Towards performance portability of AI models using SYCL-DNN

Muhammad Tanvir – Staff Software Engineer – ML libraries

IWOCL – 10-12 May 2022
Outline

• Deep Learning Challenges
• Problem Statement – performance portability
• How does SYCL help?
• Performance Evaluations
• Deep Learning on RISC-V
• Conclusions
Deep Learning Challenges

• **Diversity** of Technologies and Techniques

• **Migrating** between Deep Learning frameworks

• Multiple implementations of same **NN Optimization algorithms**

• **Maintainability** of various version of low-level backend/libraries integrated in existing high-level frameworks

• **Hardcoded** implementation of inference engines for a restricted set of hardware

• Multiple implementations of same **NN algorithms** across various target **HW**
Deep Learning Challenges

- **Diversity** of Technologies and Techniques.
- **Migrating** between DL framework.
- Multiple implementations of Same **NN Optimization algorithms**.
- **Maintainability** of various version of low-level libraries integrated in existing high-level frameworks.
- **Hardcoded** implementation of Inference engines for a restricted set of hardware.
  - Multiple implementations of same **NN algorithms** across various target **HW**
Problem Statement

• Multiple implementations of same **NN algorithms** across various target **HW** – **is there a way around this?**

• An ideal solution(s) should:
  • Be **portable** across different platforms
  • Require **little (to no) changes** to the actual kernel code
  • Yield **acceptable performance** as compared to vendor optimized code
  • Have **backward compatibility (maintainability)**
SYCL

• Supports cross-platform portability
  • Different implementations of SYCL compilers provide a variety of targets

• Is maintainable
  • Open-source implementations

• Kernel Code modifications?

• Performance Portability?
• **Kernel Code Modification?**

• **Possible Solution**: Use C++ template meta programming to write highly parametrized kernels (for the most compute intensive operations)

• **Benefits**:
  • Reuse the same kernel code
  • Modify (tune) the template parameters to maximize performance on target hardware
• **Performance Portability?**

• **Possible Solution:**
  - Tune the template parameters of the kernel to best match the underlying hardware*
  - Expose maximum performance out of the tuned kernel

• **Acceptable Performance:**
  - Within 70%-80% range of vendor optimized code's performance

*e.g. https://github.com/codeplaysoftware/sycl-blas/blob/master/tools/auto_tuner/gen/intel_gpu.json
SYCL

SYCL-DNN

- Conv
- Batchnorm
- Pool
- Softmax etc

https://github.com/codeplaysoftware/SYCL-DNN

SYCL-BLAS

- GEMM
- GEMV
- Reduction etc

https://github.com/codeplaysoftware/sycl-blas

© 2022 Codeplay Software Ltd.
Performance Evaluations

1. Identify the most compute intensive operation(s)
2. Choose the target hardware
3. Tune SYCL code based on the target hardware
4. Evaluate performance
5. Repeat steps 2-4 for remaining target hardware
Performance Evaluations

VGG Demo

VGG-16

Image Classification

90% = American Staffordshire Terrier
10% = cat
Performance Evaluations

1. Identify the most compute intensive operation

- Conv: `sycl::conv2d`
- BiasAdd: `sycl::biasAdd`
- Relu: `sycl::relu`
- Pooling: `sycl::pooling`
- Gemm: `sycl::gemm`
- Softmax: `sycl::softmax`
Performance Evaluations

1. Identify the most compute intensive operation

```
Conv -> sycl::conv2d
BiasAdd -> sycl::biasAdd
Relu -> sycl::relu
Pooling -> sycl::pooling
Gemm -> syclblas::gemm
Softmax -> sycl::softmax
```

GEMM
Performance Evaluations

2. Choose the target hardware

- Intel CPU
- Intel GPU
- NVIDIA GPU
- RISC-V
Performance Evaluations

3. Tune SYCL code based on the target hardware

- **Target Hardware**
  - Intel(R) Core(TM) i7-6700K CPU @ 4 GHz
  - Intel Corporation HD Graphics 530

