IMAGINATION TENSOR TILING

By Alex Pim, Vice President of AI Research & Architecture, Imagination.

Introduction

For system designers trying to meet the ever-increasing challenges in the automotive industry, the need to reduce external DDR system bandwidth in hardware-accelerated inference systems rank high on the list. Not only does a decrease in DDR bandwidth equate to a reduction in power consumption, but it also reduces the processing latency of the networks running on the system.

This article focuses on how Imagination Tensor Tiling technology inside the IMG Series4 neural network accelerator (NNA) has been specifically designed to help SoC designers achieve these aims.

To be as efficient as possible, a deep neural network accelerator must use as little external memory bandwidth as it can: this reduces the overall power consumption of the system, and decreases the time to inference. In most cases the available DDR bandwidth figure given to a deep neural network accelerator is a theoretical maximum – in reality, this is often limited to a much lower value.

To help explain this, the diagram below shows how as we reach the upper limit in available bandwidth of the system, on-chip memory (OCM) size has little effect, as significant bandwidth usage allows the input and coefficient buffers to be serviced with minimum latency. Conversely, a lower limit in available bandwidth means very large OCM sizes are needed to execute the network without noticeable latency.

By varying the input and coefficient buffer size, local minima for each network configuration can be found, but what we actually want is to shift this bandwidth/OCM plot for all networks to the left – DDR bandwidth usage *and* physical OCM size.

From a silicon-area point of view, laying down OCM is costly and can end up negating other area optimisations to the core.

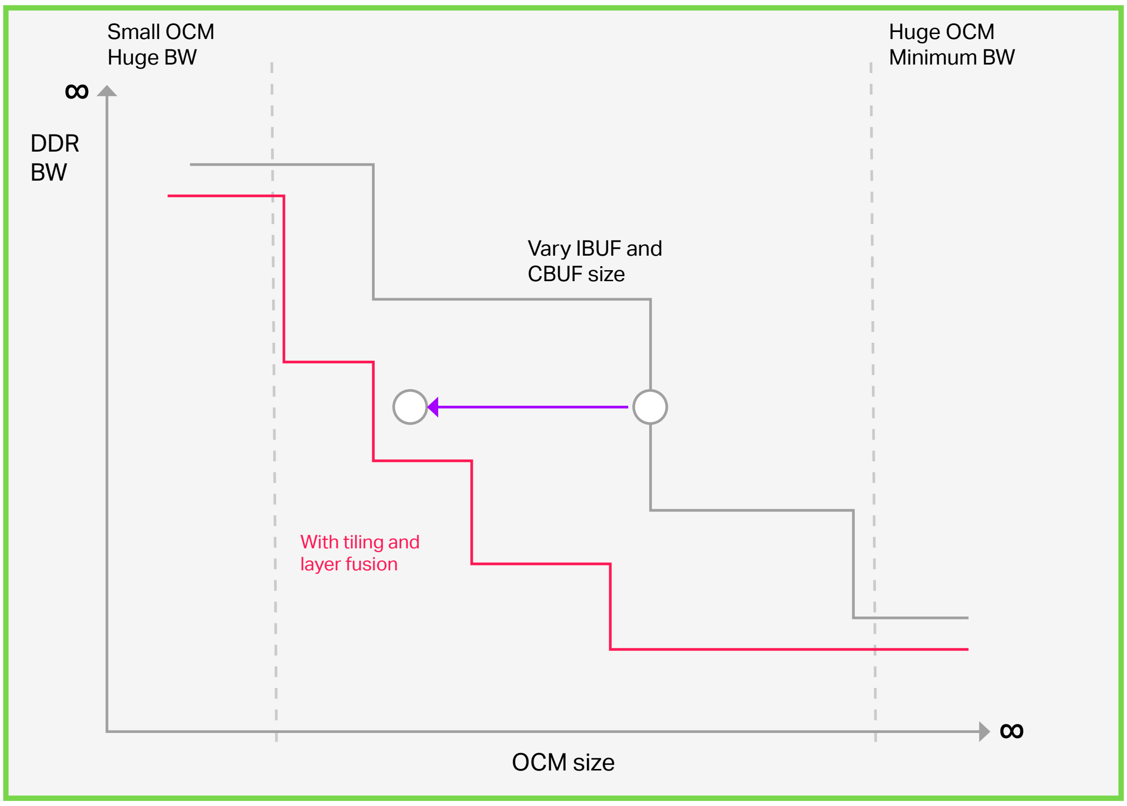


Figure 1 – Goals of Imagination Tensor Tiling and layer fusion

Imagination Tensor Tiling (ITT) efficiently packages up tensors into tiles, processed in groups, where all the intermediate data is stored in local on-chip memory – thus minimising data transfers between consecutive layers of the neural network**.**

The process described in this paper uses a combination of "*tiling*", and "*layer fusion*", where tiling divides a complete tensor into tiles for running in multiple processing passes and layer fusion merges the tiled operation from multiple layer groups together.

