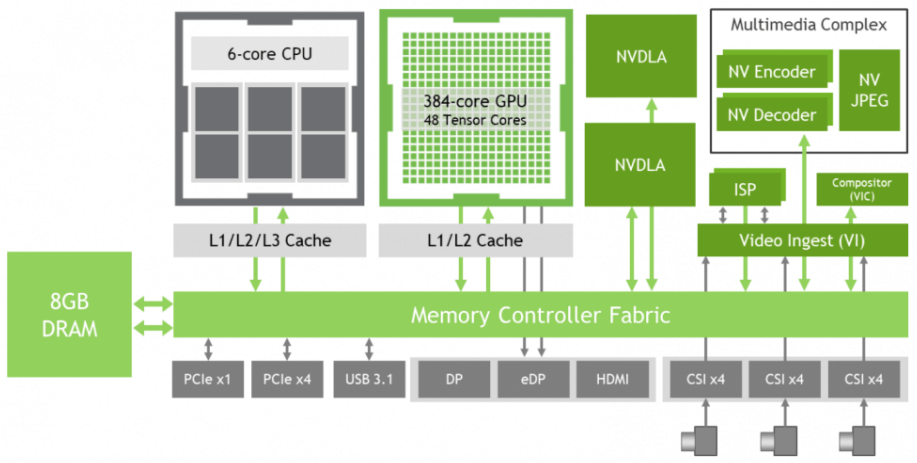
FOTV: A generic device offloading framework for OpenMP

Jose Luis Vázquez and Pablo Sánchez
University of Cantabria

Table of contents

Table of contents

Motivation: The landscape



Nvidia Jetson Xavier NX block diagram, https://developer.nvidia.com/blog/jetson-xavier-nx-the-worlds-smallest-ai-supercomputer/

- Increasing number of embedded heterogeneous architectures
  - Growing complexity
  - Covering new market segments
    - More accelerator types – FPGA, DSP, TPU...
    - New applications – Edge computing, mobile computing, multicore microcontrollers...
- New development model: Accelerator offloading
  - Multiple new and different frameworks

Motivation: OpenMP Limitations

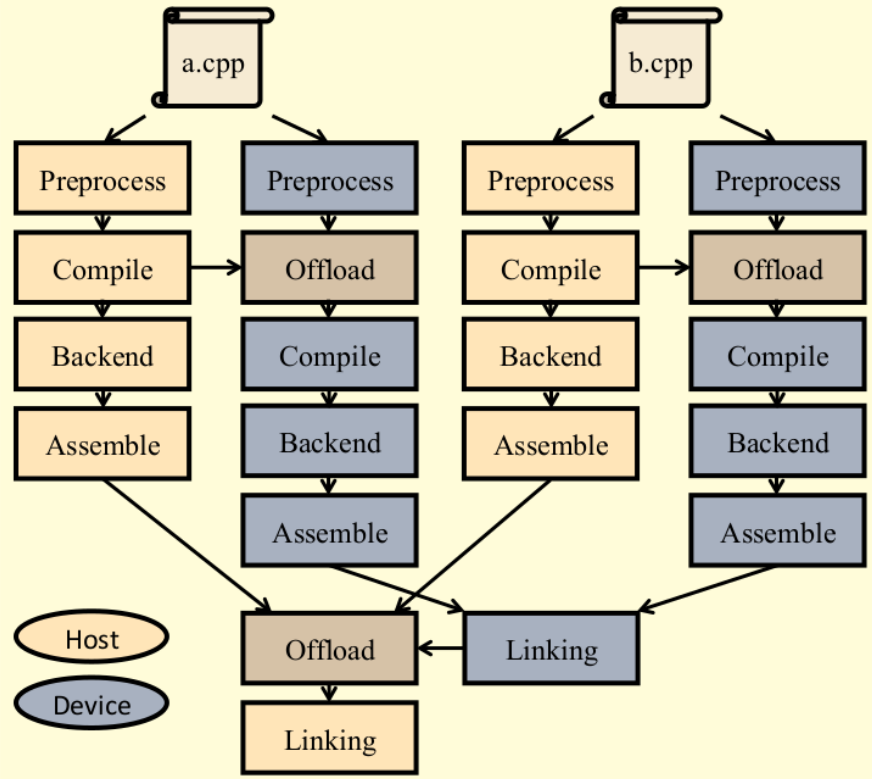
- OpenMP 4.0 introduces offload support through target directive
- Limitations of the approach
  - OpenMP requires target and host code generation in parallel
    - Support limited to host compiler support
      - New target support requires host compiler modification
      - Other frameworks only require target specific compilers
    - Target-specific optimizations limited to host compiler
      - Both device and host code are exactly the same
        - Example: CPU/GPU oriented code is normally inefficient for FPGA

