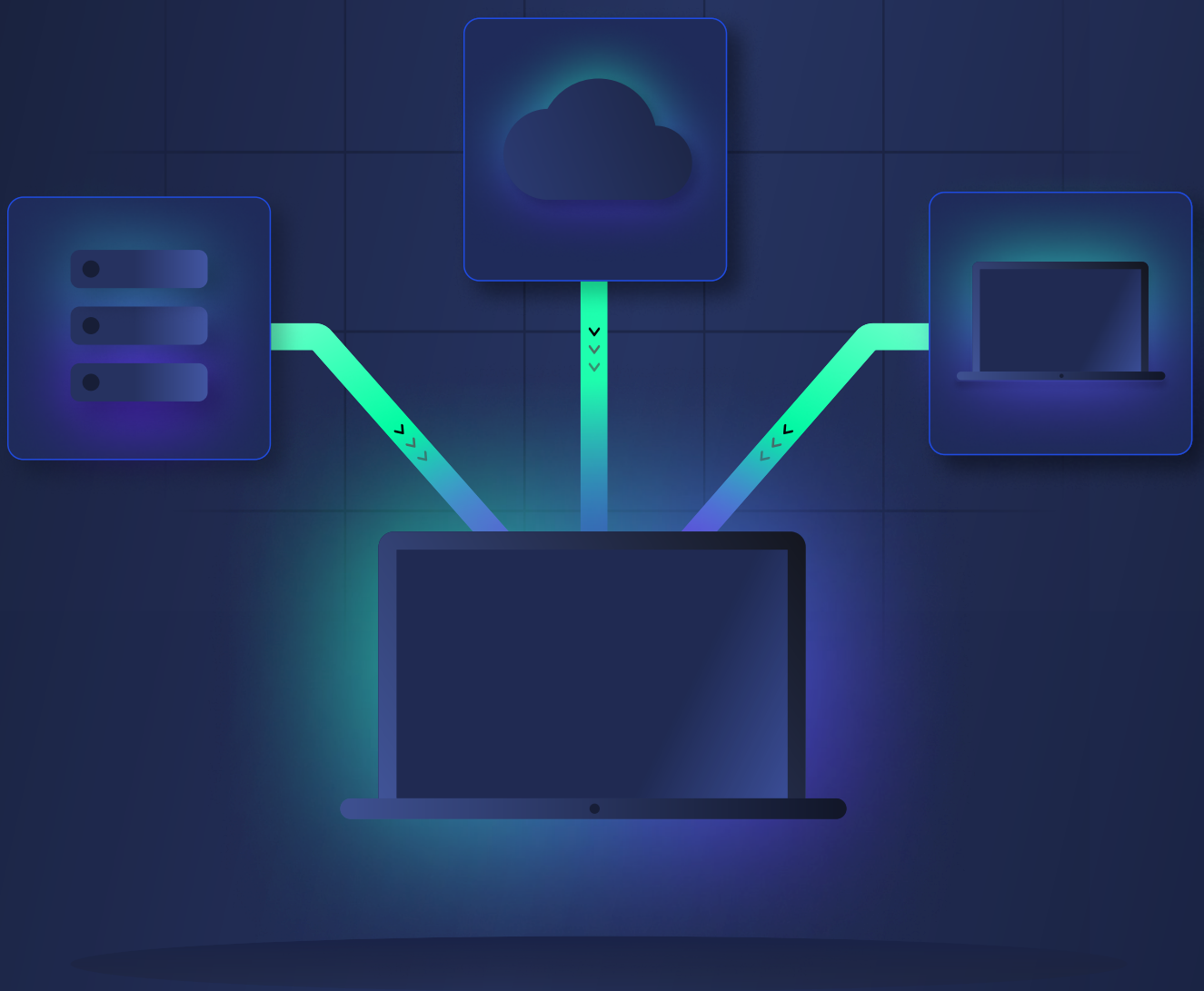
3 Ways You Can Cut Build Times, Not Corners



INCREDIBUILD

Introduction

Much like the Earth's rotation around the Sun, or the way the coffee machine always packs in at the most inconvenient time, builds are an unavoidable part of life for some developers.

And thanks to being a major bottleneck, builds are the bane of many a development cycle.

