2022 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Caching Tool Run Results in Large-Scale RTL Development Projects

Ashfaq Khan

Intel Corporation

accellera SYSTEMS INITIATIVE

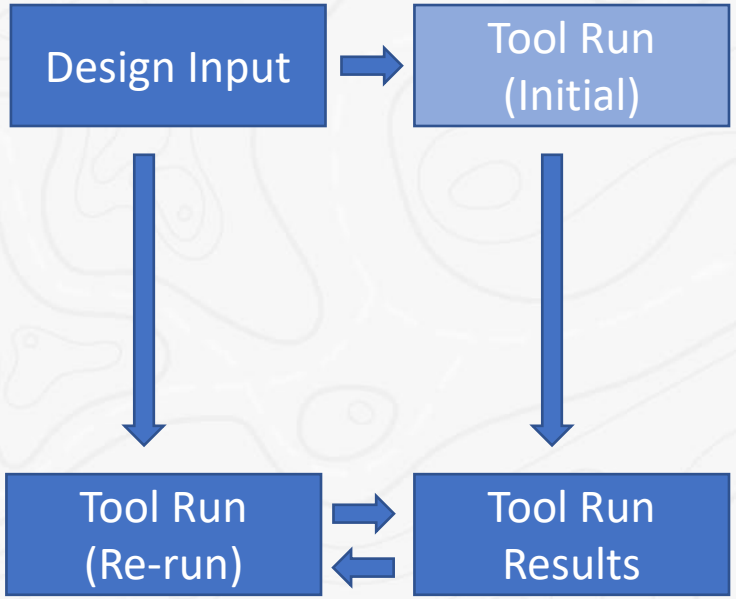
Outline

- Why cache (re-use) EDA tool run results - what does it even mean?
  - What is special about large-scale projects when it comes to caching?
- Challenges and considerations in caching tool run results
- Various caching methods and their pros & cons
- Implementation details - with more focus on the proposed method
- Results
- Conclusion - including asks from the EDA industry

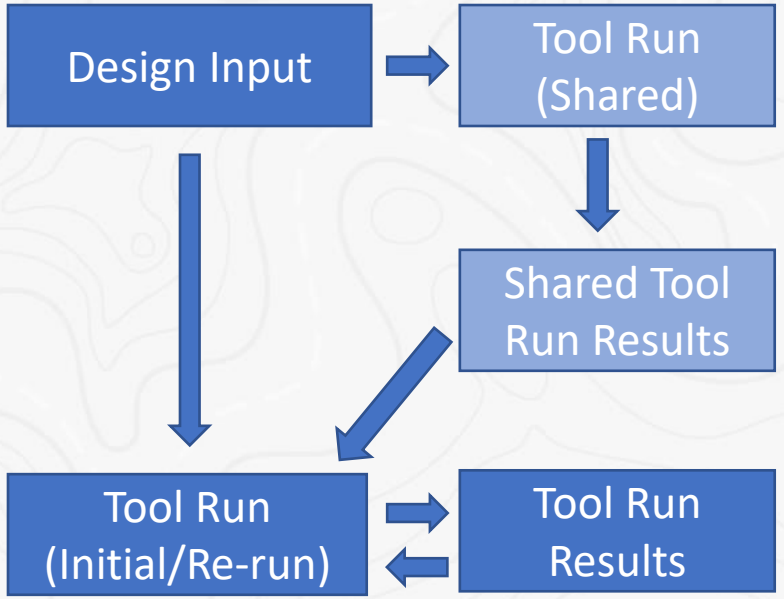
Caching EDA tool run results

- Observations:
  - RTL Designers' activities rarely involve changing ~100% of the design
  - Iterative EDA Tool runs with significant portion of the design unchanged
  - Major opportunity for resource optimization!
- Caching EDA tool run results
  - Re-use tool-generated content
  - Reduce or eliminate certain tool runtime
  - Opportunity to save human/engineering resources
  - Opportunity to save compute/machine resources
- Example:
  - Re-compiling a design for simulation while only changing one line of code!

