Integrated Distributed ThinLTO

Konstantin Belochapka and Katya Romanova

LLVM Developers’ Meeting, October 2023

ThinLTO

- An ease of use and adoption contributed to ThinLTO’s popularity.
- Users had to add one option to the compiler and the linker command line (-flto=thin), this was very easy.

Distributed ThinLTO

- Distributed ThinLTO is not as commonly used as ThinLTO.

- Distributed ThinLTO is quite complicated to integrate with the majority of build systems, mainly because the build-rule dependencies that a build system needs to construct the build graph are not known in advance. The dependencies become available midway through the Distributed ThinLTO build-process, after the ThinLink phase completes, and we know the list of import files.

- Some build systems support Distributed ThinLTO build-flow (Bazel, Buck2).

Enabling Distributed ThinLTO for existing build projects is hard

- Rewrite your build projects for Bazel or Buck2 from scratch.
- Modify your existing software build-project written for makefiles, ninja, autotool or cmake (but this is hard).

Enabling Distributed ThinLTO for existing build projects is hard

To use Distributed ThinLTO in an existing makefile project, it will be required to:

1. Invoke ThinLink step, that for every of the bitcode files will generate a list of files that ThinLTO needs to import from.
2. Analyze these import dependencies (it could be done by implementing a script that parses the content of the import files).
3. Write a script to generate a set of backend compile processes command lines.
4. Feed a set of the backend compile processes to the distribution system and specify which dependent files need to be copied to the remote node for each of the compile processes.
5. Identify which files failed to compile and re-do the compilation for these files on the local machine.
6. Perform a final link phase linking all the generated native object files.

Enabling Distributed ThinLTO for existing build projects is hard

- An archive needs to be unpacked; its individual members need to be passed to the linker for ThinLink step surrounded by --start-lib/–end-lib pair.
- ThinLTO backend compile needs to be invoked for each of the individual archive constituents.
- Preventing name collision prevention when unpacking the same archives in different parallel processes is an additional challenge.

Enabling Distributed ThinLTO for existing build projects is hard

- Not a trivial task even for a very experienced build master.
- A lot of custom scripts needs to be implemented.

Integrated Distributed ThinLTO approach

In Sony, we implemented Integrated Distributed ThinLTO approach where all the tasks that I just described are done within the linker. We coupled Distributed ThinLTO with Sony's proprietary distribution system called SN-DBS.

If a project already has ThinLTO enabled and wants to start using distributed ThinLTO, only a couple of small things are needed:

- To deploy a distribution system.
- To modify the makefile or buildscript by adding "--thinlto-distribute" option to the linker command line.

Integrated Distributed ThinLTO approach

To start using Distributed ThinLTO our users just to add one additional option to the linker command line (“--thinlto-distribute”).

Our Integrated Distributed ThinLTO project has been in production for several months.

Upstreaming Integrated Distributed ThinLTO project

- We would like to upstream our project.
- RFC has been submitted recently https://discourse.llvm.org/t/rfc-integrated-distributed-thinlto/69641.
- We plan to support the Integrated Distributed ThinLTO functionality on Linux and use the open-source distribution system, Icecream (IceCC).

Distributed ThinLTO and Icecream integration

Some changes had to be done in Icecream side to allow us to hook it up with the distributed ThinLTO. Luckily, Icecream is an Open Source project.

Here is an abbreviated list of things that we had to do in Icecream to teach it to do code generation:

- Support –thinlto-index-only and -flto option for the IceCC compiler wrapper.
- Add LLVM IR bitcode files to the list of supported input files format.
- Support transferring a set of input files (instead of one file which was a limitation) both on Icecream client (IceCC) and Icecream server (iceccd deamon) sides.
- Implement an extension to the Icecream server, so that it could do CodeGen in multiple remote execution environments in parallel (to avoid file name collisions).
- Submit our changes to the open source (IceCream project) and have them accepted.

Distributed ThinLTO integrated with other distribution systems.

- Distributed ThinLTO projects will generate a generic build script (JSON file), which will contain the list of compilation command lines, the locations of the output native object files and the list of dependencies to be copied on the remote node.

- To support integration of Distributed ThinLTO with any other distribution system other than IceCream and SN-DBS, one need to write a custom script that knows the specifics of a particular distribution system. It will take the information in the generic JSON file, convert it to the custom (distribution system specific) Makefile/Fast build .FB file/JSON/Incredibuild XML/etc, and invoke the remote build system.

After this custom script is written once (by Sony or someone else) and open-sourced, all other users will be able to take advantage of using Distributed ThinLTO with this particular build/distribution system with minimal efforts.

All the users will have to do is to specify that the distribution is desired and to pass the name of the build/distribution as a parameter (***--thinlto-distribute=icecream/distcc/fastbuild/incredibuild).***

