# Incredibuild Acceleration Benchmark: **CMake projects in Visual Studio**

INCREDIBUILD

## Table of Contents

Starting with VS2017, Visual Studio comes bundled with Modern CMake. This makes it very easy to follow the CMake Visual Studio workflow to be adopted even for large projects. Now that CMake supports project generation and build for C# language, we expect CMake Visual Studio to become more popular in the Windows world.

Here is a summary of why using Visual Studio CMake for your project makes sense:

- Maintaining multiple projects and solutions in different branches is no longer needed. They are autogenerated from CMakefiles.txt file.
- A new version of Visual Studio does not necessitate the migration of solutions and projects. It is easy to be on the cutting edge of Visual Studio versions.
- Merging parallel work from different branches becomes a lot easier since there are no project/solution changes to be merged.
- Maintaining batch or Perl scripts is no longer necessary, as CMake can automate builds.
- CI/CD pipelines built over CMake are a lot more maintainable than custom solutions based on homegrown scripts.

**TIP:** Use modern CMake. If your project is still using CMake versions below 2.6, spend some time and effort to move to the newer versions. The thumb rule is to use the version of CMake that came after your compiler version.

## Using Visual Studio + CMake + Incredibuild

In addition to CMake, Visual Studio also comes bundled with Incredibuild which can now be used seamlessly to accelerate CMake builds from within Visual Studio.

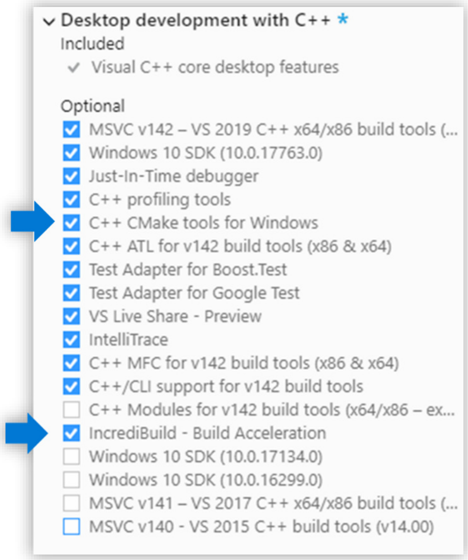
The backbone of Incredibuild's offering, Virtualized Distributed Processing™ enables a workload that consists of multiple, concurrent processes to be automatically and dynamically distributed to hundreds, and even thousands of idle CPUs on remote machines across your network or public cloud.

As CMake builds and C++ builds in general consist of hundreds of compilation tasks that can be executed in parallel, having hundreds of cores at your disposal can highly accelerate build times, which is exactly what distributed computing is all about.

Virtualized Distributed Processing™ can even use idle CPUs on remote machines while users are working on them – operating in the background. In organizations that have hundreds of machines, the aggregated number of idle CPUs in any given moment can easily be in the thousands. These are wasted cores that Incredibuild recaptures to accelerate time consuming workloads in need of computing power.

Incredibuild runs processes on remote machines in a secure sandbox. Everything a process requires to run properly is dynamically emulated by Incredibuild from the local host to the remote machine. **This means all you need to install on remote machines is the Incredibuild Agent – there's no need to install Visual Studio, nor your source code or any other build tools**. Any output generated by the remotely executed process - std output, errors, return codes, files generated, etc. – is automatically synched back to the local host, as if the process had been executed locally.

In the Visual Studio Installer, ensure that the checkboxes for "C++ CMake tools for Windows" and "IncrediBuild – Build Acceleration" are selected:



Desktop development with C++ *

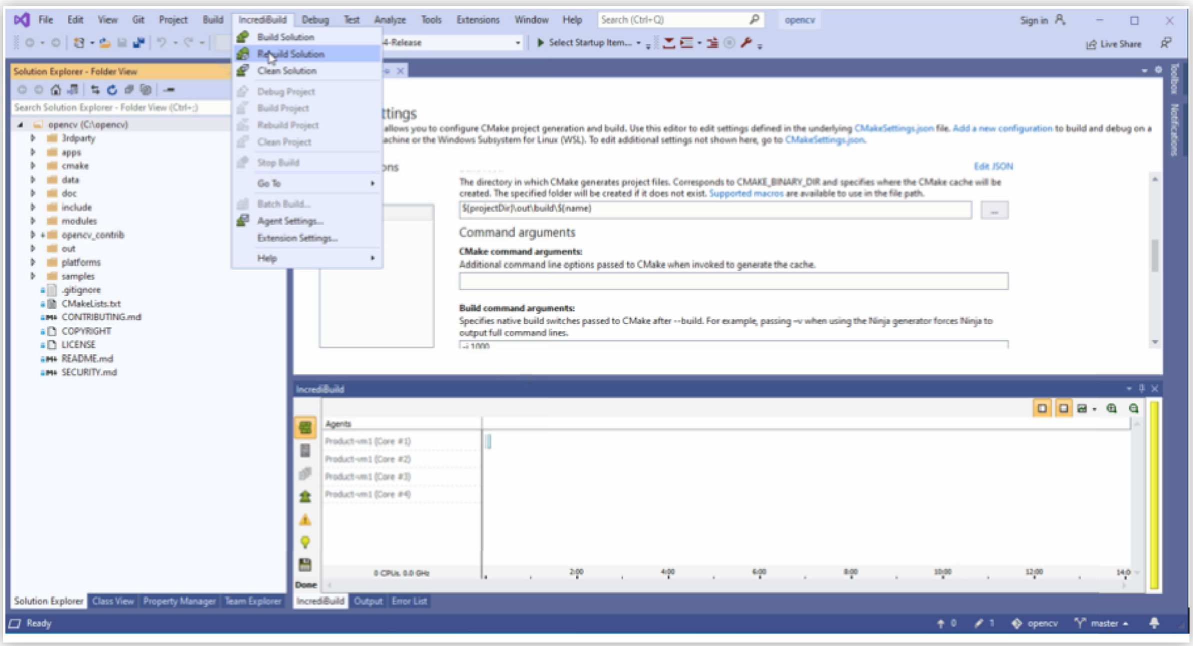
Included

✓ Visual C++ core desktop features

Optional

- [x] MSVC v142 – VS 2019 C++ x64/x86 build tools (...
- [x] Windows 10 SDK (10.0.17763.0)
- [x] Just-In-Time debugger
- [x] C++ profiling tools
- [x] C++ CMake tools for Windows
- [x] C++ ATL for v142 build tools (x86 & x64)
- [x] Test Adapter for Boost.Test
- [x] Test Adapter for Google Test
- [x] VS Live Share - Preview
- [x] IntelliTrace
- [x] C++ MFC for v142 build tools (x86 & x64)
- [x] C++/CLI support for v142 build tools
- [ ] C++ Modules for v142 build tools (x64/x86 – ex...
- [x] IncrediBuild - Build Acceleration
- [ ] Windows 10 SDK (10.0.17134.0)
- [ ] Windows 10 SDK (10.0.16299.0)
- [ ] MSVC v141 – VS 2017 C++ x64/x86 build tools (...
- [ ] MSVC v140 - VS 2015 C++ build tools (v14.00)

Once installed, an Incredibuild toolbar and menu appear in the Visual Studio development environment, offering Incredibuild's distributed Build and Rebuild operations.



ttings

allows you to configure CMake project generation and build. Use this editor to edit settings defined in the underlying CMakeSettings.json file. Add a new configuration to build and debug on a achine or the Windows Subsystem for Linux (WSL). To edit additional settings not shown here, go to CMakeSettings.json.

The directory in which CMake generates project files. Corresponds to CMAKE_BINARY_DIR and specifies where the CMake cache will be created. The specified folder will be created if it does not exist. Supported macros are available to use in the file path.

`${projectDir}\out\build\${name}`

Command arguments

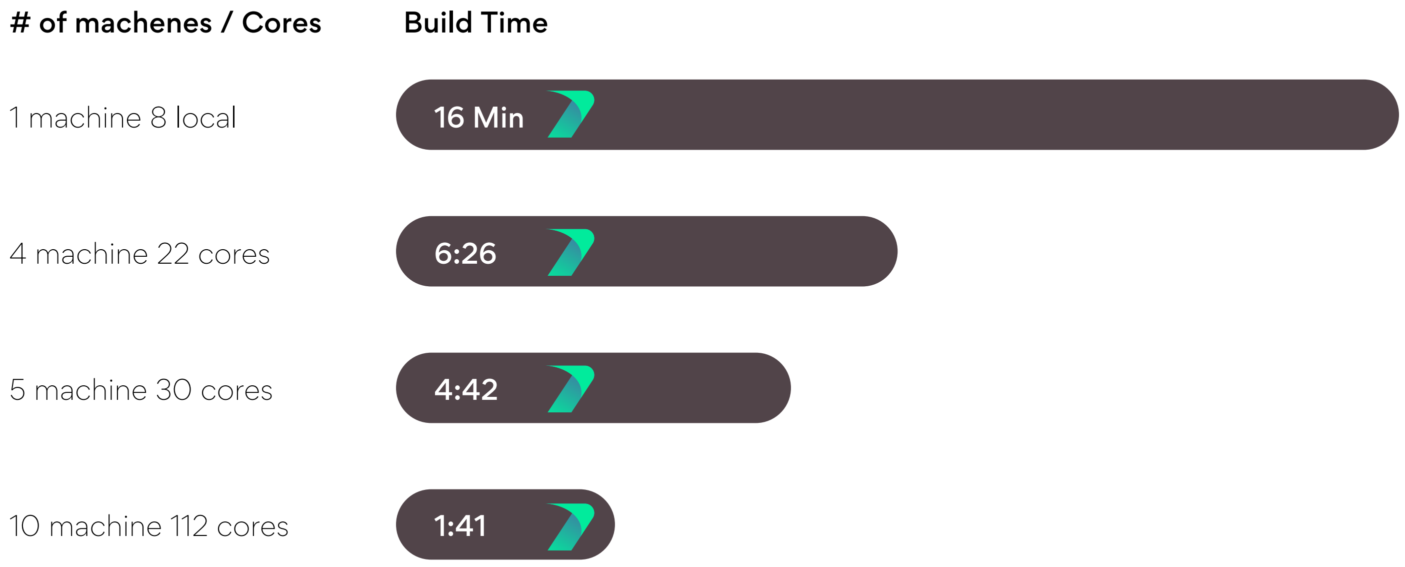
**CMake command arguments:**
Additional command line options passed to CMake when invoked to generate the cache.