Caching scenarios - individual vs. team



Caching/Re-using tool run results by a single user



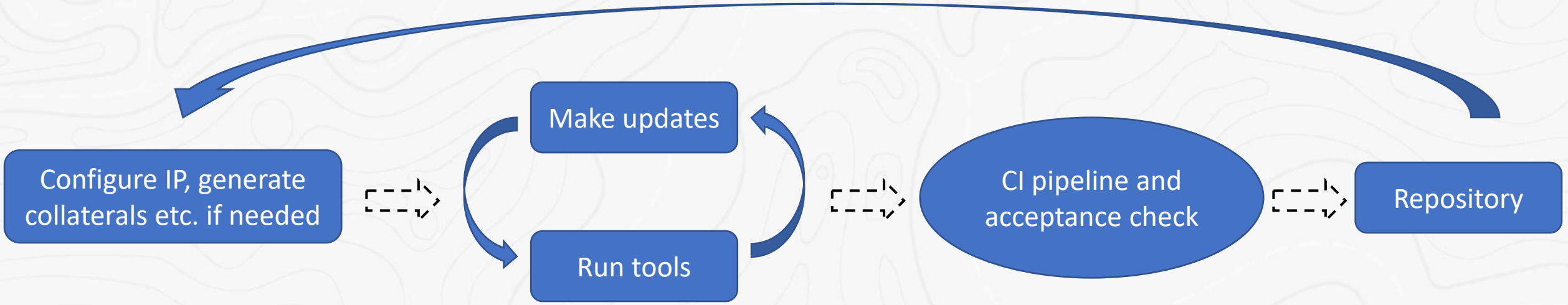
Caching/Re-using tool run results within a team

Challenges in caching tool run results

- Content Re-usability
- Save & Restore vs. Re-generate
- Cache Hit Rate
- Cache/Re-use Infrastructure
  - Correctness

Ensuring Correctness: Continuous Integration (CI)

- Using cached content could add risk of using incorrect/stale content
- Best to have some guaranteed point in design where no cache is used
- Having a CI infrastructure makes this simple
  - But this can be done through other Quality Assurance mechanisms as well



Various caching methods

- Method 1 - Storing cache as part of the repository
  - Method 1A: Every user is allowed to store the content.
  - Method 1B: Only one or more designated designers are allowed to store the content.
  - Method 1C: The CI infrastructure handles the storing.
- Method 2 - Storing cache outside of the repository
  - Method 2A: Mechanism to detect changes in input to the tool, cached content is only retrieved when there’s no change
  - Method 2B: User decides whether to retrieve the cache or not

Pros & Cons of various caching methods

Method	Key Feature	Pros	Cons
1A: Store in Repo, by all users	Every user submits content to store	1. Cached content always available as part of the repo	1. Doesn't work well for large/complex content. 2. Issue in CI while content merging
1B: Store in Repo, by designated users	Only designated users submit content to store	1. Cached content always available as part of the repo	1. Doesn't work well for large/complex content. 2. Restrictive work model
1C: Store in Repo, by CI infrastructure	CI infrastructure stores as part of the repository (upon successful checks)	1. Cached content always available as part of the repo	1. Doesn't work well for large/complex content. 2. Adds complexity in CI infrastructure

