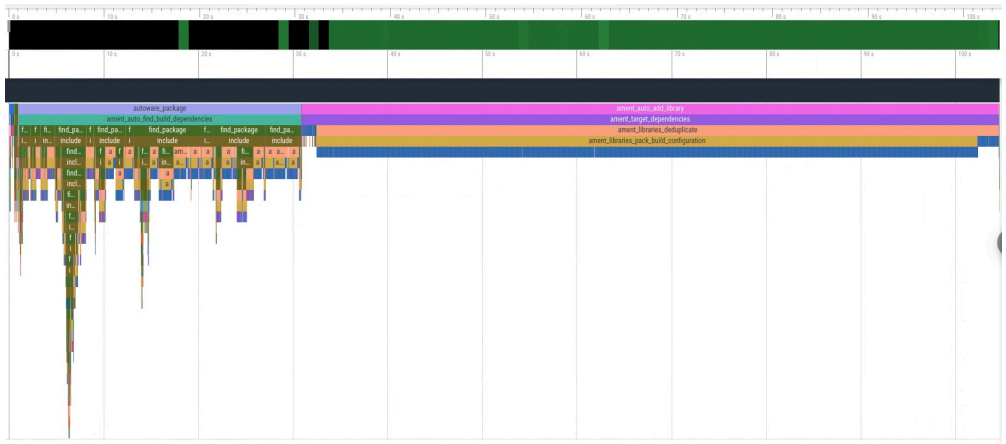
Beyond `ament_auto` #3491

VRichardJP started this conversation in **Design**

VRichardJP on May 12, 2023 Collaborator edited

Context

Autoware uses extensively the `ament_auto` macros. Unfortunately, these macros are slow when packages have a lot of dependencies, which is the case of many Autoware packages. For example, this is what cmake execution on `behavior_path_planner` looks like:



This problem could be fixed upstream, however a fix would not land until the next next release of ROS2 (not Iron, but J release). In other words, we can't expect an upstream fix before the next 1~2 years :(

How much time is spent in CMake exactly?

You can build Autoware with `--profiling-format=google-trace --profiling-output=cmake_profile.json` to trace cmake execution on each package. There are a lot of packages, so you can use this simple script to quickly find which packages are slow:

```
#!/bin/bash
# Usage: ./min_max_profile.sh | sort -k2 -n
for f in $(find build -name "cmake_profile.json"); do
  echo "$f" 1>&2
  min=$(jq 'map(.ts | values) | min' $f)
  max=$(jq 'map(.ts | values) | max' $f)
  diff=$(echo "$max - $min" | bc)
  echo $f $diff
done
echo "--- done ---" 1>&2
echo "" 1>&2
```

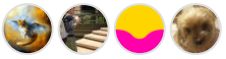
Category

Design

Labels

component:system

4 participants



Reported value is ~CPU time in microseconds:

```
$ ./min_max_profile.sh | sort -k2 -n
build/autoware_lint_common/cmake_profile.json 353345
build/autoware_cmake/cmake_profile.json 492184
build/tier4_planning_launch/cmake_profile.json 671057
build/livox_description/cmake_profile.json 695467
build/map4_localization_launch/cmake_profile.json 705889
...
build/eagleye_rt/cmake_profile.json 57116249
build/behavior_path_planner/cmake_profile.json 65776780
build/behavior_velocity_planner/cmake_profile.json 77437622
build/trajectory_follower_node/cmake_profile.json 89506475
build/static_centerline_optimizer/cmake_profile.json 120522910
```

In total, that is 2528 seconds (~42 minutes) of ~CPU time spent in CMake on my machine (if you have 8 cores on your machine and live in a perfect world, that is at least 5 minutes). Of course it is not possible to reduce this time to 0, but I hope I can convince everyone there might be some room for improvement on the cmake side. As reported on the `ament_cmake` github issue, CMake is slowed down by 2 things:

- deep recursive `find_package`: this is a tricky problem to solve, because dependencies are always added, never removed. So the deeper a package is, the longer it will take to find all dependencies. Essentially, the only way to make this faster is for ament to generate faster FindXXX.cmake files. I have recently made a PR that drastically reduce `find_package()` processing time.
- `ament_auto_add_library` / `ament_auto_add_executable`: this is due to the ament auto macros going through the same dependencies over and over. It does not have to be like this, and it could be easily fixed by using modern CMake targets.

