## DocumentRevision History

<table>
<thead>
<tr>
<th>Date Published</th>
<th>Revision</th>
<th>Revision Change Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 2017</td>
<td>0.5</td>
<td>alpha Release</td>
</tr>
<tr>
<td>February 2017</td>
<td>0.6</td>
<td>Add hardware information</td>
</tr>
<tr>
<td>April 2017</td>
<td>0.8</td>
<td>Add Beta information</td>
</tr>
<tr>
<td>May 2017</td>
<td>0.9</td>
<td>Align with Silver release</td>
</tr>
<tr>
<td>June 2017</td>
<td>1.0</td>
<td>Align with Product release</td>
</tr>
</tbody>
</table>
Disclaimer

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at Intel.com, or from the OEM or retailer.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel, the Intel logo, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2016-2017 Intel Corporation. All Rights Reserved.
# Table of Contents

## Abbreviations

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

## 1. Intel Deep Learning Inference Accelerator Overview

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Set</td>
<td>9</td>
</tr>
</tbody>
</table>

## 2. Board Specification

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® DLIA add-in card Specifications</td>
<td>11</td>
</tr>
<tr>
<td>Components and Interfaces</td>
<td>11</td>
</tr>
<tr>
<td>Mechanical Specification</td>
<td>12</td>
</tr>
<tr>
<td>System Compatibility List</td>
<td>13</td>
</tr>
</tbody>
</table>

## 3. Supported Operating Systems

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CentOS 7.0</td>
<td>14</td>
</tr>
<tr>
<td>CentOS 7.2</td>
<td>14</td>
</tr>
<tr>
<td>CentOS 7.3</td>
<td>14</td>
</tr>
</tbody>
</table>

## 4. System Setup

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install Intel® DLIA card</td>
<td>15</td>
</tr>
<tr>
<td>Configure host BIOS</td>
<td>15</td>
</tr>
<tr>
<td>Install OS</td>
<td>15</td>
</tr>
<tr>
<td>Network setting</td>
<td>15</td>
</tr>
<tr>
<td>Install Intel® DLIA software using installer</td>
<td>15</td>
</tr>
</tbody>
</table>

## 5. Software Stack

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
</tr>
</tbody>
</table>

## 6. Software Installation

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Software and Install Environment</td>
<td>17</td>
</tr>
<tr>
<td>Installation</td>
<td>17</td>
</tr>
<tr>
<td>Un-install Intel® DLIA</td>
<td>19</td>
</tr>
</tbody>
</table>

## 7. Upgrade FPGA Image Using dlia_flash_and_program

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dlia_flash_and_program</td>
<td>20</td>
</tr>
</tbody>
</table>

## 8. Updating FPGA Image using dlia_ipselector

<table>
<thead>
<tr>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
</tr>
</tbody>
</table>

## 9. Building Caffe and MKL_DNN from Source Code

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build Caffe</td>
<td>22</td>
</tr>
</tbody>
</table>
Abbreviations

DLIA: Intel® Deep Learning Inference Accelerator
CNN: Convolutional Neural Networks
MKL-DNN: Intel® Math Kernel Library for Deep Neural Networks
FPGA: Field Programmable Gate Arrays
RPD: Raw Programming Data
Caffe: Intel Distribution of the software Caffe, originally developed by Berkeley Vision and Learning Center (BVLC)
1. Intel Deep Learning Inference Accelerator Overview

Intel® Deep Learning Inference Accelerator (Intel® DLIA) is a turnkey inference solution that accelerates convolutional neural network (CNN) workloads for image recognition. Intel DLIA comes pre-programmed with image recognition models that can be used right out of the box for multiple network topologies.

Intel Deep Learning Inference Accelerator (Intel® DLIA) mainly consists of two components:

- **Hardware:** Gen3 x 16 host interface add-in card, based on Intel® Arria 10 FPGA
  - PCIe® x8 electrical
  - PCIe® x16 power & mechanical
- **Software:** Integrated deep learning stack with industry-standard libraries and frameworks
  - Deep Learning Accelerator (DLA) IP
  - Integration with Caffe and Intel® MKL-DNN
  - Sample applications

Intel DLIA is built for data center applications for real-time or offline video / image recognition and classification. This add-in card is designed to be plugged in host systems through PCIe®, and used to accelerate computationally intensive CNN primitives in optimized FPGA hardware. The solution works seamlessly with Intel® Xeon® family processors.

