OpenMP Device Offloading to FPGA Accelerators

Lukas Sommer, Jens Korinth, Andreas Koch
Motivation

- Increasing use of heterogeneous systems to overcome CPU power limitations
Motivation

• Increasing use of heterogeneous systems to overcome CPU power limitations

• FPGAs increasingly used for implementation of accelerators in HPC systems (e.g. Microsoft Azure)
Motivation

• Increasing use of heterogeneous systems to overcome CPU power limitations

• FPGAs increasingly used for implementation of accelerators in HPC systems (e.g. Microsoft Azure)

• Programming of heterogeneous systems is non-trivial
Motivation

• Increasing use of heterogeneous systems to overcome CPU power limitations

• FPGAs increasingly used for implementation of accelerators in HPC systems (e.g. Microsoft Azure)

• Programming of heterogeneous systems is non-trivial

• Desirable: Programming with a single, portable code base
OpenMP Device Offloading

- Denote target regions to execute on device

```c
#pragma omp target 
   map(to:x[0:SIZE]) \ 
   map(tofrom:y[0:SIZE])
{

   Target Region

}
```
OpenMP Device Offloading

- Denote target regions to execute on device
- Specify which and how data is transferred to device memory

```c
#pragma omp target
map(to:x[0:SIZE]) \ 
map(tofrom:y[0:SIZE])
{
#pragma omp parallel for
for(i=0; i<SIZE; i++)
    y[i] = a*x[i]+y[i];
}
```
OpenMP Device Offloading

- Denote target regions to execute on device
- Specify which and how data is transferred to device memory
- Use additional parallel constructs inside target region (also target-specific, e.g. teams, distribute,...)

```c
#pragma omp target 
 map(to:x[0:SIZE]) \ 
 map(tofrom:y[0:SIZE])
{
    #pragma omp parallel for [...] 
    for(i=0; i<SIZE; i++) {
        y[i] = a*x[i]+y[i];
    }
}
```
Goal

- Implement mapping of target regions to FPGA accelerators in LLVM Clang
  - Preserve FPGA-specific pragmas (e.g. Vivado HLS)
  - Automated flow from OpenMP-annotated input program to FPGA bitstream + software executable
Goal

- Implement mapping of target regions to FPGA accelerators in LLVM Clang
  - Preserve FPGA-specific pragmas (e.g. Vivado HLS)
  - Automated flow from OpenMP-annotated input program to FPGA bitstream + software executable

- Extend LLVM OpenMP Runtime
  - Manage data-transfers between host and FPGA
  - Control device execution on the FPGA accelerator
ThreadPoolComposer

- Toolchain to fast-track implementation of FPGA-based accelerators in heterogeneous systems
- Synthesize accelerator from kernel code

TPC is available as open source: https://goo.gl/qTsU3B
ThreadPoolComposer

- Toolchain to fast-track implementation of FPGA-based accelerators in heterogeneous systems
- Assemble (multiple) instances of different kernels in top-level design, combined with standardized host- and memory connection

TPC is available as open source: https://goo.gl/qTsU3B
ThreadPoolComposer

- Toolchain to fast-track implementation of FPGA-based accelerators in heterogeneous systems
- Control execution and data-transfer using two-layered API
  - Higher-level TPC API is device/platform-agnostic, allows for portable implementation (write once, run everywhere)

TPC is available as open source: https://goo.gl/qTsU3B
Compilation Flow

• Start from a single, portable source file
Compilation Flow

- Start from a single, portable source file
- Standard host-compilation, including fallback if offloading fails
Compilation Flow

- Start from a single, portable source file
- Standard host-compilation, including fallback if offloading fails
- One device-specific compilation flow per device type
  - Limited to extracted target regions
Compilation Flow

- Custom Clang toolchain for TPC-based offloading to FPGA accelerators
  - Identified with new LLVM target triple
  - Preserves FPGA-specific pragmas, e.g. Vivado HLS pragmas
  - Yields three artifacts
Compilation Flow

- TPC-specific software binary
  - Entry point for FPGA device execution
  - Transfers kernel arguments and launches hardware execution using TPC API
Compilation Flow

- TPC-specific software binary
  - Entry point for FPGA device execution
  - Transfers kernel arguments and launches hardware execution using TPC API
  - Included in the combined binary
Compilation Flow

- Hardware kernel code extracted from target region
Compilation Flow

- Hardware kernel code extracted from target region

- Description of input argument types
Compilation Flow

- Hardware kernel code extracted from target region
- Description of input argument types
- TPC automates synthesis from kernel code and description to full FPGA design

No additional user input required!
Compilation Flow

- Hardware kernel code extracted from target region
- Description of input argument types
- TPC automates synthesis from kernel code and description to full FPGA design
- Resulting bitstream features standardized host- and memory connection
Runtime Flow

