Extending HPCToolkit for GPU-accelerated Systems

John Mellor-Crummey
Department of Computer Science
Rice University
johnmc@rice.edu
Outline

- OpenMP 4.5 - 5.0
- OMPT API for accelerators
- OMPT implementation with accelerators
- HPCToolkit
  - interface with accelerator programming models
  - measurement
  - attribution
  - code-centric presentation
- Unexpected Challenges
- Remaining work
  - HPCToolkit
  - libomptarget
OpenMP 4.5 and OpenMP 5.0

- Offload computation to accelerators
- Avoid data movement for each target construct

```c
Example target_data.3.c
#include <math.h>
#define COLS 100
void gramSchmidt(float Q[][COLS], const int rows)
{
    int cols = COLS;
    #pragma omp target data map(Q[0:rows][0:cols])
    for(int k=0; k < cols; k++)
    {
        double tmp = 0.0;
        #pragma omp target map(tofrom: tmp)
        #pragma omp parallel for reduction(+:tmp)
        for(int i=0; i < rows; i++)
            tmp += (Q[i][k] * Q[i][k]);
        tmp = 1/sqrt(tmp);
        #pragma omp target
        #pragma omp parallel for
        for(int i=0; i < rows; i++)
            Q[i][k] *= tmp;
    }
}
```

Figure credit: OpenMP Standards Committee, OpenMP Application Programming Interface Examples. Version 4.5.0, November 2016.
OpenMP 5 API for Target Devices

- Device-independent host callbacks for target devices
  - `ompt_callback_device_initialize`
  - `ompt_callback_device_load`
  - `ompt_callback_target_enter/exit target region`
  - `ompt_callback_target_map`
  - `ompt_callback_target_data_op alloc delete transfer_to_device transfer_from_device`
  - `ompt_callback_target_submit launch kernel`
  - `ompt_callback_device_unload`
  - `ompt_callback_device_finalize`

- Device-specific API for target devices

<table>
<thead>
<tr>
<th>Entry Point String Name</th>
<th>Type Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;ompt_get_device_num_procs&quot;</td>
<td>ompt_get_device_num_procs_t</td>
</tr>
<tr>
<td>&quot;ompt_get_device_time&quot;</td>
<td>ompt_get_device_time_t</td>
</tr>
<tr>
<td>&quot;ompt_translate_time&quot;</td>
<td>ompt_translate_time_t</td>
</tr>
<tr>
<td>&quot;ompt_set_trace.ompt&quot;</td>
<td>ompt_set_trace.ompt_t</td>
</tr>
<tr>
<td>&quot;ompt_set_trace.native&quot;</td>
<td>ompt_set_trace.native_t</td>
</tr>
<tr>
<td>&quot;ompt_start_trace&quot;</td>
<td>ompt_start_trace_t</td>
</tr>
<tr>
<td>&quot;ompt_resume_trace&quot;</td>
<td>ompt_resume_trace_t</td>
</tr>
<tr>
<td>&quot;ompt_flush_trace&quot;</td>
<td>ompt_flush_trace_t</td>
</tr>
<tr>
<td>&quot;ompt_stop_trace&quot;</td>
<td>ompt_stop_trace_t</td>
</tr>
<tr>
<td>&quot;ompt_advance_buffer_cursor&quot;</td>
<td>ompt_advance_buffer_cursor_t</td>
</tr>
<tr>
<td>&quot;ompt_get_record.type&quot;</td>
<td>ompt_get_record.type_t</td>
</tr>
<tr>
<td>&quot;ompt_get_record.ompt&quot;</td>
<td>ompt_get_record.ompt_t</td>
</tr>
<tr>
<td>&quot;ompt_get_record.native&quot;</td>
<td>ompt_get_record.native_t</td>
</tr>
<tr>
<td>&quot;ompt_get_record.abstract&quot;</td>
<td>ompt_get_record.abstract_t</td>
</tr>
</tbody>
</table>
OpenMP 5 Implementation Requirements

- Works with or without a tool that supports OMPT
- Works with tool support for OMPT
  - enabled
  - disabled
- OpenMP implementation strategies require demand-driven implementation
  - clang-generated heterogeneous binaries
    - constructor prior to main loads code onto device using libomptarget
GPU-accelerated OpenMP 5 Application

(Host executable and/or libraries contain CUBINs)

