Google

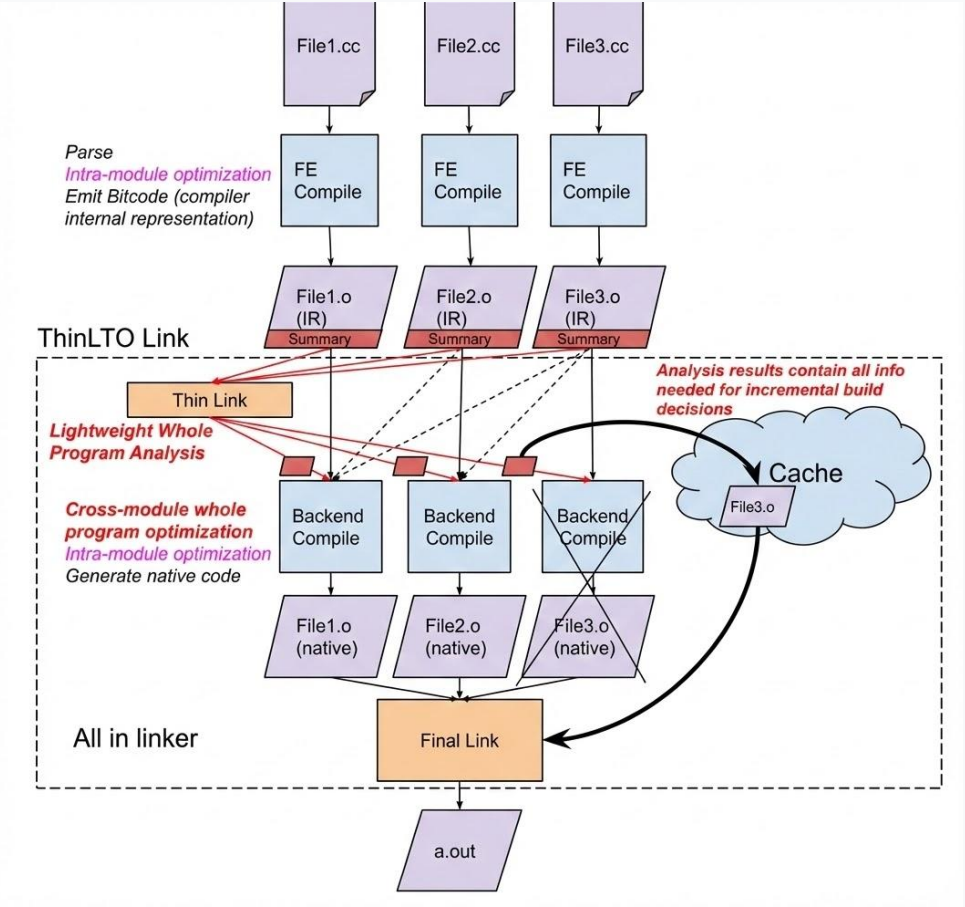
Distributed ThinLTO for Linux kernel

Rong Xu (xur@google.com)

LPC 2025

In process build

- Frontend (FE) compilation:
  - Generates IR and summary
  - Fully parallel
- Linker does the remaining steps:
  - Thin-link: merges summary and generates index files
  - Backend compilation: fully parallelized and invoked via in-process callback to the compiler
  - Final link
- Incremental build: Caching

