2D convolution algorithm and system deployment based on AI Engine multi-core parallelism

ZhenDong Zheng zzd1411@mail.ustc.edu.cn
QianYu Cheng qycheng@mail.ustc.edu.cn
BinZe Jiang jiangbinze@mail.ustc.edu.cn
University of Science and Technology of China

1 Algorithms and System Design

1.1 Background technology and hardware environment

2D convolution is an operation on matrix data and is widely used in convolutional neural networks commonly used in traditional image processing and AI fields. In the 2D convolution layer, the convolution kernel is a small-scale 2D weight matrix. Its working mechanism is to ”slide” the convolution kernel on the 2D input data, and perform matrix multiplication on some elements of the current input. The results are then pooled into a single output pixel and combined into the convolution result. The convolution kernel is a weighted summation of a certain part, which corresponds to the local perception of the global image. Its principle is that when observing an object, we can neither observe each pixel nor the whole at one time, but start from the local Get to know each other.

Due to the calculation and memory access characteristics of the 2D convolution algorithm, the algorithm cannot achieve good performance on the general-purpose processor platform. The heterogeneous computing platform represented by GPU and FPGA can bring better acceleration energy efficiency to this type of algorithm. Compare. This design uses the hardware resources provided by the Xilinx VCK5000 Versal PCIe accelerator card to efficiently implement the two-dimensional convolution algorithm.

Traditionally, scalar processors (such as CPUs) are very efficient in complex algorithms with different decision trees and extensive libraries, but are limited in performance scaling; The aggregate efficiency is higher, but affected by the storage structure, there is a higher delay and a lower energy efficiency ratio. Programmable logic such as FPGAs, which can be precisely tailored to specific computing functions, perform best in real-time applications and irregular data processing, but associated algorithm changes introduce significant compile time.

VCK5000 provides ARM Cortex-A72, FPGA, DSP, AI Engine and other resources, and supports scalar, vector computing and custom computing. These computing engines can be customized and interconnected through a high-speed on-chip network. Among them, AI Engine is a vector processor that supports very

long instruction word (VLIW), single instruction multiple data (SIMD), and multi-core processing. As a hard-deployed processor core, users can directly write data-level parallelism for AI Engine. The kernel program directly transfers data from the board's DDR4 memory to the AI Engine for calculation through the programmable logic (PL) side. Compared with FPGA, AI Engine does not require complex hardware considerations, data parallel programming is simpler, and is suitable for adaptive AI inference applications. VCK5000 provides an array of 400 AI Engines for users to use.

1.2 Host-Board Interaction Platform

Based on the VCK5000 board in the X86-64 host environment, this design provides a platform that can run 2D convolution algorithms for 4K (3840*2160) resolution images in batches. As shown in **Figure 1**, the specific execution process is as follows:

1. The host sends the image data to the DDR4 memory input buffer reserved by the board;
2. Execute the data distribution kernel on the PL side of the board, split the image data into multiple 64×32 blocks, and distribute the data to several AXI-Stream interfaces in sequence to pass the data to the corresponding AI Engine core;
3. The AI Engine array on the board receives the data and performs a two-dimensional convolution operation on the 64×32 image;
4. Execute the data reorganization kernel on the PL side of the board, retrieve part of the output of each AI Engine processing block through the AXI-Stream interface, reorganize the overlapping data between processing blocks into a complete image, and write it into the output buffer;
5. The host side retrieves the convolution result data from the reserved DDR4 memory output buffer.

1.3 AI Engine 2D convolution kernel

The AI Engine is mainly modified with reference to the filter2d implementation in the Vitis Vision Library to process the int type matrix convolution with a fixed size of 64×32. When processing, consider padding for the number of blocks in advance, copy the data according to the following rules, and then perform convolution:

1. padding The data at the four corners of the subsequent matrix are the same as the data at the four corners of the original matrix;
2. padding The data in the first column/row of the matrix are the same as the data in the first column/row of the original matrix; The data in the last column/row of the matrix after
3. padding is the same as the data in the last column/row of the original matrix.

Since AI Engine supports SIMD programming using intrinsics, the calculation statements in the convolution kernel can implement multiplication and accumulation operations of multiple numbers at one time, reducing the computational overhead of convolution operations.