You might need to start a new build for many different reasons - from source changes and schedules to post-process events - and even incremental builds on a branch can significantly slow down the development cycle.

It's not hard to see why these rising build times have been an issue for developers. **In a 2023 survey, nearly 81% of C++ developers saw build times as a pain point, with up to 43% defining it as a major pain point.**

But a lot of the most popular languages and tools, such as C++ and Unreal Engine, are famous for their long build times, and the difficulty in improving them.

And it's about more than just tools — as software becomes better, larger, and more sophisticated, it takes longer to build it. Similarly, distributed or hybrid teams, multiple locations for assets, and converging pipelines from across multiple different areas of your organization rack up more time on the build clock.

In fact, long build times are also becoming more common.

Some developers might often spend weeks aggregating work from different projects into one, and spending many more weeks analyzing build logs manually to rewrite the entire build process in an effort to speed things up.

You get the picture.

No matter who you are, or what industry you're in, it's easy to see that long build times are spiraling out of control.

And yet so many developers see this as a necessary evil — all part of the process; an inconvenient but inevitable part of their job.

But what are long builds actually costing you, and how can you radically reduce the time they take to save time and budgets?

In this whitepaper, we will take a look at what we mean by a "long build", what the hidden costs might be to both your project and your developers, and three strategies to avoid sending your build times into a black hole.

What does a “long build” actually mean?

Builds can’t be immediate — as you’ll likely already know, most are far from instant.

But that doesn’t mean that all build times are made equal, either. In fact, there are plenty of factors that can impede your dev cycle: interrupted work processes, clunky workflows, and regular context-switching all impact how long your builds become.

To look into build times in more detail, let’s group them into three broad categories:

- **Short or unobtrusive wait** - these are fast builds that don’t require you to stop what you’re doing, or context-shift (that’s shift from one unrelated task to another). Incremental builds - where your software products are being incrementally designed, implemented, and tested - often work like this, and allow developers to keep their train of thought.
- **Noticeable wait** - this category is pretty self-explanatory. In these cases, you’re not fully context-switching, but you might get a long enough break in your workflows to consider doing something else in the meantime.
- **Full build wait** - this category is the most problematic for your builds. Very long builds can see projects screech to a grinding halt, with devs waiting until a new build completes so they can go back and see what actually happened. Common things you’ll see during full build waits are context-switching, broken trains of thought, and even boredom.

The not-so-hidden costs of a long build

No matter how you measure it, it's clear that long builds translate into higher costs. And some of the statistics calculated around business costs are pretty startling:

- StackOverflow's 2023 Developer Survey found that the annual cost of a developer in the US is around $165,000 per year, taxes and all. (Already, that's a $15,000 increase - 10% higher - compared to the previous year's Developer Survey figures.)

- Across the entire year, this developer cost breaks down to around $83 (USD) an hour.

- If a developer spends an hour waiting for a build and doesn't engage in other tasks during that period, the company is still spending an average of $83 for that developer's time. This means you're missing the opportunity for your developer to use that hour to actively build more code.

"Context-switching must be the answer then!" we hear you cry. Well, yes and no.

While context-switching could help developers get more done during build waits in theory, context-switching has actually been found to be more distracting and prevents focused and productive work.

In fact, studies by the University of California found that it actually takes an average of 23 minutes and 15 seconds to get back to their original task at hand after context-switching.

That's also looking at a wider net of people and professions — as development work often requires really deep focus and attention, this figure is likely to be much higher for devs.

Spending hours context-switching, and not working on a project, means that on top of resource costs - including your cloud or on-premise infrastructure costs, network costs, for example - you're also wasting your teams' valuable time.

By a significant margin.

Depending on how many times your devs need to wait for builds and change tasks for the day, this could eat into roughly three hours of an eight-hour day.

So companies could be waving goodbye to almost $250 a day per developer on context-switching alone.

