Application Note

Optimizing TI Deep Learning Performance via Memory and DDR Bandwidth Reduction

Adam Hua, Reese Grimsley

ABSTRACT

TIDL is TI's AI inference framework operating on TDA4x and AM6xA series processors, utilizing the built-in C7xMMA AI accelerator for efficient AI model inference. The C7xMMA, as a dedicated AI inference accelerator, features a complex architecture. While the TIDL inference framework has extensively optimized its resource allocation to maximize utilization efficiency, high memory bandwidth consumption can still occur during model inference. To further leverage inference resources and reduce memory usage, models running on TIDL require additional optimization. This document details model optimization methods aimed at reducing DDR bandwidth consumption.

Table of Contents

Trademarks

1 Introduction

Current SoCs capable of AI model inference typically employ one of two architectures: those integrating general-purpose GPUs, and those incorporating dedicated AI inference accelerators, commonly termed NPUs. TI's AI-accelerated SoCs in TDA4x and AM6xA portfolios adopt the latter approach, with their NPU often referred to as C7xMMA. This name originates from the NPU's two components: the C7000 series floating-point digital signal processor and a Matrix Multiply Accelerator (MMA). The C7x series DSP core runs an RTOS, handling data scheduling and non-linear processing within the model. The MMA, which is deeply coupled with the C7x, is responsible for linear algebra operations like matrix multiplication and 2D convolutions that make up >99% of the computation requirements for most neural networks.

TI provides the TI Deep Learning (TIDL) inference framework, offering a unified interface for easy invocation by the high-level operating system (*e.g.,* Linux, QNX). Invocation methods are beyond this document's scope, we assume reader familiarity with the relevant interfaces and will focus on model optimization techniques. Users leverage TIDL tooling to compile models for a particular processor. TIDL then deploys quantized, compiled models onto the NPU, allowing users to invoke inference using the TIDL runtime (TIDL-RT), tivxTIDLNode or open-source runtimes (OSRT) like ONNX Runtime or Tensorflow-Lite.

TIDL memory read/write bandwidth refers to the load on the DDR interface. For instance, if a single inference frame requires reading 100MB from DDR (including model weights, input, and intermediate layer feature maps) and writing 50MB to DDR (including model output and intermediate feature maps), achieving a 30 fps frame rate demands a total DDR read/write bandwidth of 4.5 GB/s. Given that single-channel DDR4 might offer actual bandwidth around 8 GB/s, TIDL model inference can consume significant bandwidth.

Some devices like AM67A and TDA4VH include multiple instances of the C7x. Parallel inference tasks running across parallel accelerators will have further implication on the DDR utilization and contention, although processors like TDA4VH will include multiple DDR interfaces. System-wide DDR contention should be considered when viewing system-level performance, but the optimizations in this document remain beneficial for reducing DDR utilization in the first place to improve model performance.

Optimizing DDR bandwidth requires some understanding of the C7xMMA cache structure, effectively utilizing TIDL tools, and potentially modifying the model.

2 C7xMMA Cache Structure

DDR bandwidth optimization starts with comprehending the TI C7xMMA's memory hierarchy, simplified in the figure below.

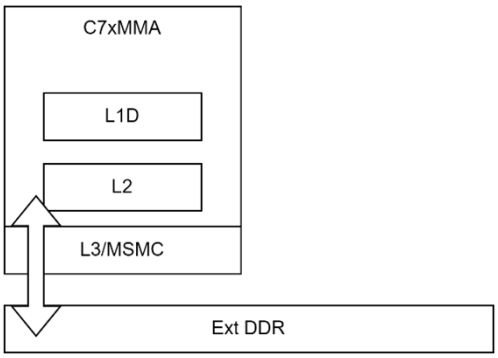

Figure 2-1. C7xMMA Three-Level Cache Structure

The C7xMMA employs a three-level cache structure. Beyond external DDR, it incorporates internal L1D, L2, and L3/MSMC caches. L1D is the smallest and closest to the compute core (typical size 16KB). L2 is somewhat more distant (typical sizes 224KB, 448KB), but tightly coupled to data movement mechanisms for the MMA. L3 on TDA4x is the Multicore Shared Memory Controller (MSMC), while on other SoCs, it isSRAM managed individually by each C7xMMA. Note: The L1D, L2, L3 terminology here corresponds to the TIDL framework's description; chip datasheets may refer to L1P, L1D, L2, and in some SoCs, L3 (i.e. MSMC on TDA4VM). The size of L2 and L3 regions can be found in a device_config.cfg file included in tidl tools

