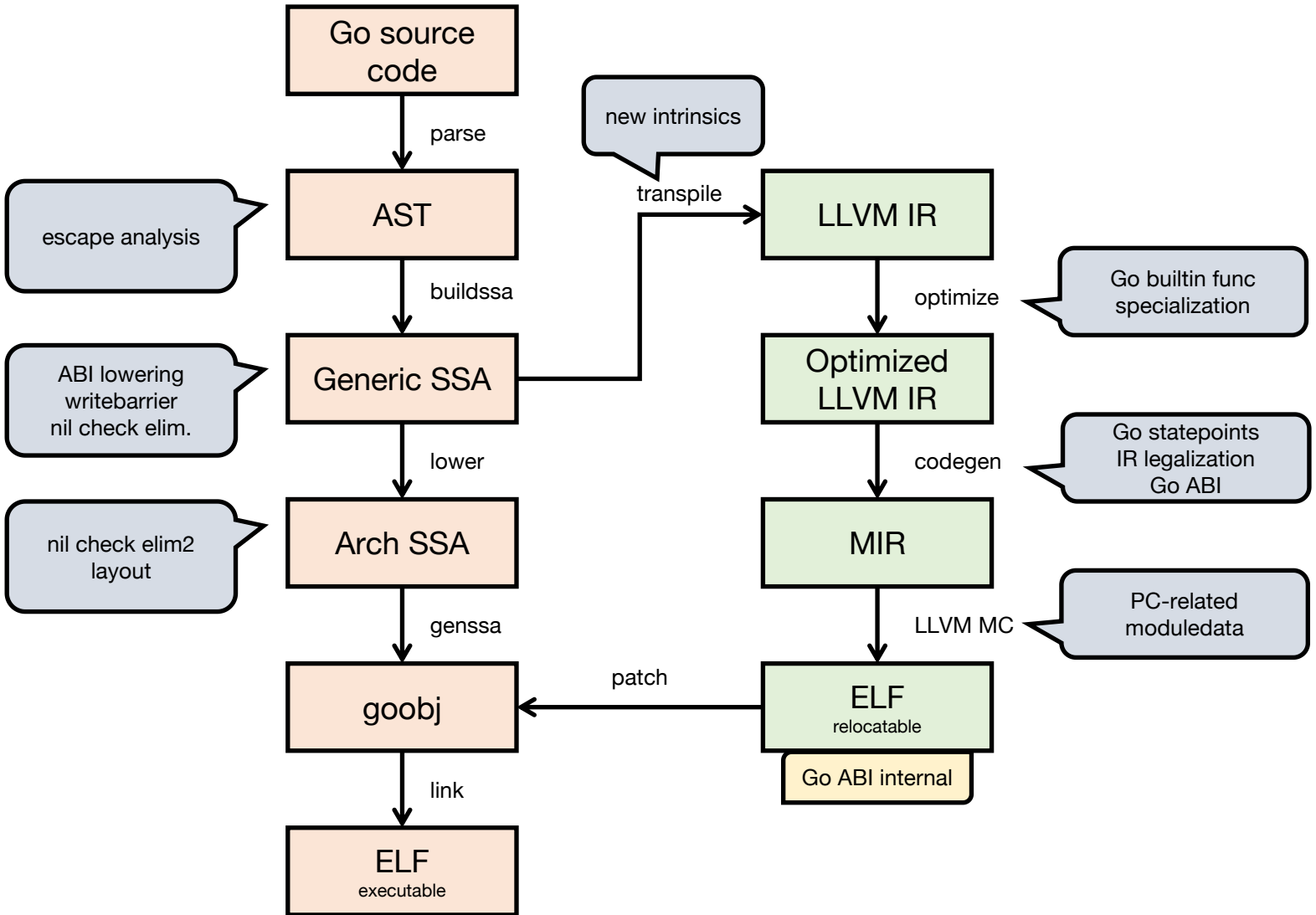
https://go.googlesource.com/go/+/refs/heads/dev.regabi/src/cmd/compile/internal-abi.md

The Go Runtime and moduledata

- moduledata:
  - metadata generated by the compiler and used by the runtime
  - Support stack walking, stack trace, garbage collection and reflection API

https://cloud.google.com/blog/topics/threat-intelligence/golang-internals-symbol-recovery/

TangoLLVM: LLVM as a Backend



1. run the Go compiler to generate a *goobj*
   - reuse pass before lowering
2. transpile Go SSA to LLVM IR
3. run the LLVM backend to compile a function again
4. extract symbol data and relocation from ELF/MachO and patch them back to *goobj*
   - avoid supporting goobj in LLVM
5. fallback to the Go compiler if we cannot patch

The Go Garbage Collector

- Go runtime can allocate objects on stack and in heap
  - Copy stack objects when stacks are growing
  - No moving/copy in heap objects

- Garbage collection happens at statepoints
  - Statepoint: a call instruction with a computed stack map indicating where to find ***GC pointers*** on stack

- Go ABI has no callee-saved registers
  - Values in registers are spilled to stack at a statepoint.