Long builds waste time

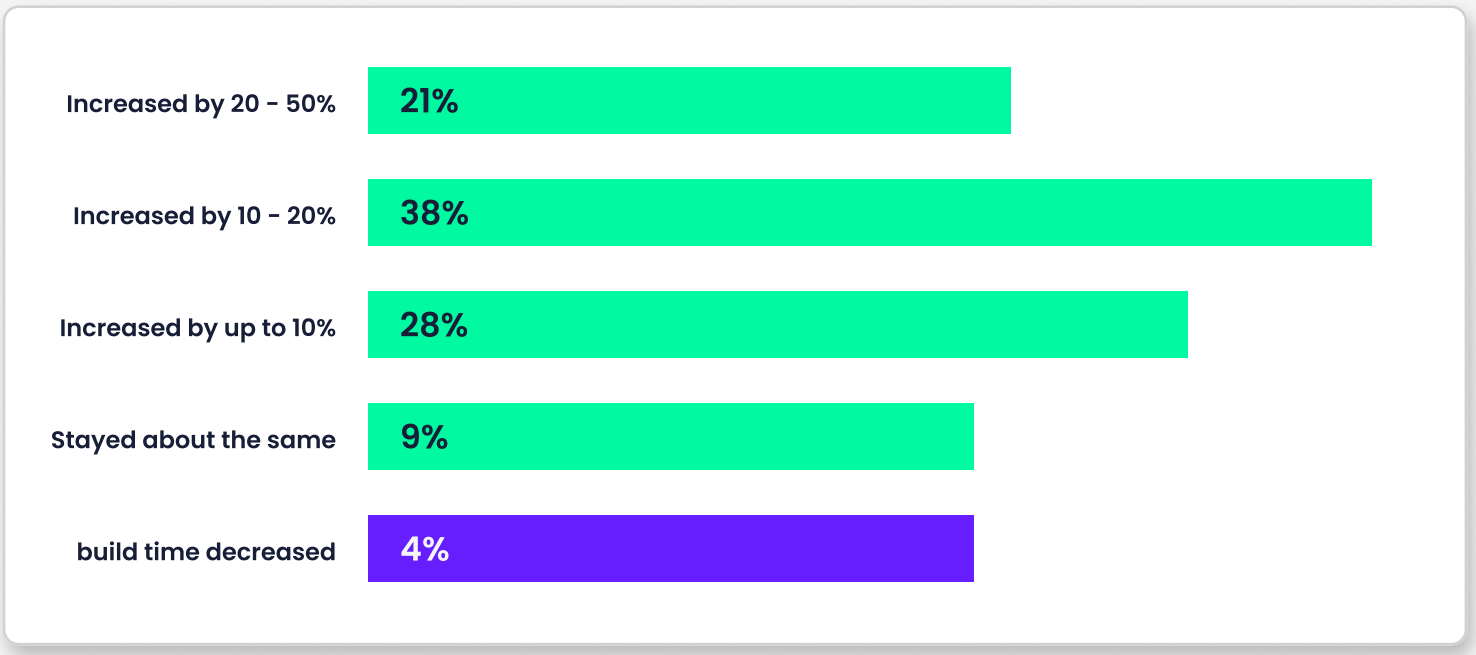
We've looked into the monetary costs of long builds and context - switching - but what about the development time lost to waiting around?