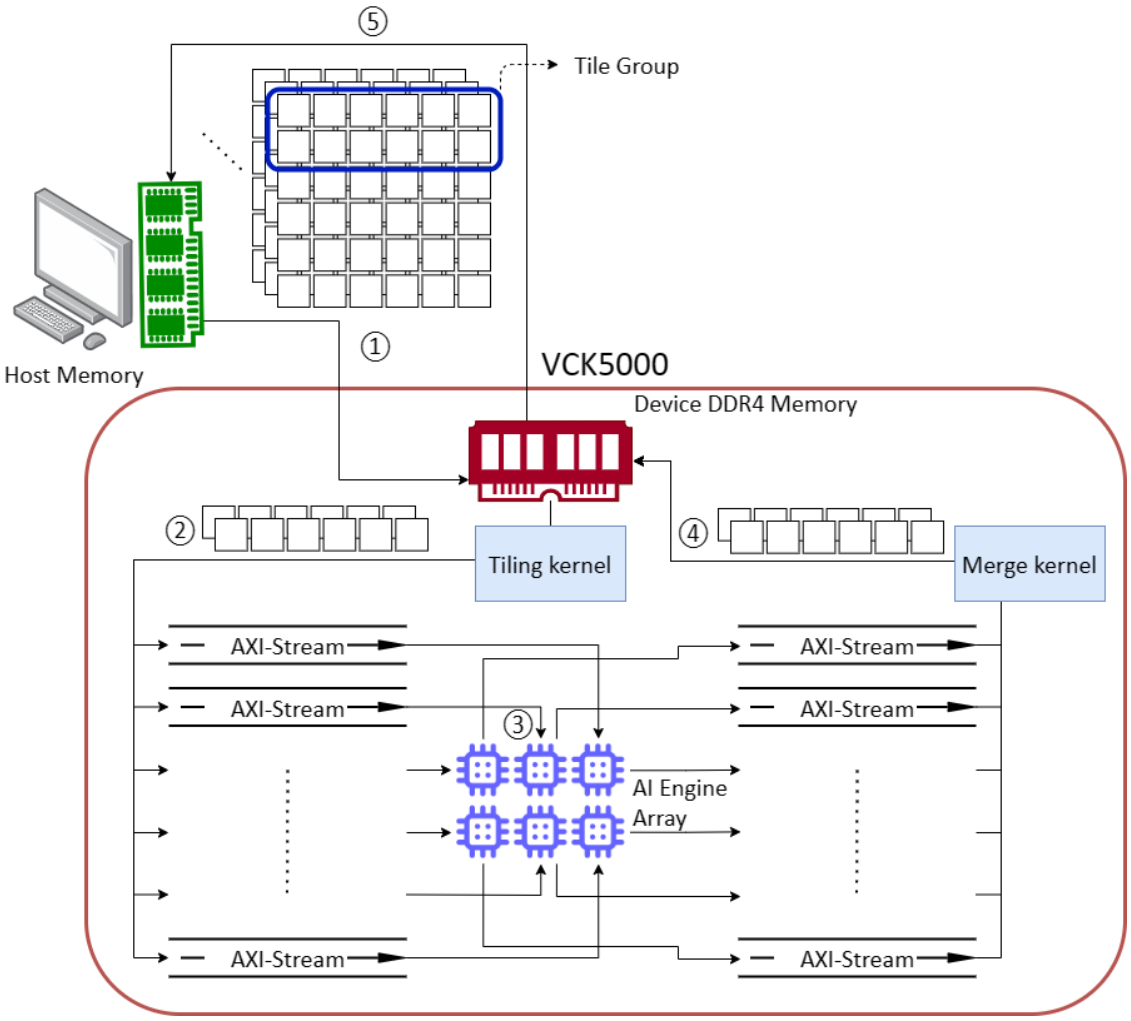

Figure 1: Overall platform framework

1.4 Data distribution and reorganization

1.4.1 Data Distribution Kernel

In order to enable the system to support image input of any scale, we use 64×32 as the block size to process the input data transmitted from the DDR4 memory of the board from top to bottom and from left to right. In order to realize the parallel processing of multiple AI Engines, we organize the blocks sequentially in groups of 7 blocks. In one round of data transmission, each block in a group of block data is transferred through multiple AXI-Stream channels They are respectively transmitted to the corresponding AI Engines to perform convolution operations, and after the calculations are completed, the execution results of each AI Engine are reorganized.

Due to the data multiplexing characteristics of the 2D convolution operation, if the block data with no overlap between the two blocks is simply passed in and convolved, the complete result cannot be obtained by combining the convolution output of each block . To this end, we designed a PL-side data distribution kernel

with a more complex input block mechanism, so that each block passed into the AI Engine has a width of 2 with the upper, lower, left, and right blocks of the input data. overlapping. Therefore, the PL end can restore the complete convolution output through the output of the AI Engine. The block diagram of the complete image is shown in **Figure 2**.

