CUDA-OMP —
Or: Breaking the Vendor Lock

Performance Portable Programming Through OpenMP as Target Independent Runtime Layer

Johannes Doerfert\textsuperscript{1,2}, Mark Jasper\textsuperscript{1}, Joseph Huber\textsuperscript{3,4}, Khaled Abdelaal\textsuperscript{5}, Giorgis Georgakoudis\textsuperscript{1}, Thomas Scogland\textsuperscript{1}, Konstantinos Parasyris\textsuperscript{1}

\textsuperscript{1} LLNL: Lawrence Livermore National Laboratory
\textsuperscript{2} ANL: Argonne National Laboratory (past, email is dead)
\textsuperscript{3} ORNL: Oak Ridge National Laboratory (past, email is dead)
\textsuperscript{4} AMD
\textsuperscript{5} University of Oklahoma (ANL intern)
Motivation
Motivation

CUDA

AMD GPU

NVIDIA GPU

CPU

Intel GPU
Motivation

hipify?

HIP

CUDA

AMD GPU

NVIDIA GPU

CPU

Intel GPU
Motivation

- Source Language Fragmentation
- Variable (Performance) Portability
- Source Porting requirement
Motivation

- Source Language Fragmentation
- Variable (Performance) Portability

CUDA

OpenMP

AMD GPU

NVIDIA GPU

CPU

Intel GPU
Motivation

- Source Language Fragmentation
- Variable (Performance) Portability
- Source Porting requirement
Approach
Approach

- CUDA
- OpenMP
- AMD GPU
- NVIDIA GPU
- CPU
- Intel GPU
Approach
Approach

CUDA → LLVM → Clang → OpenMP Runtime →

AMD GPU

NVIDIA GPU

CPU

Intel GPU
Approach

CUDA → LLVM → OpenMP Runtime → LLVM-IR+ →

- AMD GPU
- NVIDIA GPU
- CPU
- Intel GPU
Approach

- HIP
- CUDA
- SYCL
- OpenMP

LLVM
Clang

OpenMP Runtime

AMD GPU
NVIDIA GPU
CPU
Intel GPU
Overview – CUDA via OpenMP Offload Compilation

- CUDA
- OpenMP
- Offload Compilation

Diagram:
- CUDA
- clang
- Device compilation
- Host compilation
- Embedding
- Vendor runtimes
- NVIDIA GPU
- Execution
Overview – CUDA via OpenMP Offload Compilation
Overview – CUDA via OpenMP Offload Compilation
Overview – CUDA via OpenMP Offload Compilation

New Offload Driver
- Language agnostic
- “Classic” design
- Static library support
- LTO-capable
Overview – CUDA via OpenMP Offload Compilation

New Offload Driver
- Language agnostic
- “Classic” design
- Static library support
- LTO-capable

Novel Embedding
- Language agnostic
- Metadata enriched
- Multi-device support
- ELF-tooling available
Overview – CUDA via OpenMP Offload Compilation

New Offload Driver
- Language agnostic
- “Classic” design
- Static library support
- LTO-capable

Novel Embedding
- Language agnostic
- Metadata enriched
- Multi-device support
- ELF-tooling available

CUDA API Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs
## Wrapped (User) CUDA APIs

### CUDA API calls used by each benchmark

<table>
<thead>
<tr>
<th>API call</th>
<th>XSbench</th>
<th>RSbench</th>
<th>LULESH</th>
<th>SU3</th>
<th>Triad</th>
<th>miniFE</th>
</tr>
</thead>
<tbody>
<tr>
<td>cudaMalloc</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>cudaMallocHost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>cudaMemcpy</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>cudaMemcpyAsync</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>cudaFree</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>cudaFreeHost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>cudaMemcpy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>cudaMemcpyAsync</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>cudaDeviceSynchronize</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cudaThreadSynchronize</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>cudaMemcpyProperties</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cudaMemcpyAsync</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>cudaStreamCreate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Overview – CUDA via OpenMP Offload Compilation

New Offload Driver
- Language agnostic
- “Classic” design
- Static library support
- LTO-capable

Novel Embedding
- Language agnostic
- Metadata enriched
- Multi-device support
- ELF-tooling available

CUDA API Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs
Overview – CUDA via OpenMP Offload Compilation

New Offload Driver
- Language agnostic
- “Classic” design
- Static library support
- LTO-capable

Novel Embedding
- Language agnostic
- Metadata enriched
- Multi-device support
- ELF-tooling available

CUDA API Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs

CUDA Builtin Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs
Overview – CUDA via OpenMP Offload Compilation

New Offload Driver
- Language agnostic
- “Classic” design
- Static library support
- LTO-capable

Target libm.a
- Map to target math impl.
- Multi-(sub-)target support
- Enable math optimization

Novel Embedding
- Language agnostic
- Metadata enriched
- Multi-device support
- ELF-tooling available

CUDA API Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs

CUDA Builtin Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs
Overview – CUDA via OpenMP Offload Compilation

New Offload Driver
- Language agnostic
- “Classic” design
- Static library support
- LTO-capable

Target libm.a
- Map to target math impl.
- Multi-(sub-)target support
- Enable math optimization

Novel Embedding
- Language agnostic
- Metadata enriched
- Multi-device support
- ELF-tooling available

CUDA API Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs

CUDA Builtin Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs
Overview – CUDA via OpenMP Offload Compilation

New Offload Driver
- Language agnostic
- “Classic” design
- Static library support
- LTO-capable

Target libm.a
- Map to target math impl.
- Multi-(sub-)target support
- Enable math optimization

Results:
- Portable CUDA
- Interoperable
  CUDA (+ HIP) + OpenMP
- Access to OpenMP features

Novel Embedding
- Language agnostic
- Metadata enriched
- Multi-device support
- ELF-tooling available

CUDA API Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs

CUDA Builtin Wrappers
- Map to OpenMP RT
- Use existing & new APIs
- Incl. compiler used APIs
Evaluation
<table>
<thead>
<tr>
<th>Execution Time (s)</th>
<th>Power9 + V100</th>
<th>AMD + MI50</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.75</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.75</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance Evaluation – XSBench

- **Power9 + V100**
  - **Execution Time (s)**
  - *Vendor-cc*

- **AMD + MI50**
  - *Vendor-cc*
Performance Evaluation – XSBench

Power9 + V100

Execution Time (s)

AMD + MI50

vendor-cc  clang-cc

vendor-cc  clang-cc
Performance Evaluation – XSBench

Graph showing execution time (s) for different combinations of architectures and compilers:

- **Power9 + V100**
  - vendor-cc
  - clang-cc
  - cuda-omp-cc

- **AMD + MI50**
  - vendor-cc
  - clang-cc
  - cuda-omp-cc
Performance Evaluation – RSBench

Graphs showing execution times for different compilers and architectures:
- Power9 + V100:
  - Vendor-cc: approximately 2 seconds
  - Clang-cc: approximately 2 seconds
  - CUDA-OMP-cc: approximately 1 second
- AMD + MI50:
  - Vendor-cc: approximately 1 second
  - Clang-cc: approximately 1 second
  - CUDA-OMP-cc: approximately 1 second
Performance Evaluation – Triad (Stream)

Power9 + V100

- vendor-cc
- clang-cc
- cuda-omp-cc

Execution Time (s)

AMD + MI50

- vendor-cc
- clang-cc
- cuda-omp-cc