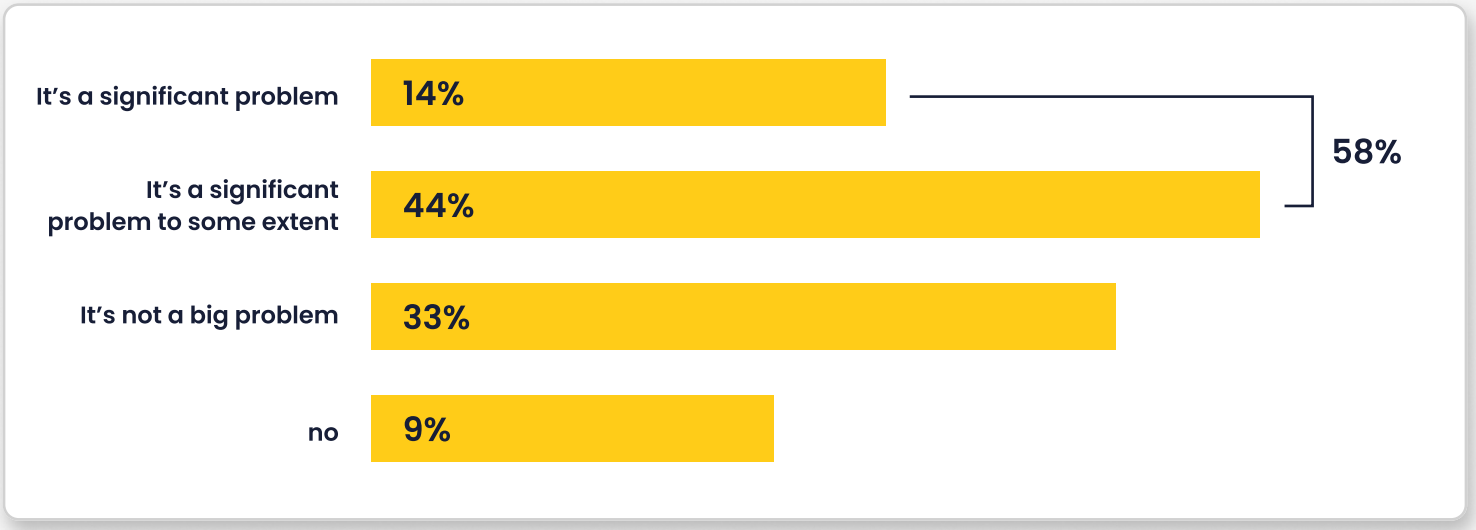
The builds of today take a long time. And it's getting longer.

Instead of allowing developers to build quickly, test, and get results, long build times slow down the development cycle and have a direct impact on productivity. In many cases, developers need to wait for builds to finish before they can move on to the next task, so this wait is literally time down the drain.

According to our "Big Dev Build Times" Survey Report in 2022":

- While build times can vary significantly, running a build takes an average of 20 minutes.
- Almost 25% of respondents spent more than 30 minutes on each build, jumping up to 32% for CI/Release builds.
- Respondents spent an average of 57 minutes a day waiting for builds to finish.
- 96% of respondents said that build time increased by an average of 15.9% in 2021.
- 69% of companies spent over 10 minutes running a build, with 24% of companies spending more than 30 minutes.
- In addition, 98% of respondents admitted they waste time waiting for builds to finish.
- Slow build times are a problem for 91% of companies. Only 33% said they don't consider them a huge problem, but 58% said they're a significant problem or a problem to some extent.
- Build wait times also showed to vary by industry. In 2022, the defense and military industry averaged the slowest builds at 72 minutes, while the banking industry averaged the fastest builds at 38 minutes.
- The three industries that suffered the most from slow build times were defense and military (73%), IT services (67%), and automotive (63%).





Even the fastest build wait time of 38 minutes in the banking industry adds up to a lot of wasted time in terms of productivity.

By slowing down the iteration cycles, long builds reduce the ability to quickly test and "fail fast" - or quickly spotting and fixing errors.
Instead, long builds mean it's more likely that errors are tucked away for later - a nasty surprise for the developer who finds it when they're finally able to test again.

With less time to iron out last-minute kinks before release, DevOps teams and IT departments then have to race against the clock to fix the problems which, again, involves spending even more money.

Three strategies to cut down your build times

1. Build caching

Caching is the process of storing multiple copies of data in a temporary location so they can be accessed faster at a later time. It's used for software applications, servers, and web browsers, among other things.

But the one key advantage of caching is that it means users and apps don't need to start from scratch whenever something is booted up. Websites use caching to speed up the loading process for web pages, for example.

So devs can use caching in some handy ways to speed up build times.

In software builds, whether they are being built incrementally or from scratch, this means that some build outputs and artifacts can be stored for reuse later — a powerful strategy for significantly reducing build times with minimal effort.