The combination of these two processes (termed Imagination Tensor Tiling) reduces external DDR bandwidth consumption by up to an incredible 96.25%, and perfectly complements Imagination's highly scalable and flexible IMG Series4 multi-core architecture.

So, let us take a deeper look into how we have achieved this.

The Imagination Tensor Tiling (ITT) process can be split into three phases:

Phase 1

Prediction of the tile size and subsequent tile creation

Phase 2

Creation of tile groups (TG) and allocation of on-chip-memory (OCM)

Phase 3a

Runtime tile execution (single-core)

Phase 3b

Runtime tile execution (multi-core)

Phase 1:
Prediction of the tile size and subsequent tile creation

A “layer group” is a collection of layers, grouped together, matching the order with which they are processed by the hardware. This results in a map of data relationships and dependencies described in tile-formations, which are chosen to utilise the highest overlap of reusable data for the network case being compiled for.

The ITT algorithm is based on a concept that we start with the *final* layer group configuration (the output from the network) and then propagate the confines of this configuration as a goal, backwards through the network layers, building up dependencies and tile size deltas as we approach the start of the network.

Using a single-shot cost model for predicting the tile sizes, we divide the tensor to be tiled into base and sub-tiles, where the base tile expands, but the sub-tiles remain a constant size.

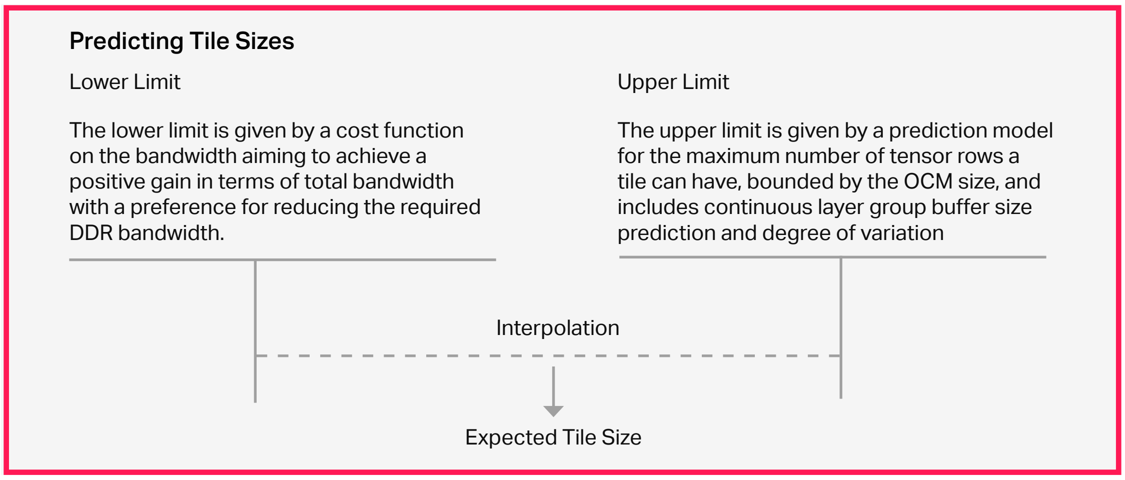

Predicting Tile Sizes
Lower Limit
The lower limit is given by a cost function on the bandwidth aiming to achieve a positive gain in terms of total bandwidth with a preference for reducing the required DDR bandwidth.
Upper Limit
The upper limit is given by a prediction model for the maximum number of tensor rows a tile can have, bounded by the OCM size, and includes continuous layer group buffer size prediction and degree of variation
Interpolation
Expected Tile Size

Figure 2 -Tile size prediction method

The tensor is tiled by lines, so each tile will always have the original tensor width. The prediction of tile sizes is based on tensor size, coefficient size, on-chip ram (OCM) size and some other pre-computed gradients. We can demonstrate this with an example of how we might tile a ResNet V1-50 deep neural network to run on our IMG Series4 neural network accelerator:

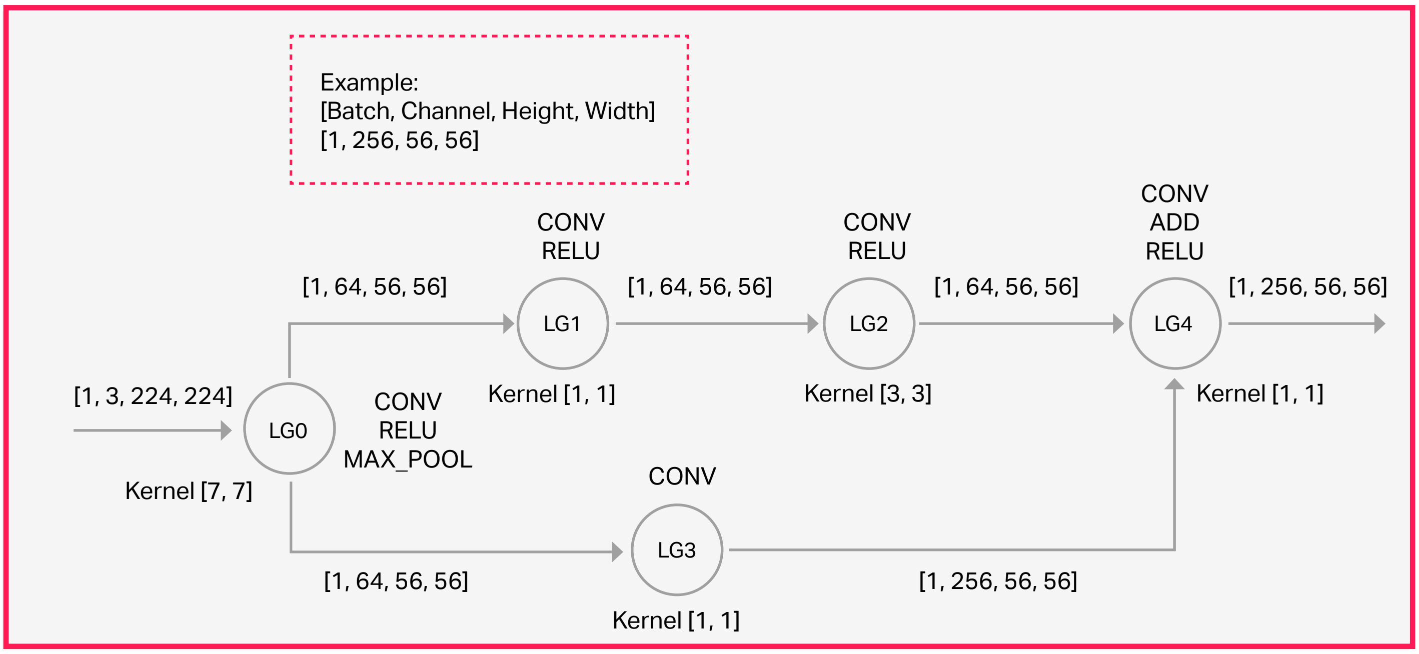


Figure 3 - Phase 1: ResNetV1-50 layer groups

First, the algorithm would take layer group 4 (LG4 shown in the diagram above), which is at the end of the layer processing and has a tensor shape (batch, channel, height and width) of [1, 256, 56, 56], and subsequently create a tile arrangement comprising a base tile LG4#0 and sub-tiles LG4#1-LG4#5.

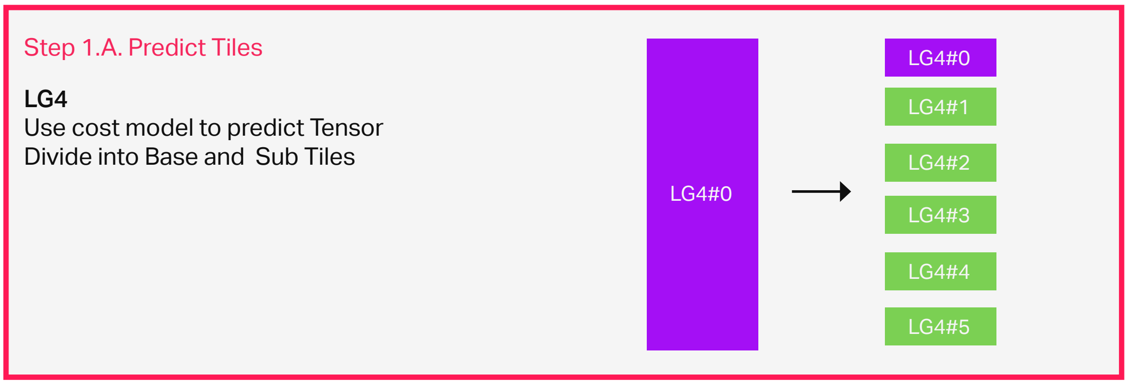

Step 1.A. Predict Tiles
LG4
Use cost model to predict Tensor
Divide into Base and Sub Tiles
LG4#0
LG4#0
LG4#1
LG4#2
LG4#3
LG4#4
LG4#5

Figure 4 - Phase 1: Predict tile size and delineate tile

LG4 needs to have 56 lines split into tiles, which for this example might be 5 sub tiles of 10 lines and 1 base tile of 9 lines. The tile sizes would depend on the output of the single-shot cost model shown in figure 1.

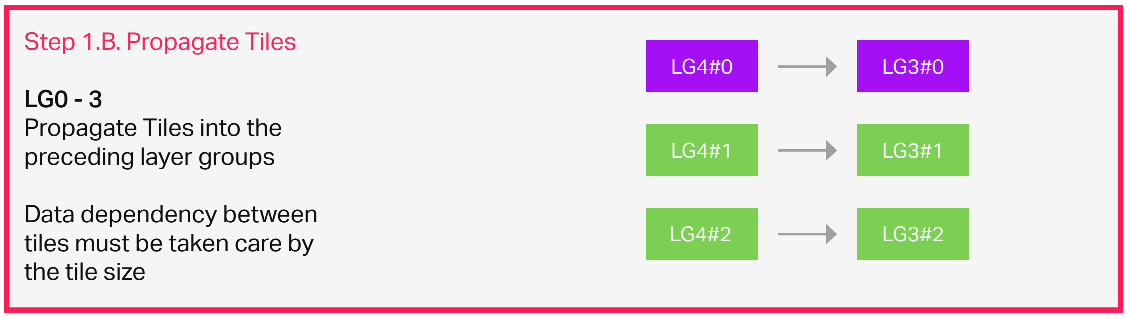

Step 1.B. Propagate Tiles
LG0 - 3
Propagate Tiles into the preceding layer groups
Data dependency between tiles must be taken care by the tile size
LG4#0
LG3#0
LG4#1
LG3#1
LG4#2
LG3#2

