Fused-Tiled Layers: Minimizing Data Movement on RISC-V SoCs with Software-Managed Caches

Victor J.B. Jung[†], Alessio Burrello[‡], Francesco Conti[*] and Luca Benini[†*]

[†]Integrated Systems Laboratory (IIS), ETH Zürich
[*]Department of Electrical, Electronic and Information Engineering, University of Bologna
[‡] Interuniversity Department of Regional and Urban Studies and Planning, Politecnico di Torino

Abstract

The success of Deep Neural Networks (DNNs) and their high computational requirements pushed for large codesign efforts aiming at DNN acceleration. Since DNNs can be represented as static computational graphs, static memory allocation and tiling are two crucial optimizations. Hence, System-on-chips (SoCs) specialized for DNN acceleration commonly features a multi-level software-managed memory hierarchy. In such architecture, layer-wise tiling, i.e., splitting each layer into multiple sub-nodes, is commonly used; however, while reducing memory occupation, it can increase the total memory transfer, ultimately causing costly off-chip memory copies, which impact energy efficiency and create memory bottlenecks. This work proposes Fused-Tiled Layers (FTL), a novel algorithm for automatic fusion between tiled layers. We leverage the flexibility and efficiency of a RISC-V (RV32) heterogeneous SoC to integrate FTL in an open-source deployment framework, which we tune for RISC-V targets. We demonstrate that FTL brings up to 60.1% runtime reduction for a typical Multi-Layer Perceptron (MLP) stage of Vision Transformers (ViTs) due to the reduction of off-chip transfer and on-chip data movement by 47.1 %.

Introduction

With the increased interest in efficient execution of DNNs at the edge, hardware architectures and DNN compilers have been proposed. While ARM-based architectures have a high level of maturity in both hardware and software ecosystems that allows for optimized execution of modern DNN models, the RISC-V ecosystem has proliferated in terms of novel hardware architectures, but there is room for improvement in terms of software support.

Specifically, while popular DNN compilers such as CubeAI[1], TVM [1], or IREE[2] present streamlined support for ARM-based SoCs, important features of modern RISC-V SoCs, such as multi-level memory hierarchy, are not considered. In particular, layer fusion is a well-known technique performed by DNN compilers to avoid materializing intermediate tensors that increase memory footprint and bandwidth. In RISC-V-based platforms with multi-level memory hierarchy, it is particularly critical to avoid materializing huge intermediate activations in Last Level Cache (LLC) memory. Therefore, this paper presents a new flexible algorithm tailored to RISC-V SoCs with a multi-level memory hierarchy, *Fused-Tiled Layers (FTL)*, to minimize memory transactions by fusing a series of layers:

1. FTL formulates the tiling of each DNN layer as a constraint optimization problem, where each output tensor dimension is linked to input tensor dimensions via a linear transformation, allowing us to merge several layers to generate valid layer fusion solutions for any layer combination. By doing so, we minimize transfers from L2 memory to LLC.
2. We benchmark FTL with a ViT's MLP on a reduced version of the heterogeneous RISC-V SoC Siracusa [2], with, and without the Neural Processing Unit (NPU). Compared to the layer-per-layer tiling strategy, we demonstrate a runtime reduction of 28.8 % when only using the 8-cores cluster of RISC-V cores and 60.1 % when using the cluster and the NPU.

Methodology

We integrate FTL into Deeploy[3], an open-source bottom-up DNN deployment framework that generates optimized bare-metal C requiring minimal runtime support. We rely on kernels using the extended *RV32IMCF-XpulpV2* Instruction Set Architecture (ISA) featuring hardware loops, post-increment load-store, and Single Instruction Multiple Data (SIMD) instructions. To move tiles of tensors across the memory hierarchy, we use Direct Memory Access (DMA) engines that rely on the flexibility of RISC-V systems to perform 3D transfers. Fig 1 provides a visual representation of the different steps (numbered from 1 to 4) of FTL. In step ①, we attribute a variable for each tensor dimension related to the given operator. Then, we formulate the constraints for the tiling of the single operator in step ②. There are three kinds of constraints: the *geometrical constraints* describe the

*This work has received funding from the Swiss State Secretariat for Education, Research, and Innovation (SERI) under the SwissChips initiative.