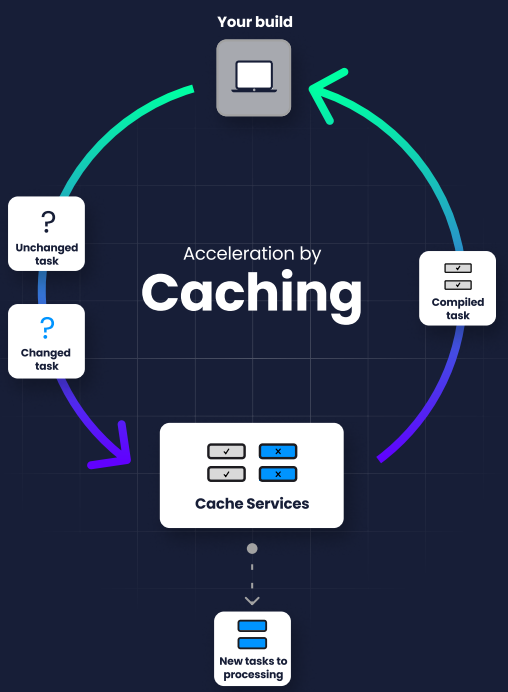
This table includes more advantages to using caching for your builds:

Boosts developer productivity	True "work from anywhere"without impacting speed
• Don't need to completely rebuild when switching branches or introducing new code • Minimizes wait times • Reduces redundancy • Optimizes resource use • Empowers devs to keep focus and momentum when processes are expedited • Encourages devs to adopt best practices • Results in more streamlined and effective software development workflows	• Minimizes the impact of network latency • Uses downstream bandwidth instead of upstream bandwidth, so devs can work seamlessly from various locations • Addresses challenges of remote and distributed work environments, which are becoming more prevalent • Reduces dependence on centralized build servers • Allows devs to be more responsive to changing business needs and market demands

Reuses cached data between CI builds	Allows faster builds
• Prevents time-wasting • Optimizes the way devs use resources • Streamlines continuous integration processes • Stores and retrieves previously downloaded artifacts and intermediate build states for more efficient builds • Helps companies optimize infrastructure costs and reduce resource consumption • Minimizes the need for extensive compute resources and storage capacity • Allows organizations to scale development environments more effectively and allocate resources where needed most • Offers a more cost-effective approach to resource management • Helps companies achieve better efficiency and ROI through software development	• Significantly reduces build time by pulling from cached versions of previous builds • Shifting left to build per commit is more possible with shorter dev times • Faster response times for developers • Better coping with crunch times — the more builds executed, the more efficient the cache • Faster time for bug resolution and time to market • Lowers costs — especially when using cloud resources for computing • Saves on CI server licenses, as each CI server can complete more builds in parallel

How does Incredibuild handle build caching?

We've developed a distinctive build caching method built on our platform's strong and tested parallel distribution technology. This technology involves low-level system-hooks injected into processes, similar to how anti-virus software operates. With this approach, we can automatically track every file read and other inputs accessed by a process. This seamless and generic mapping capability helps identify task dependencies, relieving both you (the user) and the tool from added complexities.



2. Write build-friendly code

No matter how good your tools, the easiest way to cut build times is to write code that's designed for faster builds.

When writing code, it's easy to focus on your convenience and take shortcuts to write things quicker.

(In fact, we've already written about how to enhance your overall code quality with GitHub on our blog — check it out here.)

But these code-writing practices aren't always compatible with fast builds — they might add dependencies, make code more complex than it needs to be, or simply just add needless lines of code that need to be compiled.

Here are three easy ways to resolve this issue.

Reduce dependencies
Trimming down dependencies is vital for efficient builds. The more you intertwine all your different files, components, modules, and layers, the more complex your build becomes, and your build times will grow longer and longer.

Instead, it's always best to streamline your build as much as you can. Spot the vital dependencies, pinpoint those you could get rid of, and assess which decoupling efforts need more work.

As your internal code quality improves, you'll notice your compilation times will speed up.