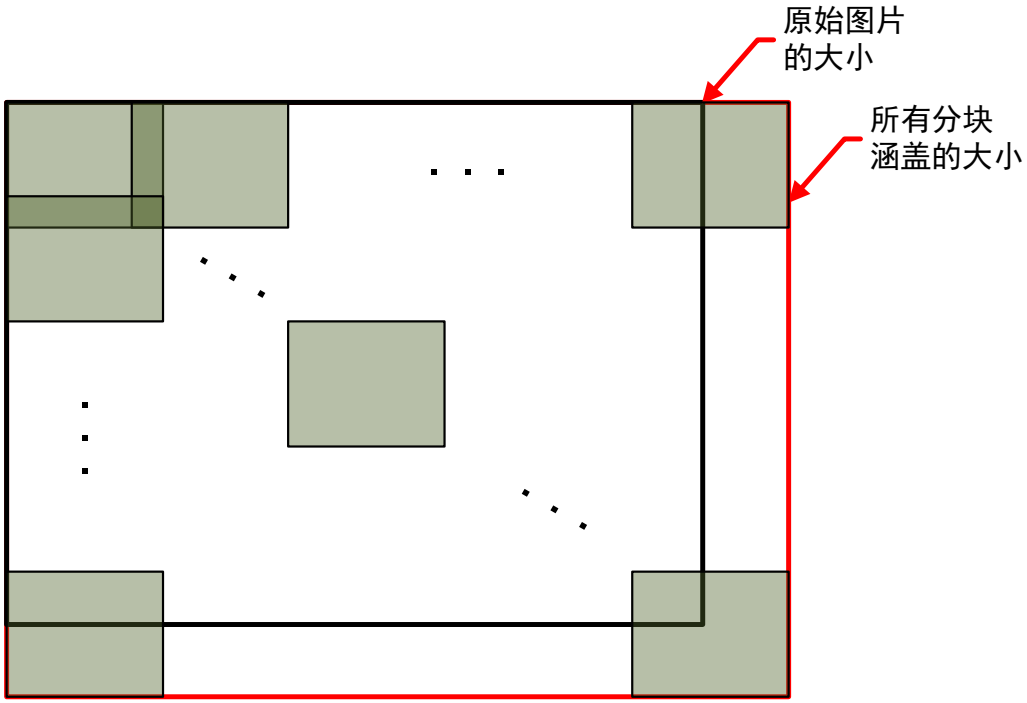

Figure 2: Data block layout

It is worth noting that for the data block of the last row or column, due to the preset convolution padding rules, the data distribution kernel needs to copy the valid data of the last row or column in the block to the next row or next column. In addition, the part without valid data in the 64×32 block is all filled with 0.

1.4.2 Data reorganization kernel

Since the AI Engine will pre-padding and then perform convolution on each block of data received, part of the results of block convolution need to be discarded during reorganization. To this end, we designed a special PL-side data reorganization kernel behind the AI Engine array, combining the output of each block according to its order in the original picture, and will be located in the upper left corner, lower left corner, upper right corner, lower right corner, The blocks in the first row, the last row, and other blocks are classified and processed. Similarly, during the execution of the reorganized kernel, data is also transmitted through the 256-bit data width memory output interface. The specific rules for classification processing are as follows:

For the block located in the upper left corner, the convolution result of the last column and row is wrong

(except for special cases where the full image size is exactly 64×32). As shown in **Figure 3**(1), the pink area represents the data obtained after padding by AIE, and the red area represents the wrong result. The reorganization kernel directly writes the block convolution results into the final output, and the erroneous results are overwritten by the subsequently written block output. When the block results are spliced from left to right and top to bottom, subsequent blocks will overwrite the erroneous data with the correct results.