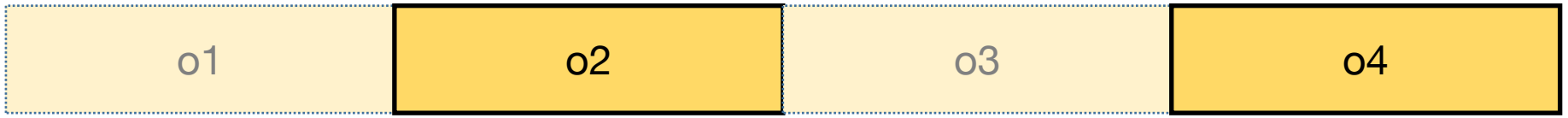
Support Go Garbage Collection in LLVM

- Based on GCStatepoints
  - https://llvm.org/docs/Statepoints.html
  - Identify pointers at LLVM IR
  - Keep pointer infos during codegen/lowering
  - Emit stack map in LLVM MC

- Challenges:
  - Most existing passes are not aware of GC
    - e.g., LICM can move a GEP pointer out of the loop where the base pointer of the GEP may be null

GC Pointers

- GC Pointer: an LLVM IR value with pointer types in TangoLLVM [$]
  - A GC pointer can be read by the garbage collector to locate the containing (stack or heap) object



- Valid GC pointer
  - Any pointer within live objects: [o2.base, o2.base + o2.size)
- Invalid GC pointer
  - bad pointer: null + offset
  - zombie pointer: o2.base + o2.size = o3.base
    - GC will mark o3 instead of o2

[$] unsafe pointers with integer types are discussed later

Identify Invalid GC Pointers

- Only GEP can lead to invalid GC pointers
  - invalid base pointer: null + non-zero offset
  - invalid offset: out of bounds

- Identify valid GC pointers in IR using static analysis
  - null pointer analysis/null pointer check elimination
  - pointer element size propagation [$]

- An invalid GC pointer is still safe if it does not live through any statepoint
  - move/clone GEP right before each use
    - may nullify LICM

[$] TangoLLVM is based on LLVM 19 and uses opaque pointer types.

Unsafe Pointers

- Pointer-like integer values are also GC pointers
  - IntToPtrInst and PtrToIntInst
    - ptr -> int -> int arithmetic -> ptr

- Track the validity of pointer-like integers using static analysis
  - fallback to the Go compiler if we cannot determine whether a pointer-like integer value is a valid GC pointer

https://pkg.go.dev/unsafe#Pointer

Evaluation

- The go1 benchmark suite
  - `baseline`: the Go compiler
  - `llvm`: llvm without lto and pgo
  - `lto`: llvm with lto only
  - `pgo`: llvm with lto and pgo
- Each benchmark was repeated 20 times