For the slow `find_package`, there is a some chance the fix will land on humble. But for the slow `ament_auto` macros, the fix would not be available until J ROS2 release.

Ok, so what do we do?

Option 1: bear with the long cmake time until `ament_auto` macros are fixed.
Option 2: get rid of `ament_auto` macros, and use base `ament_cmake` macros instead.
Option 3: fork of `ament_auto`, and fix the fork.
Option 4: use an alternative to "ament_auto" macros, that would be simple, fast and perfectly fit autoware's use case.

Option 1 is simple, but you can guess it is not my favorite.
Option 2 would be a regression in my opinion. It would solve the speed issue, but would also inflate all cmake files and increase the chances of messing up packages (i.e. most likely require more maintenance).
Option 3 could be a decent option. However, changing the behavior of "ament_auto" macros would impact all packages, not just autoware packages, so it would mean maintaining backward compatibility (the reason the upstream "ament_auto" would not add breaking changes to its humble release in the first place).

I let you guess which option I worked on :-)

An alternative to ament_auto

Over the last few days, I have written a PoC `ament_cmake_extension` package. It serves the same purpose than `ament_cmake_auto`: to provide an easy-to-use abstraction layer on top of `ament_cmake` base macros. Its API is very close to `ament_cmake_auto`, so transitioning from `ament_cmake_auto` is often just a matter of a few `sed`. In my opinion, it provides a simpler, faster and more foolproof API than "ament_auto" macros. One key difference is that it uses exclusively modern CMake targets instead of the classic `_LIBRARIES`, `_INCLUDE_DIRS` and `_DEFINITIONS` variables, which makes cmake files both cleaner and faster.

It creates valid ament packages, so it is possible to mix packages created with `ament_cmake_auto` and `ament_cmake_extension`. In order to test it in Autoware, I have added a simple option to `autoware_cmake()` macros, so that it is simple to switch between the two APIs:

```
--- a/autoware_cmake/cmake/autoware_package.cmake
+++ b/autoware_cmake/cmake/autoware_package.cmake
@@ -13,6 +13,8 @@
 # limitations under the License.

 macro(autoware_package)
+  cmake_parse_arguments(_ARG "USE_AMENT_EX" "" "" ${ARGN})
+
   # Set compile options
   if(NOT CMAKE_CXX_STANDARD)
     set(CMAKE_CXX_STANDARD 17)
@@ -45,8 +47,13 @@ macro(autoware_package)
   endif()

   # Find dependencies
-  find_package(ament_cmake_auto REQUIRED)
-  ament_auto_find_build_dependencies()
+  if(_ARG_USE_AMENT_EX)
+    find_package(ament_cmake_extension REQUIRED)
+    ament_ex_find_package_dependencies()
+  else()
+    find_package(ament_cmake_auto REQUIRED)
+    ament_auto_find_build_dependencies()
+  endif()
```

To use `ament_ex` macros instead of `ament_auto`, just add `USE_AMENT_EX` flag to `autoware_package()`:

```
find_package(autoware_cmake REQUIRED)
autoware_package(USE_AMENT_EX)
```

As I reported in the original issue, a few adjustments are necessary on autoware cmake files side. This is because many packages create ill-formed targets, or rely on dirty hacks that break as soon as you try to export/import targets. Fixing the cmake files is a bit boring but not difficult. Once it's done, we end up with very clean cmake files and of course fast build!

Is it really faster?