Figure 5 - Phase 1: Propagate tiles into preceding layer groups

These tiles are then propagated back-wards through the network by one tensor processing step and data dependencies are created so that for example, base tile LG4#0 has a dependency on a base tile LG3#0 and sub-tiles LG4#1 and LG#2 have dependencies on sub-tiles LG3#1 and LG3#2 respectively.

The tensor dimensions for LG3 and LG2 are still 56 lines, and the kernel size [1,1] remains unchanged between processing steps so the base tile size does not need to change for this step.

However, when we propagate LG2 back to LG1, you can see that the kernel size changes from [1,1] to [3, 3], requiring an additional line to be brought forward from LG1 to LG2.

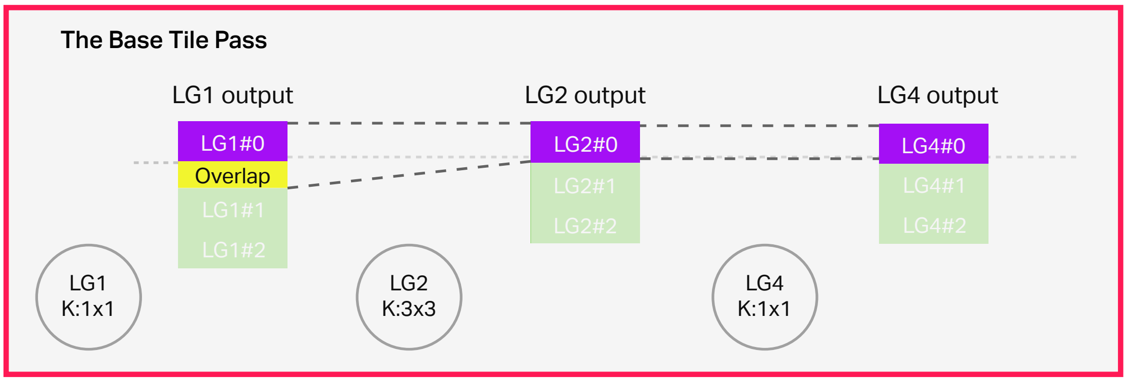


Figure 6 - Phase 1: Expanding base tile to create overlap

In response, we resize the base tile on this step from nine lines to 10 lines, resulting in a new tile configuration of five sub tiles of 10 lines, and one sub tile of 10 lines instead of 9 as per the previous step. This allows the correct line data to be accessible for processing tensor LG2, with the kernel size of [3,3] as the network topology dictates.

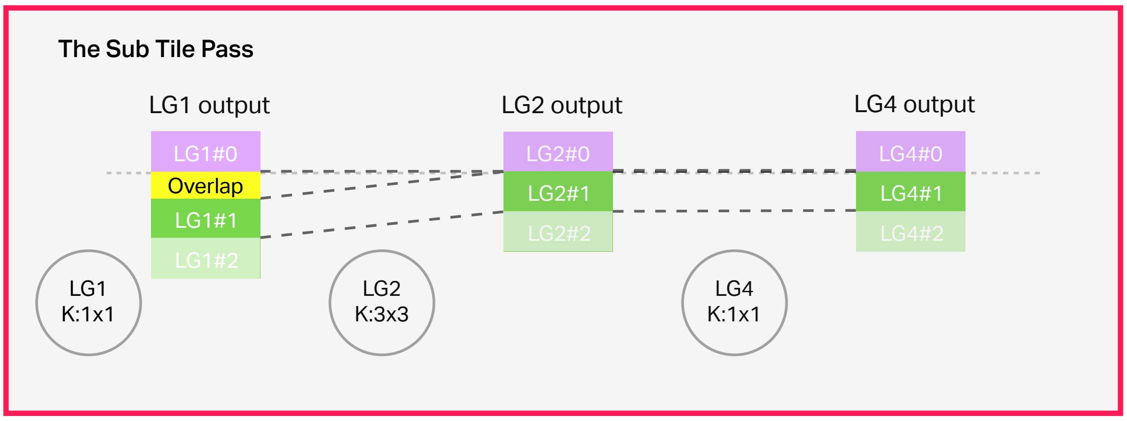


Figure 7 - Phase 1: Process continues until all layers have been processed in this layer group

This operation continues until we reach the start of the network or a tile group boundary is hit, where all tile transitions and relationships will have been mapped.

