INCREDIBUILD

Accelerate Android AOSP 16 and AOSP 15 builds by up to 10X

In this article you'll learn how to accelerate AOSP builds using shared cache, distributed computing for achieving:

- ✓ **Faster build time**
- ✓ **Increased CI throughput**
- ✓ **Reduced cloud spend**

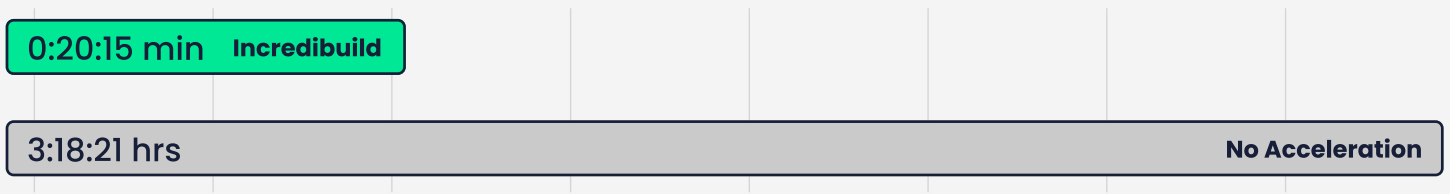
- Baseline vs Incredibuild accelerated Android AOSP16 and AOSP15 build times on a 32-core machine

A >6x build time acceleration ratio over baseline on a 32-core machine.

EXECUTIVE SUMMARY

Incredibuild is offering a build acceleration solution purpose-built for Android builds at scale. By seamlessly combining distributed computing, a generic shared cache, and comprehensive build observability, Incredibuild accelerates Android builds by 6x on a 32-core machine. It integrates directly with existing CI tools to reduce build wait times, increase host throughput, and improve resource efficiency without requiring any changes to your code, build scripts, or toolchain. This document demonstrates how Incredibuild delivers consistent performance gains and full visibility into your AOSP-based build pipelines, whether on-premises, in a hybrid environment, or scaled across multiple teams.

A similar benchmark on AOSP 16 running on a common 16-core developer workstation shows an acceleration ratio of ~10x faster build time over baseline!



THE COST OF WAITING

> *"The collective annual cost of developer waiting time averages $3,000,000/year."*
>
> CircleCI

The AOSP build process, while robust, is inherently time-consuming. A full clean build can take several hours, consuming vast computational resources. More critically, even incremental builds often involve recompiling large portions of the codebase due to complex dependencies. This "build-from-scratch" behavior for every change creates several critical pain points for development teams:

- **Developer frustration and Idle time**: Engineers spend valuable time waiting for builds to complete, disrupting their workflow and delaying the feedback loop. This unproductive time directly impacts morale and team efficiency.
- **Reduced CI throughput**: Build queues become a major pain point, especially during peak development times. Each build server can only process a limited number of builds per day, leading to longer wait times and infrastructure sprawl.

- **Increased infrastructure costs**: Long compute-intensive build times necessitate powerful, always-on build servers and substantial cloud-based compute resources, driving up operational expenses.

The current paradigm is one of redundant work, where every developer and CI agent may be compiling the same unchanged source files, a process that is costly, inefficient, and unsustainable for large-scale development.

THE SOLUTION

The Incredibuild platform addresses these challenges by intelligently eliminating redundant work and accelerating the non-redundant work that's left, through two core technologies: multi tool Shared Cache and Distributed Computing. This approach transforms the build process from a series of isolated, repetitive tasks into a collaborative, cumulative effort.

