RECHNER-ARCHITEKTUR INFIII

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

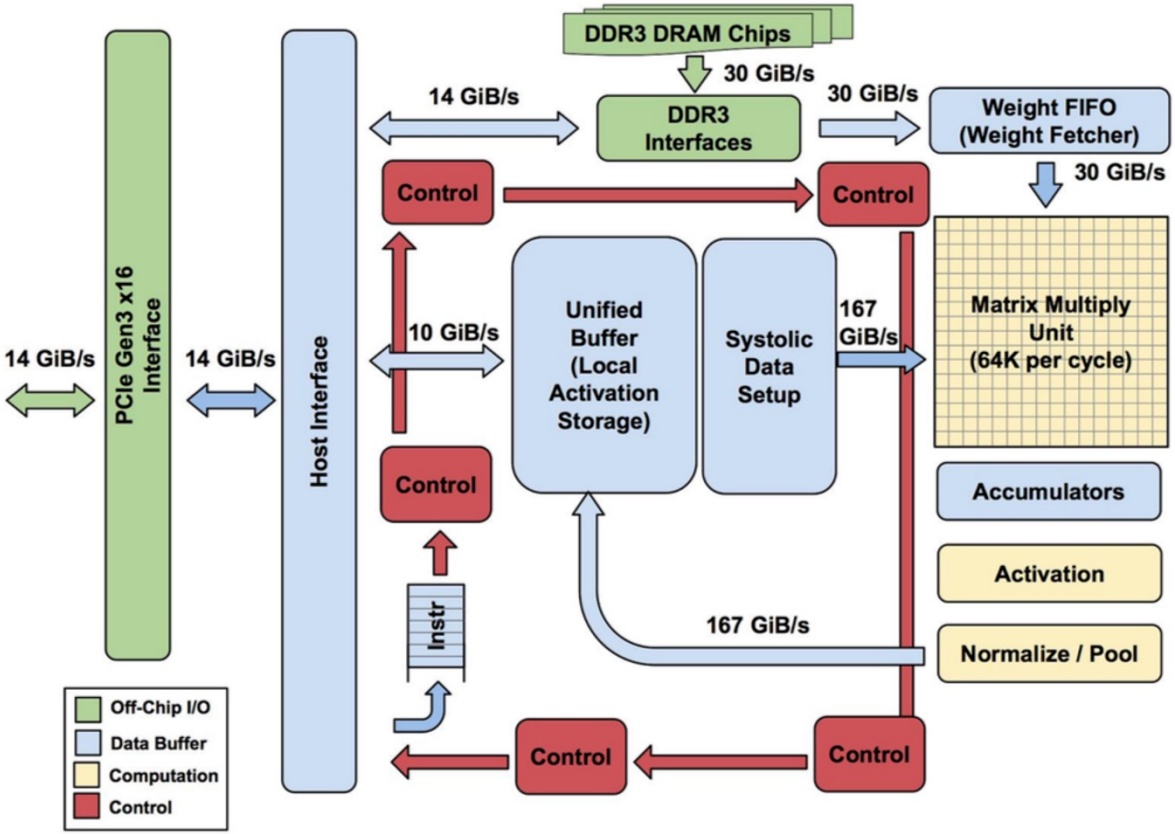
Architecture Overview

06. TPU

Philipp Gündisch, Philipp Holzinger

TPU – a DL application specific integrated circuit (ASIC):

- not an ASIC in the strict sense
- TPU still programmable logic via a CISC instructionset
  → adaptable to many different DL applications (networks)
- hardware structure optimized with respect to dataflow in neural networks
- ALU optimized for matrix operations, which are common in neural networks (e.g.: convolutional layer, fully connected layer)
- more energy-efficient for DL-tasks than traditional CPU/GPU-approaches