goos: linux
goarch: amd64
pkg: test/bench/go1
cpu: Intel(R) Xeon(R) w5-3435X

	go	llvm		lto		pgo	
	sec/op	sec/op	vs base	sec/op	vs base	sec/op	vs base
BinaryTree17-8	1332.0m ± 6%	1218.3m ± 1%	-8.53%	823.2m ± 1%	-38.20%	791.3m ± 14%	-40.59%
Fannkuch11-8	1.694 ± 1%	1.618 ± 0%	-4.50%	1.639 ± 0%	-3.25%	1.631 ± 0%	-3.75%
FmtFprintfEmpty-8	13.705n ± 1%	13.070n ± 1%	-4.63%	10.145n ± 1%	-25.98%	6.771n ± 1%	-50.59%
FmtFprintfString-8	26.49n ± 1%	21.92n ± 1%	-17.25%	19.93n ± 1%	-24.75%	16.59n ± 1%	-37.37%
FmtFprintfInt-8	34.91n ± 1%	27.05n ± 1%	-22.52%	27.07n ± 1%	-22.46%	17.42n ± 0%	-50.09%
FmtFprintfIntInt-8	53.46n ± 1%	45.73n ± 1%	-14.47%	45.08n ± 1%	-15.69%	28.23n ± 0%	-47.19%
FmtFprintfPrefixedInt-8	61.76n ± 1%	57.46n ± 1%	-6.95%	52.60n ± 1%	-14.82%	52.02n ± 1%	-15.76%
FmtFprintfFloat-8	68.11n ± 1%	62.20n ± 1%	-8.68%	63.12n ± 0%	-7.33%	59.06n ± 1%	-13.29%
FmtManyArgs-8	192.4n ± 1%	171.1n ± 0%	-11.05%	170.1n ± 1%	-11.59%	156.2n ± 0%	-18.79%
GobDecode-8	2.137m ± 0%	1.967m ± 0%	-7.94%	1.875m ± 1%	-12.24%	1.452m ± 0%	-32.06%
GobEncode-8	1.504m ± 1%	1.455m ± 1%	-3.21%	1.384m ± 0%	-7.97%	1.227m ± 1%	-18.42%
Gzip-8	132.1m ± 1%	121.6m ± 0%	-7.99%	120.5m ± 0%	-8.76%	121.9m ± 0%	-7.72%
Gunzip-8	15.24m ± 0%	15.83m ± 0%	+3.83%	16.01m ± 0%	+5.03%	14.37m ± 0%	-5.72%
HTTPClientServer-8	36.19µ ± 1%	35.13µ ± 1%	-2.94%	34.56µ ± 1%	-4.49%	33.16µ ± 2%	-8.37%
JSONEncode-8	3.784m ± 0%	3.297m ± 1%	-12.86%	2.831m ± 1%	-25.17%	2.699m ± 1%	-28.66%
JSONDecode-8	20.60m ± 0%	17.91m ± 1%	-13.09%	18.09m ± 1%	-12.21%	14.44m ± 0%	-29.92%
Mandelbrot200-8	2155862.5000n ± 1%	0.2530n ± 1%	-100.00%	0.2548n ± 1%	-100.00%	0.2543n ± 1%	-100.00%
GoParse-8	1.788m ± 0%	1.658m ± 1%	-7.27%	1.527m ± 1%	-14.60%	1.398m ± 1%	-21.82%
RegexpMatchEasy0_32-8	26.80n ± 1%	25.90n ± 1%	-3.36%	24.75n ± 1%	-7.63%	17.56n ± 1%	-34.48%
RegexpMatchEasy0_1K-8	85.39n ± 1%	82.75n ± 1%	-3.10%	82.11n ± 1%	-3.85%	78.53n ± 0%	-8.04%
RegexpMatchEasy1_32-8	20.73n ± 2%	21.88n ± 1%	+5.52%	20.34n ± 2%	-1.90%	15.00n ± 0%	-27.66%
RegexpMatchEasy1_1K-8	128.70n ± 1%	112.00n ± 0%	-12.98%	105.65n ± 0%	-17.91%	90.22n ± 0%	-29.90%
RegexpMatchMedium_32-8	380.5n ± 1%	309.6n ± 0%	-18.64%	276.2n ± 0%	-27.41%	188.5n ± 1%	-50.48%
RegexpMatchMedium_1K-8	15.75µ ± 1%	12.06µ ± 0%	-23.46%	11.80µ ± 1%	-25.10%	10.68µ ± 1%	-32.21%
RegexpMatchHard_32-8	650.6n ± 0%	534.0n ± 0%	-17.93%	514.6n ± 0%	-20.90%	419.4n ± 1%	-35.53%
RegexpMatchHard_1K-8	19.11µ ± 1%	15.94µ ± 6%	-16.60%	15.47µ ± 1%	-19.07%	12.95µ ± 1%	-32.24%
Revcomp-8	235.3m ± 1%	181.3m ± 1%	-22.97%	212.2m ± 0%	-9.83%	207.8m ± 1%	-11.68%
Template-8	25.08m ± 1%	20.09m ± 1%	-19.92%	19.05m ± 1%	-24.03%	14.39m ± 1%	-42.64%
TimeParse-8	110.45n ± 1%	88.19n ± 1%	-20.15%	79.14n ± 1%	-28.35%	77.23n ± 0%	-30.07%
TimeFormat-8	118.2n ± 1%	143.8n ± 1%	+21.70%	137.0n ± 1%	+15.86%	103.2n ± 0%	-12.69%
geomean	20.71µ	10.96µ	-47.06%	10.40µ	-49.79%	8.829µ	-57.37%