Pros & Cons of various caching methods

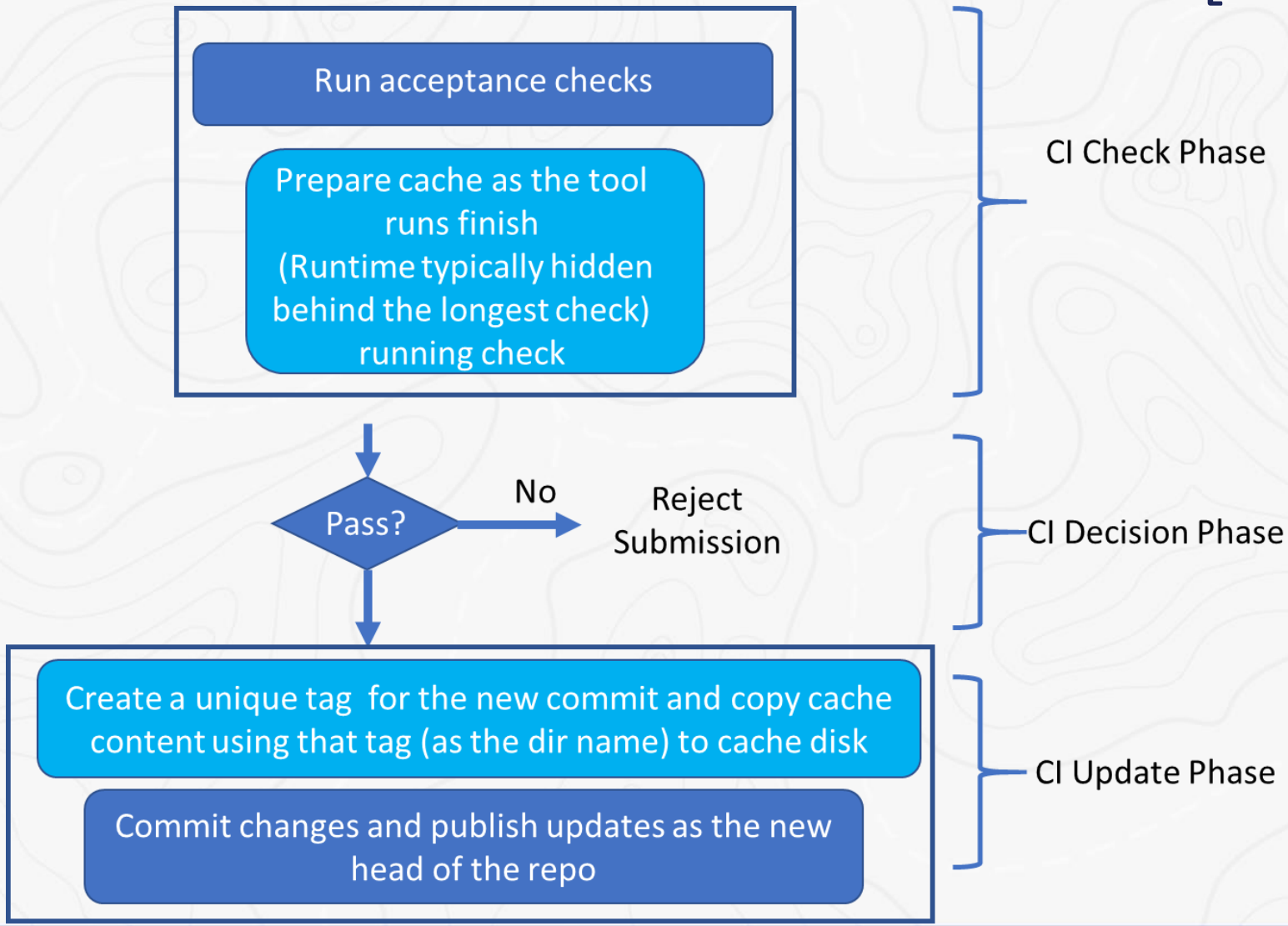
Method	Key Feature	Pros	Cons
2A: Store separately, Decision to retrieve is based on automatic detection of input change	CI infrastructure stores (upon successful checks) in a separate location than the repository	1. Cached content available for copying for as long as project decides to keep the data 2. Scales to any type of collateral and any size 3. No major change in work model	1. Content retrieval involves difficult setup and high maintenance cost due to the need for automated detection of input change
2B: Store separately, Decision to retrieve left to the user Proposed Method	CI infrastructure stores (upon successful checks) in a separate location than the repository	1. Cached content available for copying for as long as project decides to keep the data 2. Scales to any type of collateral and any size 3. No major change in work model 4. Easy setup	1. Doesn't track change in input collaterals.