Phase 2:
Creation of tile groups (TG) and allocation of on-chip-memory (OCM)

Now that we have performed a goal-based back-propagation scan of the network, we are able to arrange the tiles that have been created into optimised *tile groups* based on tile size, current OCM space and other tile-grouping rules designed to minimise external DDR memory bandwidth, while maintaining maximum network execution performance.

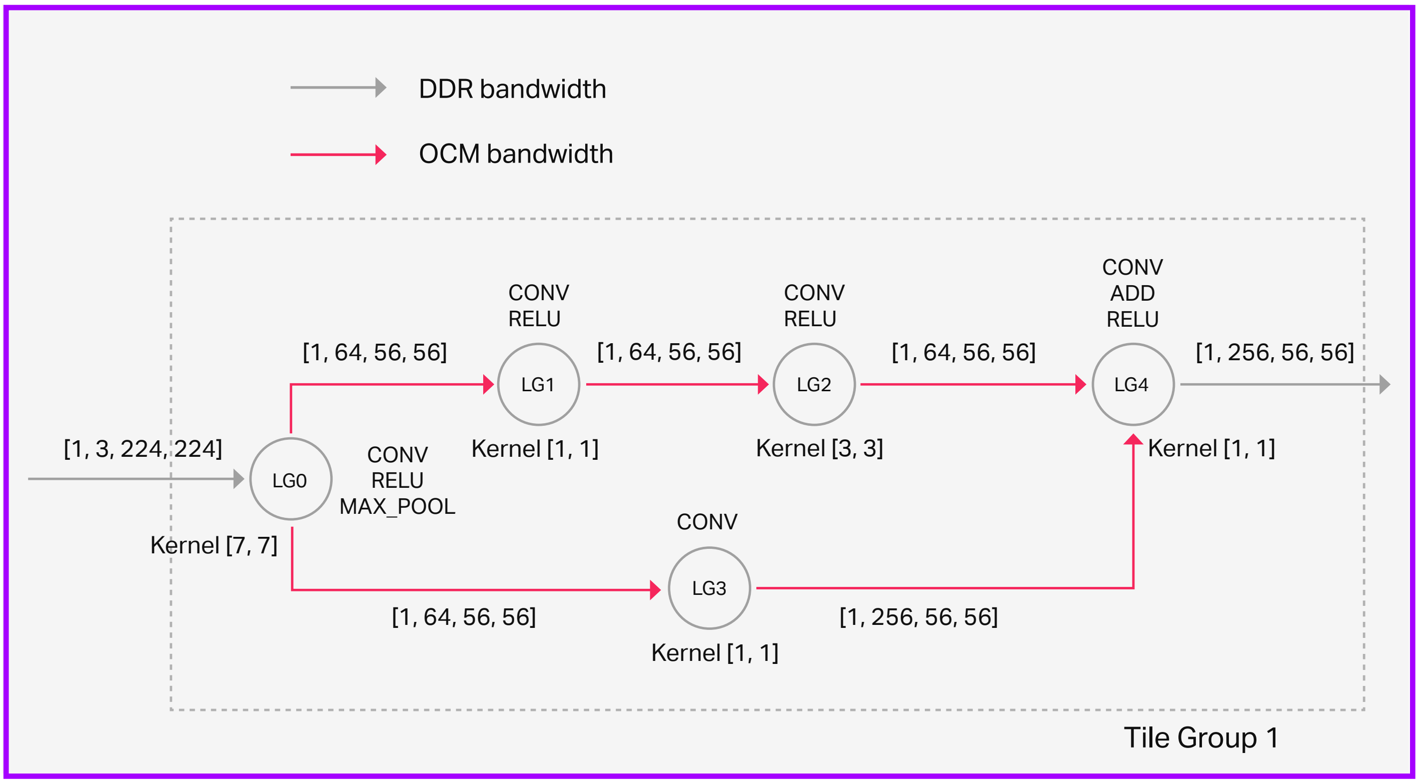


Figure 8 - Phase 2: Create tile groups and allocate OCM

We can see in this example that branches in the network flow can be fused into the same tile group; all layer groups LG0 to LG4 become one tile group – this is what we refer to as *layer fusion.*

Step 2.A. Create Tile Group

LG0 - 4
Create optimised tile groups base on the tiles size, OCM spaces and other tile grouping.

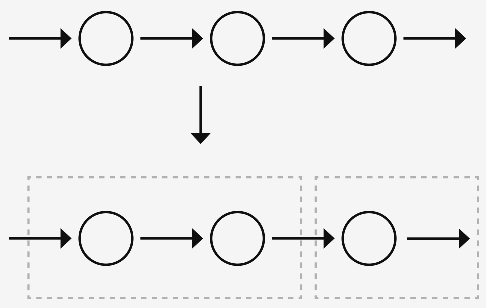


Figure 9 - Phase 2: Create optimised tile groups

As we create new tile groups, we partition and allocate the on-chip memory. Buffers are optimised based on the lifetime of the tiles and the region to which they have been allocated.