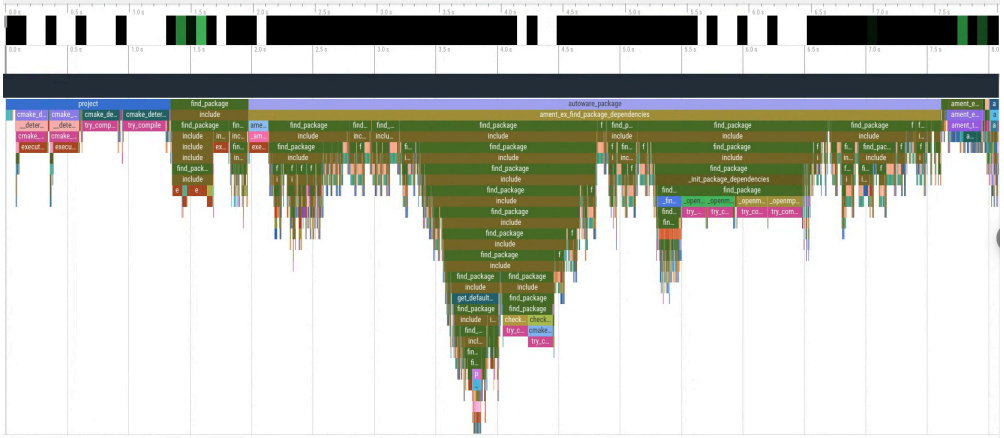
As proof-of-concept, I have applied this PR, sed-ed all autoware.universe packages to make use of `ament_ex` macros, then spent a couple hours fixing all the build issues. If you are curious how `ament_ex` macros are used, I have pushed my changes here:
https://github.com/VRichardJP/autoware.universe/tree/ament_ex

Using the same script to report each cmake invocation time, now I get:

```
$ ./min_max_profile.sh | sort -k2 -n
build/autoware_lint_common/cmake_profile.json 321976
build/ament_cmake_extension/cmake_profile.json 365491
build/map4_localization_launch/cmake_profile.json 631369
build/tier4_planning_launch/cmake_profile.json 658787
build/autoware_cmake/cmake_profile.json 688222
...
build/tensorrt_yolox/cmake_profile.json 12911722
build/heatmap_visualizer/cmake_profile.json 13489099
build/static_centerline_optimizer/cmake_profile.json 15131815
build/tensorrt_yolo/cmake_profile.json 20309413
build/obstacle_velocity_limiter/cmake_profile.json 36402255
```

The total CPU time is now 1057 seconds (~18 minutes), more than twice as fast!.
Note: For this experimentation I have only modified autoware.universe packages. Other packages would gain from using `ament_ex` macros, but maybe not as much.

This is what the cmake trace looks like on `behavior_path_planner` now (to compare with the old one above):



And clean build are of course faster:

- before:

```
real    50m51.785s
user    302m5.073s
sys     20m42.933s
```

- after:

```
real    44m46.038s
user    261m18.154s
sys     19m3.339s
```

It's more than 10% faster. Not only CMake execution is faster, but also the compilation itself.

A transition would takes some time (~250 packages...), but I think the long term gain is worth it.
What do you think?

↑ 5 👍 7 👀 1

5 comments · 9 replies

Oldest | Newest | Top

kenji-miyake on May 12, 2023

Thank you for your work and proposal!
I personally prefer Option 3, but Option 4 may be acceptable as a short-term.

@xmfcx @mitsudome-r @yukkysaito What do you think?

↑ 2 0 replies

VRichardJP on May 12, 2023 Collaborator Author edited

Just to add a few details, I don't only propose to move away from `ament_auto` only for the performance. I find `ament_auto` has some design issues or questionable defaults, which would not be fixed upstream before ROS2 J release (if ever fixed).

So, implementing an ament_auto "v2" (whether it is a fork or a new package) would also be the opportunity to add "missing" features, or improve the macros behavior. A few examples:

- There is no construct to properly link and export system libraries (e.g. PCL). For instance, there are many cmake files in Autoware that do something like:

```
find_package(PCL REQUIRED)
target_include_directories(myTarget ${PCL_INCLUDE_DIRS})
target_link_libraries(myTarget ${PCL_LIBRARIES})
```

and forget about `target_compile_definitions(myTarget ${PCL_DEFINITIONS})` or `ament_export_dependencies(PCL)`, which may cause dependency issues on downstream packages (unless PCL is slapped in over and over, which is what is done actually.)