Implementation details of Method 1 and 2A

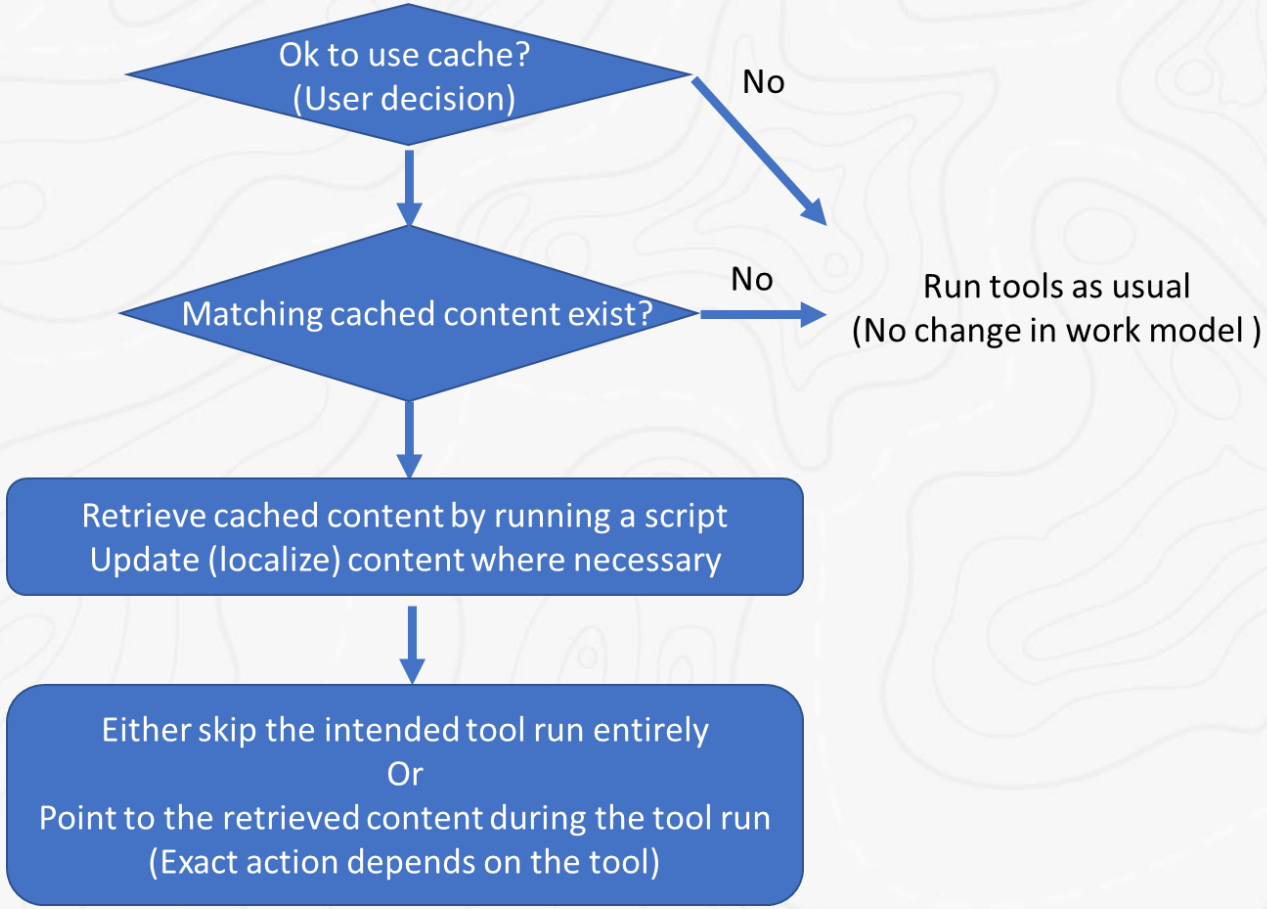
- Method 1A*, 1B*:
  - Users save (check-in) manually
- Method 1C*:
  - CI saves (delete content first then add, no merging of cached content)
  - Cached content are excluded from consideration when users look for changes in their local repo
- Method 2A:
  - Content retrieval is decided by checking if relevant inputs changed
    - Before caching, a checksum of the inputs is created and saved
    - Before retrieval, a checksum of the inputs is created and checked against cached content
    - Identifying inputs can be done by manually creating it, or automatically identifying all inputs (tracing) and then excluding the ones that are not relevant

* May also use/need retrieval check like Method 2A for repeated runs

Implementation details of Method 2B [Save]



Implementation details of Method 2B [Restore]

