Working with NVIDIA Tegra BSP and Supporting Latest CUDA Versions

Leon Anavi
Konsulko Group
leon.anavi@konsulko.com
leon@anavi.org
Yocto Project Summit 2019
Konsulko Group

- Services company specializing in Embedded Linux and Open Source Software
- Hardware/software build, design, development, and training services
- Based in San Jose, CA with an engineering presence worldwide
- http://konsulko.com/
Agenda

- Introduction to CUDA, Jetson and Tegra
- Challenges in supporting CUDA for the Yocto Project and OpenEmbedded
- Solutions for building custom Linux distributions with latest CUDA version with Yocto Project and OpenEmbedded
- Flashing, booting and testing
- Conclusions
CUDA, Jetson & Tegra
CUDA

- **Compute Unified Device Architecture**
- Parallel computing platform and programming model developed by NVIDIA that allows seamlessly to use GPU for general purpose computing
- Appropriate for 3D graphics as well as a wide range of machine learning and artificial intelligence (AI) applications
- Initially released in 2007
NVIDIA Jetson

- Leading AI computing platform for GPU-accelerated parallel processing with CUDA on mobile and embedded devices
- Bringing the power of modern AI to embedded systems with ARM CPUs for robotics and autonomous machines
- Jetson models feature the Tegra ARM SoC
## NVIDIA Jetson

<table>
<thead>
<tr>
<th>JETSON</th>
<th>CUDA</th>
<th>AVAILABLE THROUGH</th>
</tr>
</thead>
<tbody>
<tr>
<td>TK1</td>
<td>6.5</td>
<td>Jan 2024 (for the CPU) / 2025 for Toradex Apalis TK1</td>
</tr>
<tr>
<td>TX1</td>
<td>10</td>
<td>Jan 2021</td>
</tr>
<tr>
<td>TX2</td>
<td>10</td>
<td>Apr 2022</td>
</tr>
<tr>
<td>TX2i</td>
<td>10</td>
<td>April 2028</td>
</tr>
<tr>
<td>Xavier</td>
<td>10</td>
<td>Jan 2025</td>
</tr>
<tr>
<td>Nano</td>
<td>10</td>
<td>Jan 2025</td>
</tr>
</tbody>
</table>
NVIDIA L4T (Linux for Tegra)

- Part of NVIDIA JetPack SDK
- Uses NVIDIA SDK Manager with graphical user interface
- Includes a reference filesystem derived from Ubuntu 18.04
- For more details:
  https://developer.nvidia.com/embedded/jetpack
## CUDA & GCC/G++

<table>
<thead>
<tr>
<th>CUDA VERSION</th>
<th>REQUIRED GCC/G++ VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>6.5</td>
<td>4.8</td>
</tr>
</tbody>
</table>
Building a custom GNU/Linux distribution with Yocto Project & OpenEmbedded for Tegra
Benefits

Benefits for using the Yocto Project and OpenEmbedded instead of L4T reference filesystem derived from Ubuntu:

- Custom distribution based on Poky, the reference distribution of the Yocto Project, for the exact needs of particular embedded device
- Flexibility to select init system, display server protocol, desktop environment, etc.
- Option to integrate software over the air update mechanism
BSP Layers

- meta-tegra
  https://github.com/madisongh/meta-tegra

- meta-jetson-tk1
  https://github.com/cubicool/meta-jetson-tk1

- meta-toradex-tegra
  http://git.toradex.com/cgit/meta-toradex-tegra.git
<table>
<thead>
<tr>
<th>Yocto Project Release Codename</th>
<th>Yocto Project Release Version</th>
<th>Yocto Project Release Date</th>
<th>Minimal Supported GCC Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warrior</td>
<td>2.7</td>
<td>April 2019</td>
<td>8.3</td>
</tr>
<tr>
<td>Thud</td>
<td>2.6</td>
<td>November 2018</td>
<td>7.3</td>
</tr>
<tr>
<td>Sumo</td>
<td>2.5</td>
<td>April 2018</td>
<td>7.3</td>
</tr>
<tr>
<td>Rocko</td>
<td>2.4</td>
<td>October 2017</td>
<td>6.4</td>
</tr>
<tr>
<td>Pyro</td>
<td>2.3</td>
<td>May 2017</td>
<td>5.4</td>
</tr>
<tr>
<td>Morty</td>
<td>2.2</td>
<td>November 2016</td>
<td>5.4</td>
</tr>
<tr>
<td>Krogoth</td>
<td>2.1</td>
<td>April 2016</td>
<td>4.9</td>
</tr>
<tr>
<td>Jethro</td>
<td>2</td>
<td>November 2015</td>
<td>4.8</td>
</tr>
</tbody>
</table>
Meta-tegra