The figure below shows cache usage during a typical layer's inference, involving four operations. Operation 1 is DMA transferring data directly from DDR to L2. Operation 2 moves data from L3 to L2. Operation 3 transfers data from L2 to L3. Operation 4 moves data from L3 to DDR. Operations 2 and 3 are over ten times more efficient than 1 and 4. Utilizing the previous layer's feature map can lead to three scenarios: only Operation 1 (if the input layer and previous layer output reside entirely in DDR); only Operation 2 (if the previous feature map fits completely in L3/MSMC); or both Operations 1 and 2 (if the previous output is too large for L3, partially stored in DDR). After computing the current layer's feature map, Operation 3 is prioritized to move data to L3. If L3 capacity is exceeded, Operation 4 stores the surplus in DDR. Weight values are always stored in DDR and fetched directly to L2 when needed.

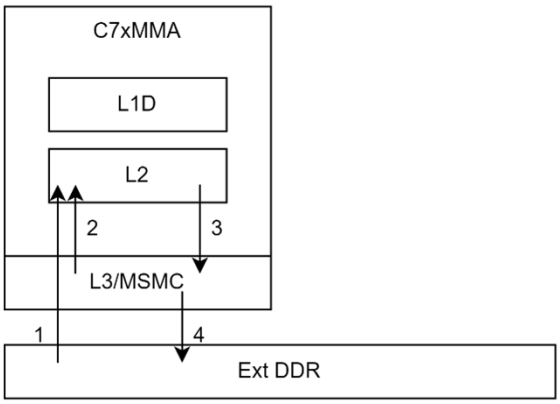

Figure 2-2. C7xMMA Cache Operations

This three-level cache architecture significantly boosts inference efficiency by avoiding slow DDR read/writes during compute cycles and conserving DDR bandwidth. The key to improving efficiency and saving bandwidth lies in maximizing L3 utilization, preventing feature map storage in DDR. The next section covers analyzing model memory usage.

3 Model DDR Read/Write Analysis for a Compiled TIDL Model

The TI model compilation tool provides corresponding interfaces to facilitate fast DDR read/write analysis. When compiling the model, a bufinfolog_0.csv file will be generated in the output folder. To be precise, it will be in a folder with naming like artifacts/tempDir/$SUBGRAPH_NAME_tidl_net, where the SUBGRAPH_NAME is model dependent. Exact names of the folder structure may have slight changes in different SDK releases.

It is important to note that when using the RTOS (i.e., tidl_model_import.out) tool, you must ensure that a perfSimConfig is provided and that the tool executes without errors. In any scenario where model compilation fails, the errors must be resolved before analyzing bandwidth consumption. Any error logs during the compilation process may lead to inaccurate analysis results.

The table below explains the key fields in the bufinfolog CSV file.