libomptarget

libomptarget.nvptx

libcuda

libomp

libomp

OMPT

OMPT

OMPT

OMPT

libcupti

Linux

HPCToolkit’s libhpcrun
OMPT Initialization for Accelerators

<table>
<thead>
<tr>
<th>tool</th>
<th>application</th>
<th>libomp</th>
<th>libomptarget</th>
<th>libomptarget.nvptx</th>
</tr>
</thead>
<tbody>
<tr>
<td>__lib_csu_init</td>
<td></td>
<td>calls __tgt_register_lib</td>
<td>initiates demand loading</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>target_register_lib</td>
<td>of plugins</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>libomp_libomptarget</td>
<td>force OpenMP initialization</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls lot_init</td>
<td>calls lot_init</td>
<td></td>
</tr>
<tr>
<td>ompt_start_tool</td>
<td></td>
<td>ompt_pre_init</td>
<td>calls tool_initializer</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>tool_initializer(lump_lookup)</td>
<td>calls tool_device_load</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>lomp_lookup(...)</td>
<td>calls tool_device_initialize</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>lot_init(lump_lookup)</td>
<td>obtain all device</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>__tgt_rtl_init_device</td>
<td>callbacks from libomp</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>libomptarget_rtl.ompt_init</td>
<td>calls libomptarget_rtl.ompt_init</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>calls pl_init(lump_lookup)</td>
<td>(pl_init, pl_fini)</td>
<td></td>
</tr>
<tr>
<td>tool_device_initialize</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tool_device_load</td>
<td></td>
<td>__tgt_rtl_load_binary</td>
<td>calls tool_device_load_callback</td>
<td></td>
</tr>
</tbody>
</table>
GPU-accelerated Application using CUDA
(Host executable and/or libraries contain CUBINs)

libomptarget

libomptarget.nvptx

libcuda

libcupti

OMPT

OMPT

OMPT

libomp

HPCToolkit’s libhpcrun

Linux
HPCToolkit Measurement of GPUs

- Registers for callbacks associated with target devices
  - device control
    - device_initialize/finalize
    - device_load/unload
  - target operations
    - target_region, target_submit, target_data_op
  - buffer_request/complete

- Computes non-overlapping relocation of CUBIN functions

- Adds CUBINs to the load map

- Processes buffer of events delivered by CUPTI Activity API
  - PC samples: relocates PCs to facilitate source correlation
  - kernel invocations
  - explicit data copies
  - implicit data copies (page faults)

- Correlates with context using CUPTI external correlation ids
HPCToolkit Attribution

• HPCToolkit’s hpcstruct performs binary analysis of heterogeneous binaries
  — host binary
  — embedded CUBIN segments

• Analysis of CUBINs
  — relocates functions so that they are non-overlapping
  — recovers program structure
    – inlined code and line map for unoptimized binaries (with -G)
    – line map only for optimized binaries (with —generate-line-info)
  — associates structure with code addresses
    – handles both unoptimized and optimized CUBINs

• Produces program structure file
  — load module for host
  — load module for each cubin
  — each load module contains
    – files, functions, inlined functions, statements
Code-Centric Attribution for OpenMP
Code-Centric Attribution for CUDA
Unexpected Challenges - I

• Challenge: extra threads
  — CUDA helper thread
  — CUPTI helper threads
    – CUPTI spawns a pthread every time it launches a kernel coordinate measurement of asynchronous operations?

• Approach
  — modify HPCToolkit’s libmonitor to record return address associated with pthread_create call
  — ignore a thread spawned by any of NVIDIA’s libraries
    – recognize libraries by an API function they supply rather than by name
Unexpected Challenges - II

• Large overhead for PC Sampling with CUPTI

• Assessing the situation
  — Test case: LLNL’s rajaperfsuite
    – uses RAJA portability layer to offload kernels to a GPU
  — Observe overhead for turning on the CUPTI Activity API to measure GPU performance using PC Sampling
CUPTI User Space Overhead for PC Sampling

- memset added to CUDA launch to support PC Sampling with CUPTI accounts for 28% of total execution time
CUPTI Kernel Overhead for PC Sampling

- `nv_alloc_system_pages` added to CUDA launch to support PC Sampling with CUPTI accounts for 42% of total execution time.
CUPTI Kernel Overhead for PC Sampling
Remaining Work: HPCToolkit

• **hpcrun**
  - upgrade OMPT support from TR4 to OpenMP 5 standard
    asynchronous assembly of calling contexts mediated by wait-free operations on data structures
  - integrate GPU support to allow both CUDA and OpenMP 5 in the same execution
  - add support for sample-based tracing of GPU activity
  - complete support for sparse metric sets
    - many GPU metrics
    - few nodes in CCT have GPU metrics
    - goal: avoid space cost of empty GPU metrics almost everywhere
  - test support for OpenACC

• **hpcstruct**
  - integrate support for parsing dot CFGs for NVIDIA CUBINs
    - enable us to attribute GPU kernel performance at the loop level
  - compute approximate call tree on GPUs
    - when there is a single call to a function, know its calling context
    - when there are multiple calls, proportionally attribute cost to callers

• **hpcviewer**
  - needs top-down support for analyzing GPU metrics

• **hpctraceviewer**
  - needs support for displaying traces of GPU kernel executions
Remaining Work: libomptarget

- Refine OMPT support for use of libomptarget without OpenMP
- Upstream changes to libomptarget
- Hand off OMPT GPU support to IBM for direct integration into LOMP
Unmet Needs from NVIDIA

- **API for unpacking .nv_fatbin segments**
  - NVIDIA has refused to provide header file or API
  - Complicates binary analysis of heterogeneous binaries constructed with NVIDIA nvcc
    - CUDA and OpenACC

- **API for computing control flow graphs for CUBINs**
  - Currently, execute nvdisasm and parse its output

- **CUPTI Activity API for PC sampling has significant overhead**
  - Long time spent initializing memory (profile buffers?) in both user space and the kernel when PC sampling is enabled

NVIDIA has committed to working on this one for Volta