**Outstanding Performance / Power / Price per inference**

- Energy efficient inference
- Scalable throughput gains several times better than CPU alone
- Lower TCO for high throughput systems

**Fit within Intel® Xeon® Processor Infrastructure**

- Multiple 1U and 2U Server System options
- PCIe® Gen3 x8 enables fast communication between host and adapter

**Flexible and Portable software architecture**

- Integrated deep learning stack with Caffe and Intel® Math Kernel Library for Deep Neural Networks (MKL-DNN) APIs
- Accelerate six CNN primitives in the FPGA
- Support CPU fallback of primitives not implemented by FPGA (hybrid FPGA/CPU)
- Unified user experience and code portability across Intel product families
1.1. Feature Set

- Board Feature Set

<table>
<thead>
<tr>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Form Factor</strong></td>
</tr>
<tr>
<td>• Full-length, full-height, single width PCIe* add-in card</td>
</tr>
<tr>
<td>• 277mm x 98.4mm</td>
</tr>
<tr>
<td>• Max. Component height: 14.05mm</td>
</tr>
<tr>
<td><strong>FPGA</strong></td>
</tr>
<tr>
<td>Intel® Arria 10 FPGA</td>
</tr>
<tr>
<td><strong>TDP</strong></td>
</tr>
<tr>
<td>50–60 Watts</td>
</tr>
<tr>
<td><strong>TFLOPS</strong></td>
</tr>
<tr>
<td>Up to 1.5</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
</tr>
<tr>
<td>• Two banks DDR4 with ECC, 4 GBytes (x72) each</td>
</tr>
<tr>
<td>• Up to 2133MPTS</td>
</tr>
<tr>
<td>• 1024 MBytes of Flash memory for booting FPGA (512 MB for user and 512 MB for backup)</td>
</tr>
<tr>
<td><strong>PCI Express Configuration</strong></td>
</tr>
<tr>
<td>PCIe* 3.0 x8 electrical and x16 mechanical host interface</td>
</tr>
<tr>
<td><strong>Cooling</strong></td>
</tr>
<tr>
<td>Actively cooled</td>
</tr>
</tbody>
</table>

- Intel® Arria 10 FPGA Feature Set

<table>
<thead>
<tr>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SerDes Transceivers</strong></td>
</tr>
<tr>
<td>Up to 17 Gbps</td>
</tr>
<tr>
<td><strong>Logic Elements Available</strong></td>
</tr>
<tr>
<td>Up to 1150K</td>
</tr>
<tr>
<td><strong>Embedded Memory</strong></td>
</tr>
<tr>
<td>Up to 53 Mb</td>
</tr>
<tr>
<td><strong>LVDS Performance</strong></td>
</tr>
<tr>
<td>Up to 1.6 Gbps</td>
</tr>
<tr>
<td><strong>18x19 Variable-Precision Multipliers</strong></td>
</tr>
<tr>
<td>Up to 3,300</td>
</tr>
</tbody>
</table>

- Software Feature Set

<table>
<thead>
<tr>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network topologies</strong></td>
</tr>
<tr>
<td>AlexNet, GoogLeNet, CaffeNet, LeNet, VGG-16, SqueezeNet, custom-developed</td>
</tr>
<tr>
<td><strong>Framework</strong></td>
</tr>
<tr>
<td>Caffe</td>
</tr>
<tr>
<td><strong>Pre-programmed IP</strong></td>
</tr>
<tr>
<td>• Intel® Deep Learning Accelerator IP (DLA IP)</td>
</tr>
<tr>
<td>• Accelerates CNN primitives in FPGA: convolution, fully connected, ReLU, normalization, pooling, concat.</td>
</tr>
<tr>
<td>• Networks beyond these primitives are computed with hybrid CPU+FPGA</td>
</tr>
<tr>
<td><strong>Libraries</strong></td>
</tr>
<tr>
<td>Intel® Math Kernel Library for Deep Neural Networks (MKL-DNN)</td>
</tr>
</tbody>
</table>

The software includes image recognition models which can be used out of the box. Custom CNN algorithm can be implemented using Intel Caffe framework, or MKL-DNN libraries, both of which are prepackaged and pre-integrated.

Intel DLI/A comes pre-programmed with the DLA IP on the FPGA which accelerates six CNN primitives:
- Convolution
- Rectified Linear Unit (ReLU)
• Local Response Normalization (LRN)
• Pooling (max and average)
• Fully-connected
• Concatenation

These primitives are enabled through Caffe and Intel® MKL-DNN which provide unified deep learning APIs. The whole software stack is optimized for performance. Reconfiguration to the primitives are made at the upper layers of the stack, through MKL-DNN and Caffe, abstracting the low level FPGA programming complexity.

Network topologies that use primitives beyond these six are configured on the host and computed with hybrid CPU + FPGA.

Validated network topologies include AlexNet, GoogleNet, SqueezeNet, VGG-16, CaffeNet and LeNet.
2. Board Specification

Intel Deep Learning Inference Accelerator (Intel DLIA) is a hardware, software, and IP solution used for deep learning inference applications.

2.1. Intel® DLIA add-in card Specifications

The Intel Deep Learning Inference Accelerator (Intel DLIA) board consists of an Intel® Arria 10 FPGA. The board communicates with the host CPU, through PCIe bus. The FPGA chip is preprogrammed to accelerate six CNN primitives with optimal performance. In this scenario, several end devices will be connecting to the server, and sending their data (images) to be processed in this server for the classification/scoring/inference problems.