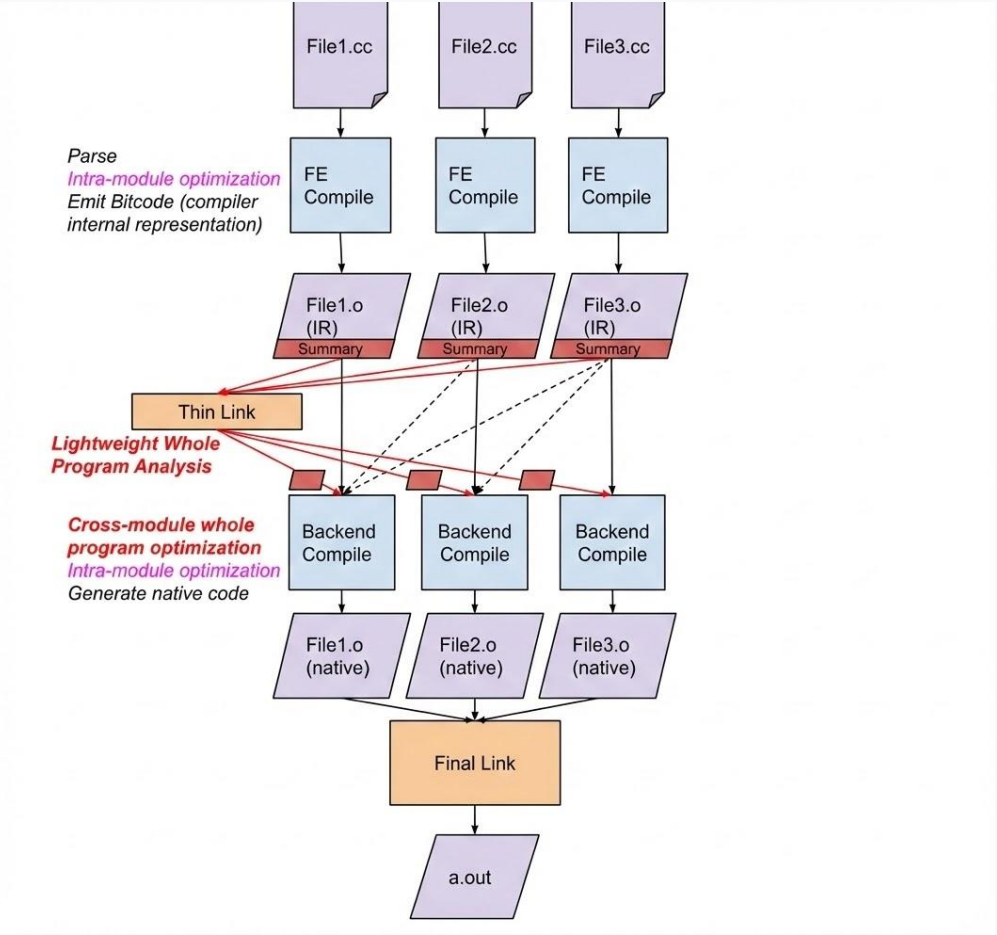


Distributed build

Distributed build exposes the internals and users manage each sub-step:

- BE compilation is user-invoked
  - Native objects are available to users
  - Incremental builds are managed via native objects
- Thin-link and Final-link are done by linker

"Distributed" here is to differentiate with previous mode. It works well with distributed systems but does not require a distributed environment.

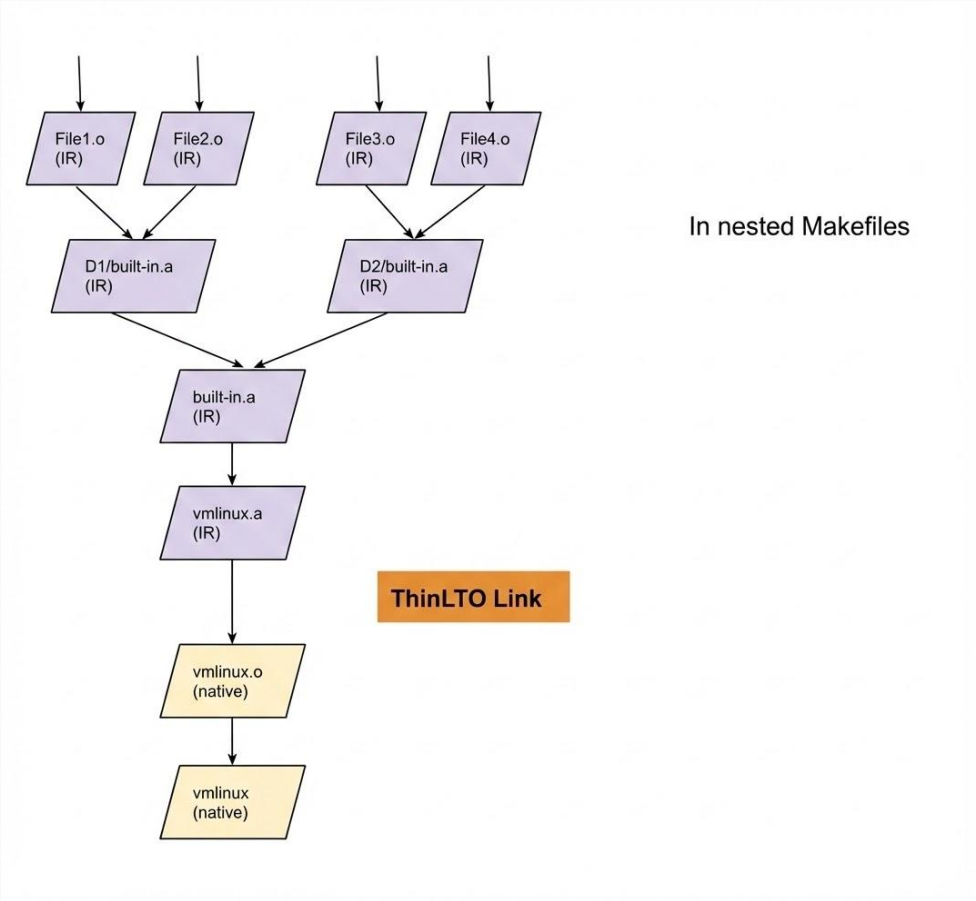


Why distributed build

- In-process mode – easier to integrate into build system, but
  - No final objects
  - A single set of options for BE compile
- Exposes the final native objects
  - Livepatch: find the changed final objects much easier
  - Objtool: the same as non-LTO builds
- BE compilation can have its specific compiler options
  - Help Debug / performance tuning
- Better supported and tested