Proposal: Future Offload Target Virtualization (FOTV)

- New generic OpenMP device
  - Not device specific
    - Facilitates new device OpenMP support
  - Supports external loading at runtime
  - Can contain multiple different non-natively supported devices
- Code extractor included in the target toolchain
  - Generates target region code and metadata
  - Facilitates the integration between OpenMP and external, device-specific toolchains

Table of contents

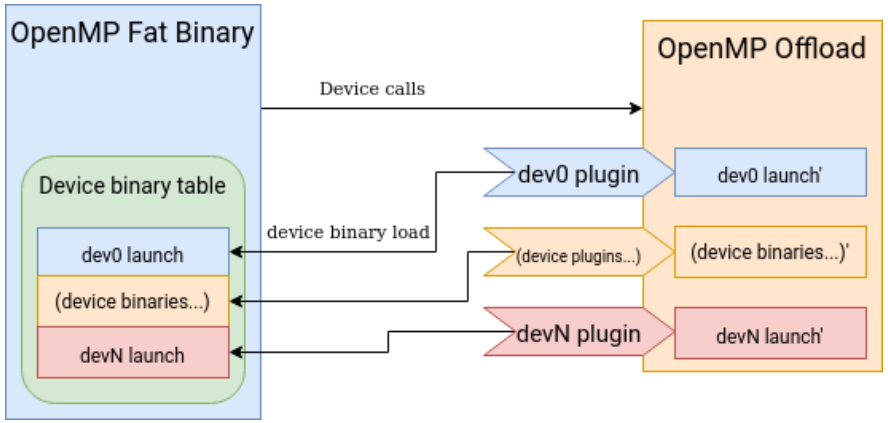
Traditional compilation process (Clang)



LLVM/Clang offloading compilation diagram, S. F. Antao et al. "Offloading Support for OpenMP in Clang and LLVM", LLVM-HPC2016, Salt Lake City, Utah, USA, November 13-18, 2016.

- Target code is generated in parallel to host code
  - Uses compiler IR as source – requisite interface
- A bundler wraps the target code with OpenMP offloading infrastructure before linking
  - Fixed format
  - Libomptarget plugins expect this format

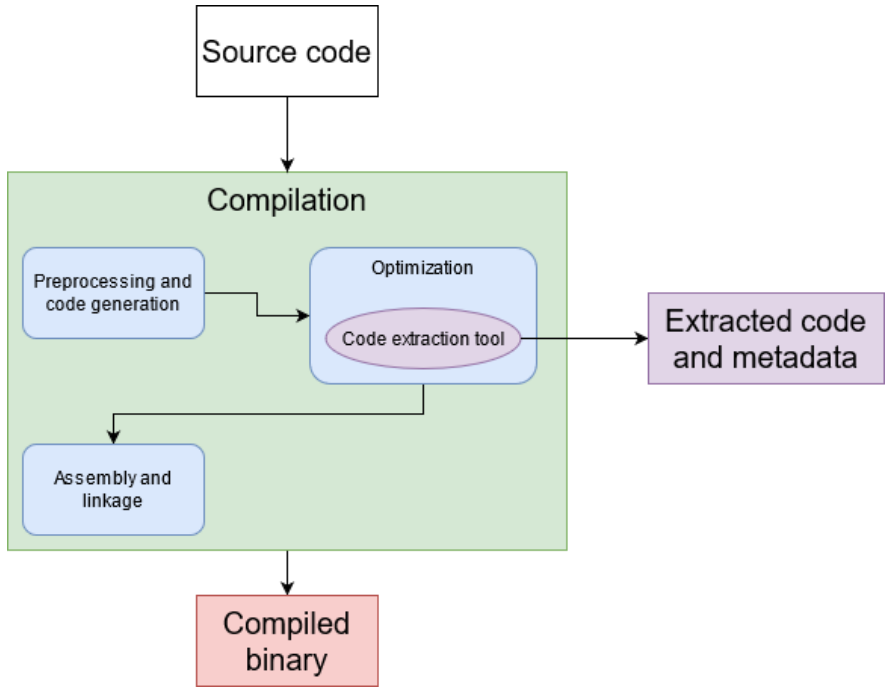
Traditional execution process



Standard OpenMP Offloading execution model. Device binaries load strictly from the fat binary through the device plugins