vectorization

goos: linux
goarch: amd64
pkg: test/bench/go1
cpu: Intel(R) Xeon(R) w5-3435X

	go	llvm		lto		pgo	
	sec/op	sec/op	vs base	sec/op	vs base	sec/op	vs base
BinaryTree17-8	1332.0m ± 6%	1218.3m ± 1%	-8.53%	823.2m ± 1%	-38.20%	791.3m ± 14%	-40.59%
Fannkuch11-8	1.694 ± 1%	1.618 ± 0%	-4.50%	1.639 ± 0%	-3.25%	1.631 ± 0%	-3.75%
FmtFprintfEmpty-8	13.705n ± 1%	13.070n ± 1%	-4.63%	10.145n ± 1%	-25.98%	6.771n ± 1%	-50.59%
FmtFprintfString-8	26.49n ± 1%	21.92n ± 1%	-17.25%	19.93n ± 1%	-24.75%	16.59n ± 1%	-37.37%
FmtFprintfInt-8	34.91n ± 1%	27.05n ± 1%	-22.52%	27.07n ± 1%	-22.46%	17.42n ± 0%	-50.09%
FmtFprintfIntInt-8	53.46n ± 1%	45.73n ± 1%	-14.47%	45.08n ± 1%	-15.69%	28.23n ± 0%	-47.19%
FmtFprintfPrefixedInt-8	61.76n ± 1%	57.46n ± 1%	-6.95%	52.60n ± 1%	-14.82%	52.02n ± 1%	-15.76%
FmtFprintfFloat-8	68.11n ± 1%	62.20n ± 1%	-8.68%	63.12n ± 0%	-7.33%	59.06n ± 1%	-13.29%
FmtManyArgs-8	192.4n ± 1%	171.1n ± 0%	-11.05%	170.1n ± 1%	-11.59%	156.2n ± 0%	-18.79%
GobDecode-8	2.137m ± 0%	1.967m ± 0%	-7.94%	1.875m ± 1%	-12.24%	1.452m ± 0%	-32.06%
GobEncode-8	1.504m ± 1%	1.455m ± 1%	-3.21%	1.384m ± 0%	-7.97%	1.227m ± 1%	-18.42%
Gzip-8	132.1m ± 1%	121.6m ± 0%	-7.99%	120.5m ± 0%	-8.76%	121.9m ± 0%	-7.72%
Gunzip-8	15.24m ± 0%	15.83m ± 0%	+3.83%	16.01m ± 0%	+5.03%	14.37m ± 0%	-5.72%
HTTPClientServer-8	36.19µ ± 1%	35.13µ ± 1%	-2.94%	34.56µ ± 1%	-4.49%	33.16µ ± 2%	-8.37%
JSONEncode-8	3.784m ± 0%	3.297m ± 1%	-12.86%	2.831m ± 1%	-25.17%	2.699m ± 1%	-28.66%
JSONDecode-8	20.60m ± 0%	17.91m ± 1%	-13.09%	18.09m ± 1%	-12.21%	14.44m ± 0%	-29.92%
GoParse-8	1.788m ± 0%	1.658m ± 1%	-7.27%	1.527m ± 1%	-14.60%	1.398m ± 1%	-21.82%
RegexpMatchEasy0_32-8	26.80n ± 1%	25.90n ± 1%	-3.36%	24.75n ± 1%	-7.63%	17.56n ± 1%	-34.48%
RegexpMatchEasy0_1K-8	85.39n ± 1%	82.75n ± 1%	-3.10%	82.11n ± 1%	-3.85%	78.53n ± 0%	-8.04%
RegexpMatchEasy1_32-8	20.73n ± 2%	21.88n ± 1%	+5.52%	20.34n ± 2%	-1.90%	15.00n ± 0%	-27.66%
RegexpMatchEasy1_1K-8	128.70n ± 1%	112.00n ± 0%	-12.98%	105.65n ± 0%	-17.91%	90.22n ± 0%	-29.90%
RegexpMatchMedium_32-8	380.5n ± 1%	309.6n ± 0%	-18.64%	276.2n ± 0%	-27.41%	188.5n ± 1%	-50.48%
RegexpMatchMedium_1K-8	15.75µ ± 1%	12.06µ ± 0%	-23.46%	11.80µ ± 1%	-25.10%	10.68µ ± 1%	-32.21%
RegexpMatchHard_32-8	650.6n ± 0%	534.0n ± 0%	-17.93%	514.6n ± 0%	-20.90%	419.4n ± 1%	-35.53%
RegexpMatchHard_1K-8	19.11µ ± 1%	15.94µ ± 6%	-16.60%	15.47µ ± 1%	-19.07%	12.95µ ± 1%	-32.24%
Revcomp-8	235.3m ± 1%	181.3m ± 1%	-22.97%	212.2m ± 0%	-9.83%	207.8m ± 1%	-11.68%
Template-8	25.08m ± 1%	20.09m ± 1%	-19.92%	19.05m ± 1%	-24.03%	14.39m ± 1%	-42.64%
TimeParse-8	110.45n ± 1%	88.19n ± 1%	-20.15%	79.14n ± 1%	-28.35%	77.23n ± 0%	-30.07%
TimeFormat-8	118.2n ± 1%	143.8n ± 1%	+21.70%	137.0n ± 1%	+15.86%	103.2n ± 0%	-12.69%
geomean	17.64µ	10.96µ	-10.21%	10.40µ	-15.02%	8.829µ	-28.24%

LICM

Q&A

Thank you