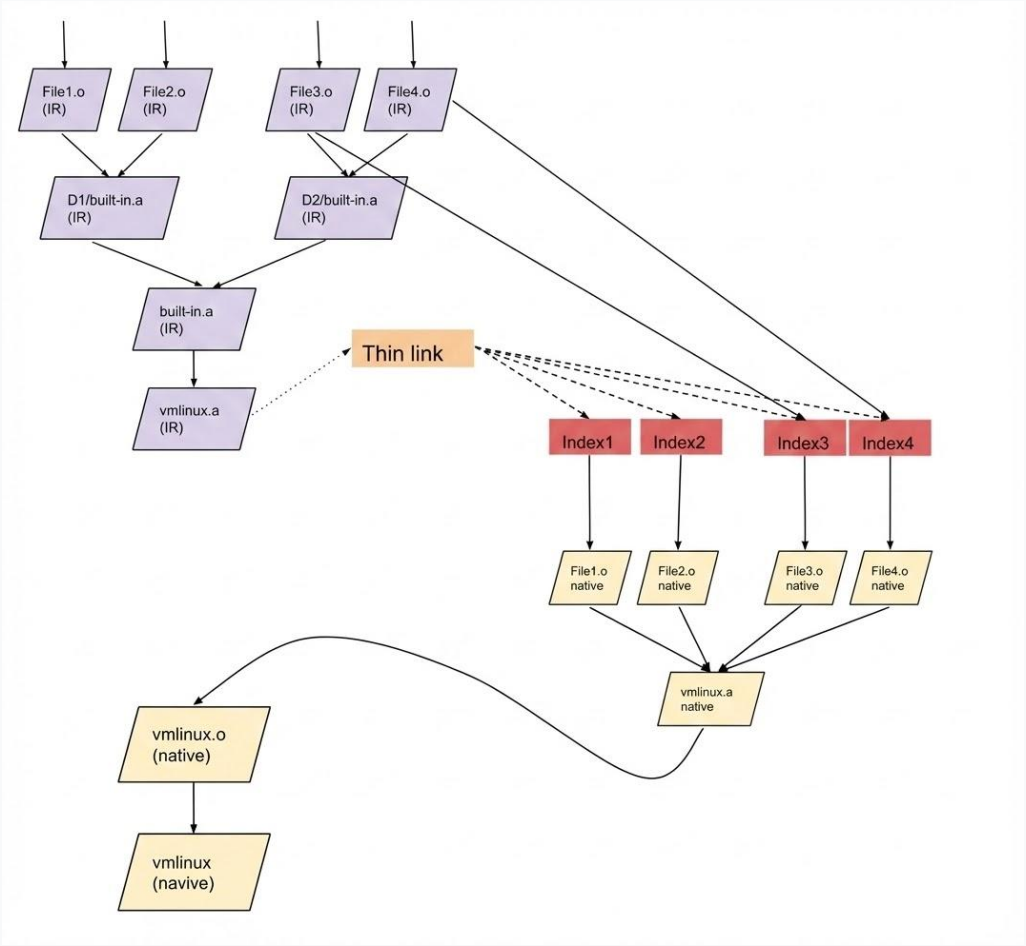
In-process ThinLTO kernel build flow

- Sub-dir Makefiles: generate IR objs, thin-archived into built-in.a
- All sub-dir built-in.a are thin-archived into root's built-in.a
- vmlinux.a remains in IR format
- vmlinux.o is a relocatable native object



Implementation

- Configuration: CONFIG_LTO_CLANG_THIN_DIST
- Thin-link: flag "--thinlto-index-only"; generate index files
- Backend (BE) Compilation:
  - Final objects: %.thinlto-native.o
  - Initial approach: Regeneration from the nested Makefiles
  - Latest approach (Co-developed by Masahiro Yamada):
    1. Uses the file list in vmlinux.a.
    2. Utilizes the options stored in %.o.cmd.
  - Build performance is about the same



Build comparison

- In-process and distributed build produce different binary:
- Option handling:
  - In-process mode turns on –ffunction-sections by default and cannot be disabled
  - Some “-O0” and “-O3” are ignored in BE compilation
  - -falign-loops=1 is ignored in BE compilation
- Function orders are slightly different

Build time comparison

Overall Build Time Comparison

Time	Non LTO	In-process ThinLTO	Distributed ThinLTO
real	44.30	118.71	58.88
user	2222.23	2278.43	2324.16
sys	322.93	324.78	441.47

"Fairer" Comparison (Matching Build Options)

Time	In-process (–thinlto_jobs=96)	Distributed with function-sections on
real	119.10	120.65
user	2352.93	2398.40
sys	335.06	447.39

Build time comparison

Distributed ThinLTO Stage Breakdown

Time	FE (Front End)	Thin-link	BE (Back End)	Final link	Final link (w/ function section)
real	30.94	4.46	10.42	16.30	83.95
user	2053.65	8.79	308.09	19.16	84.83
sys	319.34	50.02	124.78	49.69	54.61

Summary

- Better options handling
- Better fit into kBuild
- Should be the default for kernel ThinLTO build

References:

1. LLVM discourse post: https://discourse.llvm.org/t/rfc-distributed-thinlto-build-for-kernel/85934
2. Patch: https://lore.kernel.org/all/20251028182822.3210436-1-xur@google.com/