Experimenting with Link Time Optimization

Jon Degenhardt

Silicon Valley D Language Meetup

Dec 14, 2017

Today's meetup: Testing the LTO support released with LDC 1.5.0

- Agenda:
  - Link Time Optimization overview
  - LTO support in LDC 1.5.0
  - Overview of the benchmark applications: eBay's TSV utilities
  - Benchmark results: Improved runtimes, smaller executables

- About me
  - Search relevance and search engine architecture at eBay
  - Programming in D for about two years

Link Time Optimization (LTO)

- Whole program optimizations at link-time
- Interprocedural optimizations difficult or impossible when considering only part of the program (e.g. an individual source file)
- Supported by both GCC and LLVM
- LLVM approach
  - Compilation: Write LLVM IR bitcode to .o files rather than machine code
  - Link-time: Read LLVM bitcode from all files. Pass the program to LLVM optimization modules
  - GCC LTO uses a similar approach

Link Time Optimization (cont)

- LLVM's LTO requires special support in the linker
  - macOS: Supported by system linker (Xcode)
  - Linux: GNU "gold" linker. Uses a plugin architecture to support optimizers
  - LLD: New LLVM linker, supports LTO natively
- Full vs Thin LTO
  - Full – Loads a program's entire IR code into memory for optimization
    - General issue: Significant memory use, long compile times
  - Thin – Loads module "summaries" instead of full IR. Retains most optimization benefits, but with faster builds and less memory use.
  - Thin and Full are not compatible, all code used in a build must be built using the same method.

LTO support in LDC

- LDC 1.1.0 (Jan 2017) – Initial LTO support
  - macOS supported out-of-the-box via Xcode linker
  - Linux requires separate install of the GNU “gold” linker
- LDC 1.2.0, 1.3.0 – Ongoing fixes and improvements
- LDC 1.4.0 (Sep 2017)
  - Ships with LLVM LTO plugin for the ‘gold’ linker. Linux support out-of-the-box.
  - `ldc-build-runtime` tool
    - Downloads D standard library source (druntime, Phobos) and compiles with LTO.
    - Interprocedural optimizations across D standard library and application code!!
- LDC 1.5.0 (Nov 2017) – Critical bug fixes
  - LTO with D standard libraries is now viable
  - Experimental support for Windows LTO

LTO Benchmarks

- Not many published LTO benchmarks
  - Published benchmarks show mixed results. Executable size reduction is common. Runtime performance improves on occasion.
  - Common sentiment: Programs most likely to benefit are those that have not been hand optimized.
- Basis for this benchmark report: eBay's TSV utilities
  - One of many tools used for large data set processing. Filtering, statistics, sampling, etc.
  - Written in D as part of an exercise exploring the language
  - Benchmark well compared to similar tools written in native languages
    - March 2017 study: https://github.com/eBay/tsv-utils-dlang/blob/master/docs/Performance.md
  - Takeaway: Benchmark study suggests at least a reasonable level of optimization
  - Latest TSV utilities release is built with LTO on Travis-CI

March 2017 benchmarks: Top-4 in each test (No LTO)

Benchmark	Tool/Time	Tool/Time	Tool/Time	Tool/Time
CSV-to-TSV	***csv2tsv***	csvtk	xsv	
(2.7 GB, 14M lines)	27.41	36.26	40.40	
Summary statistics	***tsv-summarize***	Toolkit 1	Toolkit 2	Toolkit 3
(4.8 GB, 7M lines)	15.83	40.27	48.10	62.97
Numeric row filter	***tsv-filter***	mawk	GNU awk	Toolkit 1
(4.8 GB, 7M lines)	4.34	11.71	22.02	53.11
Regex row filter	***tsv-filter***	GNU awk	mawk	Toolkit 1
(2.7 GB, 14M lines)	7.11	15.41	16.58	28.59
Column selection	***tsv-select***	mawk	GNU cut	Toolkit 1
(4.8 GB, 7M lines)	4.09	9.38	12.27	19.12
Join two files	***tsv-join***	Toolkit 1	Toolkit 2	Toolkit 3
(4.8 GB, 7M lines	20.78	104.06	194.80	266.42

- *Macbook Pro, 16 GB RAM, SSD drives. Times in seconds. Comparison includes 9 separate tools (C, Rust, Go)*

Benchmark results with LTO: macOS

Compiler	LTO	csv2tsv	tsv-summarize	tsv-filter (numeric)	tsv-filter (regex)	tsv-select	tsv-join
LDC 1.2.0	None	29.41	15.33	4.28	7.85	4.05	20.84
LDC 1.2.0	App; Thin	23.99	15.70	4.25	7.54	4.04	20.66
LDC 1.2.0	App; Full	23.86	15.59	4.25	7.54	4.05	20.73
LDC 1.5.0	None	25.54	22.52	4.96	7.78	4.28	21.33
LDC 1.5.0	App; Thin	25.70	22.55	5.01	7.65	4.19	21.24
LDC 1.5.0	App; Full	24.10	21.81	5.16	7.60	4.21	21.38
LDC 1.5.0	D libs; Thin	21.48	10.44	3.65	7.14	4.05	20.11
Delta from 1.2.0/None		**27%**	**32%**	**15%**	**9%**	**0%**	**4%**
Delta from 1.5.0/None		**16%**	**54%**	**26%**	**8%**	**5%**	**6%**

