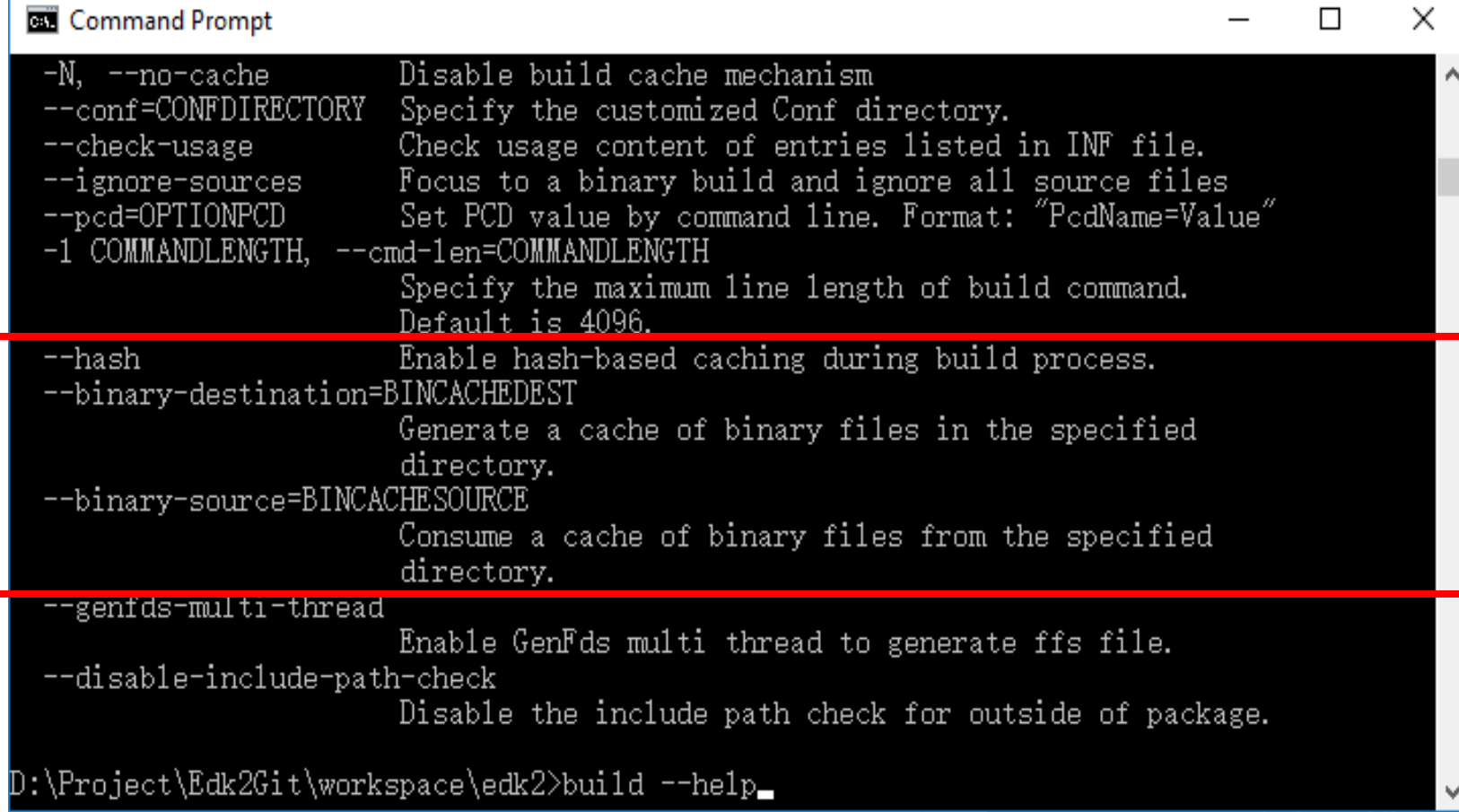


Edk2 Build Cache Enhancement

Steven Shi

Edk2 build cache option

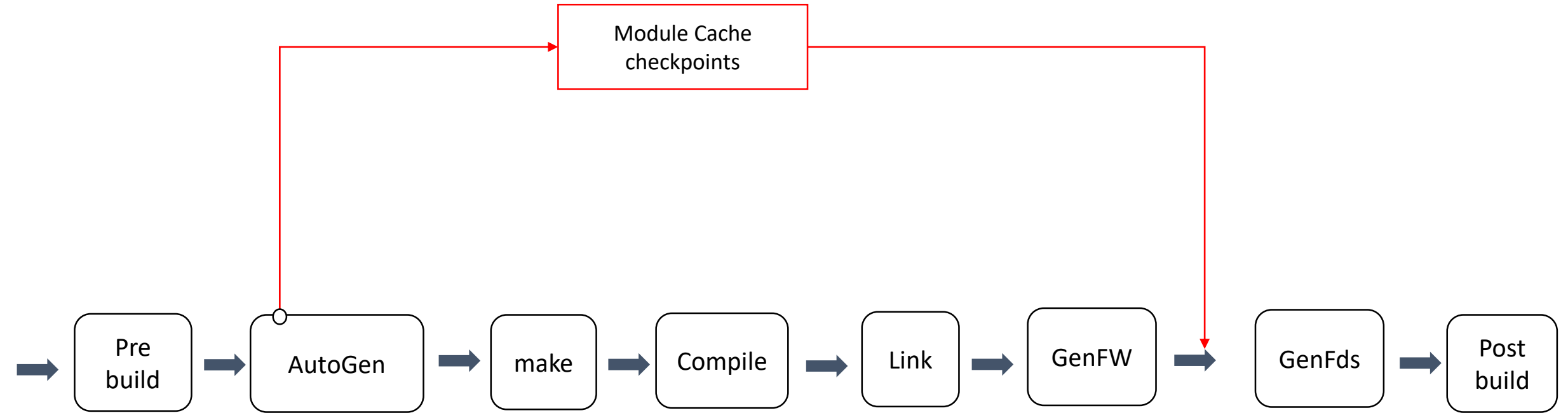
- build -help
- build -p OvmfPkg/OvmfPkgIa32X64.dsc -a IA32 -a X64 -t VS2015x86 -b NOOPT --hash --binary-destination=BinCache
- build -p OvmfPkg/OvmfPkgIa32X64.dsc -a IA32 -a X64 -t VS2015x86 -b NOOPT --hash --binary-source=BinCache



```
Command Prompt
-N, --no-cache          Disable build cache mechanism
--conf=CONFDIRECTORY    Specify the customized Conf directory.
--check-usage           Check usage content of entries listed in INF file.
--ignore-sources        Focus to a binary build and ignore all source files
--pcd=OPTIONPCD          Set PCD value by command line. Format: "PcdName=Value"
-1 COMMANDLENGTH, --cmd-len=COMMANDLENGTH
                        Specify the maximum line length of build command.
                        Default is 4096.
--hash                  Enable hash-based caching during build process.
--binary-destination=BINCACHEDEST
                        Generate a cache of binary files in the specified
                        directory.
--binary-source=BINCACHESOURCE
                        Consume a cache of binary files from the specified
                        directory.
--genfds-multi-thread   Enable GenFds multi thread to generate ffs file.
--disable-include-path-check
                        Disable the include path check for outside of package.

D:\Project\Edk2Git\workspace\edk2>build --help
```

Current build cache in edk2 build process



Current build cache problem: easy to cache miss

- Current build cache try to skip the Autogen parsing as much as possible
- Not accurately parse the module really used info, e.g. the include header files
- Use over approximate and redundant source code as dependency files for single module cache, which is monolithic and couple many other modules together
