Towards Performance Portability through an Integrated Programming Eco-System for Tensor Algebra

Gokcen Kestor, Roberto Gioiosa, Mark Raugas
Pacific Northwest National Laboratory
Outline

• Introduction and Background
  ▪ Tensor Algebra COmpiler (TACO)
  ▪ Minos Computing Library (MCL)

• TACO-MCL Integrated Software Stack

• Initial Results

• Conclusions
Motivation

• The Cambrian era is upon us:
  ▪ Hardware landscape:
    ✓ Many custom accelerators are being developed
    ✓ Each HW design has its own interface, performance and energy profile
  ▪ Software landscape:
    ✓ Complex workflows (simulations + in-situ data analytics, simulations + AI)
    ✓ Many programming languages and frameworks (from C/C++ to Python, TensorFlow, etc.)

• Program and performance portability has become a major concern:
  ▪ Current HPC systems: ORNL Summit, LLNL Sierra, SNL Trinity
  ▪ Next HPC systems: ORNL Frontier, LLNL El Capitan, ANL Aurora
• Expecting multi-device systems with several classes of devices within a single SoC (e.g., CPUs, GPUs, AI engines, FPGAs, …)
• Programming such systems is challenging!
Proposal: A Portable Hardware/Software Stack

- Scientists express their algorithm with high-level DSLs that provide domain-specific programming abstractions
- Compiler lowers DSL code to device-specific, highly-optimized code
- Dynamic runtime coordinates access to computing resources and data transfers
Tensor Algebra COmpiler (TACO)

- TACO is a fast and versatile library for linear and tensor algebra
- C++ and Python extension to support complex tensor expression
  - Mostly focused on sparse tensor algebra*
- Automatically generate
  - Sequential CPU code
  - Parallel OpenMP code
  - NVIDIA CUDA GPU code

* Not all sparse tensor algebra operations are supported

TACO Example

```c
TACO Example

```y(i) = A(i, j) * x(j)`

CUDA code generation for sparse matrix-dense vector computation

```c
C(i, k) = A(i, j) * B(j, k)`

OpenMP code generation for dense DGEMM computation

```c
P3HPC Workshop

September 2, 2020
TACO Software Stack

- TACO Python
- TACO C++
- TACO Compiler
- Sequential
- OpenMP
- CUDA
- X86 CPU
- NVIDIA GPU
- AMD GPU
- FPGA
- AI Engine
TACO-MCL: Integrated Programming Eco-System for Tensor Algebra

- **TACO C++/Python Language**
- **TACO-MCL Compiler**
- **MCL Runtime**
- **Heterogeneous Devices**

- Automatically generate portable MCL host code and OpenCL kernels
- Break long expressions into smaller kernels for multi-device execution
- Analyze data and control flow dependencies to maximize asynchronous execution
- Asynchronous task execution and overlapping of data transfers and computation
- Load balancing and resource management
- Multi-applications support

September 2, 2020
The Minos Computing Library (MCL)

• Framework for programming extremely heterogeneous systems
  ▪ Programming model and programming model runtime
  ▪ **Abstract low-level architecture** details from programmers
  ▪ **Dynamic** scheduling of work onto available resources

• Key programming features:
  ▪ Applications factored into tasks
  ▪ **Asynchronous** execution
  ▪ Devices are managed by the scheduler
  ▪ Co-schedule **independent applications**
  ▪ Simplified APIs and programming model (based on OpenCL)

• Flexibility:
  ▪ Scheduling framework
  ▪ **Multiple scheduling algorithms** co-exist
  ▪ Code portability
  ▪ Resources allocated **at the last moment**

Scaling Up and Down

Same code runs on all these systems without modification
By using MCL with TACO backend, TACO applications will:

- Leverage a broader classes of computing devices
- Execute in multi-device environments
- Execute in a multi-application environment
- Exploit sophisticated scheduling and load balancing algorithms
```
#include <iostream>
#include "taco.h"

using namespace taco;

int main(int argc, char* argv[]) {
  Format csr({Dense, Sparse});
  Format csf({Sparse, Sparse, Sparse});
  Format sv({Sparse});
  Tensor<double> A("A", {2,3}, csr);
  Tensor<double> B("B", {2,3,4}, csf);
  Tensor<double> c("c", {4}, sv);

  // Insert data into B and c
  B(0,0,0) = 1.0;
  B(1,2,0) = 2.0;
  B(1,2,1) = 3.0;
  c(0) = 4.0;
  c(1) = 5.0;

  IndexVar i, j, k;
  A(i,j) = B(i, j, k) * c(k);

  std::cout << A << std::endl;
}
```
Experimental Results 1/2

• CCSD(1) method from NWChem
  ▪ Coupled cluster (CC) methods are commonly used in the post Hartree-Fock ab initio quantum chemistry and in nuclear physics computation.
  ▪ The CC workflow is composed of iterative set of excitation (singles (S), doubles (D), triples (T), and quadruples (Q)) calculations

• Testbed:
  ▪ NVIDIA DGX-1 V100
  ▪ 2x Intel Xeon E5-2680, 768GB memory
  ▪ 8x NVIDIA V100, 16GM memory, NVLink
Experimental Results 2/2

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>TACO (seconds)</th>
<th>TACO/MCL (seconds)</th>
<th>Speedup w.r.t. TACO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>0.85</td>
<td>0.168</td>
<td>5.086</td>
</tr>
<tr>
<td>Medium</td>
<td>39.07</td>
<td>7.05</td>
<td>5.58</td>
</tr>
<tr>
<td>Large</td>
<td>1209.93</td>
<td>223.10</td>
<td>5.43</td>
</tr>
</tbody>
</table>

- TACO applications automatically scale to use all GPUs
- All problem sizes show scalability
- Expect similar speedups with larger problems
- Not ideal speedup -- WIP
Conclusions

- Program and performance portability has become a major concern
- Current and future systems feature multi-device, multi-class accelerators
  - Programming and porting applications on such systems is extremely difficult
  - Each device class has its own programming model
  - Need to manage data locality, load balancing, correctness, and resource utilization
- We developed an approach that attempts to solve the problem with an integrated software stack:
  - Users develop applications using high-level DSLs
  - Compiler lower code to targets
  - Runtime manages data locality, load balancing, and computing resources
- With TACO-MCL, original TACO applications gain:
  - Access to non-NVIDIA resources (AMD/Intel GPUs, FPGAs, AI engines)
  - Transparent and automatic access to multi-device systems
  - Transparent execution in multi-applications environments (complex workflows)
Thank you

POC: Gokcen Kestor
gokcen.kestor@pnnl.gov
P3HPC Workshop