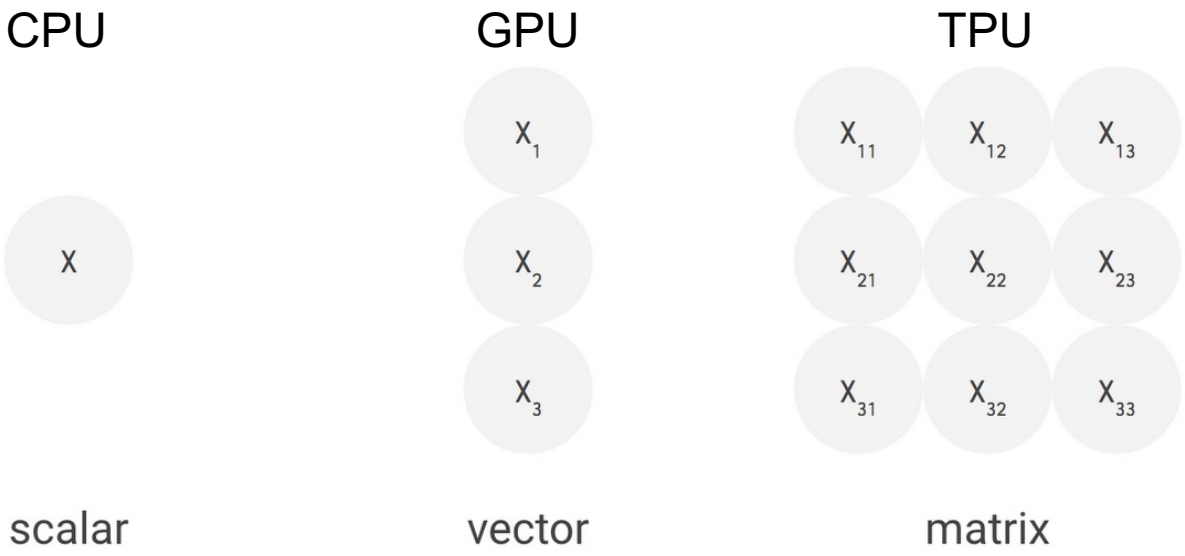


- **Matrix-Multiply-Unit MXU**
  256x256x8bit multiply add
- **Unified Buffer**
  24MB Registers
- **Activation Unit**
  hardwired activation functions

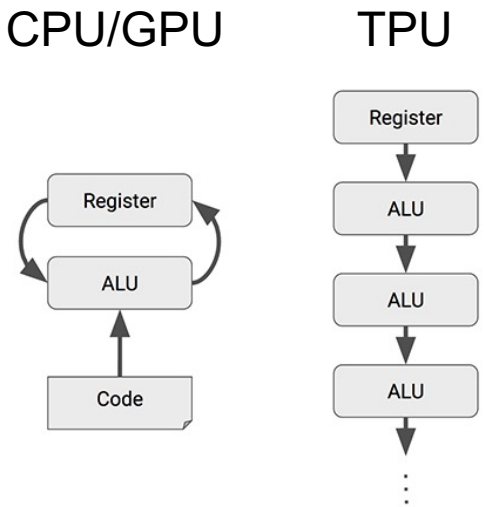
Typical TPU instructions

TPU Instruction	Function
Read_Host_Memory	Read data from memory
Read_Weights	Read weights from memory
MatrixMultiply/Convolve	Multiply or convolve with the data and weights,accumulate the results
Activate	Apply activation functions
Write_Host_Memory	Write result to memory