- there is no construct to handle targets that can't be created with `ament_auto_add_library` (e.g. CUDA libraries must use `cuda_add_library()`)

↑ 2 0 replies

xmfcx on May 15, 2023 Maintainer

Wow this was a lot to take in lol.
I've never profiled a CMake build before, getting to know how much time is spent there, I've learned a lot of things today, thank you for working on this.

So, a lot of time is spent on:

- Recursive `find_package()` calls. (I don't know if it's possible to improve this)
- `ament_libraries_deduplicate()` function. (It's also called within find_package() calls too)
  - Your PR fixes this.

And also you've created this ament_cmake_extension package and also modified your fork of Autoware Universe to make use of it.

Since I'm not too experienced with CMake that much, replacing ament_cmake_auto with ament_cmake_extension which is built from ground up scares me a bit.

> **Option 3:** fork of ament_auto, and fix the fork.
> **Option 4:** use an alternative to "ament_auto" macros, that would be simple, fast and perfectly fit autoware's use case.
> **Option 3** could be a decent option. However, changing the behavior of "ament_auto" macros would impact all packages, not just autoware packages, so it would mean maintaining backward compatibility (the reason the upstream "ament_auto" would not add breaking changes to its humble release in the first place).

What about something between 3 and 4?
Option 5:

- Copy ament_cmake_auto out of `ament_cmake` and create a new package with a different name like yours.
- Fix the problems in this new package with your suggestions.
- Use it in the Autoware instead of `ament_auto_xx` calls.
- This way, the change is more gradual and controlled, less chance of unexpected failures in the future.

This is because I find it risky to make such a drastic change. What do you think **@VRichardJP** **@kenji-miyake** ?

↑ 1 👍 1 6 replies

Show 1 previous reply

yukkysaito on May 16, 2023 Maintainer edited

@VRichardJP The results of this survey are very informative for me. Thank you 👍

@xmfcx **@VRichardJP**

> This is because I find it risky to make such a drastic change.

Yes, I think there is a risk.
How about the following approach?

1. Copy ament_cmake_auto out of ament_cmake and create a new package with a same name in autoware.
2. Fix the problems in this new package with your suggestions.
3. In the future, when changes are introduced into the original ament_cmake_auto, remove the copied ament_cmake_auto.

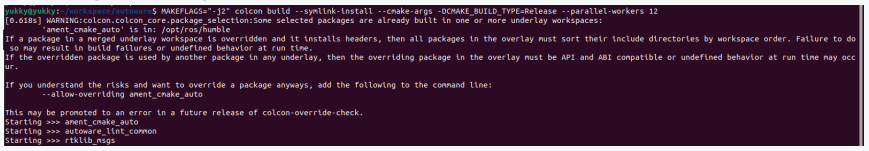
I believe that the package in the workspace should have priority. Sorry if I am wrong. (It should do so in ROS 1, but I haven't tested it in ROS 2.)I'll test it.
Since it is the same ament_auto, code compatibility can be maintained.

yukkysaito on May 16, 2023 Maintainer

I did some testing.

- with `--allow-overriding ament_cmake_auto`, ament_cmake_auto in the workspace was used.

- without `--allow-overriding ament_cmake_auto` , ament_cmake_auto in the workspace was used. But there is warning message.



- with `--packages-ignore ament_cmake_auto` , `/opt/ros/humble/share/ament_cmake/ament_cmake_auto` was used.

VRichardJP on May 16, 2023 Collaborator Author edited

> 3. In the future, when changes are introduced into the original ament_cmake_auto, remove the copied ament_cmake_auto.

I don't want to leave any misunderstanding here: there is a chance the changes I propose would never land in the `ament_cmake_auto` package. Maybe in a `ament_cmake_auto2`, but that's exactly what `ament_cmake_extension` tries to be. This is because several of the changes I introduce are breaking:

- starting from exporting cmake targets (what makes `ament_cmake_extension` faster and more robust). If you try to export malformed target (e.g. a dependency it is linked to is not exported), ament CMake will raise a build error.
- then, many of the small QoL changes, which makes it easier to work with targets, are as many breaking changes as well.

Cmake targets require extra care, but there are worth it. There are faster, simpler to use, won't cause dependency issues downstream...