- Fat binary contains all target executables
  - Specific OpenMP format
  - Generated by bundler
- OpenMP infrastructure uses plugin libraries for device support
  - Uses the fat binary format
  - Plugin design flexibility limited by fat binary philosophy

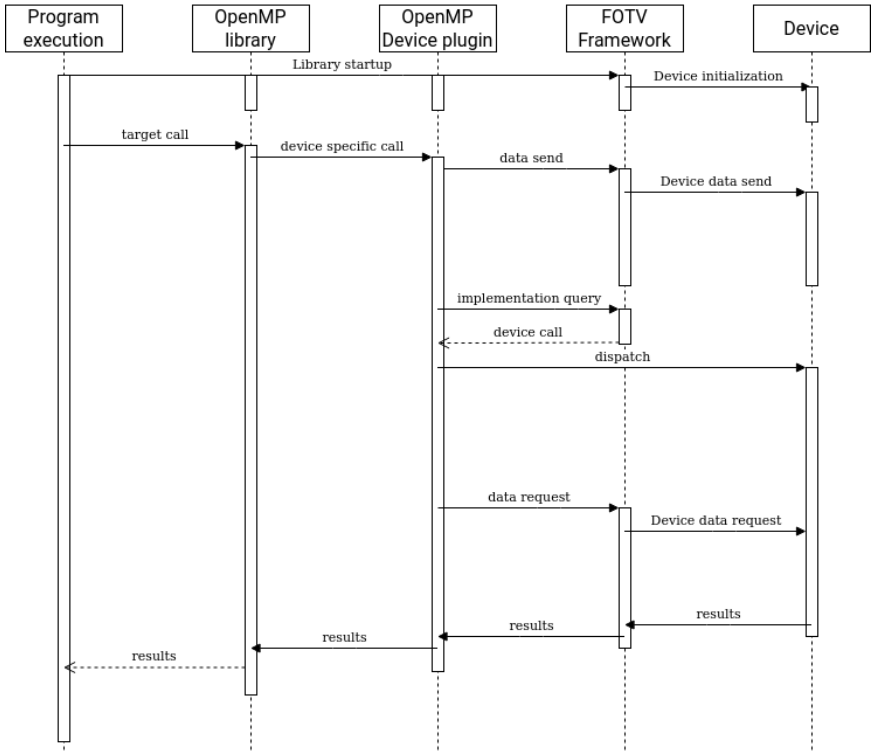
Proposed compilation process



Simplified compilation diagram with embedded extractor

- The proposed compilation process produces a standard binary and a number of code and metadata files
  - Code for target regions
  - Metadata for target region scope, shape, and target data regions
- Source code can serve as interface between the host compiler and the device specific tools
  - It can be optimized for specific devices