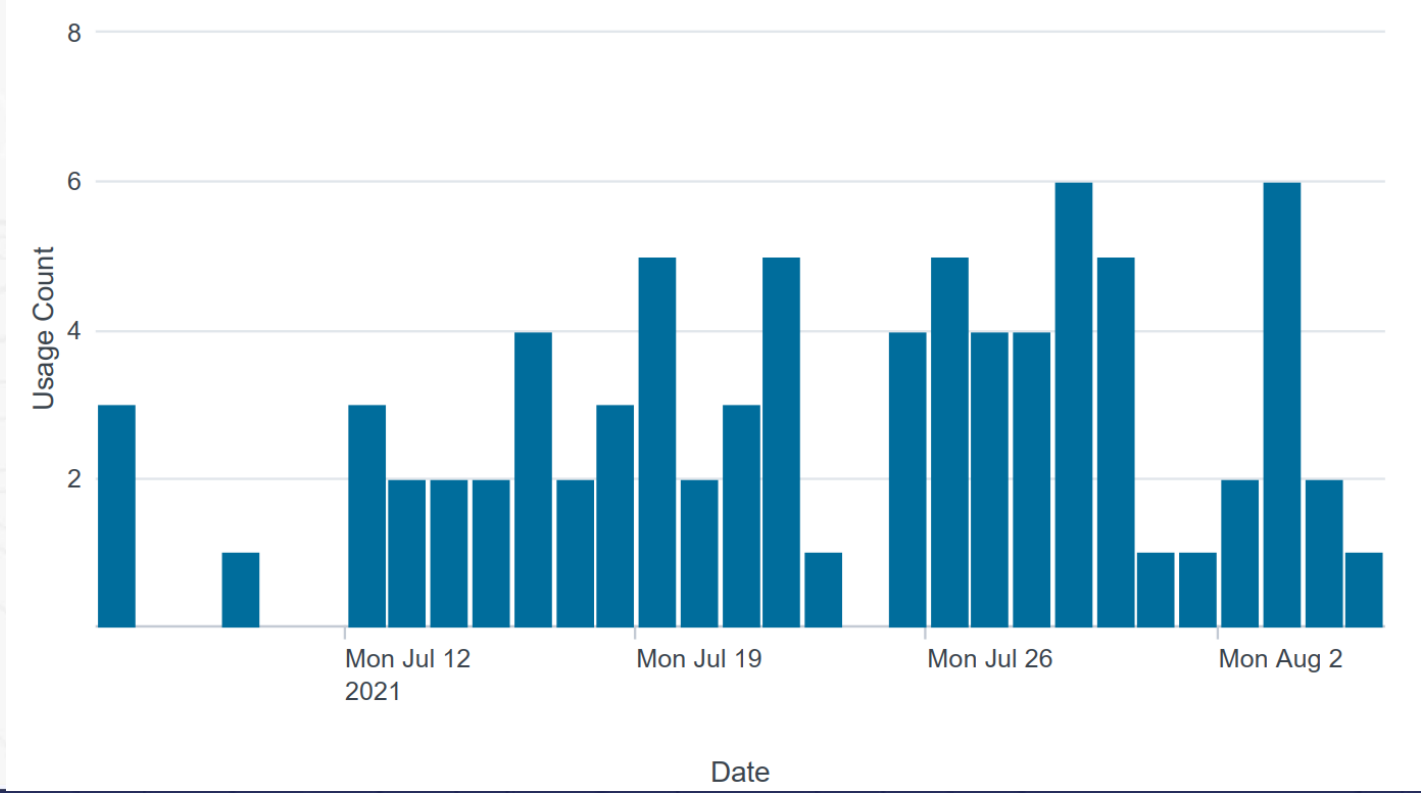


Pseudo code for Method 2B

```
Define Cache_sections
        Define section name
        Define what content to copy (paths of files or directories relative to the top of the repo or run area)
        Define any necessary pre-cache processing or post-retrieval processing

If User is CI then
        Target_sections = all (CI copies all cacheable content to the central disk)
        From_Dir = <User's run area>
        To_Dir = <Central Cache Disk>
        Tag = <Create a unique tag>
Else
        Target_sections = <User's command line input; Default is all>
        From_Dir = <Central Cache Disk>
        To_Dir = <User's run area>
        Tag = <Retrieve the latest unique tag> (Git example: git describe --abbrev=0 --tags --first-parent --match "<unique tag pattern>")

If User is NOT CI then
        Exit if cached content does not exist for Tag in the From_Dir

For each Section in Target_sections
        If User is CI then
                Perform any pre-cache processing on the content to be cached for this Section

        Copy content for that Section from From_Dir to To_Dir

        If User is NOT CI then
                Perform any post-retrieval processing on the content that was retrieved for this Section
```