In this example, normal tiles go into a SWAP region which services short lifetime buffers, overlap tiles go into a special HEAP region and the coefficients are stored in the OCM. The resultant memory configuration is created at compile time, so completely reproducible for debug purposes.

Step 2.B. Allocate OCM

Tile Group 1
While creating tile groups, allocate the OCM spaces.

Buffers are optimised depends on the life time of each region of any tiles.

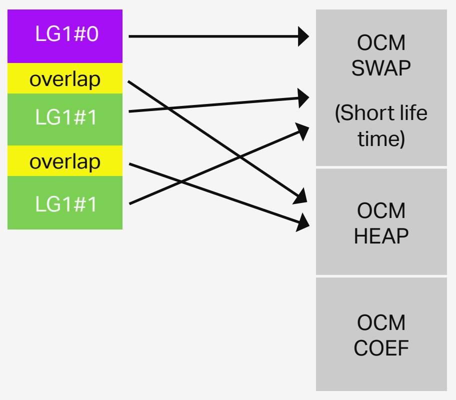
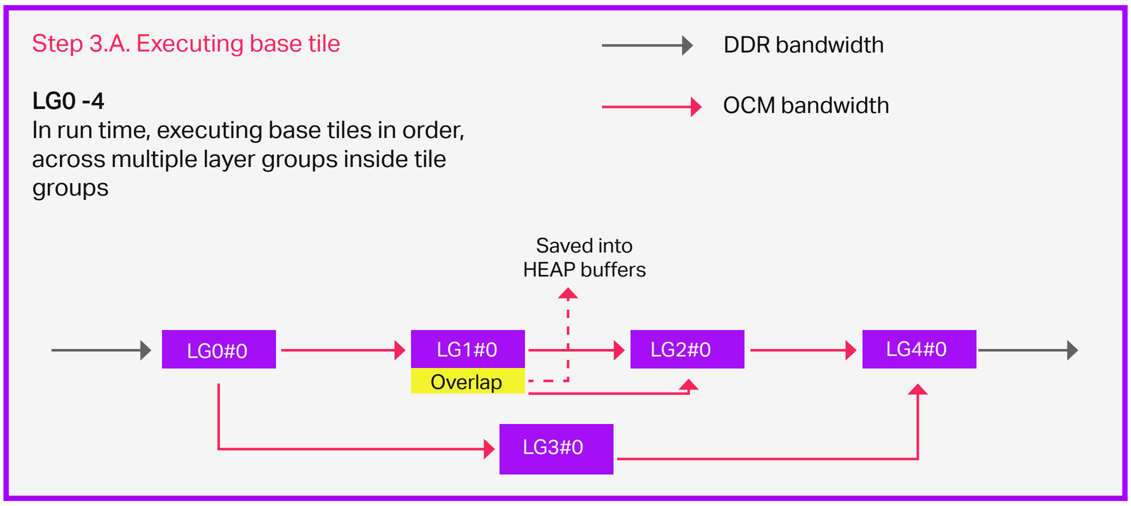


Figure 10 - Phase 2: Allocate OCM as we create optimised tile groups

Phase 3a:
Runtime tile execution (single-core)

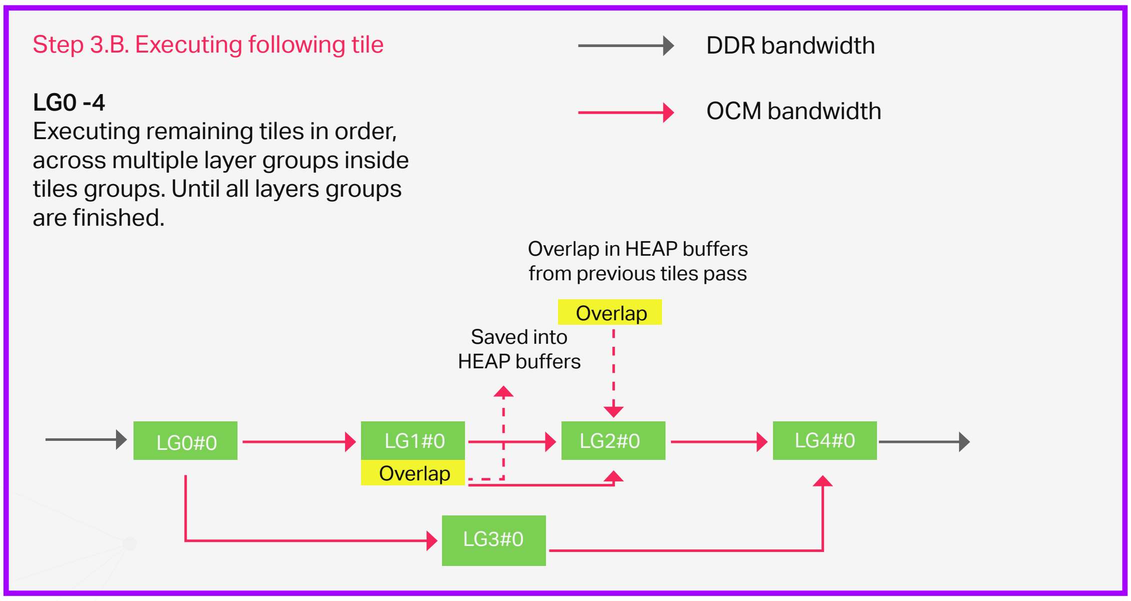
In this phase we execute all base tiles, across all the layer groups inside the tile group while keeping the overlapped data in the buffer for the next tile pass.