- Yocto/OE BSP for Tegra devices
- Hosted in GitHub under MIT License
- Started by Matt Madison
- More than 700 commits by 16 contributors
- https://github.com/madisongh/meta-tegra
Meta-tegra Supported Devices

- Jetson-TX1 development kit (Linux4Tegra R32.2.1, JetPack 4.2.2)
- Jetson-TX2 development kit (Linux4Tegra R32.2.1, JetPack 4.2.2)
- Jetson AGX Xavier development kit (Linux4Tegra R32.2, JetPack 4.2.2)
- Jetson Nano development kit (Linux4Tegra R32.2.1, JetPack 4.2.2)
- Jetson-TX2i module (Linux4Tegra R32.2.1, JetPack 4.2.2)
- Jetson-TX2 4GB module (Linux4Tegra R32.2.1, JetPack 4.2.2)
Meta-tegra Releases

- Jethro
- Krogoth
- Morty
- Rocko
- Sumo
- Thud
- Warrior
- Master (to become Zeus)
Before You Start

- Install NVIDIA SDK Manager
- Login and download JetPack packages for the targeted machine
Getting Started (1/2)

- Recommended host distribution is Ubuntu
- Since JetPack 4.2 manual download through NVIDIA SDK Manager with NVIDIA Developer Network login is required
- Set the path to the downloaded packages in variable `NVIDIA_DEVNET_MIRROR`, for example:

  ```
  NVIDIA_DEVNET_MIRROR = "file:///home/leon/Downloads/nvidia/sdkm_downloads"
  ```

- SDK Manager downloads a different CUDA package depending on the Ubuntu version, by default for 18.04. If you are using 16.04 set:

  ```
  CUDA_BINARIES_NATIVE = "cuda-binaries-ubuntu1604-native"
  ```
Set machine, for example:

```
MACHINE = "jetson-tx2"
```

Whitelist commercial license flags:

```
LICENSE_FLAGS_WHITELIST = "commercial"
```

Generate script for flashing bootloader, rootfs, and other necessary artifacts to the on-board eMMC of Jetson Developer kit:

```
IMAGE_CLASSES += "image_types_tegra"
IMAGE_FSTYPES = "tegraflash"
```
Putting the Pieces Together

<table>
<thead>
<tr>
<th>CUDA VERSION</th>
<th>REQUIRED GCC/G++ VERSION</th>
<th>LATEST YOCTO/OE RELEASE WITH THIS GCC/G++ VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>7</td>
<td>Thud (2.6)</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>Rocko (2.4)</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>Pyro (2.3)</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>Pyro (2.3)</td>
</tr>
<tr>
<td>6.5</td>
<td>4.8</td>
<td>Jethro (2)</td>
</tr>
</tbody>
</table>
The big problem: How to use the latest CUDA on the latest release of the Yocto Project?
Solutions

- Use the newest release of the Yocto Project with appropriate GCC version, for example CUDA 10 on Thud with GCC 7
- Use an external toolchain with the appropriate GCC version
- Use backported GCC version on the latest release of the Yocto Project, for example GCC 7 from `meta-tegra/contrib`
- Add required by CUDA GCC version as a secondary toolchain and use it only for building CUDA recipes
Solution 1: Older Compatible Release

- Identify a release of the Yocto Project which contains recipes for a compatible GCC version supported by the targeted CUDA version, for examples CUDA 10 requires GCC 7 which is available in release Thud (2.6)

