Software

A Case Study on OpenCL vs GPU Assembly for Machine Learning Performance

Roy Oursler, PhD

About Me

- AI Algorithm Engineer at Intel working on oneDNN
- Experienced in CPU and GPU optimization
  - Started CPU optimization work with ISA-L in 2015.
    - Library focusing on Compression, Erasure Codes, Hashing, and Encryption
  - Moved to oneDNN CPU optimization in 2019
  - Switched to oneDNN GPU optimization in 2020.

AGENDA

Introduction to oneDNN

X^e GPU Architecture

OpenCL vs nGEN Assembly Comparison

Outcome

Introduction to oneDNN

- Open-source high performance library for deep learning workloads
- Support heterogeneous computing
  - Current focus is on CPU and GPU
- Provide cross platform support
  - Linux, Windows, Mac

Introduction to oneDNN: Requirements

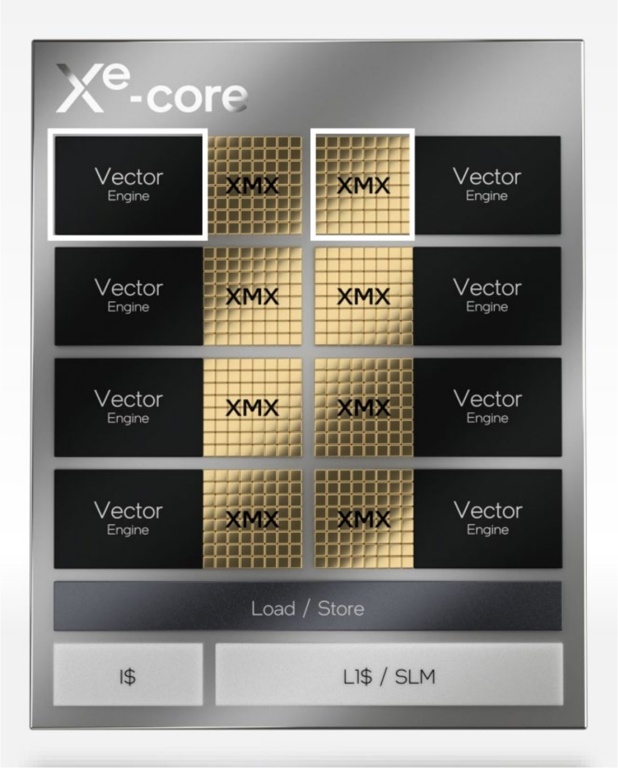
- Require runtime choice of optimal implementation for a given problem and hardware architecture
  - To support this, oneDNN uses a JIT based architecture
- Support Multiple Data Types:
  - int8, f16, bf16, f32
- Support Multiple Data Formats

GPU Architecture

GPU Architecture: X^e-core

- X^e-core is made of
  - Vector and Matrix Engines
  - Shared Resources
    - SLM/Cache
    - Load/Store
- X^e Matrix Extension (XMX) provides instructions to accelerate matrix multiplication
    - Provides access to the dpas/dpasw instructions which perform matrix multiplication