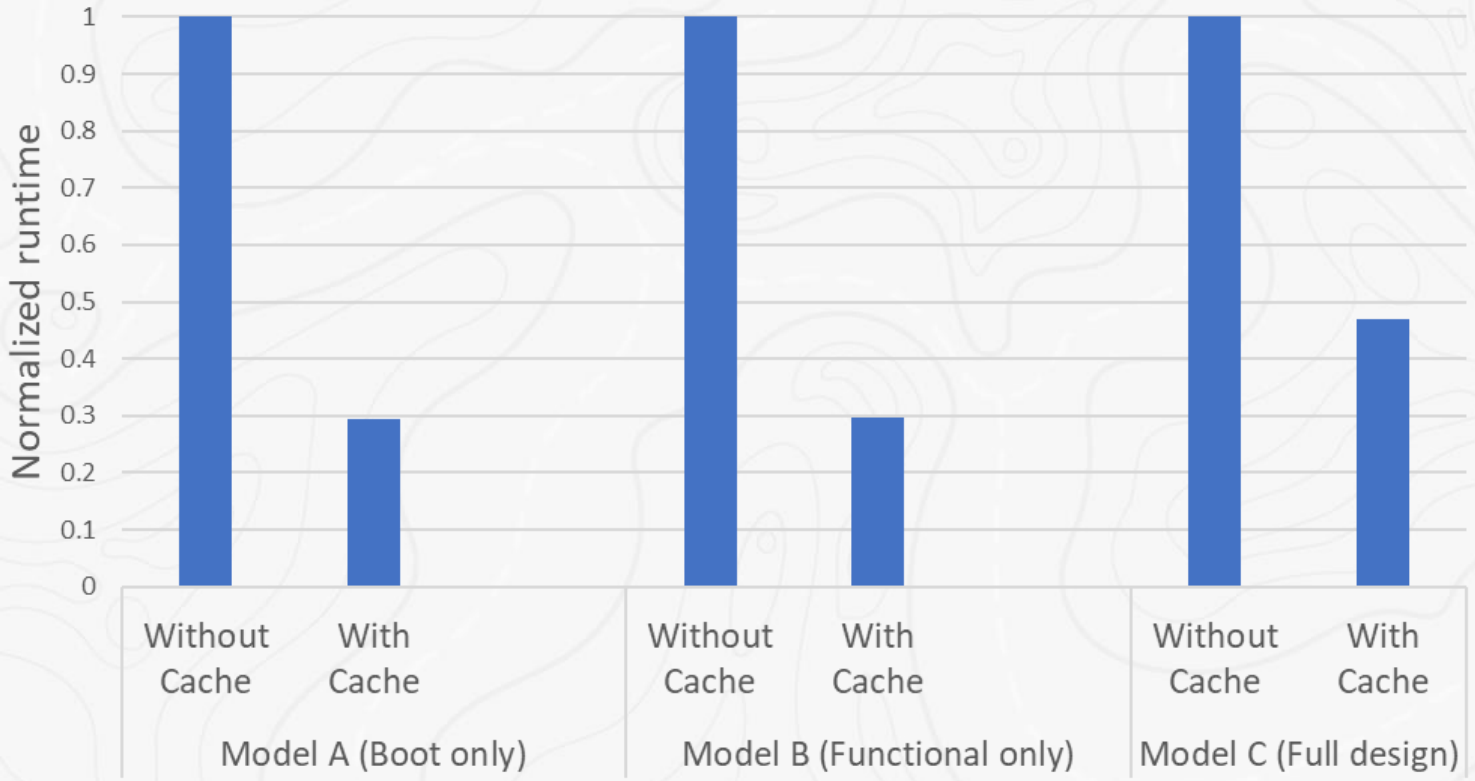
Results

- Usage of the caching method 2B in a production project in Intel
  - Cached Content: Various code generation needed before sim compilation
  - Target audience: Validation engineers (~20)



Results

- Reduction achieved in building simulation models
  - Generating the binary executable, does not include test runtime



Conclusion

- Caching or re-using EDA tool run results can significantly improve tool runtimes while reducing compute usage
- Net result is increase in designer productivity and faster project TAT
- What we need from the EDA industry:
  - Tool run results should be self-contained (portable, reusable etc.)
  - Tool should have a way to quickly evaluate how much of its input has changed
  - Tool should be able to re-use existing tool run results
    - Both directly from the current user work area and from a shared area

There's a lot of untapped potential here for the EDA industry and users!

Acknowledgements

- Thanks to Narasimhan Iyengar for his support on the deployment of the proposed caching method at Intel.