<table>
<thead>
<tr>
<th>CUDA VERSION</th>
<th>REQUIRED GCC/G++ VERSION</th>
<th>LATEST YOCTO/OE RELEASE WITH THIS GCC/G++ VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>7</td>
<td>Thud (2.6)</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>Rocko (2.4)</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>Pyro (2.3)</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>Pyro (2.3)</td>
</tr>
<tr>
<td>6.5</td>
<td>4.8</td>
<td>Jethro (2)</td>
</tr>
</tbody>
</table>

Solution 2: External Toolchain (1/2)

- Download a compatible external toolchain, for example from Linaro or ARM
- Use Yocto/OE layer `meta-linaro/meta-linaro-toolchain`  
  https://git.linaro.org/openembedded/meta-linaro.git/about/
- Set external toolchain in `conf/local.conf`:
  
  ```
  TCMODE = "external-linaro"
  EXTERNAL_TOOLCHAIN = "/opt/toolchains/gcc-x86_64_aarch64-linux-gnu"
  ```
- Disable unsupported by GCC 7 flags

  ```
  DEBUG_PREFIX_MAP = "-fdebug-prefix-map=${WORKDIR}=/usr/src/debug/${PN}/${EXTENDEDPE}${PV}-${PR} \\
  -fdebug-prefix-map=${STAGING_DIR_HOST}=" \\
  -fdebug-prefix-map=${STAGING_DIR_NATIVE}=" \\
  "
  ```
Solution 2: External Toolchain (2/2)

- Replace `--enable-default-pie` with `--disable-pie`, set by security_flags.inc in Poky, to successfully configure and compile `native-glibc` if you plan to build an SDK

  GCCPIE ?= "--disable-pie"

- Create `.bbappend` files that remove unsupported flags from CPPFLAGS, for example `missing-attributes`, for all recipes failing to build with an older compiler due to similar issues.

  cc1: error: -Werror=missing-attributes: no option -Wmissing-attributes

- For more details:

  https://github.com/madisongh/meta-tegra/wiki/Using-linaro-gcc7-for-CUDA-support
Solution 3: Backported GCC 7

- Yocto/OE layer meta-tegra branch master contains a sublayer `contrib` with backported recipes for GCC/G++ 7
- Add layers to `conf/bblayers.conf`
- Specify GCC and CUDA versions in `conf/local.conf`

```conf
GCCVERSION = "7.%"
require contrib/conf/include/gcc-compat.conf

CUDA_VERSION="10.0"
CUDA_BINARIES_NATIVE = "cuda-binaries-ubuntu1604-native"
```
One more thing...

- Add sample applications to be able to quickly verify that CUDA is properly running on the device after booting

- For example, add to `conf/local.conf`:

```plaintext
IMAGE_INSTALL_append = " cuda-samples"
```
Testing
Flashing an Image

- Enter recovery mode on Jetson developer kit by holding the recovery button (REC) and press reset (RST) or by interrupting u-boot and executing:

```bash
enterrcm
```

- Connect Jetson dev kit to the build machine with USB cable

- Extract the archive created by bitbake and flash it:

```bash
tegadir=$(mktemp -u /tmp/tx2.XXXXXXXX)
unzip tmp/deploy/images/jetson-tx2/core-image-minimal-jetson-tx2.tegraflash.zip -d $tegradir
cd $tegradir
sudo ./doflash.sh
```
USB to UART cable can be attached to pins 8 (TX), 9 (GND) and 10 (RX) of J21 on Jetson TX2 developer kit.
Testing CUDA

- Run `/usr/bin/cuda-samples/deviceQuery`
- If everything works as expected it will print `Result = PASS`
- After that run the other sample application `/usr/bin/cuda-samples/UnifiedMemoryStreams`
Conclusions
Conclusions

- Yocto/OE layer meta-tegra provides the required BSP for building custom embedded distributions for modern Tegra devices
- Latest CUDA version requires older GCC/G++ version which are not available in the latest Yocto Project release
- There are different ways to overcome the shortcomings with CUDA requirements by using backported GCC/G++, an external toolchain or an older Yocto release
Thank You!

Useful links:

- https://github.com/madisongh/meta-tegra
- https://github.com/madisongh/meta-tegra/blob/master/README
- https://github.com/madisongh/meta-tegra/wiki/Using-linaro-gcc7-for-CUDA-support