GPU Architecture – OpenCL Extensions

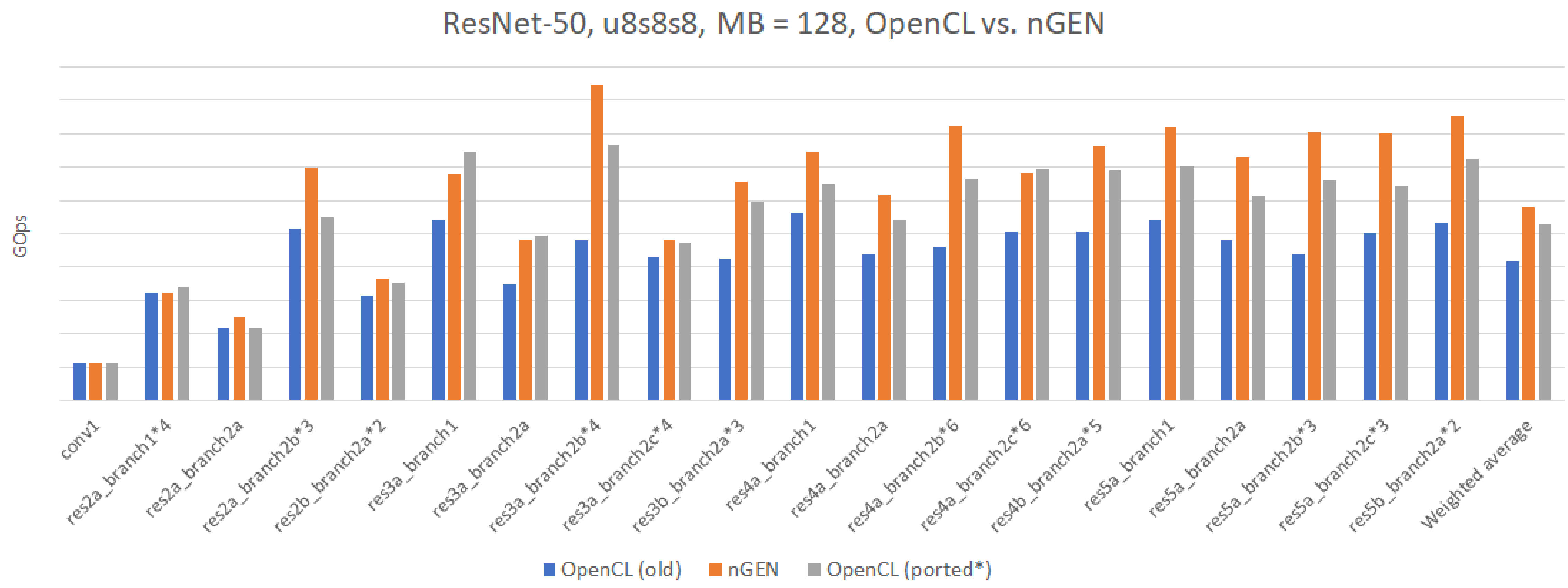


- cl_intel_subgroups
  - Take advantage of X^e-core architecture
- cl_intel_bfloat16_conversion
  - Current WIP pull request
  - Gives access to bfloat16 data type
- Various Compiler built-ins
  - Gives access to functionality like the XMX unit
  - Some extensions are being made to give longer term access to XMX like the SPIR-V extension SPV_INTEL_joint_matrix and SYCL extension sycl_ext_oneapi_matrix

GPU Architecture – nGEN

- nGEN is a C++ library for assembly generation on Intel GPUs
- Supports AOT and JIT kernel generation
- Inspired by Xbyak library for JIT assembly on x86

OpenCL vs nGEN Assembly: Results



OpenCL vs nGEN Assembly: Binary Analysis

- OpenCL implementation emitted shorter read instructions
- Cases with padding/non-multiple sizes resulted in OpenCL C implementation emitting extra conditions.
- Neither issue appears fundamental to using OpenCL C.

OpenCL vs nGEN Assembly: Takeaway

- OpenCL C can get essentially equivalent performance
- Pure assembly is poor for productivity (surprise of the year)
  - Experiment with assembly did reveal gaps with OpenCL C implementation
- OpenCL C is slow for JIT compilation
  - Takes approximately 500 ms vs about 10ms for nGEN Assembly
  - SPIR-V is estimated to improve OpenCL C performance by 3x, but that is not enough.
- It is challenging to prevent implementation proliferation for both OpenCL C and nGEN due to different data-types, formats, architecture, and problem-specific optimizations.
  - Challenging to determine what will break an OpenCL compiler optimization heuristic
  - Hard to compose different (and potentially conflicting) optimizations

Outcome

- Neither solution was deemed good enough
  - OpenCL compile time was too slow and maintaining multiple implementations is a lot of work
  - nGEN alone was too much work to implement all desired optimizations
- Switched to implementing an assembly generator.
  - Uses a custom IR targeted for our use case, loosely based on Halide
  - Specify optimization transforms to perform
- Code generation is significantly faster
  - Generation is around 50 ms
  - With further work, code generation performance could be improved further
- Allows more effective composition of optimizations

Questions?