Proposed execution process



Runtime structure with the proposed framework

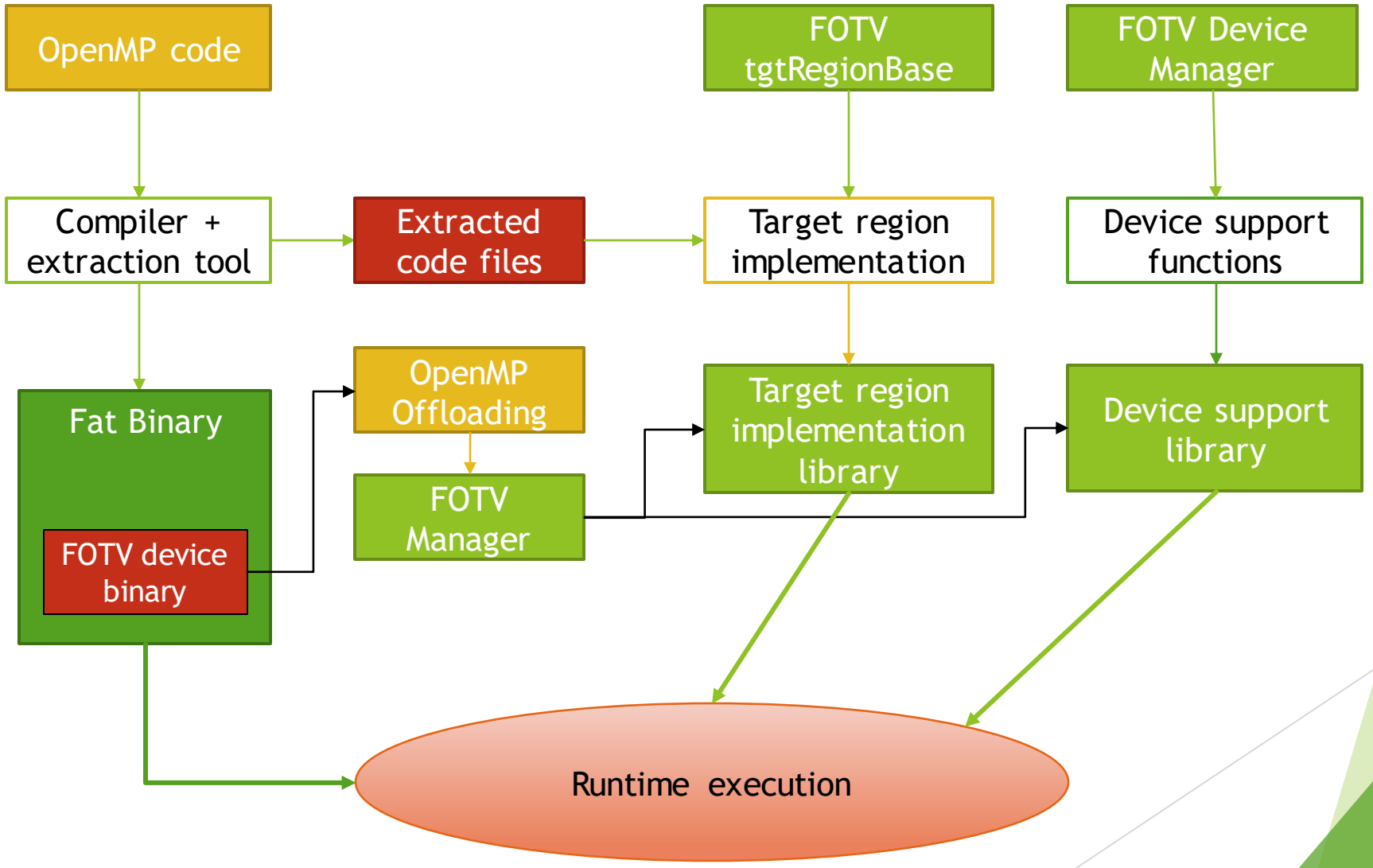
- OpenMP interfaces with runtime library
- Runtime library provides custom device support
  - Initialization, data management, implementation management
  - Can operate with other frameworks
- Runtime library provides implementation querying
  - Execution for the specific device configuration

Advantages to proposal

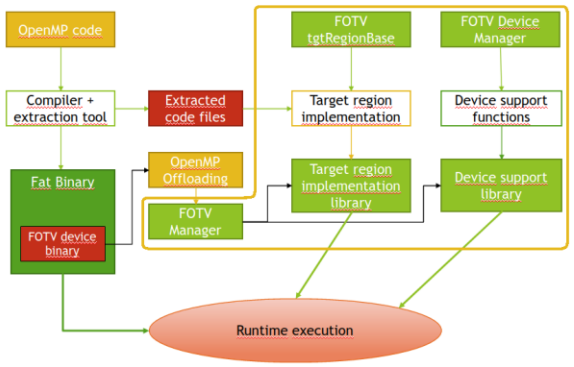
- Code extraction system gives easily tunable sources for device regions
- Runtime loading allows for off-line target device code generation
- No host compiler modifications beyond the FOTV framework
- New devices and regions can be implemented on demand