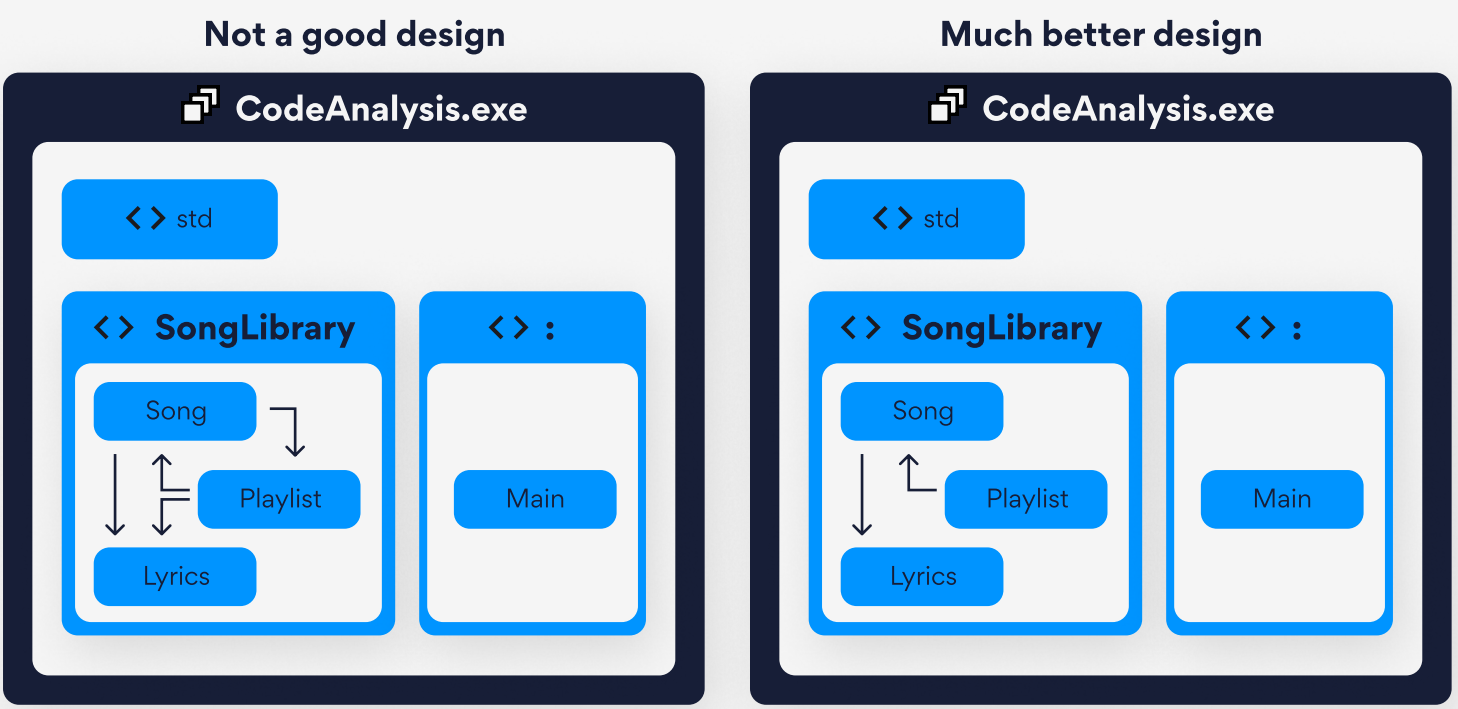
If in doubt, stick to the Dependency Inversion principle — a well-known SOLID principle — for build-time reduction. Are your header files suitable for a compilation unit? If not, it's time to cut them adrift. Tools like include-what-you-use are great for purging these dependencies from your existing codebases.

Avoid breaking code into too many small functions

Breaking code down can massively increase the amount of time it takes to call it - or function call overhead. This can also have a massive impact on your build times.

Instead, try to hit the sweet spot between optimizing manually, compiler options, and good coding practices. This will not only improve your performance, but it also means you're not introducing more unnecessary complex code or getting in the compiler's way of optimizing your code.

Compiler options like '-qipa' or these compiler options for Mac can help minimize the impact and improve your optimizations.



Use precompiled headers

Finally, this is one of our favorite ways to cut down your C++ compilation time. Precompiled header files are like time-savers in a digital world. These files store rarely modified headers in an intermediate form, accelerating local compilations without any downsides.

Precompiled headers are binary files generated from C++ header files, already parsed and pre-processed. When you're in the process of building, precompiled headers check if they've been previously built, skipping them if they have, which can slash compilation times by up to six times.

(On a side-note: be cautious with distributed builds. While precompiled headers can speed things up locally, they tend to aggregate units in distributed scenarios, potentially slowing down the process. So make sure you keep an eye on changes in precompiled headers; if they're updated frequently, their benefits may dwindle.)

Another thing to think about is upgrading your compiler. Modern compilers are constantly improving, and getting more efficient at optimizations and code generation while speeding up compilation. Switching from an older compiler to a newer one can noticeably improve your C++ compile times.

