HALO: A Compiler Framework for Chiplet Architectures

Weiming Zhao
Weifeng Zhang
Background

- AI needs more-than-Moore computing power
  - Massive computing power needed
    - Huge dataset for training
    - Massive model parameter
  - Exponential growth
    - 10x/year increase in demand
    - Moore’s law: 1.4 x / year

- To tackle the challenge
  - Scale up
    - Domain Specific Architecture (DSA)
  - Scale out
    - More cores / accelerators
Chiplet: A Promising Approach

• Chiplet
  • Integrate multiple dies (chiplets) into one package
    • Allows heterogeneous dies
    • Very low latency on die-to-die communication

• Why Chiplet for AI ?
  • Specific AI accelerator
  • End-to-end acceleration
  • Easy for processors tiling
  • Ultra fast inter-core communication
  • Optimize for various scenarios
    • Power, latency, throughput, storage, cost, security, ...
  • Shorter time-to-market
    • Critical to rapid evolving AI algorithm

Scale Up
Scale Out
Software: The Roadblock

• Scale-Up: Fragmented Software Ecosystem
  • Diversified HW. 😊 → vendor-specific SW. stack 😞
  • Difficult to scale-up
    • Tremendous porting effort
    • Delays time-to-market
  • Lack of interoperability
    • Suboptimal performance

• Scale-out: More of a Software Challenge
  • Workload parallelization
  • Distributed computing system (esp. for chiplets)
Fragmented Software Ecosystem

Frameworks may not fit for some use cases

Overall Effort: \( M \) (frameworks) * \( N \) (accelerators)

vendor-provided software stack

No interoperability, no collaboration
Motivation & Our Philosophy

**Motivation**
Make software an enabler, not a blocker

- Significantly reduce overall effort
- Fully unleash the power of chiplets for AI
  - Easy to scale-up with more powerful HW
  - Easy to orchestrate different chiplets
  - Easy to scale-out with many processors

- Efficient and Flexible
  - No dependency on specific AI framework
  - Low memory resource footprint
  - Low runtime overhead

**Philosophy**

1. An unified SW layer to isolate HW discrepancy
2. Make AI algorithms run efficiently on it
Proposed Solutions (1/2)

1. A Unifying Programming Model for AI Computing
   • Open Deep Learning API (ODLA)
     • APIs for:
       (1) AI operations and pre/post-processing
         • Extensible for user-defined ops & tasks
       (2) Runtime Management
         • Device management
           • Capabilities, topologies, etc.
         • Execution management
           • Context management
           • Data movement
           • Synchronized/Asynchronous execution
     • Abstract representations for devices, computations, data, etc.
     • Implementation agnostic
       • HW specific implementation
     • Simple C99-based API
2. An Optimizing Compiler Framework
   • Heterogeneity Aware Lowering & Optimization (HALO)
     • Compiles AI algorithms & workflows → ODLA-based code
     • Optimizations
       • Classical compiler optimizations
       • AI optimizations
       • Heterogeneous devices support
         • Device placement
         • Parallelization & sharding
System Overview

Heterogeneity Aware Lowering and Optimization (HALO)

- AI Model & Workflow $\rightarrow$ ODLA code
- Heterogeneity-aware optimization
- Device placement & scheduling
- Standalone executable (no framework needed)
- Efficient for small chiplets
- Quick support of new chip

Overall Effort = N (frameworks) + M (accelerators)
HALO Overview

- Multi-grained IR
  - Multi-level optimization
  - Easy to extend & customize

- Optimizations
  - Device-independent optimization
  - Device-dependent optimization
  - Partitioning & placement

- Generated code
  - Free of AI framework
  - Memory efficient
  - Low runtime overhead

HALO Overview diagram with various tools and components including TF, ONNX, Caffe, TFLite, HALO Parser, HALO IR, Op Fusion, Classical Optimizations, Shape Inference, Cost Model, Device Placement, ODLA CodeGen, LLVM CodeGen, and LLD Linker.
Evaluations

Compact Code

AI Framework Runtime Sizes vs ODLA Code Sizes (KB)

Size (KB)

<table>
<thead>
<tr>
<th>Framework</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF Lite RT</td>
<td>1422</td>
</tr>
<tr>
<td>ONNX RT</td>
<td>10394</td>
</tr>
<tr>
<td>Caffe RT</td>
<td>4714</td>
</tr>
<tr>
<td>ResNet 50 V2 on ODLA</td>
<td>100</td>
</tr>
<tr>
<td>Yolo V5 on ODLA</td>
<td>173</td>
</tr>
<tr>
<td>BERT on ODLA</td>
<td>246</td>
</tr>
</tbody>
</table>

(weights are not included)

Best fit for small chiplets (IoT, edge, etc.)
Evaluations

Efficient Execution

Latency of ResNet 50 on Telsa T4 (ms)

Latency of BERT SQUAD on Telsa T4 (ms)
Case Study

• Smart Factory
  • Analog instrument reading, fire detection, security, ...
  • Mixed AI algorithms
    • Caffe, TensorFlow
  • 3 HW platforms
    • Restrictions on storage & memory
  • Planned as a 6-month project for a team
  • With HALO/ODLA
    • For each HW, 1 engineer, ~2 weeks to implement ODLA RT lib
      • no prior experience on ODLA API
      • no need to access actual AI model
    • Finished in 2 months!
    • Recently migrated to new HW (~3 man-week)
Call for Actions

• Build unified software ecosystem
  • Open sourced @ github.com/alibaba/heterogeneity-aware-lowering-and-optimization)
    • Partnered with Intel, Graphcore, Qualcomm, Nvidia, Cambrian, ...
    • Supports 10+ HW platforms
    • Please join us!

• New programming paradigm for heterogeneous chiplets?
  • HBM, D2D interface (BoW)
    • new programming model? new language?
    • new distributed heterogeneous computing system?
Thanks