**Build command arguments:**
Specifies native build switches passed to CMake after --build. For example, passing -v when using the Ninja generator forces Ninja to output full command lines.

`-j 1000`

## How much faster are CMake builds with Incredibuild?

The following data is based on running CMake with MSBuild by compiling the popular core OpenCV open-source project to establish a known base line. This example uses the Ninja generator, but all the Visual Studio generators are supported. Here are the results with Incredibuild:



All in all, we got a 9.5X performance boost which means developers can spend more time building great code in the zone and less time waiting for code to build. In this specific use-case, adding additional cores to the Incredibuild pool, on top of the 100 cores used in this example, will result in even better compile time.

**The cherry on top:** you can use the same Incredibuild infrastructure to accelerate your CMake project under your CI/CD of choice for the full experience and to also accelerate unit tests that are part of your build or other compute intensive processes such as code analysis, code signing, various test types and more.

## Here's how we ran the benchmark?

### Step 1 – cloud and infrastructure

We have decided to use the AWS c5 family infrastructure to run the benchmark, to reach extensive compute power and flexibility. Also, we have used it in a pure cloud model. Due to the nature of the infrastructure, we used RDP access, resulting in optimal results, speed wise, latency wise and security level wise.

### Step 2 – Installing and license

Installing Incredibuild is super simple, a few clicks and you are good to go.

When installing on a new machine, Incredibuild is the only tool you'll need to install – Incredibuild will perform the distribution, utilizing the Incredibuild proprietary VE, which handles the distribution of the tool sets as well.

We used the latest official version of Incredibuild and proceeded to install the coordinator, which acts as the mediator machine and as the license server.

The coordinator monitors who requested build compilation assistant, and which machines are free to aid the running build.

After installing the coordinator, we loaded the license onto it, installed the same version on 4 additional machines, verified they all point at the coordinator IP, and once all machines were registered on the coordinator monitor, the environment was ready.

## Step 3 – Checking for build requirements

Running CMake / OpenCV requires either Visual Studio, or command line invocation.

As Incredibuild supports both options, and we have used Visual Studio, we needed to update the license to support Visual Studio.

## Step 4 – preparing the build

We have Installed Visual Studio on one of the cloud machines which will be the initiator (the machine which launches the build).

We downloaded and installed Open CV and set it to run through CMake.

We were than ready to launch the build.

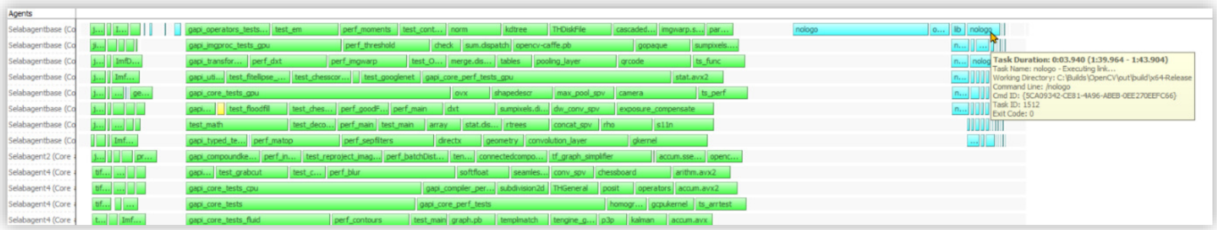
## Step 5 – running the build

We have run the build several times to produce benchmark numbers and assess performance.

First, we ran the build on only one machine. This allowed us to understand how long the build compiles on one machine, with Incredibuild and without.

The benchmarks involved a different number of cores, using all helpers.

## Step 6 – Analyzing build results using Incredibuild Build Monitor



```
Task Duration: 0:03.940 (1:39.964 - 1:43.904)
Task Name: nologo - Executing link...
Working Directory: C:\Builds\OpenCV\out\build\x64-Release
Command Line: /nologo
Cmd ID: {5CA09342-CE81-4A96-ABEB-0EE270EEFC66}
Task ID: 1512
Exit Code: 0
```

Analyzing the results, it showed the build which utilized all cores presented the best results of **1 minute and 42 seconds**.

The build monitor launched automatically as the build starts, providing us a with a real time glance of machines used, workloads currently running and the various build parts. The blue, representing linking, provided us with the ability to identify bottlenecks within the workload.

## To learn more visit our website or download our free license.