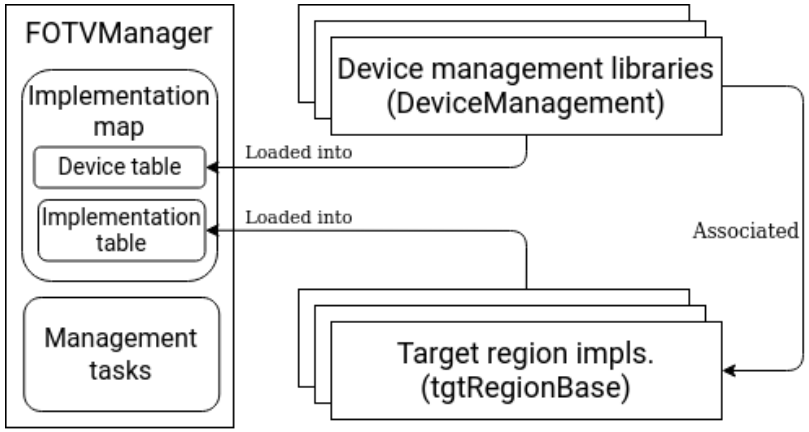
Table of contents

End-to-End architecture



The Runtime Library components

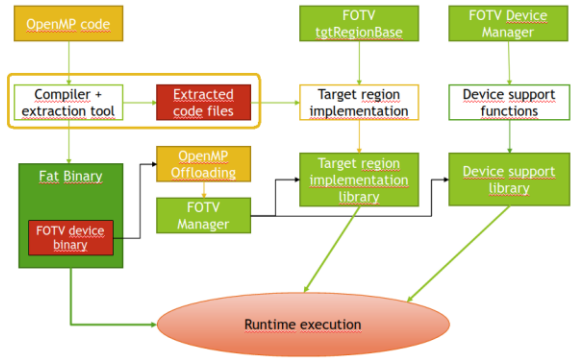




Runtime diagram. The central manager loads device libraries and implementation libraries and associates them through the device library registration routine.

- FOTVManager acts as central hub
  - Self-registered lists of devices, target region implementations and cached region-to-device associations
  - Device querying functions for the associated implementations
  - Interface between FOTV plugin and user-defined device functions
- Device Manager is the user-defined device support interface
  - Includes device management functions (registration, data management, startup/pause)
    - Also includes target implementation register
    - Extensible
- TgtRegionBase is an extensible target region wrapper
  - Includes registration info

The code extractor




```
"canny_region_l46":{
    "source_file":"../canny.cpp",
    "generating_pragma":"#pragma omp target map(to:img_gray[0:width*height], width,
    "pragma_start_line":46,
    "pragma_end_line":46,
    "region_start_line":47,
    "region_end_line":108,
    "map_clauses":[
        {
            "map_type":"to",
            "mapped_vars":"img_gray, width, height, sobel_x, sobel_y, gausian"
        },
        {
            "map_type":"tofrom",
            "mapped_vars":"img_edges"
        }
    ],
    "original_variables":[
        {
            "type":"unsigned char *",
            "name":"img_edges",
            "size":1,
            "isFirstPrivate":0
        },
```

Sample output from the extraction tool

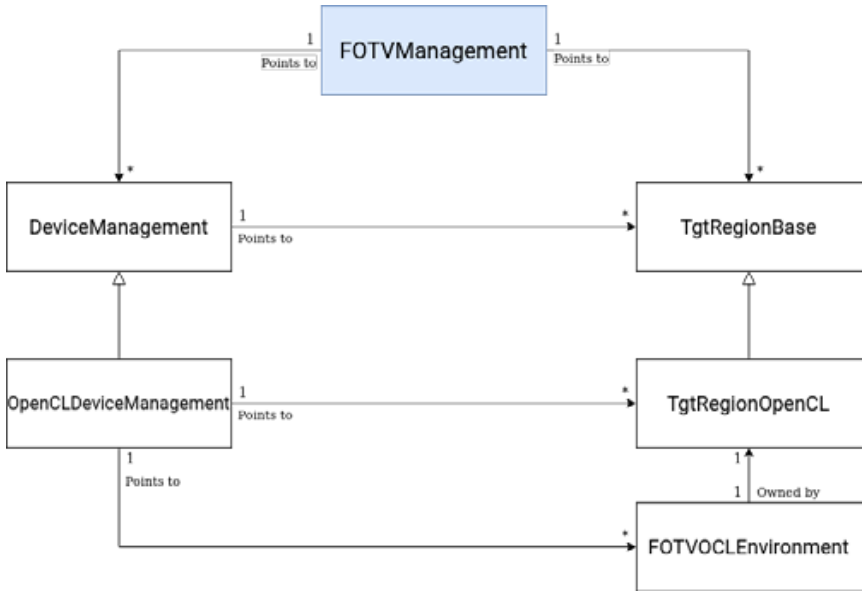
- Run as an optimization pass from the compiler
- Extracts source code and metadata from the region in a json format
  - Loop nesting level
  - Data mapping and direction
  - Function symbol (used for querying)
  - Source code location