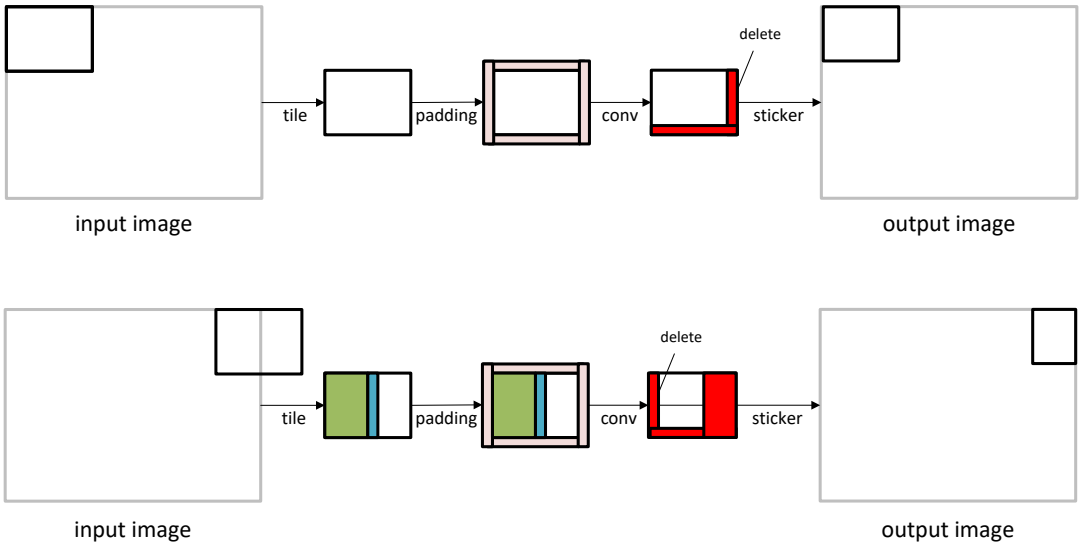

Figure 3: invalid data processing

For the block in the upper right corner, the convolution result of the first column and the last row is wrong. In addition, a part of the input on the right side of this block does not belong to the original valid data, as shown in **Figure 3**(2), the data in the green area belongs to the original valid data, and the data in the blue area is the original valid data Duplication of the last column. The reorganization kernel directly writes the data in the block convolution result that does not exceed the original image area except the first column to the final output, because subsequent blocks will still overwrite the wrong data in the last line with the correct result. Blocks in the lower left and lower right corners are processed according to similar rules.

In addition to the above cases, the convolution results of the first row, last row, and last column of the block located in the first column are wrong. In the same way as above, the reorganized kernel directly writes the data except the first row in the block convolution result to the final output. Chunking of the first row is handled according to similar rules. For the block located in the last column, the convolution results of the first row, last row, and first column are wrong. Similar to the situation in the upper right corner, a part of the input on the right side of this block does not belong to the original valid data. The reorganized kernel directly writes the data in the block convolution result that does not exceed the original image area except the first row and the first column to the final output. The chunking of the last line is handled according to similar rules.

For other blocks that are not at the edge, the convolution results of the first row and the last row are wrong. The reorganized kernel directly writes the data except the first row and the first column in the block convolution result to the final output.

2 Experimental analysis and prospect

The experiment uses the Heterogeneous Accelerated Computing Cluster (HACC) provided by Xilinx, which provides a 16-core Intel Xeon Gold 6246R X86-64 processor, 16GB of memory, and a VCK5000 board platform that supports the PCIe Gen4x8 protocol and QDMA. The board ultimately runs the board platform hardware at 300MHz.

The platform finally deployed 7 AI Engines and corresponding AXI-Stream input and output channels. At the same time, a 256-bit data-width port was deployed for the board DDR4 memory to the PL side core, effectively utilizing the DDR4 storage bandwidth.

The AIE compilation result is shown in **Figure 5**, blue represents AI Engine, and purple represents cache. The implementation of two-dimensional convolution consumes 17 blocks in the AIE array in total, 7 blocks for AI Engine calculation, 12 blocks for caching, and 17 blocks for streaming.

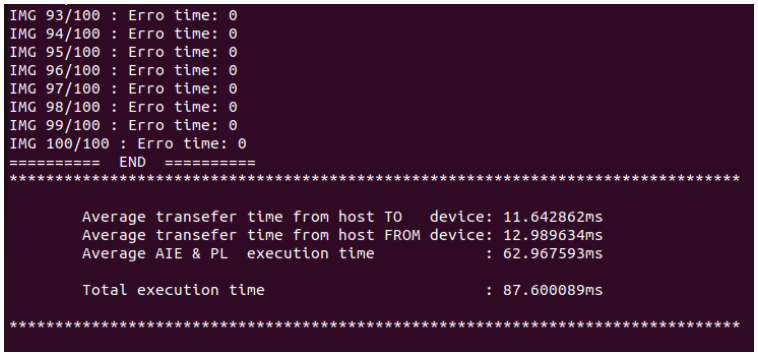

Figure 4: Run result

We used randomly generated 32-bit matrix data to test the platform, and processed 100 images in batches. Experimental data shows that the average time for the platform to process a single piece of 4K resolution data is 87ms, and the processing frame rate is 11.42fps. Among them, the delay of data transmission and transmission is 24.63ms, which is close to half of the total time, indicating that the calculation overhead does not affect the overall performance of the system, and the bottleneck of running the two-dimensional convolution algorithm is mainly in memory access.