2.2. Components and Interfaces

Size

Full length, full height, single width PCIe* card
- PCIe Gen3 x16 host interface
- PCIe x8 electrical
- PCIe x16 power & mechanical

FPGA Chip

- Intel® Arria 10 FPGA @ 275MHz

Performance

- Up to 1.5 TFLOPS

Maximum TDP

- 60W TDP

Memory

- 2 Banks 4G DDR4 SDRAM @ 2133 MPTS

Cooling

- Actively cooled

This section lists the components and interface on the board and locations. Figure 1. shows key components locations on the board.
2.3. Mechanical Specification

Intel® DLIA is a full length, full height, single width PCIe® 3.0 x16 add-in card.
2.4. **System Compatibility List**

The Intel Deep Learning Inference Accelerator card is compatible with the following server systems. This section will be updated as additional server systems are validated by Intel or by the specified server supplier.

- Intel
  - Intel® Server System R2000WT Family

---

Note: Please contact third party system manufacturers directly to verify ordering options required to support Intel Deep Learning Inference Accelerator; the system names here may need OEM modifications to fully support this device.
3. Supported Operating Systems

Intel® DLIA primarily supports CentOS (http://vault.centos.org). The following are the versions of CentOS that are verified:

3.1. CentOS 7.0
3.2. CentOS 7.2
3.3. CentOS 7.3
4. System Setup

4.1. Install Intel® DLIA card

1. Prepare one x16 PCIe* riser card
2. Remove black panel
3. Insert the Intel® DLIA card into the x16 PCIe* add-in card slot of chassis as shown in Figure 3
4. Using screw (C) to secure the riser card into place

4.2. Configure host BIOS

4.3. Install OS

CentOS 7.0, CentOS 7.2 and CentOS 7.3

4.4. Network setting

- Host must be able to access external network
- Set up proxy in /etc/yum.conf

4.5. Install Intel® DLIA software using installer

- Download install package from https://downloadcenter.intel.com/product/122302/
- Follow section Software Installation of this document for detailed instructions on using Intel® DLIA Software Installer
5. Software Stack

From bottom-up, Intel® Arria 10 FPGA is preprogrammed with six CNN primitives on the chip: convolution, local response normalization (LRN), rectified linear unit (ReLU), pooling, fully connected, and concatenation.

On the host side, Board Support Package (BSP) and OpenCL RunTime are needed to enable the communication with the hardware. The six primitives in the FPGA are then enabled through Intel® MKL-DNN, which is designed to provide a unified deep learning API for Intel® devices with optimized performance. With the integration of Intel® MKL-DNN to Caffe, users can build deep learning applications through DLIA stack using the Caffe framework, or directly using MKL-DNN primitive API.

To demonstrate the usage, a sample application is provided as reference, which implements an end to end object recognition application (AlexNet and GoogLeNet) for video inputs.

Finally, all software layers of stack are packed into an installer, which installs all components with a few clicks and key presses, to greatly facilitate installation and system setup from the user end.
6. Software Installation

6.1. Required Software and Install Environment

- CentOS 7.0 or higher (NOTE: Intel® DLIA has been validated on CentOS 7.0 7.2 and 7.3)

ISO files of CentOS can be downloaded from http://vault.centos.org. When installing CentOS, make sure to check option "development and creative workstation".

- Intel® DLIA installation package
  https://downloadcenter.intel.com/product/122302/

- Host must be able to access external network

- Set up proxy for YUM at /etc/yum.conf

6.2. Installation

1. Boot system with CentOS 7.0, CentOS 7.2 or CentOS 7.3

2. If booted to a graphical interface, open a terminal session:

   Applications -> Utilities -> Terminal

3. Copy Intel® DLIA install package intelDLIA-<version>.tar.bz2 to a temporary folder and un-tar it

   ```
   tar xvf intelDLIA-<version>.tar.bz2
   ```

4. Install Intel® DLIA

   ```
   cd installDLIA
   chmod +x dliaInstallCentos7-<version>.sh
   source dliaInstallCentos7-<version>.sh <option>
   ```

This script installs all required software, including Caffe, Intel®MKL-DNN, Altera® OpenCL runtime, drivers and FPGA image files. It also installs sample codes of Caffe AlexNet, Caffe GoogleNet and MKL-DNN AlexNet. These components are placed at below directories,
- Altera® OpenCL runtime and drivers
  ```
  /opt/intel/DLIA/altera
  ```

- Image file (.rpdp) for Intel® DLIA
  ```
  /opt/intel/DLIA/DLA/fpga
  ```

- Pre-built intelCaffe, MKL-DNN and OpenCV libraries
  ```
  /opt/intel/DLIA/lib
  ```

- Header files
  ```
  /opt/intel/DLIA/inc
  ```

- Sample codes
  ```
  /opt/intel/DLIA/application
  ```

- Source code of intelCaffe
  ```
  /opt/intel/DLIA/intelCaffe
  ```

- Source code of MKL-DNN
  ```
  /opt/intel/DLIA/MKLDNN
  ```

5. Verify installation
   - `aocl diagnose`
   Command `aocl diagnose` should return “DIAGNOSTIC_PASSED”
If `aocl diagnose` fails, further check status of PCIe* device and drivers
```bash
#lspci|grep -i altera
82:00.0 Class 1200: Altera Corporation Device 2494 (rev 01)
```
```bash
#lsmod|grep -i a10pl4
aclpci_a10pl4_drv 28406 0
```

Note: The installer does not modify binary image on FPGA. Please refer to Section Updating FPGA Image about how to update binary image on FPGA.