I don't like `--allow-overriding` for several reasons:

- it's an extra flag
- it impacts all the packages at once
- to keep the "ament_auto_*" macros around while "sneakily" changing their behavior is bound to puzzle many developers.

xmfcx on May 17, 2023 Maintainer

> I don't like --allow-overriding for several reasons:
>
> - it's an extra flag

The `--allow-overriding` is just for hiding the warning that comes up. It is not mandatory. Whether it exists or not, the `ament_cmake_auto` will take priority.

> - it impacts all the packages at once

It impacts all packages that are in the `autoware/src` directory, isn't that the case? As far as I know, we only link to the packages outside the Autoware workspace.

> - to keep the "ament_auto_*" macros around while "sneakily" changing their behavior is bound to puzzle many developers.

This is the only downside I can see. But we can add it to documentation with an explanation. It shouldn't have much side effects.

The upsides:

- Users don't need to learn a new CMake tool/library.
- We don't need to make too many changes in the Autoware repository.

> This is because several of the changes I introduce are breaking:
>
> - starting from exporting cmake targets (what makes `ament_cmake_extension` faster and more robust). If you try to export malformed target (e.g. a dependency it is linked to is not exported), ament CMake will raise a build error.

Is it possible to make it faster with less breaking changes?

VRichardJP on May 17, 2023 Collaborator Author edited

> - it's an extra flag
>
> The `--allow-overriding` is just for hiding the warning that comes up. It is not mandatory. Whether it exists or not, the `ament_cmake_auto` will take priority.

Sure, it is just a minor hindrance.

> > it impacts all the packages at once
>
> It impacts all packages that are in the autoware/src directory, isn't that the case? As far as I know, we only link to the packages outside the Autoware workspace.

Yes, this is what I mean. Autoware source tree does not contain only autoware packages. There are external projects aswell. If we fork and change the behavior of `ament_auto_*` macros, then these external packages could break. To have a new name means to be able to mix the old/new behavior.

> > to keep the "ament_auto_*" macros around while "sneakily" changing their behavior is bound to puzzle many developers. This is the only downside I can see. But we can add it to documentation with an explanation. It shouldn't have much side effects.

Independently of this proposal, I think we should document `ament_auto` macros behavior in the developer documentation. By reading at so many cmake files, I can tell there is a gap between what people think these macros do, and what they actually do. For instance, a document like this one for ament_cmake_auto would be nice.

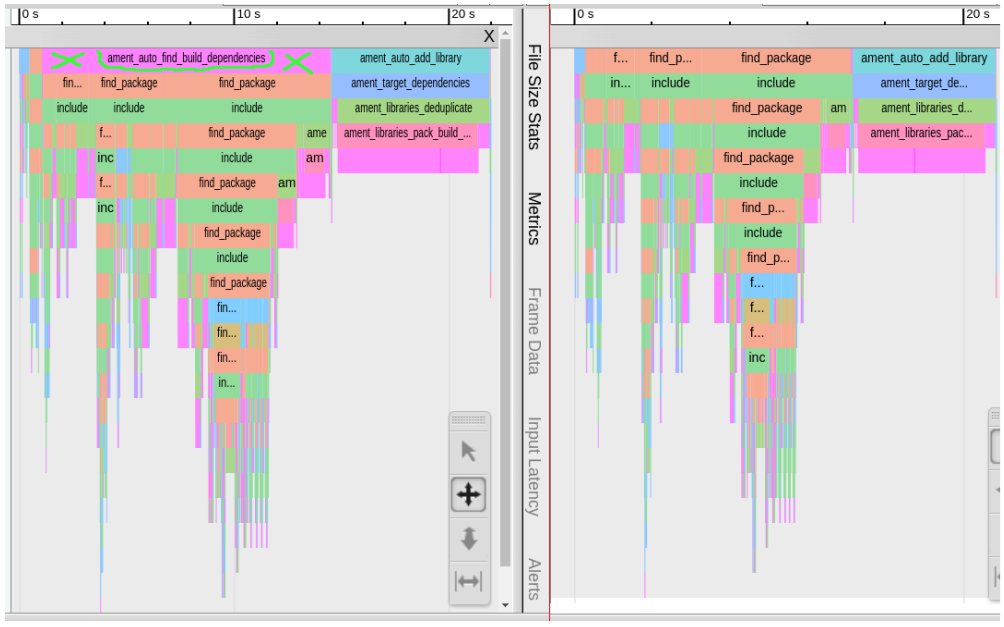
The behavior of `ament_cmake_extension` macros slightly differs from `ament_cmake_auto` ones for that reason. I tried to align what the devs would think the macros do, and what they actually do (of course it's 100% subjective).