There are still many problems with the current design. First of all, for the current design, the two tasks of fetching data from the memory port and sending data to the AXI-Stream port are executed serially. In the future, double buffering technology can be used to make memory port access and AXI-Stream data Sends can be performed concurrently, covering a certain memory access latency. Secondly, due to the current limitation

of the QDMA platform, two-dimensional convolution cannot target the allocation of storage banks for DDR4 cache data connected to multiple PL core ports to achieve more detailed memory parallel transmission. In the future, the platform design can be further improved. Optimization. We hope to further iterate the design in the future to maximize the use of board bandwidth.

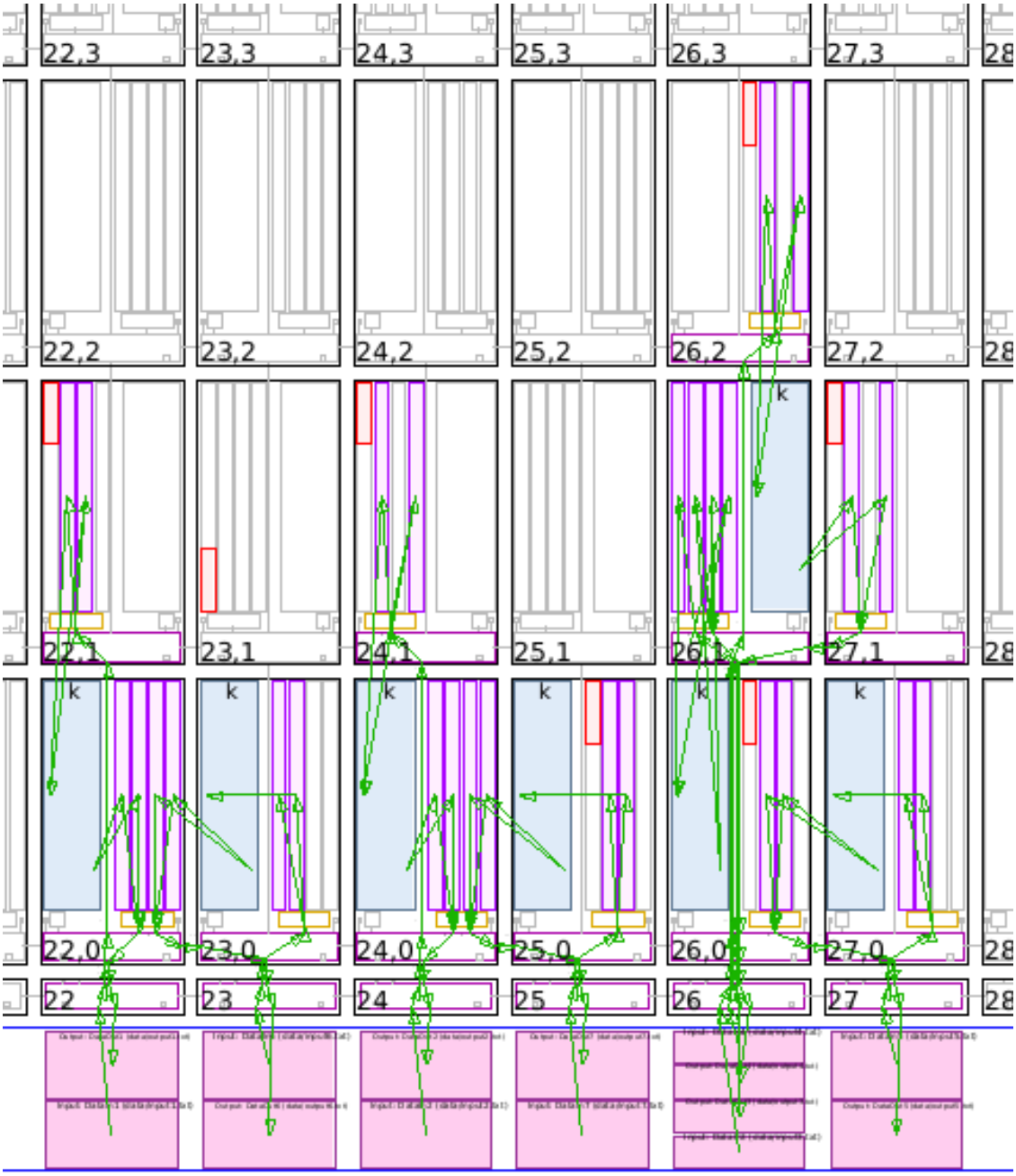

Figure 5: AI Engine array deployment layout