- Improvements in most benchmarks
- No material improvement from app-only LTO (except csv2tsv in LCD 1.2.0)
- Significant gain from including D standard libraries

Benchmark results with LTO: Linux

Compiler	LTO	csv2tsv	tsv-summarize	tsv-filter (numeric)	tsv-filter (regex)	tsv-select	tsv-join
LDC 1.2.0	None	41.74	25.46	7.03	12.34	5.88	34.10
LDC 1.5.0	None	46.84	30.34	7.61	12.12	6.25	34.01
LDC 1.5.0	App; Thin	47.38	30.05	7.91	12.48	6.30	34.36
LDC 1.5.0	App; Full	48.97	30.28	7.73	12.30	6.23	34.58
LDC 1.5.0	D libs; Full	33.44	17.87	6.20	10.48	5.94	32.65
Delta from 1.2.0/None		**20%**	**30%**	**12%**	**15%**	**-1%**	**4%**
Delta from 1.5.0/None		**29%**	**41%**	**19%**	**14%**	**5%**	**4%**

- Slower machine than macOS benchmark (commodity cloud box)
- No material improvement from app-only LTO
- LTO including D libraries is a clear improvement

Executable sizes: macOS (bytes)

Compiler	LTO	csv2tsv	tsv-summarize	tsv-filter	tsv-select	tsv-join
LDC 1.2.0	None	3,841,420	5,144,720	6,217,924	4,066,664	4,118,700
LDC 1.5.0	None	6,709,936	7,988,448	8,137,804	6,890,192	6,945,336
LDC 1.5.0	App; Thin	6,643,344	6,949,848	6,639,876	6,675,664	6,687,840
LDC 1.5.0	App; Full	6,643,344	6,949,712	6,639,844	6,676,000	6,688,392
LDC 1.5.0	D libs; Thin	2,679,184	3,082,068	3,172,648	2,734,356	2,738,700
Delta from 1.5.0/None		**60%**	**61%**	**61%**	**60%**	**61%**

Executable sizes: Linux (bytes, dynamic libc. Static adds 1.35MB)

Compiler	LTO	csv2tsv	tsv-summarize	tsv-filter	tsv-select	tsv-join
LDC 1.2.0	None	726,880	1,117,040	1,416,952	756,456	776,472
LDC 1.5.0	None	995,760	1,400,672	1,743,288	1,026,344	1,049,176
LDC 1.5.0	App; Thin	998,624	1,296,496	1,547,944	1,023,880	1,031,984
LDC 1.5.0	App; Full	998,432	1,300,792	1,547,656	1,024,312	1,036,648
LDC 1.5.0	D libs; Full	826,064	1,154,808	1,359,544	856,064	868,736
Delta from 1.5.0/None		**17%**	**18%**	**22%**	**17%**	**17%**

Example: Building with LTO

- Use code from blog post Faster Command Line Tools in D, version 4b.

- Build command, no LTO:

```
$ ldc2 -release -O faster_cmd_v4b.d
```

- Build commands, with LTO for D standard libraries:

```
$ ldc-build-runtime --reset --dFlags="-flto=thin" BUILD_SHARED_LIBS=OFF
$ ldc2 -release -O -flto=thin -L-L./ldc-build-runtime.tmp/lib faster_cmd_v4b.d
```

Concluding Remarks

- Many thanks to the LDC team for help with this work
  - Special thanks to Johan Engelen and Martin Kinkelin
- LTO is now a real option with the LDC 1.5.0 release
  - Easy to try if you are using LDC on macOS or Linux
  - Still an early technology. Good test coverage is quite valuable to detect problems.
  - Current recommendation: Use Thin LTO on macOS, Full on Linux
- Significant improvements on the TSV utilities apps
  - The big win comes from running LTO on the D standard libraries
  - Cross-module inlining likely the most significant source of performance gains
  - TSV utilities are small, build times are not an issue for either Thin or Full LTO
- Need benchmarks from a wider variety of apps
- Profile Guided Optimization (PGO) is the obvious thing to try next

References

- LDC: LLVM D Compiler wiki. The LDC compiler home page.
- Link Time Optimization (LTO), C++/D cross-language optimization, Johan Engelen's blog
- Building LDC runtime libraries. LDC docs for LTO on runtime libraries.
- ThinLTO: Scalable and Incremental LTO, LLVM Project Blog
- ThinLTO: Scalable and Incremental Link-Time Optimization. CppCon 2017, Teresa Johnson. The talk to see if you want to understand LTO.
- LLVM Link Time Optimization: Design and Implementation
- Optimizing real world applications with GCC Link Time Optimization, T. Glek, J. Hubicka. Describes building Firefox with LTO