[1] `https://stm32ai.st.com/stm32-cube-ai/`

[2] `https://github.com/iree-org/iree`

[3] `https://github.com/pulp-platform/Deeploy`

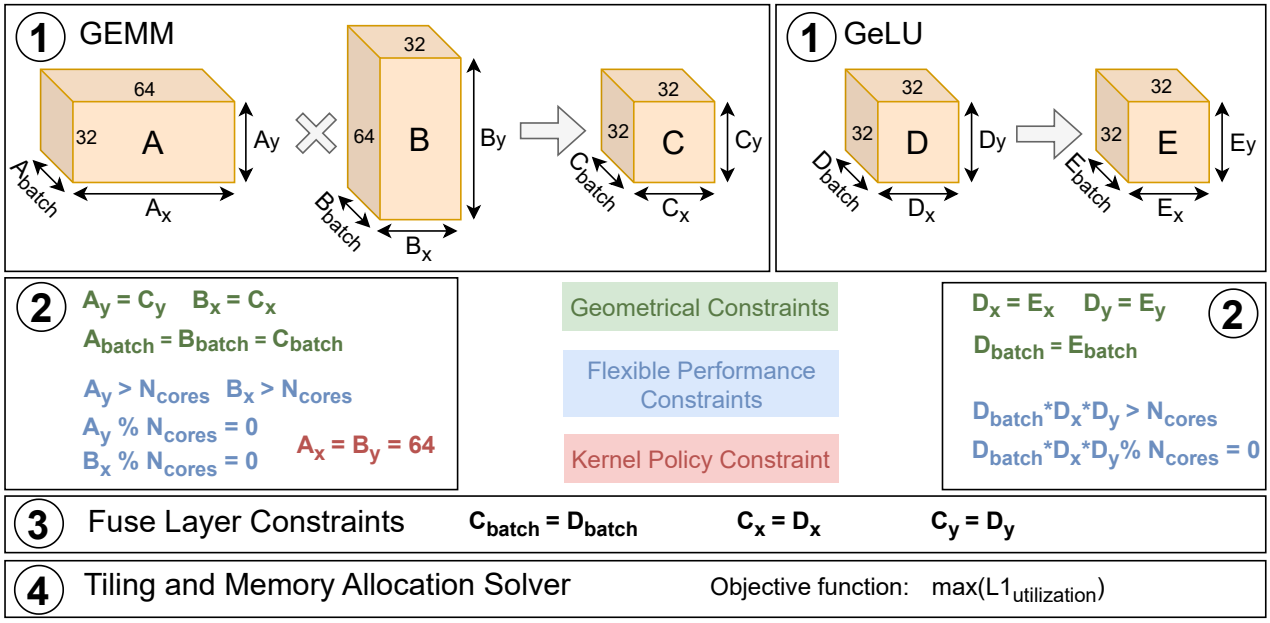

Figure 1: *Overview of the FTL on a General Matrix Multiplication (GEMM) and GeLU layer.*

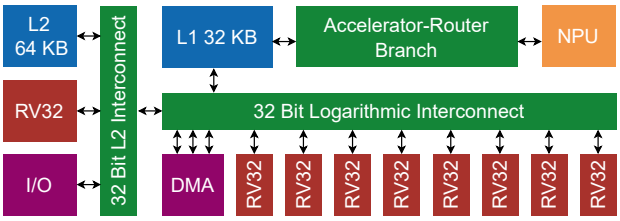

Figure 2: *Overview of the modified Siracusa SoC.*

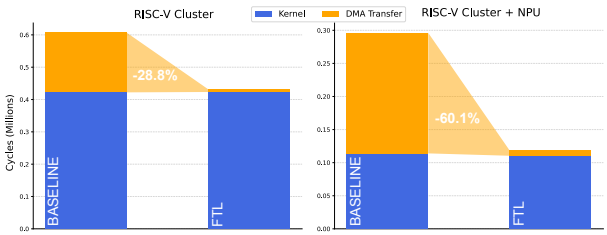

Figure 3: *Runtime comparison of ViT's MLP using layer-per-layer tiling (baseline) and FTL on the Siracusa SoC.*

data dependency between the dimensions of the output and input tensors. The *kernel policy constraints* are specific to the kernel's dataflow; Finally, we add *flexible performance constraints* to boost the hardware utilization. In step ③, we select the consecutive layers to fuse and bind the variable of their shared tensors dimension. Finally, in step ④, we solve the constraint optimization problem representing the fused layers with Deeploy's tiling and memory allocation solver.

Results

Evaluation Setup

We perform our benchmark on a reduced version of the RISC-V Siracusa [2] SoC; its architecture is described in Fig 2. The 8 RISC-V cores are using the *RV32IMCF-XPulpV2* ISA tailored to Digital Signal Processing (DSP) tasks, and the NPU is targeting GEMM and convolution. We use the GVSoC[4] event-based simulator to measure the runtime, which provides fast and accurate simulation with an error typically below 10 %.

ViT's MLP benchmark

To showcase the benefits of FTL, we benchmark a GEMM followed by a GeLU activation function. These layers are commonly found in the MLP stage of ViTs [3]. Fig 3 reports the MLP's runtime with and without FTL when using only the RISC-V cluster (left side) and using the cluster and the NPU (right side). There are two reasons to explain such runtime reduction when using FTL. First, FTL reduces the number of DMA transfers by 47.1 % by preventing the materialization of the MLP's intermediate tensor. Second, the L2 memory capacity is exceeded when materializing the MLP's intermediate tensor; hence, this tensor is stored in L3 Random-Access Memory (RAM). With FTL, we don't need to perform costly off-chip memory transfers to bring back the intermediate tensor from L3 to L1, leading to a reduction of the runtime. If double-buffering is used, FTL speeds up execution only if the kernel runtime is less than the DMA's runtime. As reported in Fig 3, this is the case when using the cluster and the NPU.

[4] `https://github.com/gvsoc`

References

[1] T. Chen et al., "Tvm: an automated end-to-end optimizing compiler for deep learning," in *USENIX OSDI'18*.

[2] A. S. Prasad et al., "Siracusa: A 16 nm heterogenous risc-v soc for extended reality with at-mram neural engine," *IEEE JSSC*.

[3] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR 2021*.