Components:

Host binary

LLVM OpenMP Runtime

Libomptarget (device-agnostic)

TPC-specific binary

TPC plugin

TPC API

Platform API

FPGA

Loads & launches

Invokes
Components:

- LLVM OpenMP Runtime Infrastructure
Runtime Flow

Components:
- LLVM OpenMP Runtime Infrastructure
- TPC-based plugin for LLVM OpenMP Runtime
Runtime Flow

Components:

- LLVM OpenMP Runtime Infrastructure
- TPC-based plugin for LLVM OpenMP Runtime
- TPC-specific software binary resulting from compilation
Runtime Flow

Components:

- LLVM OpenMP Runtime Infrastructure
- TPC-based plugin for LLVM OpenMP Runtime
- TPC-specific software binary resulting from compilation
- FPGA abstraction as provided by TPC
Runtime Flow

- Host-centric: Execution starts on the host

If target region is encountered:
Runtime Flow

- Host-centric: Execution starts on the host
  - If target region is encountered:
- Transfer data to FPGA memory
Runtime Flow

- Host-centric: Execution starts on the host

If target region is encountered:

- Transfer data to FPGA memory
- Invoke binary
- Sets kernel arguments
- Launches hardware execution
---

**Runtime Flow**

- **Host-centric:** Execution starts on the host
  - If target region is encountered:
    - Transfer data to FPGA memory
    - Invoke binary
    - Sets kernel arguments
    - Launches hardware execution
    - Transfer data back to host

---

**Host**

- **Host binary**
  - **LLVM OpenMP Runtime**
    - **Libomptarget (device-agnostic)**
      - Loads & launches
        - **TPC-specific binary**
          - **TPC plugin**
            - **TPC API**
              - **Platform API**
                - **FPGA**
Evaluation

- Proof-of-concept implementation based on development version of LLVM/Clang/LLVM OpenMP Runtime
- Evaluation using integer BLAS kernels from Adept benchmark suite
  - AXPY
  - Vector scaling
  - Vector dot product
  - Dense matrix vector multiplication
  - Euclidean norm for vector
Evaluation

- Using Xilinx Vivado HLS 2016.4
  - Kernels annotated with Vivado HLS `pipeline` pragma
  - 250 MHz kernel operation frequency

```c
#pragma omp target
map(to:x[0:SIZE])
map(tofrom:y[0:SIZE])
{
  #pragma omp parallel for[...]
  for(i=0; i<SIZE; i++) {
    #pragma HLS PIPELINE II=1
    y[i] = a*x[i]+y[i];
  }
}
```
Evaluation

- FPGA-board: VC709 (Virtex 7), 4 GiB on-board memory

- Single instance of each kernel ("single-core")

- Comparison against x86-CPU (i7-6700K, 16 GiB DDR4-RAM) running 4 OpenMP threads
Insights

- Fully functional implementation of OpenMP offloading to FPGAs
  - Custom compilation flow mapping target region to hardware kernel without additional user input
  - Extension of runtime library based on ThreadPoolComposer API
Insights

- Fully functional implementation of OpenMP offloading to FPGAs
  - Custom compilation flow mapping target region to hardware kernel without additional user input
  - Extension of runtime library based on ThreadPoolComposer API
- Program a FPGA-based heterogeneous system with a single, portable code-base
Insights

• Fully functional implementation of OpenMP offloading to FPGAs
  – Custom compilation flow mapping target region to hardware kernel without additional user input
  – Extension of runtime library based on ThreadPoolComposer API

• Program a FPGA-based heterogeneous system with a single, portable code-base

• Offloading overhead primarily dependent on size of data transferred to/from device memory
Insights

- Fully functional implementation of OpenMP offloading to FPGAs
  - Custom compilation flow mapping target region to hardware kernel without additional user input
  - Extension of runtime library based on ThreadPoolComposer API
- Program a FPGA-based heterogeneous system with a single, portable code-base
- Offloading overhead primarily dependent on size of data transferred to/from device memory
- Pipelining results in 2x speedup over non-pipelined kernel
Insights

- Single PE execution slower than quad-core X86 (6.7x/3.4x)
Insights

- Single PE execution slower than quad-core X86 (6.7x/3.4x)
- Kernels very area-efficient (1-5% logic utilization)
Insights

- Single PE execution slower than quad-core X86 (6.7x/3.4x)
- Kernels very area-efficient (1-5% logic utilization)

Future work:
- Make use of coarse-grain parallelism (e.g., teams distribute)
- Distribute computation across multiple PEs
Questions?

- Stop by at our poster
- Get in touch: sommer@esa.tu-darmstadt.de

Download open-source release of ThreadPoolComposer:
https://goo.gl/qTsU3B