- Really fast if cache hit, but cache hit is difficult
- Be easy to cache miss and often becomes slow in practice

Current build cache algorithm problem: easy cache miss

Module Cache hash =

Platform hash +
Package hash +
Libs cache hash +
ModuleFiles hash

Libs cache hash =

Platform hash +
Lib Package hash +
Lib ModuleFiles hash

Current build cache algorithm

Platform hash =

workspace metafiles (Buildrule, Target, ToolChain) +
platform metafiles (all *.dsc, *dsc.inc, *fdf, *fdf.inc)

The Platform metafiles are monolithic:

- The platform meta files couple every module together and is easy to change in development practice. Any update in platform metafile will cause all platform modules cache miss.

Current build cache algorithm

Package hash=

Package .dec meta file +

All files in the dirs which are defined in .dec file [Includes] section

Package hash grows module unnecessary dependency files very fast:

- If a module defines to depend on a package, then all the packages [Includes] files will be added as the dependency to the module, no matter whether the module really include these header files
- If a lib module depend on a package, any other module using the lib will have to include all the package [Includes] files as dependency
- Overall, current package hash cause Ovmf module depends on 30~35% of all the platform build files. On an internal server platform, module even depends on ~70% of all the platform build files and every module has ~10000 dependency files

Current build cache algorithm

ModuleFiles hash =

- Module meta file (.inf) +

- All source files defined in .inf file [Sources] section +

- Any other files in the .inf folder and subfolder (skip hidden files. E.g. .gitxxx, **missing now and need to add**)

