Exploring Profile Guided Optimization of the Linux Kernel

ian Bearman
Principal Software Engineering Manager @ Microsoft

ian.bearman@microsoft.com
@manbearian
https://www.linkedin.com/in/manbearian/
https://github.com/manbearian
Introduction
GNU/Linux Dev Tools @ Microsoft

Our Mission: Support Linux dev tooling needs for Microsoft

• Across Multiple Platforms
  • Azure Cloud
    • Half (or more) of all instance in Azure are running Linux!
  • Windows Subsystem for Linux
  • IoT (such as Azure Sphere)

• Across Multiple Features and Tools
• Correctness, Performance, and Security
Optimize Single Service Instance

Internal customer request
- Linux-hosted cloud service
- Instance runs a single service
- 64-bit x86 and ARM
- Willing to build their own kernel
- **Goal**: Maximize Performance

How can a tools team help?
- Brainstorming: Profile Guided Optimization!

Complications
- Workload isn’t fully known (service and architecture isn’t completed)!
- No benchmarks provided
Background
LTO and PGO – quick primer

**PGO - Profile Guided Optimization** *(aka Pogo, FDO, -fprofile-use)*
- Consume profile information to improve code generation
- Allow placement of code (and data) for spatial and temporal locality
- Drive inlining decisions (inline hot paths, ignore cold paths)
- Intra-function Code layout

**LTO - Link Time Optimization** *(aka LTCG, WPA/WPO/IPA)*
- Compile entire module/binary at once
- Inline across CPP files
- Interprocedural analysis and optimization
- Optimize using “whole program view”
Yuan (2014)

Yuan (2015)

Previous Research
PGO + Linux Kernel
Setup

**Software:** Ubuntu 19.10
with GCC 9.2.1, binutils 2.33, kernel 5.3

**Hardware:** Marvell Thunder X2 (ARM64)

Enabling LTO + PGO
- We reached out to Andi Kleen for help with LTO
- After a few back-and-forths (and one patch) we had LTO working
- Docs + trial-and-error all that was needed to get PGO working
Profiling the Kernel

Instrumentation-based profiling

Kernel Configuration:

- Build and install kernel with instrumentation
- Run scenario
- After run trace location is @ /sys/kernel/debug/gcov
  *.gcda; *.gcno
  owned by `root` (so chown/chmod)
Optimizing the Kernel

Remember: clean your build
- follow normal clean steps; and
- Ensure previous coverage options are disabled!

GCC expects the profile data to be in a specific location in the kernel build directory or in a flattened path

```
#home#user81#linux-build#linux-5.3.0#debian#build#build-generic#some#dir#with#a#file.gcda
```

Set build flags to add `-fprofile-use`

```
KCFLAGS="-fprofile-use=/home/user81/gcov-test/generic-instr/gcov -Wno-coverage-mismatch -Wno-error=coverage-mismatch"
```

Work around breakages

```
CFLAGS_lockref.o=-fno-profile-use -O0 to linux/lib/Makefile.
```
Results
Scenarios

redis

Popular database, cache, hash, BSD licensed built-in benchmarking (redis-benchmark)

https://redis.io
redis-benchmark on ARM64

- Baseline-5.3
- O3-5.3
- PGO-5.3
Redis

redis-benchmark on ARM

- Baseline-5.3
- O3-5.3
- PGO-5.3

Graph showing performance metrics for Redis operations such as set and get.
Conclusion & Wrap-up
Conclusion

We saw wins with PGO in Redis
  • Close to the limit for non-kernel bound scenario

We would a better measurement of core kernel performance
  • Stable benchmarks for filesystem, network, scheduler, etc.

We’d love to see more
  • Microsoft Windows heavily utilizes both LTO (LTCG) and PGO
  • Windows sees 5-20% improvements from PGO
    • we want to investigate if this is relatable

Cyclic Dependency
  • Usage drives quality; quality drives usage
Acknowledgements

Fellow Team Members
Roman "@kromych" (Microsoft)
Di "Modi" Mo (Facebook)

Other Folks
Andi Kleen (Intel)
Q & A
Jobs!

Microsoft is hiring Linux developers and folks with Linux experience!

Thank You!

ian.bearman@microsoft.com
@manbearian
https://www.linkedin.com/in/manbearian/
https://github.com/manbearian