Table of contents

Case study: OpenCL-based device

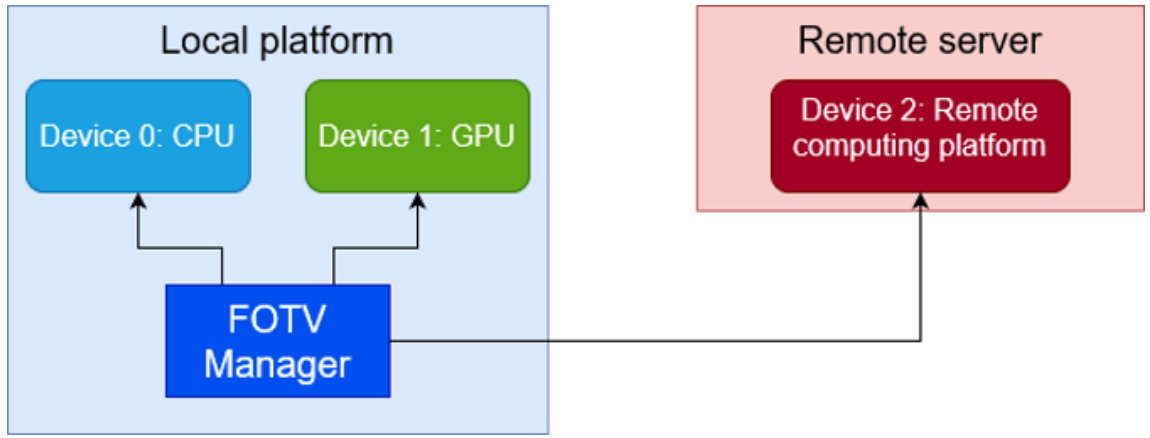
- Objective: Show flexibility and reduced overhead
- Device is implemented using an OpenCL backend
  - Goal is not to create an OpenMP to OpenCL interface
  - Showcases flexibility while being readily available
- Handles different OpenCL devices
  - Through testing we used a local GPU and a remote platform
- Benchmark was a Canny filter implemented with OpenMP

OpenCL requirements



Simplified FOTV + OpenCL diagram. The OpenCL specific structures inherit from the generic ones.

- The OpenCL device had a few quirks to iron out
  - Data movement functions are independent from variables -> difficults target region launch
  - OpenCL has its own event queue
  - OpenCL expects developers to keep track of how data regions map to kernel variables
- We implemented an extended device interface for this test case

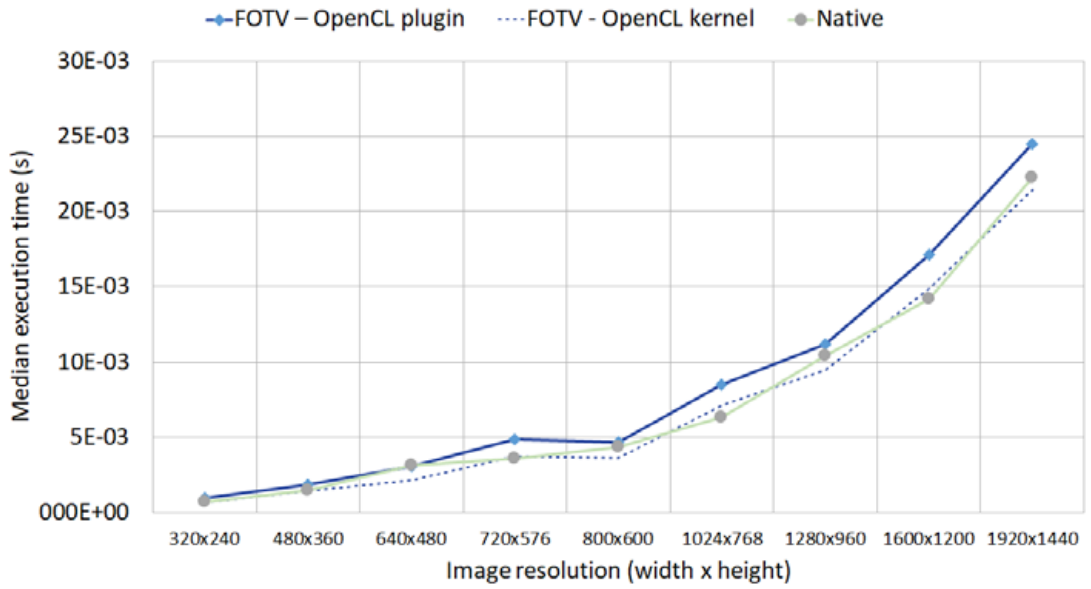



```
int y,x,index,acc;
unsigned char R,G,B,Y;

#pragma omp target teams distribute parallel for collapse(2)
    map(to:img_color[0:width*height*3], width, height) \
    map(tofrom:img_gray[0:width*height]) \
    device(dev)

for(y=0;y<height;y++) {
    for(x=0;x<width;x++) {

        index = (y*width)+x;

        B = img_color[3*index];
        G = img_color[3*index+1];
        R = img_color[3*index+2];

        acc  = (int)67  * (int)R;
        acc += (int)174 * (int)G;
        acc += (int)15  * (int)B;
        acc /= 256;
        Y = (unsigned char)acc;

        img_gray[index] = Y;
    }
}
```