### 6.3. **Un-install Intel® DLIA**

Go to the directory where you store `intelDLIA-<version>.tar.bz2`. Run below commands to uninstall Intel® DLIA package,

```bash
cd installDLIA
source ./dliaInstallCentos-<version>.sh -u
```

If you have a previous version of the SW stack installed, it is highly recommended that you run this uninstall operation prior to a new installation.
7. Upgrade FPGA Image Using `dlia_flash_and_program`

**Note:** this tool `dlia_flash_and_program` is only for upgrading FPGA image for the Intel®DLIA cards with Beta 3.1 image. If your card is pre-flashed with Beta 3.1 image, you must first upgrade FPGA image before using Intel®DLIA. Latest FPGA image file (.rpd) is located at `/opt/intel/DLIA/DLA/fpga`. Once you have upgraded Beta 3.1 image, use tool `dlia_ipselector` (Section Updating FPGA Image using `dlia_ipselector`) for future upgrades.

To check if your Intel®DLIA is pre-flashed with Beta 3.1 image, please refer to Release Notes included in DLIA Installer.

Please also note the whole process takes about 15 mins. In addition, host server may automatically reboot during this process. However, automatic reboot only happens after image upgrade process is complete. That is, automatic reboot does not impact image upgrade process at all.

In the case your host does not reboot automatically, manual reboot is not required after this image upgrade process is complete. You can start to use DLIA with new FPGA image immediately.

### 7.1. `dlia_flash_and_program`

The upgrade process will erase the User Partition on Intel® DLIA card sector-by-sector, then perform a write of the RPD design file

```
   aocl diagnose  /*get device name, eg acla10pl40*/

   dlia_flash_and_program -beta-3-1-upgrade [device_name] [rpd_file_name]
```

<device_name> is actual name (eg, acla10pl4).

For example,

```
   dlia_flash_and_program --beta-3-1-upgrade acla10pl40
   /opt/intel/DLIA/DLA/fpga/DLIA_<version>.rpd
```
8. Updating FPGA Image using `dlia_ipselector`

Intel® DLIA installer installs FPGA image files (.rpd) at
`/opt/intel/DLIA/DLA/fpga`.

Typically there are more than one image file: FPGA images are optimized based on
network topologies. To help Intel® DLIA users find the optimal FPGA image, we provide
the tool `dlia_ipselector`. Based on the prototxt file of your network,
`dlia_ipselector` automatically find and update FPGA image. The whole process is
transparent to users.

Below is how `dlia_ipselector` is invoked in a shell,

```
   dlia_ipselector [/path_to/prototxt_file_name]
```

When `dlia_ipselector` finds the optimal image, it will prompt to ask if you want to
update FPGA image. Input `y` to confirm to update FPGA image. If you are not ready for
update, enter `n`. 
9. Building Caffe and MKL_DNN from Source Code

Pre-compiled binaries of Caffe and MKL_DNN are installed at /opt/intel/DLIA/lib. In case you want to build those binaries from source code, follow the instructions in this section.

9.1. Build Caffe

Navigate to /opt/intel/DLIA/intelCaffe. Issue the following command,

```
# rm -rf build
# rm -rf .build_release
# make -j`nproc`
```

You will find the new libcaffe.so in ./build/lib folder.

9.2. Build Intel® MKL-DNN

Navigate to /opt/intel/DLIA/MKLDNN. Issue the following commands,

```
# rm -rf build
# mkdir build
# cd build
# cmake..
# make -j`nproc`
```

You can find the new libmkldnn.so in ./build/src folder.

./build/user_examples contains executables of MKLDNN sample code.

9.3. How to use newly built libcaffe.so and libmkldnn.so

First, back up original libcaffe.so and libmkldnn.so, which are in the folder /opt/intel/DLIA/lib. Then copy the newly built libcaffe.so and libmkldnn.so to /opt/intel/DLIA/lib.
10. Running Sample Code

The installer installs sample code at folder

/opt/intel/DLIA/application/example.