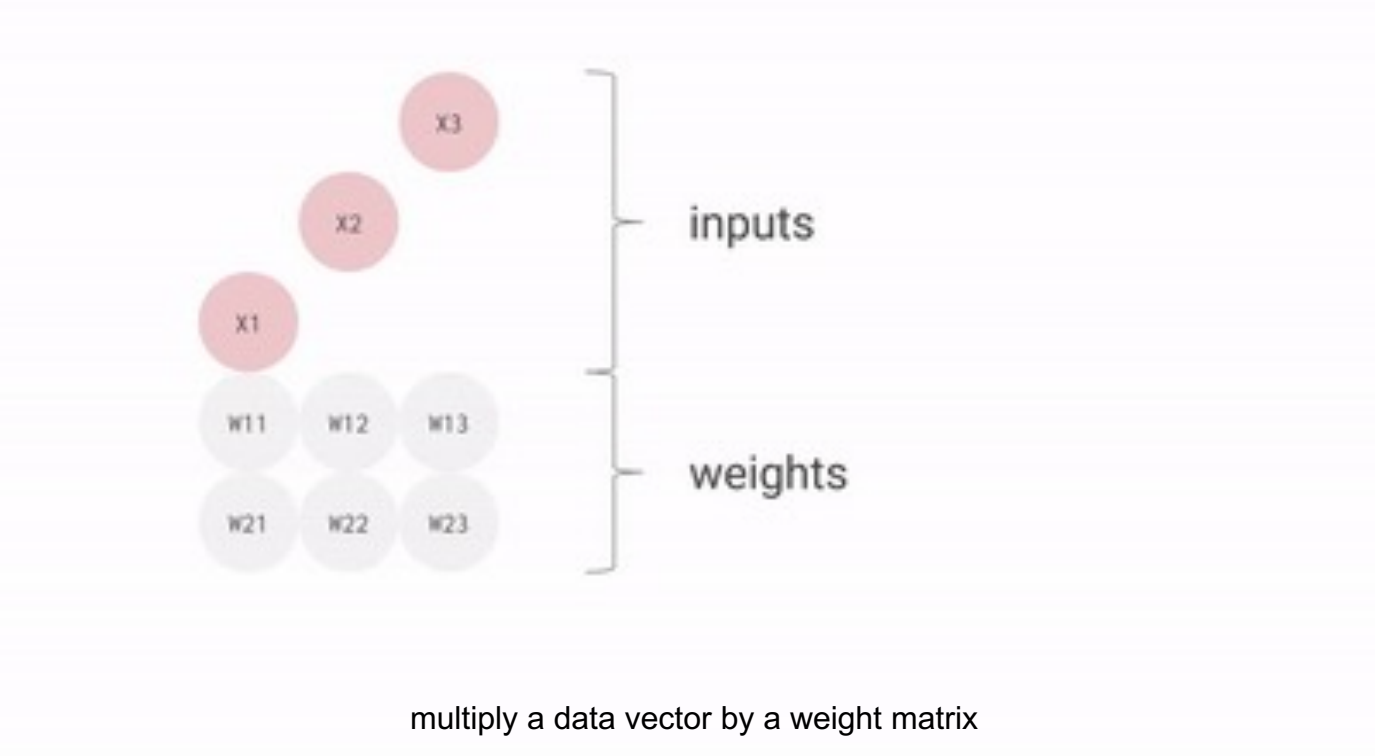
The MXU processing principle



The MXU processing principle

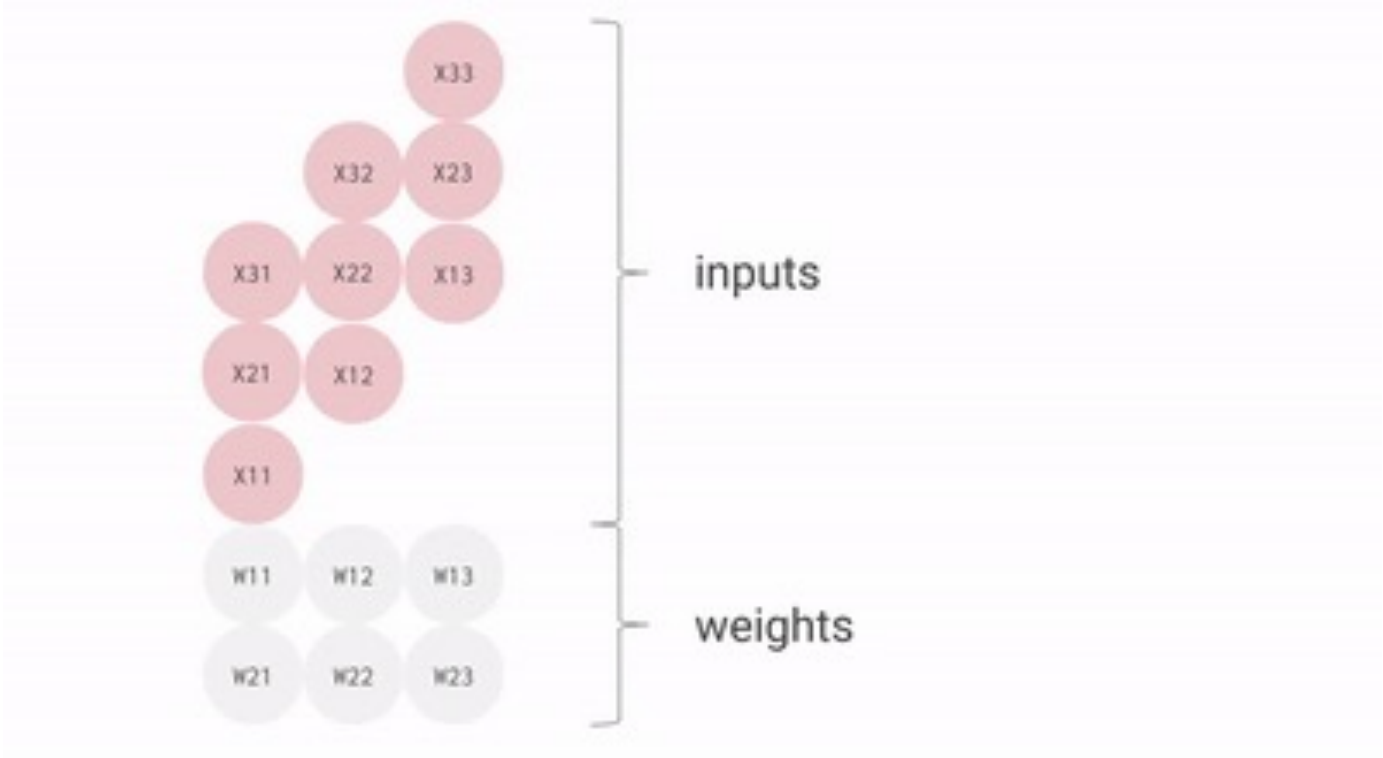


The MXU processing principle – a systolic array



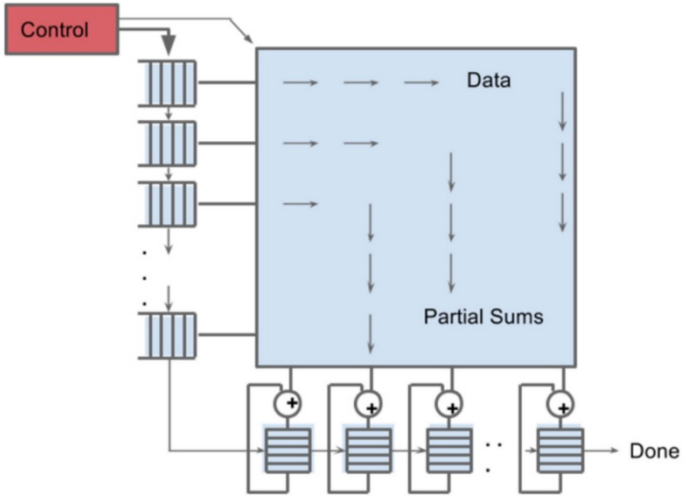
multiply a data vector by a weight matrix