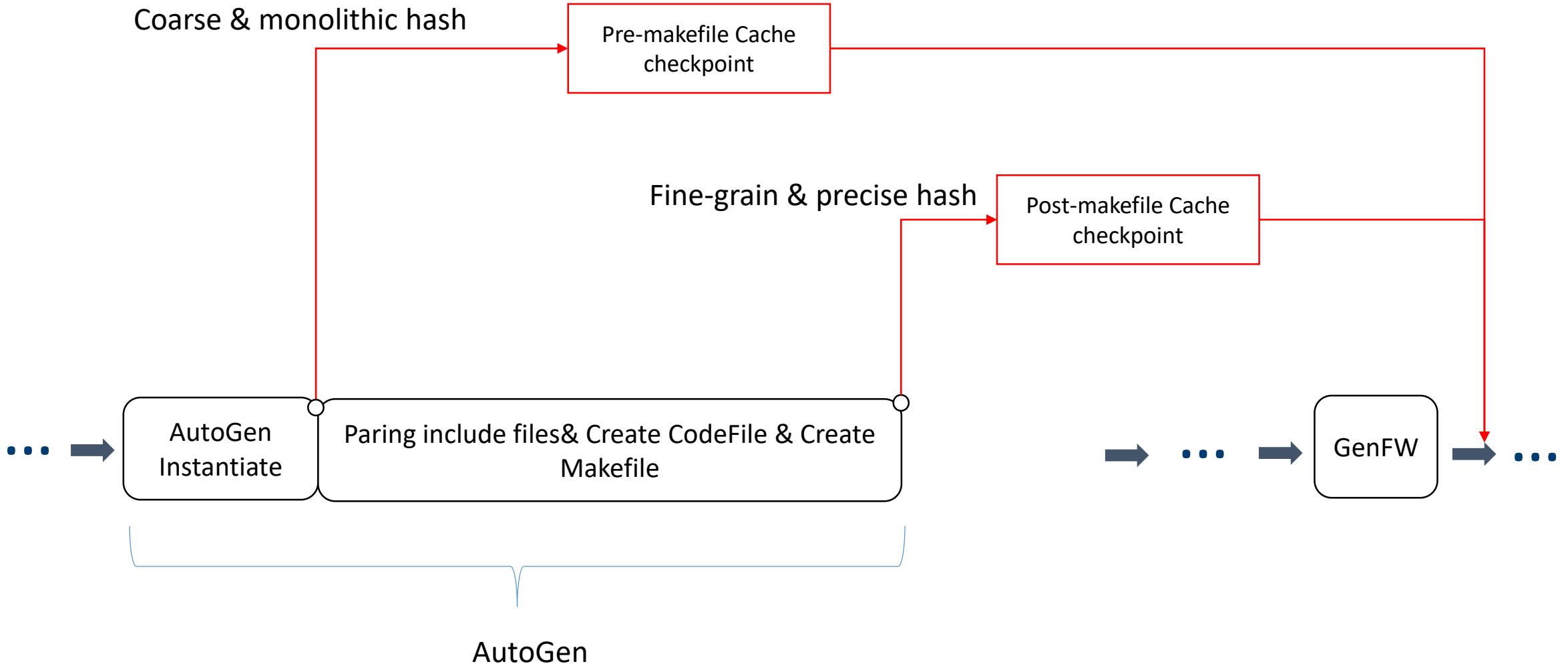
Add any other files in the INF folder because the developer might not to add all source/header files in the inf. (e.g. Openssl module)

Need to break down the platform and package hash

- Current build cache algorithm is too coarse for single module
- Need more precise info to break down the platform hash and package hash into module level
- Can leverage the module makefile to achieve module accurate info
- But we still need the coarse hash to quickly bypass no source change case

Two checkpoints for module level build cache:

Pre-makefile and Post-makefile



One more build cache checkpoint in post-makefile

Module Post-makefile hash =

Makefile hash +
Include header files hash +
AutoGen files hash +
Libs post-makefile hash +
ModuleFiles hash

} From driver module makefile generation result

Libs Post-makefile hash =

Lib Makefile hash +
Lib Include header files hash +
Lib AutoGen files hash +
Lib ModuleFiles hash

} From Lib module makefile generation result

Post-makefile build cache new hash

Makefile hash = The content of module makefile.

The Makefile content covers all workspace info (Buildrule, Target, ToolChain)

Include header files hash = The content of files in DependencyHeaderFileSet from makefile.

Autogen will searching recursively "#include" directive in file, find out all the files needed by module source file

AutoGen files hash = The content of files in AutoGenFileList from makefile.

E.g. AutoGen.c, AutoGen.h, StringH, StringIdf

Build Cache improve example via post-makefile cache in edk2

Generate build cache firstly, then clean build with and without code change.

code change: edk2 1ec05b81e59f7ed89d2e3be7214262c9626ec2a7 patch, which change the OVMF dsc files:

OvmfPkg/OvmfPkgIa32.dsc

OvmfPkg/OvmfPkgIa32X64.dsc

OvmfPkg/OvmfPkgX64.dsc

Clean Build Platform VS2015x86 toolchain	No build Cache Without code change	Only Pre-makefile Cache checkpoint Without code change	Only Pre-makefile Cache checkpoint With change	Both Pre-makefile and post-makefile Cache checkpoints With change
<i>Ovmf Ia32</i> (NOOPT)	1m:59s	27s	2m:08s	39s
<i>Ovmf X64</i> (NOOPT)	2m:12s	23s	2m:11s	35s
<i>Ovmf Ia32X64</i> (NOOPT)	2m:16s	26s	2m:20s	40s
<i>Ovmf Ia32</i> (DEBUG)	2m:43s	23s	2m:40s	38s
<i>Ovmf X64</i> (DEBUG)	2m:48s	22s	2m:48s	37s
<i>Ovmf Ia32X64</i> (DEBUG)	3m:03s	26s	3m:00s	43s

Question?

Multiple-level build caches in edk2 base tool process