Below is a list of subfolders:

./data (test data, such as video files)
./models (prototxt, caffemodel)
./object_recognition (source code, headers and Makefile)
./multi-fpga (example app for multi DLIA cards)

10.1. Running Caffe sample application

Navigate to folder

/opt/intel/DLIA/application/example/object_recognition

To build the sample application with Caffe framework, use command

```
make -j$['nproc']
```

This command will create executable classifier.

classifier accepts the following options

```
--model (#Caffe prototxt)
--weight (#Caffe binary weight)
--input (input video file)
--guil (#display classification result in GUI window)
--batch (#batch size)
--loop (#enable loop mode)
--binarymean (#mean value of video)
```

For example, to run AlexNet on Intel® DLIA

```
./classifier
   --model ../models/deploy_alexnet_dlia.prototxt
   --weight ../models/bvlc_alexnet.caffemodel
   --input ../data/Test_Vid_4.mp4
   --batch 960
   --loop
   --binarymean ../data/imagenet_mean.binaryproto
```

To run classifier on CPU, use the prototxt which specifies CPU engine. For example, to run AlexNet on CPU

```
./classifier
   --model ../models/deploy_alexnet_cpu.prototxt
   --weight ../models/bvlc_alexnet.caffemodel
   --input ../data/Test_Vid_4.mp4
```
Note: for the topologies not found in the folder ./models (e.g., VGG), Caffe prototxt and caffemodel files for Intel® DLIA can be generated by

- Download original prototxt and caffemodel files from Caffe official website
  https://github.com/BVLC/caffe/wiki/Model-Zoo
- Run tool dlia_converter to create new prototxt and caffemodel files for Intel® DLIA
- The new prototxt and caffemodel can be used with the above sample application

10.2. Running MKL-DNN sample application

Besides Caffe, the sample application also provides an example of how to implement AlexNet using MKL-DNN API. To build application for MKL-DNN, use command

```
make mkldnn -j$nproc
```

This command will create executable classifierMkldnn, which accepts the below options

```
--input      (#input video file)
--gui        (#display classification result in GUI window)
--batch      (#batch size)
--loop       (#enable loop mode)
```

Current version of sample code for MKL-DNN only supports AlexNet, trained based ImageNet 1K dataset, running on Intel® DLIA engine:

```
./classifierMkldnn
   --input ../data/Test_Vid_4.mp4
   --batch 960
   --loop
```

10.3. Example outputs of sample application

- GUI mode (--gui)
  
  For each frame, the sample app will list the first three levels of confidence,
• **Text mode**
  
  For each frame, the sample app will print out the first three levels of confidence:

  ```
  [root@montaCanyon object_recognition]# ./classifier
  frame 0
  giant panda(99%)
  Old English sheepdog(0%)
  soccer ball(0%)
  frame 1
  giant panda(99%)
  Old English sheepdog(0%)
  soccer ball(0%)
  frame 2
  giant panda(99%)
  Old English sheepdog(0%)
  dalmatian(0%)
  frame 3
  giant panda(99%)
  Old English sheepdog(0%)
  Border collie(0%)
  ```

• **Report of performance**
  
  On completion, the sample application prints out performance data (frames-per-second) of classification.
Total frames processed: 2880
Total time (seconds): 4.53593
Total preprocessing time (seconds): 0.88744
Total classification time (seconds): 3.21603
Average time per image (milliseconds): 1.57497
Average preprocessing time per image (milliseconds): 0.308139
Average classification time per image (milliseconds): 1.11668
Average image rate (img/sec): 634.931
Average classification rate (img/sec): 895.513
Average preprocessing rate (img/sec): 3245.29
11. Develop Applications for Intel® DLIA

11.1. Environment

- **Library files**
  Intel® DLIA installer installs Intel® Caffe, MKL-DNN and related DLIA libraries at
  directory /opt/intel/DLIA/lib

  ```
  [root@montaCanyon lib]# pwd
  /opt/intel/DLIA/lib
  [root@montaCanyon lib]# ll
  total 32728
  -rwxr-xr-x 1 root root 23345066 Apr 6 15:45 libcaffe.a
  -rwxr-xr-x 1 root root 6050384 Apr 6 15:45 libcaffe.so
  -rwxr-xr-x 1 root root 6050384 Apr 6 15:45 libcaffe.so.1.0.0-rc3
  -rwxr-xr-x 1 root root 3292007 Apr 14 09:17 libdilia.so
  -rwxr-xr-x 1 root root 1379145 Apr 14 09:17 libdnnm5.so
  -rwxr-xr-x 1 root root 8698816 Apr 14 09:17 libmkldnn.so
  -rwxr-xr-x 1 root root 112600418 Apr 14 09:18 libmklmq_gnu.so
  -rwxr-xr-x 1 root root 113800545 Apr 14 09:18 libmklmq_intel.so
  ```