Step 3.A. Executing base tile
LG0 -4
In run time, executing base tiles in order, across multiple layer groups inside tile groups
DDR bandwidth
OCM bandwidth
Saved into HEAP buffers
LG0#0
LG1#0
Overlap
LG2#0
LG4#0
LG3#0

Figure 11 - Phase 3: Execute base tile

We can see that the only overlap is LG1 which comes from LG2, and has a kernel size of [3,3] which requires overlap between tiles in LG1. We then continue to execute all until all sub-tiles in all layer groups have been processed.



Step 3.B. Executing following tile
LG0 -4
Executing remaining tiles in order, across multiple layer groups inside tiles groups. Until all layers groups are finished.
DDR bandwidth
OCM bandwidth
Overlap in HEAP buffers from previous tiles pass
Overlap
Saved into HEAP buffers
LG0#0
LG1#0
Overlap
LG2#0
LG4#0
LG3#0

Figure 11 - Phase 3: Execute base tile

Phase 3b:
Runtime tile execution (multi-core)

Where a tiled network can run on a multi-core cluster such as 4NX-MC2, 4NX-MC4, 4NX-MC8 and 4NX-MC8, the tiles within each layer group get distributed across all available cores, where each core executes different parts in the same layer group.

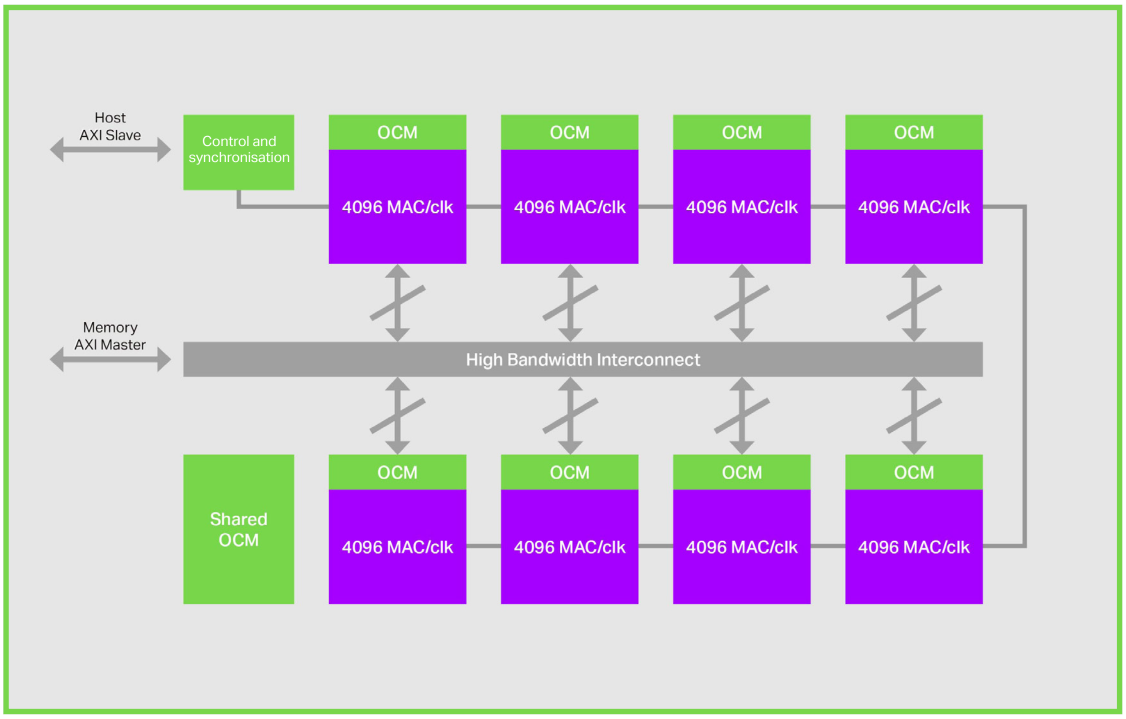


Figure 12- 4NX-MC, our scalable multicore architecture.

Additionally, each core in the cluster has its own OCM providing the most optimised performance and power consumption for the tiles being executed.

