Kernel-Based Placement for CGRA

Keyi Zhang and Mark Horowitz
Computer Science Department, Stanford University

Introduction

Existing problems in CGRA placement:

1. Traditional FPGA/CGRA placement algorithms are sequential because it iteratively refines the placement result.
2. Partition, a necessary step to achieve speed up through parallelism, usually yields sub-optimal solutions.
3. Even parallel placement suffers Amdahl's Law very early on: only a minor performance improvement increasing CPU from 8 cores to 16 cores.

Solutions:

1. Use high level information, such as kernel, from Halide can be used to partition and achieve significant speed up.
2. The new placement algorithm are able to scale in a cloud scale.

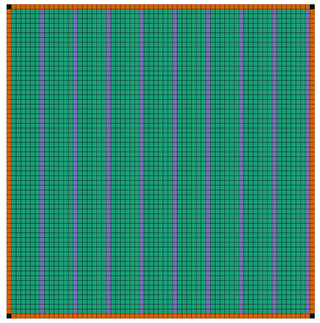

Fig. 1: Excample CGRA Design with 4K tiles, including both PE and MEM tiles

Computation Kernel for Image Processing

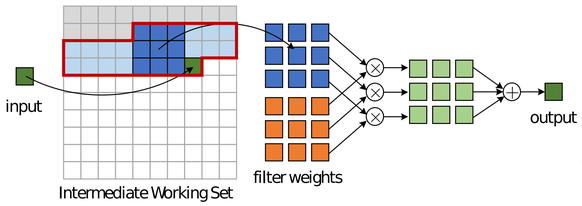

Fig. 2: Line buffer for a 3×3 convolution.

1. A lot of image processing applications, accelerated by hardware, such as convolutions, are built on the concept of **line buffers**, as shown in Figure 2.
2. The application can be viewed as a pipeline of stages computing different intermediate images by line buffers.
3. Once compiled to hardware dataflow, we call these strongly connected subgraph around linebuffers **kernel**, as shown in Figure 3.

Kernel Extraction Algorithm:

1. Topological sort the graph (a DAG)
2. Identifies the "core" of the kernels, typically it is the line buffer, where the kernel computation starts. These nodes serve as end-points.
3. Traverse through the list and mark any nodes between two end-points will be identified as a kernel. An example is shown in Figure 3 and 4.

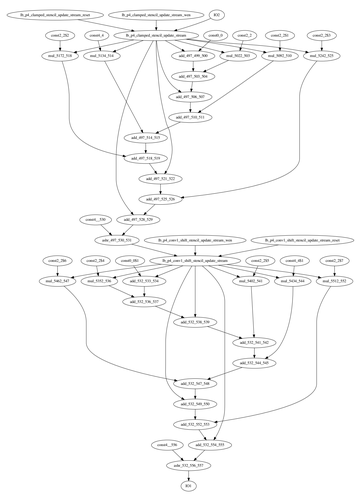

Fig. 3: Dataflow for two 3×3 convolutions (cascade)

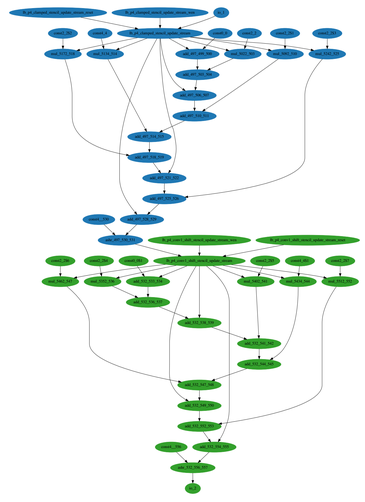

Fig. 4: Kernel extracted for two 3×3 convolutions (cascade)