The MXU processing principle – a systolic array



multiply a data matrix by a weight matrix

The MXU processing principle – a systolic array



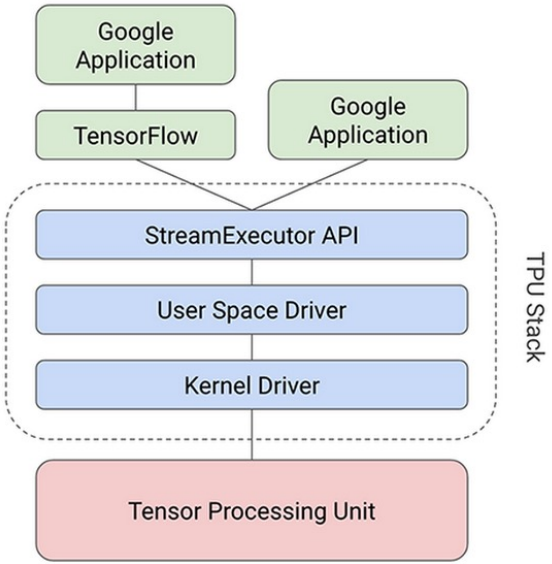
MXU structure

The MXU processing principle – a systolic array

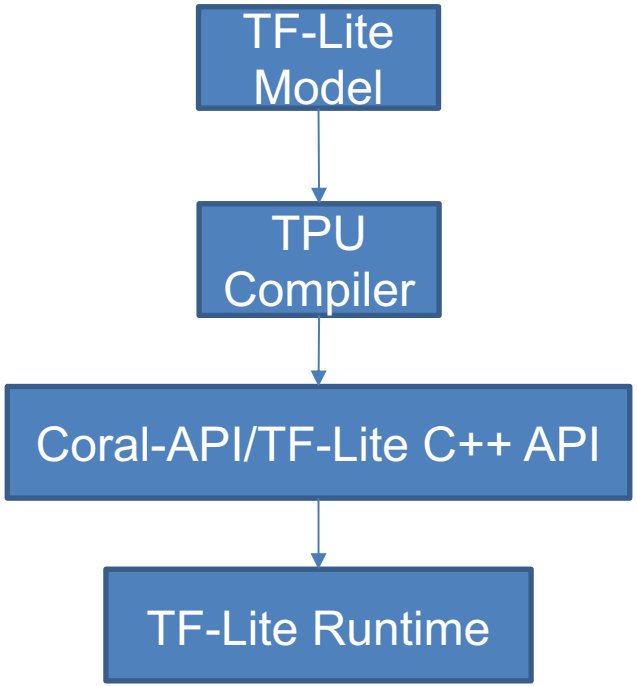
- TPU optimized for Matrix multiplication operations
  (most common operations in neural networks)
- saving lots of registers and memory accesses
- as consequence less flexible than scalar/vector processing structures
  → less suited for general purpose computations

TPU Programming Model

Workflow


Google Application
TensorFlow
Google Application
StreamExecutor API
User Space Driver
Kernel Driver
TPU Stack
Tensor Processing Unit

Workflow



DL Model

conversion into TPU compatible format

integration into C++ application

execution

Coral-API example

```cpp
int main(int argc, char* argv[]) {
  absl::ParseCommandLine(argc, argv);

  // Load the model.
  const auto model = coral::LoadModelOrDie(absl::GetFlag(FLAGS_model_path));
  auto edgetpu_context = coral::ContainsEdgeTpuCustomOp(*model)
                             ? coral::GetEdgeTpuContextOrDie()
                             : nullptr;
  auto interpreter = coral::MakeEdgeTpuInterpreterOrDie(*model, edgetpu_context.get());
  CHECK_EQ(interpreter->AllocateTensors(), kTfLiteOk);

  // Read the image to input tensor.
  auto input = coral::MutableTensorData<char>(*interpreter->input_tensor(0));
  coral::ReadFileToOrDie(absl::GetFlag(FLAGS_image_path), input.data(), input.size());
  CHECK_EQ(interpreter->Invoke(), kTfLiteOk);

  // Read the label file.
  auto labels = coral::ReadLabelFile(absl::GetFlag(FLAGS_labels_path));

  for (auto result : coral::GetClassificationResults(*interpreter, 0.0f, /*top_k=*/3)) {
    std::cout << "---------------------------" << std::endl;
    std::cout << labels[result.id] << std::endl;
    std::cout << "Score: " << result.score << std::endl;
  }
  return 0;
```

Learn more on
https://coral.ai/docs/