> Is it possible to make it faster with less breaking changes?

Maybe? I have focused mostly on using modern CMake targets because that the simplest way to make it work, but also to make things cleaner (and it's 2023!). The `ament_auto_add_library` macros and such are slow by design, because the macro must go through every single library, include directory, compilation flag that are brought by every single dependency (recursively), and make sure to remove duplicates while preserving correct order. For example, `behavior_path_planner` must go through 17268 linked libraries, most of them being duplicates. Even if processing each item is like 8ms, in total that is 2 minutes of processing. Even if the performance is improved so that it only takes 2ms for each item, that is still 30 seconds on each `add_library` call. Targets don't have this issue, first because there is no recursion (upstream target brings everything already), and also because cmake is optimized for them (no spaghetti cmake scripting involved).

xmfcx on May 15, 2023 Maintainer

> I rewrite the packages CMakeLists.txt and call find_package on each dependency instead of using ament_auto_find_build_dependencies, the package does not build any faster. So there is basically no cost for using the ament_auto_find_build_dependencies macro (great!).



I've also tried to do the same and ended up with the same result. Do you think is it possible to speed up the `find_package()` CMake function? Maybe there is a better way of making use of it? I really don't know.

↑ 1 1 reply

VRichardJP on May 15, 2023 Collaborator Author edited

To make `find_package` faster is essentially to write faster `FindXXX.cmake`. These files are normally written by hand, but in ROS the `ament_package` macro does that for you (not an `ament_auto` macro). What we can do to make it faster is:

- to have less things to find (e.g. removing unused dependencies).
- to improve the generated code, in other words the core `ament_cmake` macros. For example the `ament_deduplicate` has been written like 8 years ago and never touched since then, but it is very slow when there are a lot of dependencies. These kind of things should rather be fixed upstream.

👍 2

VRichardJP on May 22, 2023 Collaborator Author

From all the discussion, I came to realize maybe the fork option is the simplest and the least scary solution (from autoware project PoV)

Now Iron is about to be released, I guess there is a chance some of the key features can be pushed to the `rolling` branch in the weeks/months to come. I don't know if everything could land there, but I might as well try there first. As I assume the new features would not be added to humble branch upstream, that would leave the following options while keeping the `ament_auto` macros around:

- Option 1: wait for J release
- Option 3a: fork `ament_cmake` and backport the features introduced in `rolling` to our `humble` branch
- Option 3b: fork `ament_cmake` and use `rolling` branch

Because the `ament_auto` is almost never updated, I don't think merging the feature from rolling to humble branch is a problem. There would still be work on Autoware side, but certainly it would not be as heavy as changing the name of all macros.

↑ 1 👍 1 2 replies

xmfcx on May 22, 2023 Maintainer

@VRichardJP
I still think maintaining a fork of the `ament_cmake` which only has the `ament_cmake_auto` is an OK option as long as it is documented.

We should have control over the packages built in the Autoware folder regardless of the build tools we use.

Also maybe we can edit the `ament_cmake_auto` functions to have an optional flag to facilitate the new behaviors you propose. This way old users of the functions won't be affected. And we can enable the behaviors we want for packages we want.

What do you think?

VRichardJP on May 22, 2023 Collaborator Author

Yes, for example fork `ament_cmake` and add COLCON_IGNORE files everywhere but `ament_cmake_auto`.

I am thinking how the feature would be implemented upstream. Most likely, there would be an optional flag added to the `ament_auto_package()` macro (e.g. `EXPORT_TARGETS`), which would trigger the new behavior is provided.

👍 1