- **Header files**
  Headers files are placed at /opt/intel/DLIA/include

  ```
  [root@montaCanyon inc]# pwd
  /opt/intel/DLIA/inc
  [root@montaCanyon inc]# ll
  total 750
  -rwxr-xr-x 1 root root 7419 Apr 6 15:44 A0CL_Utils.h
  -rwxr-xr-x 1 root root 4986 Apr 6 15:45 caffe
  -rwxr-xr-x 1 root root 77198 Apr 6 15:44 dlia.h
  -rwxr-xr-x 1 root root 2050 Apr 6 15:44 dlia_perf.h
  -rwxr-xr-x 1 root root 2447 Apr 6 15:44 dlia_trace.h
  -rwxr-xr-x 1 root root 9032 Apr 6 15:44 dlia_types.h
  -rwxr-xr-x 1 root root 35332 Apr 6 15:44 fpga_arch.h
  -rwxr-xr-x 1 root root 77269 Apr 6 15:44 mkldnnblas.h
  -rwxr-xr-x 1 root root 47426 Apr 6 15:44 mkldnn.h
  -rwxr-xr-x 1 root root 35946 Apr 6 15:45 mkldnn_cppwrapper.h
  -rwxr-xr-x 1 root root 19637 Apr 6 15:44 mkldnn.h
  -rwxr-xr-x 1 root root 24297 Apr 6 15:44 mkldnn.h
  -rwxr-xr-x 1 root root 79835 Apr 6 15:44 mkldnn.hpp
  -rwxr-xr-x 1 root root 3438 Apr 6 15:44 mkldnn_types.h
  -rwxr-xr-x 1 root root 24485 Apr 6 15:44 mkldnn_types.h
  -rwxr-xr-x 1 root root 11420 Apr 6 15:44 mkldnn_service.h
  -rwxr-xr-x 1 root root 4324 Apr 6 15:44 mkldnn_types.h
  -rwxr-xr-x 1 root root 2143 Apr 6 15:44 mkldnn_version.h
  -rwxr-xr-x 1 root root 8291 Apr 6 15:44 mkldnn_vml_defines.h
  -rwxr-xr-x 1 root root 102809 Apr 6 15:44 mkldnn_vml_functions.h
  -rwxr-xr-x 1 root root 2125 Apr 6 15:44 mkldnn_vml.h
  -rwxr-xr-x 1 root root 3269 Apr 6 15:44 mkldnn_vml_types.h
  -rwxr-xr-x 1 root root 43324 Apr 6 15:44 mkldnn_vml_functions.h
  -rwxr-xr-x 1 root root 66542 Apr 6 15:44 mkldnn_vml_types.h
  -rwxr-xr-x 1 root root 2870 Apr 6 15:44 mkldnn_vml.h
  -rwxr-xr-x 1 root root 5526 Apr 6 15:44 mkldnn_vml_types.h
  ```
10.2 Developing Applications for Intel® DLIA

Figure 5 shows the procedure to develop Intel® applications. There are two supported APIs: Intel Caffe and Intel MKL-DNN. Intel® DLIA Converter is a tool to generate Intel® DLIA-compliant topology (prototxt and caffemodel).

- **Intel® DLIA topology converter**
  Before actual programming with Intel Caffe / MKL-DNN, it is highly recommended to first use Intel® DLIA Converter. This tool has two functions:
  - Convert original topology to Intel® DLIA-compliant topology
  - Optimize original topology to ensure high performance with Intel® DLIA

Intel® DLIA topology converter dlia_converter is installed under /opt/intel/DLIA/intelcaffe/tools. The command line below shows how to use the tool,

```
dlia_converter <orig.prototxt> <orig.caffemodel>
    <new.prototxt> <new.caffemodel>
```

<orig.prototxt> and <orig.caffemodel> are filenames of original prototxt and caffemodel files. dlia_converter creates new topology by generating
new.prototxt and new.caffemodel. These new prototxt and caffemodel files should be used when developing applications for Intel® DLIA.

- **Programming with Caffe API**
  Intel® DLIA strictly follows the programming model of Caffe, with the only extension of a new engine type MKLDNN:FPGA. For the primitives to be accelerated by Intel® DLIA, in prototxt file specify MKLDNN:FPGA as their engine type. Below is an example of a Caffe convolution layer using Intel® DLIA acceleration,

  ```
  engine: MKLDNN:DLA
  ```

  ```
  layer {
    name: "conv1"
    type: "Convolution"
    bottom: "data"
    top: "conv1"
    convolution_param { 
      num_output: 96 
      kernel_size: 11 
      stride: 4 
    }
  }
  ```

  Intel® DLIA supports the so-called hybrid engine mode. This means Caffe layers can have different engine types (e.g., MKLDNN:DLA and CAFFE).

- **Programming with MKL-DNN API**
  Intel® DLIA also exposes MKL-DNN API for users to build their deep learning applications. MKLD-N library is Intel® primitive-based deep learning library that provides optimal implementation of CNN primitives for supported devices. We direct users who are interested in programming in MKL-DNN to its official site: http://01org.github.io/mkl-dnn. The descriptions below only contain information on the DLIA extension.

  Intel® DLIA is enabled through two parameters: engine kind `engine::kind::dla`, and stream kind `stream::kind::lazy`.