Table 3-1. Description of fields in *bufinfolog* CSV files

Field	Description
Ni	Input feature map channel dimension.
No	Output feature map channel dimension.
InW	Input feature map width dimension.
InH	Input feature map height dimension.
OutW	Output feature map width dimension.
OutH	Output feature map height dimension.
In-Write-size	Actual input feature map size, calculated in Bytes. May include some small overhead for padding or the DMA bus sizing
In-Write-memSpace	L2 or DDR; All layers use L2 except the Data layer (input/output layers)
Out-Read-memSpace	Temporary storage location for current layer's computation results. Values are typically empty, L2, MSMC, or DDR. Empty indicates computed results are directly stored into Out-Write-memSpace. When a value is present, it indicates part of the current layer's output feature map needs to be stored in Out-Read-memSpace first, then read from this space and written into Out-Write-memSpace.
Out-Write-memSpace	Final storage location for current layer's feature map. Values are DDR or MSMC. DDR indicates the current layer's output feature map cannot fully fit into MSMC, so part or all of it is stored in DDR. MSMC indicates the output is entirely stored in MSMC for use by subsequent layers.
Out-Write-size	Amount of data stored in the Out-Write-memSpace by the current layer. When Out-Write-memSpace is DDR, this value represents the current layer's contribution to DDR write bandwidth consumption. May include some small overhead for padding or the DMA bus sizing
Wt-Write-memSpace	Loading location for current layer's weights. Always L2. Weight values are entirely stored in DDR and loaded into L2 when used.
Wt-Write-size	Size of the current layer's weight data. This value contributes to DDR read bandwidth consumption. May include some overhead for padding or DMA bus sizing

Among the fields mentioned above, DDR write bandwidth is primarily influenced by the value of **Out-Write-memSpace**. By reducing the output feature map size for a layer, the DDR write bandwidth consumption can be significantly decreased.

DDR read bandwidth is mainly affected by **Wt-Write-size** and **In-Write-size**. Notably, **In-Write-size** only has an impact when the output of the previous layer is partially or entirely stored in DDR.

Therefore, the key to reducing DDR read/write bandwidth lies in optimizing the model to minimize the read bandwidth caused by weights and the read/write bandwidth resulting from large feature maps.

4 Model Optimization

Based on the above, reducing DDR bandwidth relies on keeping intermediate feature maps in L3, necessitating intentional model design.

4.1 Simple Structure Models

Simple models have linear, non-branching structures. The initial section of EfficientNet shown below is entirely sequential. Here, ensuring each layer's output fits within L3 suffices. TIDL can automatically configure layers to run using L3, avoiding DDR interaction. DDR interaction will still be required when intermediate feature maps are larger than the L3 size.

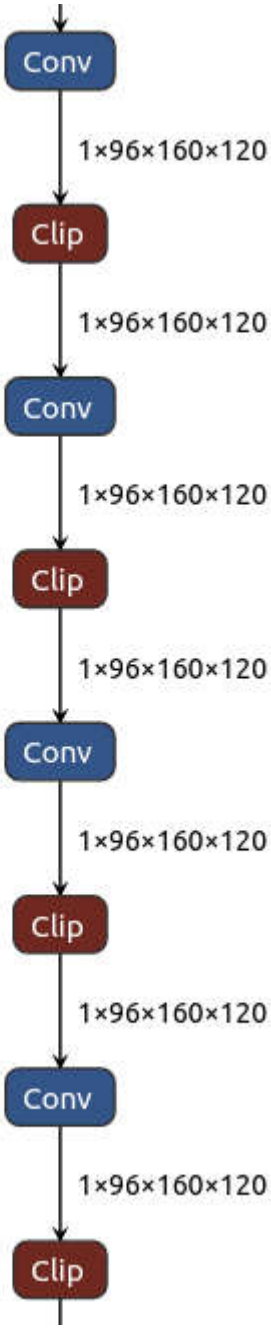

Figure 4-1. Simple Model with Sequential Layer Ordering

4.2 Complex Structure

Many models in practice have more complicated structures and graph patterns than sequential layers only. The next sections will look at a few examples of more complicated structures and what this entails for DDR and cache usage.

4.2.1 Residual Structures

Many backbone networks use residual structures like the one shown, creating localized parallel paths called "Residuals" that are beneficial during training. Residuals avoid the the vanishing gradient problem.

During compilation, TIDL simulates DDR bandwidth for different computation orders (left branch first, right first, interleaved), and will select the most efficient. It also decides whether the first Conv layer’s output stays in L3/MSMC until the add operation or is written to DDR immediately. Storing in DDR causes direct bandwidth cost, while holding in L3 may occupy memory during left branch computation, potentially forcing parts of the left branch to use DDR instead.

TIDL will select a strategy that maximizes L3 occupancy, but large intermediate feature maps may require DDR be used. In this case, it is recommeded to prioritize optimizing the size of the longer path (left-side in the figure) to avoid multiple feature-maps going to DDR as opposed to the single feature map along the skip connection.