- **Most Compute Intensive Operation**
  - GEMM

- **Tuning Methodology**
Tuning paradigm

• Every GEMM operation, in DNNs, has different compute intensity

• Most optimal solution
  • One tuned kernel per GEMM operation
  • Yields the best performance
  • Size of the library increases drastically

• Semi-optimal solution
  • One tuned kernel per DNN model
  • Yields semi-optimal performance
  • Limited no. of kernels in the library
Performance Evaluations

**Intel(R) Core(TM) i7-6700K CPU**

- SYCL-DNN
- SYCL-DNN + Tuning
- oneDNN

**Intel Corporation HD Graphics 530**

- SYCL-DNN
- SYCL-DNN + Tuning
- oneDNN

*With semi-optimal tuning regime*
Performance Evaluations

2. Choose the target hardware

- Intel CPU
- Intel GPU
- NVIDIA GPU
- RISC-V
Performance Evaluations

3. Tune SYCL code based on the target hardware

- Target Hardware
  - NVIDIA Titan RTX GPU – 24 GB DDR6

- Most Compute Intensive Operation
  - GEMM

- Tuning tool
  - Modified version of SYCL-BLAS auto-tuner*

*https://github.com/codeplaysoftware/sycl
- blas/tree/master/tools/auto_tuner
Performance Evaluations

3. Tune SYCL code based on the target hardware

• Search all possible GEMM configs* exhaustively
  • Choose a GEMM config
  • Run the entire VGG16 (DNN) model and measure performance
  • Record results for all possible GEMM configs
  • Choose the GEMM config which yields the best performance

Performance Evaluations

![NVIDIA Titan RTX Performance Chart]

- SYCL-DNN
- SYCL-DNN + Tuning
- cuDNN

VGG16 Processing Time (ms)
Performance Evaluations

2. Choose the target hardware

- Intel CPU
- Intel GPU
- NVIDIA GPU
- RISC-V
Deep Learning on RISC-V

• Target Hardware
  • RISC-V spike simulator – single core

• Software Stack
  • Acoran – The Open Acceleration Platform*

*https://www.codeplay.com/solutions/acoran/
Why Neural networks on RISC-V?

• Domain specific accelerators are required to achieve cost-effective performance on-chip
• Cost effective performance requires tuning the design to the needs of the workload required
• RISC-V ISA has a minimalist base integer instruction set and provides custom extensions
  • An ideal starting point for creating special accelerators
• More companies are looking at RISC-V to enable AI software
• Designs can benefit from the RISC-V vector extension
  • Enables vectorization for various application
  • Helps achieve high compute density on chip
The Acoran platform provides all the supporting open-source libraries and frameworks needed to build this neural network demonstration.
GEMM Operator

Main Computation

```c
for (int a = a_start, b = b_start; a <= a_end; a += blockSize, b += (blockSize * matSize)) {
    tmp += pBA[localY * blockSize + k] * pBB[localX * blockSize + k];
    // The barrier ensures that all threads have written to local
    // memory before continuing
    it.barrier(access::fence_space::local_space);
}
```

```
pc[elemIndex] = tmp;
...`
```
RISC-V/RVV Kernel compilation flow FC

Device Compiler → SYCL Kernel → Scalar LLVM IR → Codeplay Vectorizer ('vecz') → Vector LLVM IR → LLVM back-end

CPU Compiler

CPU
Deep Learning on RISC-V

VGG Demo
Conclusions

• SYCL-DNN / SYCL-BLAS have support for efficient acceleration of popular DNNs

• Acoran platform provides an end-to-end compute stack for accelerating DNNs on RISC-V
  • https://developer.codeplay.com/products/acoran/pre-alpha

• Recent Update: Adding SYCL to upstream ONNX Runtime
  • https://github.com/codeplaysoftware/onnxruntime
Thank you

muhammad.tanvir@codeplay.com

@codeplaysoft  info@codeplay.com  codeplay.com