Test platform

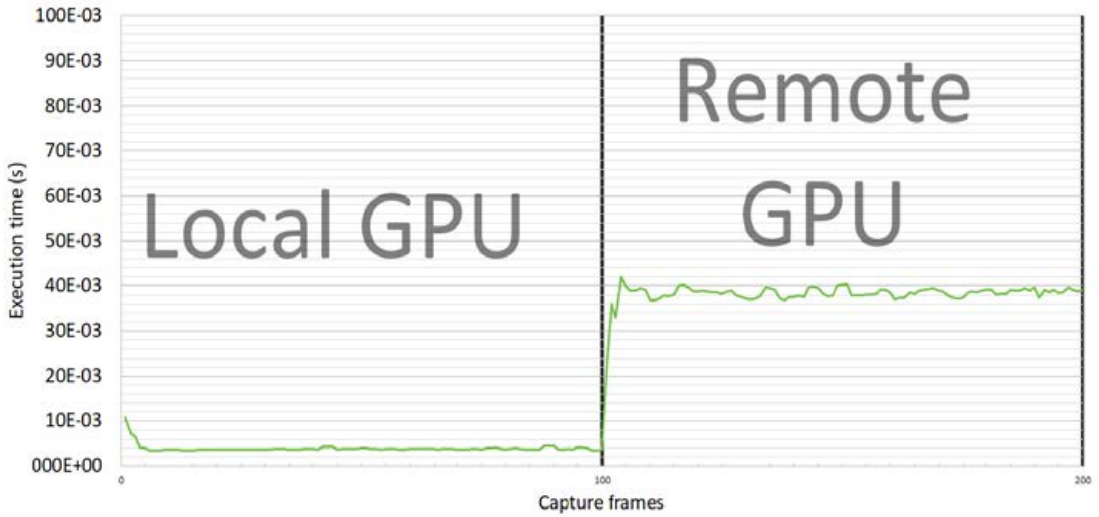
- At the top, platform diagram
  - Devices presented with their OpenMP device ID
- At the bottom, example code for a grayscale filter
  - Dev variable represents reconfiguration
  - Follows the OpenMP IDs

Execution results: Performance



- Execution time for target region of our Canny filter
  - Native = CUDA-based nVidia offload plugin
- Shows low overhead, primarily induced by OpenCL data movement
  - < 10% including OpenCL memory transfers
- At certain problem sizes the differences minimize
  - OpenCL implementation quirks

Execution results: Flexibility



- Execution of Canny filter flips over to remote execution after 100 frames
- Reconfiguration is near-instant
- Spike in execution time caused by device "warmup"
  - Runtime caching of implementation
  - Code loading in remote GPU
- Increase in execution time caused by server and network load

Table of contents

Conclusions

- We introduce a new OpenMP offloading target:
  - Interfaces with a runtime library to provide generic device support
  - Supports otherwise unsupported offload devices
- FOTV allows for device and target region flexibility
  - User-defined device support
  - Device-specific target region tuning possible
- FOTV incurs in minimum overhead against other plugins

Future work

- FOTVManager extension to support automatic load balancing
  - Option to expose a whole platform as a single OpenMP device
- Code extractor extension to automatically generate default implementations
  - Example: Full OpenCL device implementation with kernel code