Can AI fix build times?

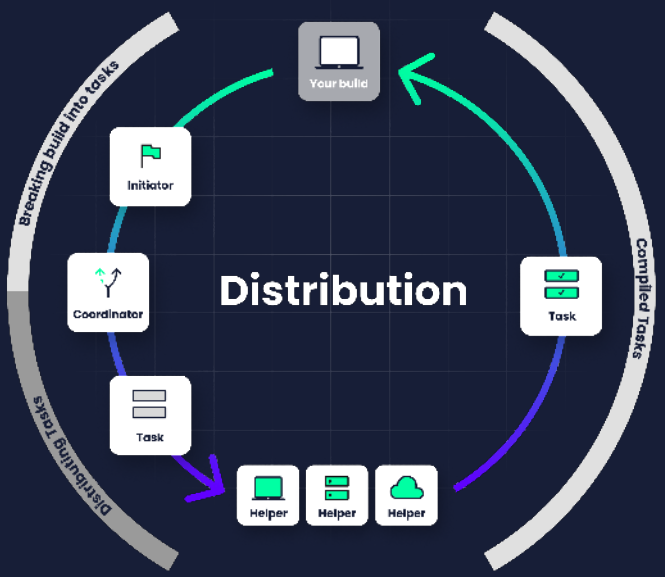
While build times are still notoriously difficult to get down, there are some hopeful fixes on the horizon. However, one major trend that could stand in the way of these fixes is artificial intelligence (AI). It's estimated that AI will actually slow down build times, creating more of a bottleneck thanks to using far more code and moving it faster into the pipeline. With 92% of US developers already using AI coding tools both in and outside of work, it's definitely an interesting one to watch for future developments!

3. Optimize your hardware and resource usage

One easy way to build faster is to have better hardware – or use your existing hardware and resources in a smarter way. Let's look at a few ways you can do this:

- Get more hardware – This is the fastest way to get faster builds - add extra memory, CPUs, hard drives, and servers. It comes with a few small issues: it's also expensive and not super scalable. To stop you from running out of space, money, and resources (or for the very unlucky, all of the above), find the point where the economies of scale start setting in and having more won't make things faster or better, just more expensive. And just don't go beyond that point.
- Enable more cloud resources – Like the point above, you can easily spin up as many resources as you need or want. But at some point, you need to pay up. More cloud instances mean higher bills, higher likelihood you'll forget about some of them, and more time spent managing them — so use with caution.
- Do more with what you already have – Unlike the other two alternatives, you can always find new ways to use your existing resources. One way to do this is to enable distributed builds and build infrastructures and pipelines that support getting more out of the existing hardware, or that prioritize more cost-effective mechanisms to use the cloud resources you have.

What are distributed builds?
Distributed builds allow you to take your builds, break them down into parts, and run smaller sub tasks and components in parallel. This reduces the time spent per build, the resources necessary per user, and engages more of your existing infrastructure instead of requiring more. You can also do this with cloud builds - here's an example we made earlier.



The long (but hopeful) road from long builds

Let's get one thing straight: Great software takes time to build, and that's not going to change anytime soon.

But there are always things you can do to mitigate these delays, and slash your build time as much as possible.

By using three key strategies — build caching, writing build-friendly code, and optimizing hardware and resource usage — to combat dragging build times, developers can minimize build times without any additional expense, time, or stress.

Developers that implement these measures can dodge the dramas of prolonged build times, foster a more agile development environment, and enhance overall project delivery efficiency.

About Incredibuild

Incredibuild is the leading platform for Development Acceleration.
Our Virtualized Distributed Processing™ and patented Build Cache technologies help you build faster and lower development costs on-prem and in the cloud so your dev teams can iterate more, fix bugs faster, and build great products.
Incredibuld also helps you get more out of your current investments in hardware and optimize cloud resources to get more bang for your buck. Incredibuild doesn't require changes to your existing tools or processes, it's a lightweight, code-free solution that most teams can get running in hours. Incredibuild is designed to accelerate distributed dev teams, even those working from home and with low bandwidth - right from the first sprint.

Find out more about how Incredibuild works: Incredibuild.com

INCREDIBUILD