  - **engine::kind::dla**
    Primitives with engine `engine::kind::dla` are accelerated by Intel® DLIA. Below code snippet shows how to create this type of engine,

    ```
    using namespace mkldnn;
    engine fpga_engine(engine::kind::dla);
    ```

  - **stream::kind::lazy**
    To maximize performance of Arria10 FPGA, primitives in DLIA engine are implemented in lazy execution mode, which is conceptually illustrated in below
Figure 6. Comparison of behaviors of eager and lazy MKL-DNN streams. (Note: lazy MKL-DNN stream fuses primitives and then submits fused primitives together to FPGA).

Compared with default stream type stream::kind::eager, stream stream::kind::lazy does not immediately executes primitives. Instead, it waits until the full execution graph is reconstructed, and processed when any data is requested. In the execution, DLIA engine greedily fuses primitives to optimize the overhead of communicating with the FPGA device. Figure 7 shows how primitive fuser merging AlexNet primitives into sequences of primitives before mapping to FPGA calls.

Figure 7. Example of how primitive fuser merges AlexNet primitives into fused primitives.

Below code snippet shows how lazy stream works,

```cpp
using namespace mkldnn;
stream lazy_stream(stream::kind::lazy);
lazy_stream.submit(primitive 1) /* primitive 1 is only enqueued*/
lazy_stream.submit(primitive 2) /* primitive 2 is only enqueued*/
```
```
  lazy_stream.submit(primitive n) /* primitive n is only enqueued*/
  lazy_stream.wait()
  /* at this point, primitives 1...n are fused together and submitted to FPGA*/

  * reorder primitive

Reorder primitives are used to synchronize data between host CPU with DLIA device. The following two examples illustrate how the input/output data in a network can be synchronized through this primitive:

Input data sent from host to DLIA device:
  auto c1_input = memory({{c1_desc, cpu_engine}, c1in.data()});
  auto c1_input_mem = memory(conv1_pd.src_primitive_desc());
  auto rdr1 = reorder(c1_input, c1_input_mem);

Output data copied from DLIA device to host:
  auto r8_output_mem = memory(relu8_pd.dst_primitive_desc());
  auto r8_output = memory({{r8_desc, cpu_engine}, r8out.data()});
  reorder(r8_output_mem, r8_output);

A set of examples on how to use MKL-DNN API can be found in the examples/ directory.

10.3 Building applications

- Choose a C++ compiler which supports C++11

  Intel® DLIA is verified with the default C++ compiler of CentOS 7.0 and CentOS 7.2 (GCC 4.8.5).

- Set up environment

  source /opt/intel/DLIA/application/example/object_recognition/setDLIAenv.sh

- In Makefile, set up flags for compiler and linker

  CPPFlags = -std=c++11 -I/opt/intel/DLIA/include
  CPPFlags += <application-specific flags>

  LFLAGS = -L/opt/intel/DLIA/lib -lcaffe -lmkl 다
  LFLAGS += <application-specific linker flags>

- There is a sample Makefile at

  /opt/intel/DLIA/application/example/object_recognition/Makefile
11.4. Limitations

- Limitations on network topology
  - Layers related to convolution must be organized in the following order:
    - Convolution -> ReLU -> Norm -> Pooling
    - However, Pooling and Norm layers can be passed.

    Below are examples of valid networks:
    - Convolution
    - Convolution -> ReLU
    - Convolution -> ReLU -> Pooling
    - Convolution -> ReLU -> norm
    - Convolution -> ReLU -> Norm -> Pooling

    And any combinations of above networks are valid networks.

    Examples of invalid networks:
    - Convolution -> Pooling -> ReLU
    - Convolution -> Norm -> ReLU
    - ReLU -> Norm
    - Norm -> Pooling
  
  - Fully-connected -> ReLU can only be followed by another Fully-connected -> ReLU
  
  - Once fully-connected operation starts, no Convolution can be executed later.

- Limitations on layer parameters
  - Input
    - Maximal number of groups is 2
  
  - Convolution
    - Convolution stride must be 1, 2 or 4, except for the first convolution layer. The first convolution layer can have any stride
    - Maximal number of channels of a Convolution layer is 1056
    - Max size of convolution filter is 49152 floats
    - The total of input and output sizes of a convolution layer cannot be larger than 786432 floats
    - The first convolution layer has following limitations:
      (image width / horizontal stride) <= 115
      (image height / vertical stride) <= 115
(filter width / horizontal stride) <= 5
(filter height / vertical stride) <= 5

- For other convolution layers (except the first convolution layer), there are following limitations

  Image width <= 115
  image height <= 115
  filter width <= 5
  filter height <= 5

- Image width (height) of convolution output cannot be larger than 112

  - Pooling
    - Only max and average pooling are supported
    - Size of pooling window cannot be larger than 7
    - Pooling stride can only be 1, 2 and 4
    - Output image width (height) of pooling cannot be larger than 56

  - Fully-connected
    - Maximal input size of fully-connected layer is 12288 floats
    - Maximal size of output of a Fully-connected layer is 4096
12. Performance Optimization

11.1 Batch sizes

Intel® DLIA can process multiple input images in parallel. Generally speaking, performance significantly improves when batch size increases.

Although batch size can be any integer number, internally DLIA is optimized to work with batch sizes in a multiple of 96. For maximal throughput and efficiency DLIA should be run with the largest batch size which the host system can support that is a multiple of 96 (e.g., 960).

11.2 OMP_NUM_THREADS

Intel® DLIA uses OpenMP to accelerate image pre-processing. Environment variable OMP_NUM_THREADS significantly affects performance. To help developers find the optimal OMP_NUM_THREADS, we provide a tool called find_optimal_openmp_thread_numbers.sh. This tool is installed by Intel® DLIA Installer at /opt/intel/DLIA/application/example/object_recognition.

Below are instructions about how to use the tool:

- Navigate to /opt/intel/DLIA/application/example/object_recognition
- Build sample app by following command
  `make`
- Run the tool
  `./find_optimal_openmp_thread_numbers.sh`

On completion, this tool will print out recommended value for OMP_NUM_THREADS. It is highly recommended to add below line to your .bashrc file to make it persistent (otherwise, the value is only valid for the current shell session).