Force-Directed Inter-Kernel Placement

After obtaining the partiton using kernels, we apply force-directed approach on these kernel-based clusters. The force $\vec{F}_i$ for cluster i is defined as:

$$\vec{F}_i = < \sum_{j=1}^{n} k_n \frac{y_i - x_j}{d_{ij}} d_{ij}^2 - \sum_{j=1}^{n} k_s \frac{x_i - x_j}{d_{ij}} S_{ij}^2, \sum_{j=1}^{n} k_n \frac{y_i - y_j}{d_{ij}} d_{ij}^2 - \sum_{j=1}^{n} k_s \frac{y_i - y_j}{d_{ij}} S_{ij}^2 >,$$

where k_n is the force constant for inter-kernel cluster connections, d_{ij} is the distance between cluster i and cluster j, k_s is the force constant for overlapping, and S_{ij} is the overlapped area between these two clusters. Notice that k_s is not a constant as if two clusters have no overlapping area, the overlapping force is zero. Hence $\vec{F}$ is not differentiable everywhere. One can approximate the overlapping force with $f_s(d_{ij}) = e^{d_{ij} - C_{ij}}$, where C_{ij} is determined by the distance between two clusters. Although we can use Jacob matrix $\vec{J}$ to derive the closed form solution with the approximation, we find that directly simulate the force will achieve very fast the decent results, as shown in Figure 5. Difference between other force-directed or partition based algorithm:

1. We place a cluster of hundreds of cells, hence reduce the problem size significantly.
2. There is no spreading force applied globally. The "spreading force" here is the overlapping force that pushes each cluster away from each other.
3. Each cluster doesn't have to be equal-sized.

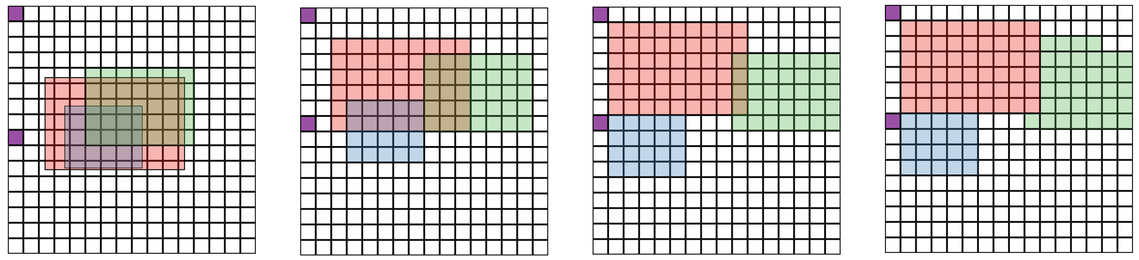

Fig. 5: Forced-directed inter-kernel placement illustration

After the force-directed placement, we apply a quick low-temperature simulated annealing with much bigger k_s to de-overlap. Then each physical cell is assigned to the cluster while further de-overlapping.

Intra-Kernel Placement

We use simulated annealing for intra-kernel placement because it works very well with non-rectangular shape, because of the de-overlapping phase in the inter-kernel placement. Because each kernel's cells are fixed on the board by the previous placement, if we use the centroid as a approximation when compute HPWL, we have an embarrassingly parallel workload, as illustrated in Figure 6.

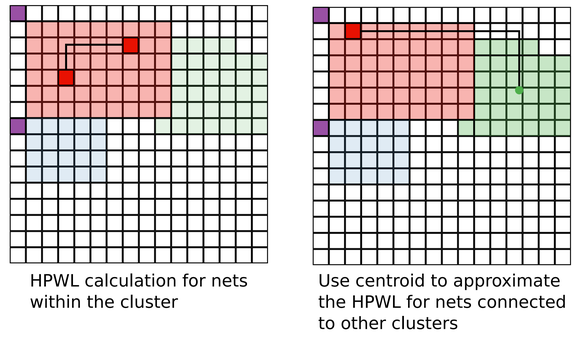

Fig. 6: Forced-directed inter-kernel placement illustration