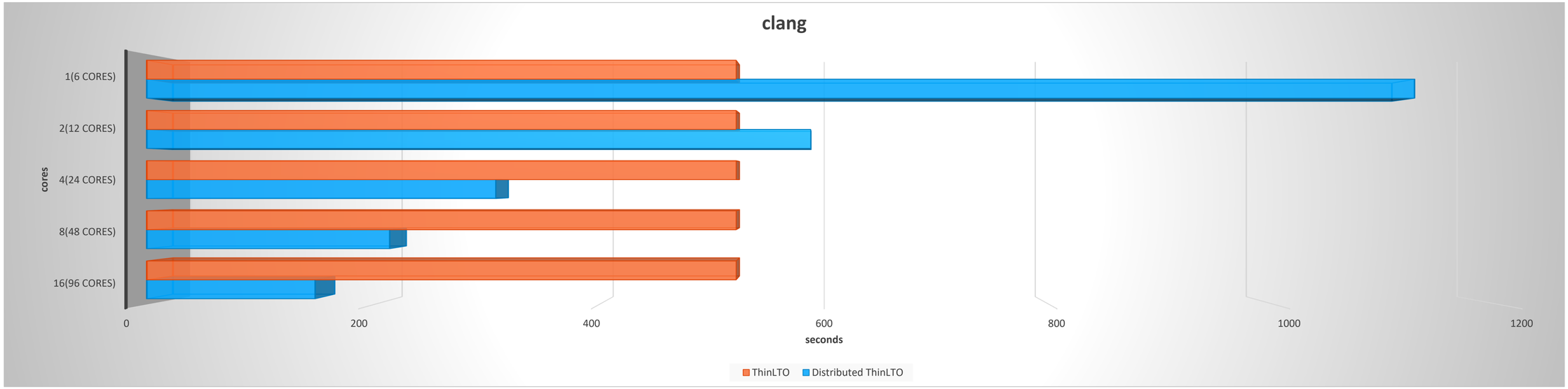
Performance

Distributed build performance obviously depends on a deployed hardware infrastructure. Bigger number of faster executor nodes alongside with fast network communications yields better distributed performance.

Distributed build performance is also very much input data dependent – typically best distributed performance can be achieved when all subjects of a build are large (more precisely require significant processing time) and, they have approximately equal size. Big number of small files significantly increase a distributed system cumulative service time against cumulative processing time. On the other hand, small number of significantly unequal files would prevent to achieve good balanced load on a pool of distributed executor nodes.

Linking time for the Clang project. Distributed ThinLTO vs. regular ThinLTO

Linking time comparison for regular (multi-threaded) ThinLTO vs Distributed ThinLTO on pools of 1, 2, 4, 8, and 16 PCs.



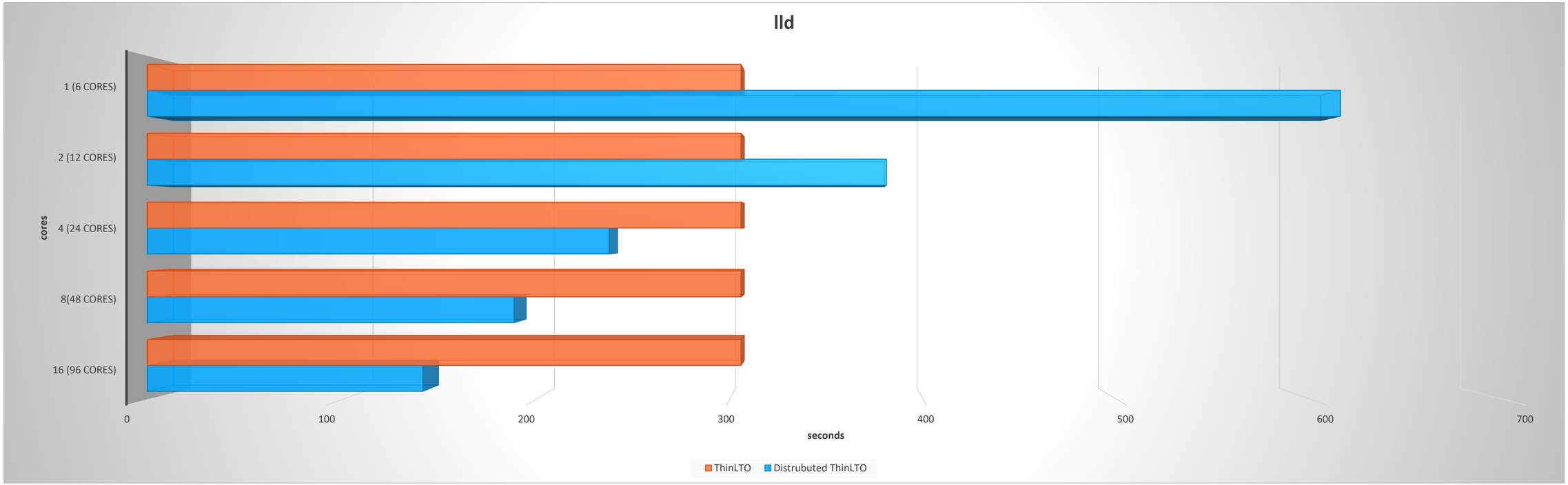
Typical PC has Xeon E5-1650 CPU – 6 physical cores, 32GB memory, 1Gb network.

Note that on 16 PCs Distributed ThinLTO is 3.5 times faster than regular ThinLTO.

Linking time for lld project. Distributed ThinLTO vs. regular ThinLTO.

Similar results as for the clang project. Starting from 4 PCs pool Distrusted ThinLTO has better performance.

Note that on 16 PCs Distributed ThinLTO is 2.2 times faster than regular ThinLTO.



Future Work

We are planning to start upstreaming our work to LLVM and LLD after our RFC is accepted. Our design is modular and the support for many other build systems could be added with the reasonable efforts.

We will also have to talk to IceCream open source developers and have our changes needed for support of Distributed ThinLTO supported.

In the future, either Sony or any other interested parties could provide custom scripts for other distribution/build systems such as Distcc/Fastbuild/Incredibuild, etc, and make them work with Distributed ThinLTO.

Thanks for listening!

Sony
Interactive
Entertainment