```bash
export OMP_NUM_THREADS=[the value found by the tool]
```

11.3 Synchronization between host and Intel® DLIA

Synchronization between host and Intel® DLIA significantly slows down Intel® DLIA. So use of synchronization should be avoided, unless necessary. Synchronization happens in the following cases,
Engine type change between connected layers
Host application writes / reads data to / from Intel® DLIA.

For example, MKL-DNN primitive mkldnn::reorder triggers data copying/synchronization between host and Intel® DLIA.

11.4 Considerations of performance of host system

While Intel® DLIA can quickly handle workload of CNN, slow host can become bottleneck of whole system.

From our experiments, the following host systems have achieved highest performance for video classification tasks

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Memory</th>
<th>Sockets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host-1</td>
<td>Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (22 Cores; 44 Threads)</td>
<td>16 x 8GB DDR4 2133</td>
<td>2</td>
</tr>
<tr>
<td>Host-2</td>
<td>Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz (14 Cores; 28 Threads)</td>
<td>8 x 8GB DDR4 2133</td>
<td>2</td>
</tr>
</tbody>
</table>
12 Using Multiple Intel® DLIA Accelerators

Typically one host supports more than one Intel® DLIA card. For example, the system below consists of two Intel® DLIA cards.

![Image](example.png)

Figure 8. Example of two Intel(R) DLIA cards

Below procedure shows how to develop applications for multiple Intel® DLIA accelerators.

- Find out how many Intel® DLIA accelerators are installed on host:
  - Query OpenCL platform ID of Intel® DLIA using `clGetPlatformIDs()` and `clGetPlatformInfo()`.

  ```
  [bryan@F99BIOS91 Desktop]$ lspci|grep -i altera
  06:00.0 Processing accelerators: Altera Corporation Device 2494 (rev 01)
  08:00.0 Processing accelerators: Altera Corporation Device 2494 (rev 01)
  [bryan@F99BIOS91 Desktop]$ 
  ```

  Intel® DLIA uses below platform vendor and platform name,
  ```
  vendor: Altera Corporation
  name: Intel(R) FPGA SDK for OpenCL(TM)
  ```

  - Based on OpenCL platform ID, query number of `CL_DEVICE_TYPE_ACCELERATOR` devices on this platform by `clGetDeviceIDs()`.
  - To accelerate tasks on Intel® DLIA with ID `n`, set environment variable `CANYON_VISTADEVICE_SELECT_CLASSIFIER` to the corresponding string of `n`. For example, to use Intel® DLIA with ID `2`,

    ```
    setenv(CANYON_VISTADEVICE_SELECT_CLASSIFIER, "2", 1);
    ```
13 Thermal Throttling

To keep the card within safe temperature levels, thermal throttling will throttle the FPGA clock speed depending on how hot the card is. This ranges from 96Mhz-243Mhz for FPGA binary image 1, and 96Mhz-230Mhz for FPGA binary image 2.

While thermal throttling is enabled by default (ENABLE_THROTTLING = 1), users can disable it by setting the environment variable ENABLE_THROTTLING to 0 (export ENABLE_THROTTLING=0).

Users can also set their temperature threshold using the environment variable TEMPERATURE_SET_POINT. This value can be set to any temperature (integer) within the range 45-85 Celsius.

Users can monitor the card(s) clock speed by setting the environment variable ACL_HAL_DEBUG=99. You can then run the sample application, and grep for clock information (e.g. ./classifier | grep “Kernel clock”).
14 Where Do I Go From Here

- Quick Start Guide
  

- Installer package
  
  https://downloadcenter.intel.com/product/122302/

- Dear Customer Letter (DCL)
  
  https://cdrd.intel.com/v1/dl/getContent/572649.htm