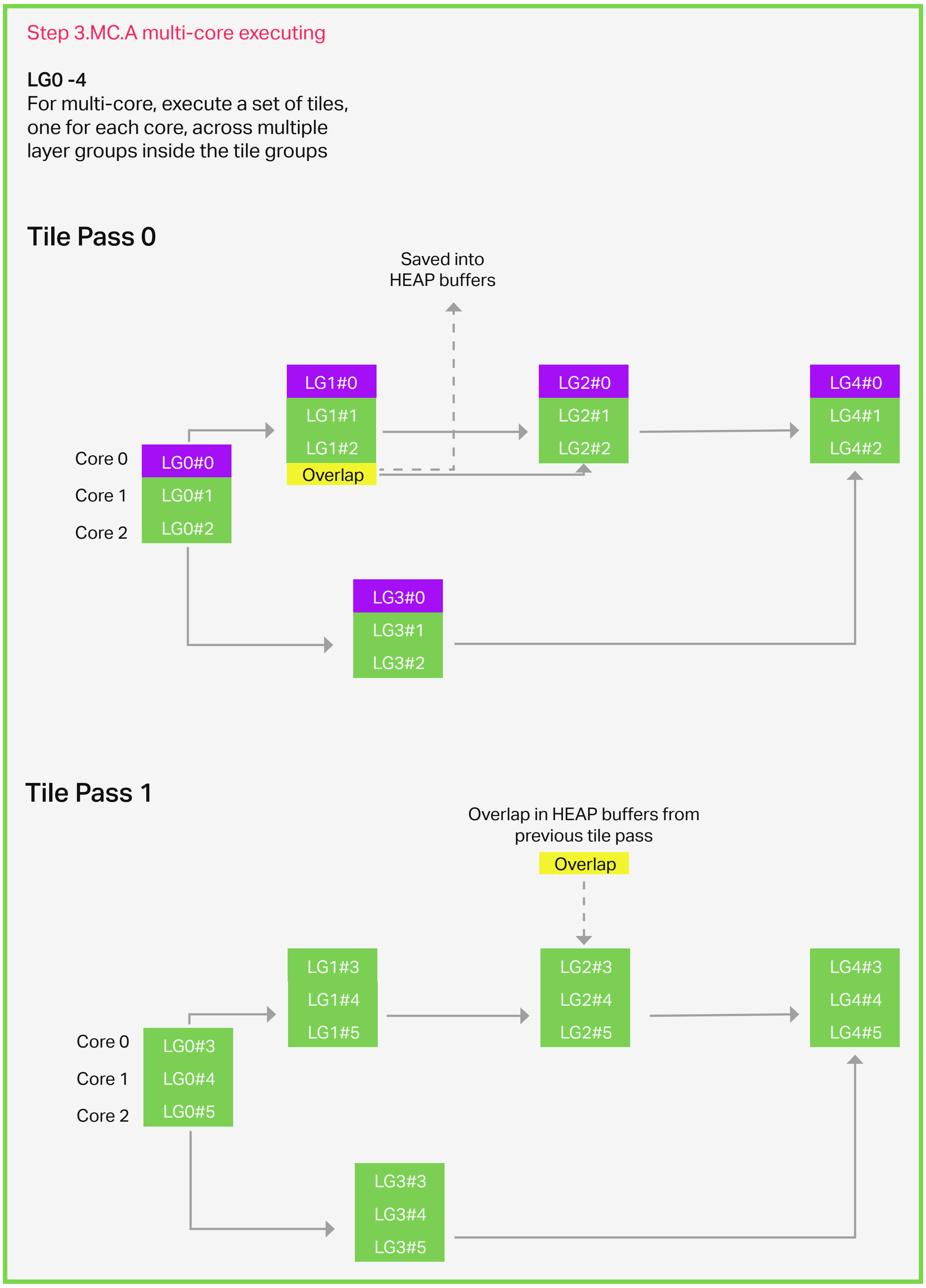


Figure 13 - Phase 3b: Execute tiles on a multi-core system

Bandwidth saving

So now that we have had a glimpse inside of Imagination's Tensor Tiling algorithm, here is the bit you have all been waiting for – ***just how much bandwidth does this Imagination's Tensor Tiling algorithm save?***

As you can see from the table below - the results are significant, hitting between 57 and 96% depending on the neural networking model.

Network	Bandwidth Savings on 4NX-MC1 (1024 OCM)
ResnetV1-50	57.60%
InceptionV3	59.58%
MobileNetV1 1.0 224	61.63%
MobileNetV1 (1080p)	52.26%
SRCNN (AI Benchmark v3.0.2)	96.25%
SSD-MobileNetV1-FPN (COCO)	41.10%
YOLOV3 (COCO)	62.79%

2MB OCM, 8-bit data – percentage of original using ITT in brackets

Unsurprisingly, the ADAS and autonomous driving compute requirements trend is upwards with strong indications that these requirements will increase by an order of magnitude within the next 3-5 years. If you'd like to find out more about our IMG Series4 NNA with ***Imagination Tensor Tiling*** then feel free to get in touch.

For more information visit:
https://www.imaginationtech.com/vision-ai/img-series4-nna/

Contact us:
https://www.imaginationtech.com/contact-us/

Imagination

www.imaginationtech.com

enquiries@imgtec.com
UK t: +44 1923 260511
US t: +1 408 530 5000