Shared cache across your stack

At the heart of Incredibuild's performance and resource optimization is a generic shared cache engine that makes sure that as long as the input data of a task that was already executed didn't change, the same task won't be re-executed by anyone again, but will be served by the shared cache instead. The shared cache stores the outputs of previous tasks and reuses them across users, machines, and CI jobs. Once a task is executed anywhere in the organization, its output is cached and the task execution doesn't need to be repeated —unless Incredibuild automatically detects that its inputs have changed. Many different tools and scripts are used as part of an AOSP build. This makes the genericness of Incredibuild's shared cache very useful as it can accelerate a large number of these various tools. The following table lists the list of tools accelerated by Incredibuild's shared cache in an AOSP build:

Tool Name	Category
clang	C++ compiler
clang++	C++ compiler
clang++	C++ linker
rustc	Rust compiler & linker
yasm	Assembler
javac	Java compiler
R8	AOSP java tool
D8	AOSP java tool
metalava	AOSP java tool
kotlin-compiler	Kotlin compiler (on JVM)
jarjar	Java packaging utility
Turbine	Java header compiler
versioner	AOSP-specific tool
aidl	AOSP-specific tool
header-abi-dumper	AOSP-specific tool
hiddenapi	AOSP-specific tool
appcompat.sh script	AOSP-specific tool

Distributed computing technology

Complementing Incredibuild's shared cache is Incredibuild's distributed computing engine that turns an ordinary workstation or cloud VM into a powerful build node. It allows any build task such as compiling, linking, or executing custom tools, to be offloaded in real-time to idle CPUs across your network, CI runners, or public cloud, effectively turning every machine into a high-performance-computer with hundreds of cores and gigs of memory.
For workloads that require additional compute power, Incredibuild's orchestrator can automatically provision cloud-based helper machines, including cost-efficient spot instances, to enable burst capacity without manual intervention. This allows organizations to elastically scale remote build cores when needed while keeping costs low.

REAL-WORLD RESULTS: ANDROID AOSP 14 BENCHMARK

To evaluate Incredibuild's impact on AOSP development pipelines, we benchmarked the build of Android 14. The test simulated real-world scenarios on 16-core, 32-core and 96-core build machines, comparing a standard build with an identical one accelerated by Incredibuild's technologies.

The results demonstrate significant performance gains and improved resource utilization which doesn't only affect a single-build duration but also the accumulation of the build-queue and the throughput of the build farm as the below table represents for a 32-core build node:

Metric	Standard CI	With Incredibuild	Impact
AOSP16 Build Duration (32-core machine)	**1 hour and 47 min**	**17 minutes**	**>6× faster builds**
Builds per CI Host per Day	**~5 builds**	**~32**	**>6× more throughput**
Queue Time (Peak Hours)	**80-100 minutes**	**<12 minutes**	**Significantly reduced developer wait times**

On a 32-core build node, a full Android Automotive AOSP build was reduced from almost two hours to 17 minutes, a 4x improvement, while on a typical 16-core developer machine, the build was reduced from 3 hours and 20 minutes to 20 minutes saving 3 hours per build!
Even on a high-end, 96-core machine, Incredibuild was able to triple the build speed while using less overall compute and memory due to its efficient use of caching which highly optimizes the throughput of such high-core machines, allowing them to run three AOSP builds in parallel instead of only one.

FINAL THOUGHTS

Incredibuild provides a comprehensive solution for accelerating AOSP build pipelines. By combining generic shared cache and distributed computing with a zero-change approach to existing toolchains, it delivers tangible benefits:

- **Up to 3–5x build acceleration** out of the box. It also leads to
- **increased CI throughput,** with more builds per host and reduced queue time , and
- **Reduced cloud compute spend** through shared caching and the ability to use cost-effective Helper Spot instances. The platform gives you a
- **360° operational view** into build processes for better control and troubleshooting. This includes a graphical build monitor that provides real-time visualization of the build execution flow, which helps identify bottlenecks and failures. Detailed hit/miss tracking tools allow you to continuously analyze and improve cache effectiveness.

Both Incredibuild's shared cache and distributed processing technologies are highly extensible due to their generic, low-level application instrumentation. This technology allows Incredibuild to provide out-of-the-box support for a wide range of compilers, linkers, build systems, and other tools. Custom vendor tools or proprietary pre- and post-processing steps, which are common in real-world Android-based builds, can also be incorporated into the caching and distribution logic with minimal configuration for additional performance gains.