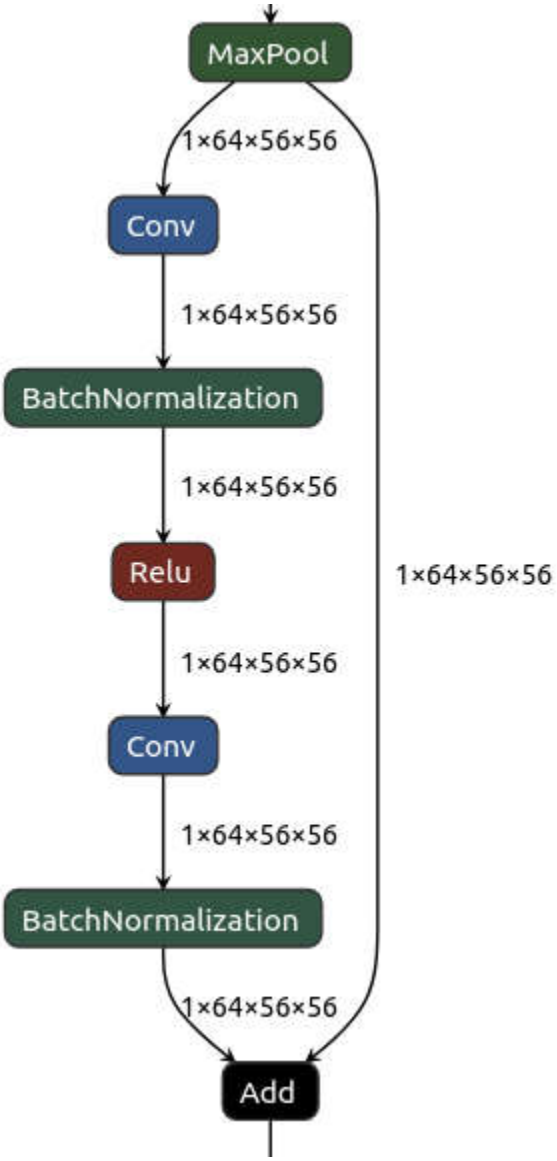

Figure 4-2. Residual structure in a neural network. The right-side path's "skip" connection has a feature map that must be stored until the left-side path completes

4.2.2 Parallel Branch Merge

Applications often involve multiple deep parallel branches merging into one, or one branch splitting into several deep parallel paths. This is especially common for multi-input neural networks. The figure shows part of a classic four-input BEV network after the gridsample operator merges paths.

Since paths are deep before merging, feature maps must be placed in DDR at the merge point, making DDR bandwidth consumption unavoidable here. Such architectures should be used only as needed to avoid excession DDR bandwidth and resulting bottlenecks. However, it may be possible to reduce the DDR read bandwidth caused by weights. The model architecture can be modified to consolidate multiple input heads and setting the batch dimension of certain model layers to a value greater than 1 such that weights need only load once.

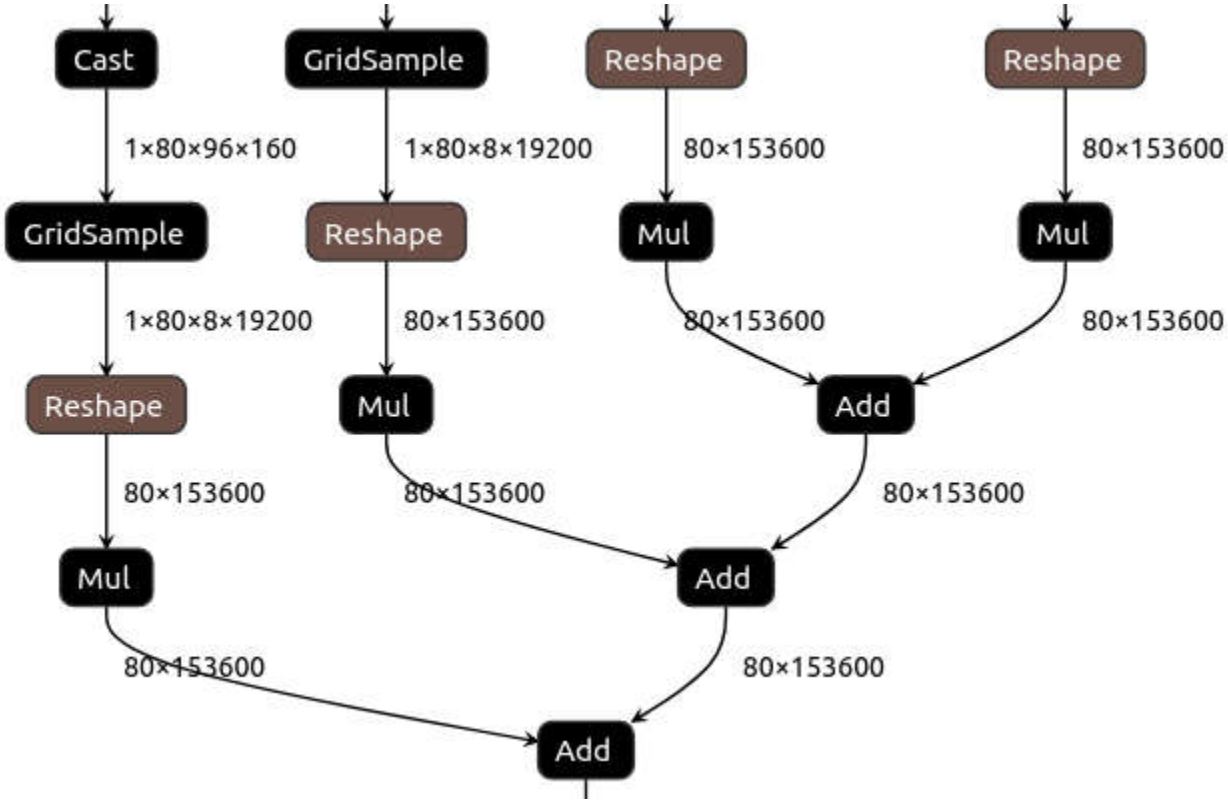

Figure 4-3. Complex structure merging multiple parallel branches

For example, in the above figure, the same backbone network precedes the GridSample layers and the feature maps at each layer of this backbone are relatively small. The four branches can be merged into two or even one, with the batch dimension adjusted accordingly. These would be followed by appropriate Slice or data-shaping layers to separate out the batches again so they can be recombined by the Add layers shown. This approach can reduce or even prevent the same weights from being repeatedly loaded, thereby lowering DDR read bandwidth overhead. This method requires attention to the size of the feature maps in the merged backbone network.

5 Summary

Model optimization for DDR bandwidth primarily involves reducing per-layer feature map size and increasing depth. DDR bandwidth consumption may be unavoidable with complex structures. TI's Model Zoo offers numerous optimized and validated models and backbones. Given the maturity of common architectures, consider replacing your model's backbone with a TI-optimized version for rapid improvement.

This document detailed methods for analyzing model DDR bandwidth consumption and optimizing models to reduce it. This is highly relevant for users of TDA4x, AM6xA series SoCs, and the TIDL inference framework. Applying these methods typically results in optimized models consuming bandwidth only for inputs and outputs, freeing significant resources for the overall system.

6 References

1. https://www.ti.com/product/AM62A7
2. https://www.ti.com/tool/PROCESSOR-SDK-AM62A
3. https://www.ti.com/product/AM67A
4. https://www.ti.com/product/TDA4VM
5. https://www.ti.com/product/TDA4VE-Q1
6. https://www.ti.com/product/TDA4VM-Q1
7. https://www.ti.com/product/TDA4AL-Q1
8. https://www.ti.com/product/TDA4VL-Q1
9. https://www.ti.com/product/TDA4VP-Q1
10. https://www.ti.com/product/TDA4VH-Q1
11. https://www.ti.com/product/TDA4VEN-Q1

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you fully indemnify TI and its representatives against any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale, TI’s General Quality Guidelines, or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025