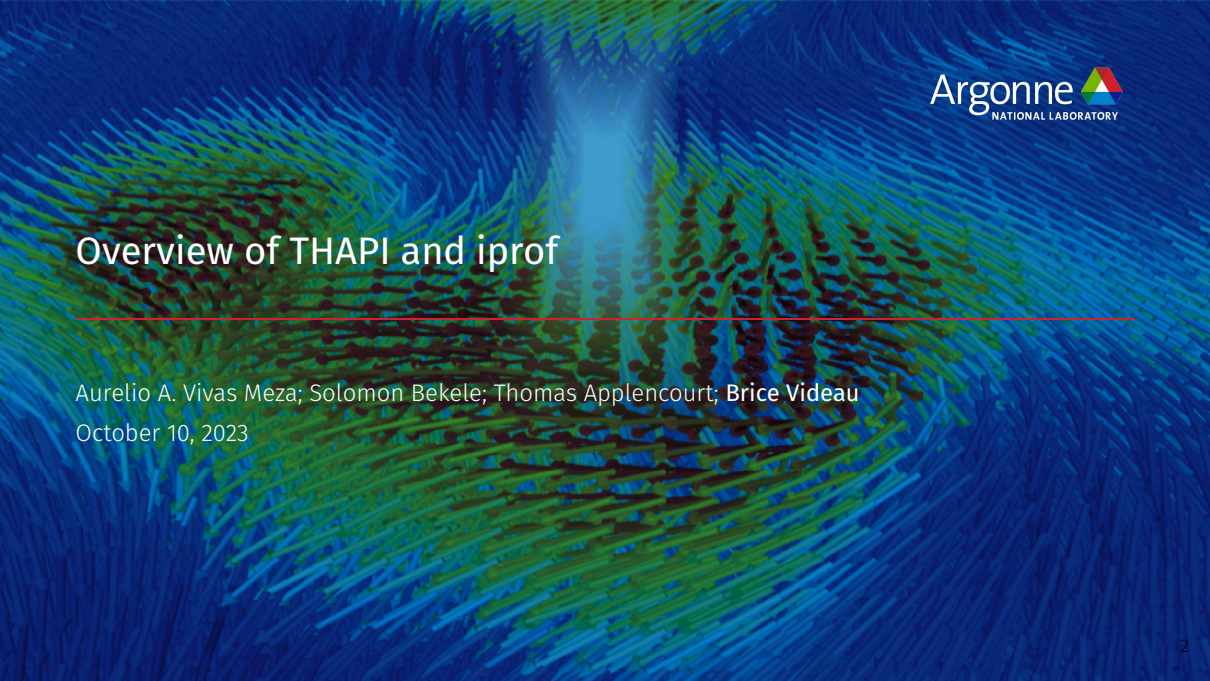
October 10-12, 2023

ALCF Hands-on HPC Workshop



Overview of THAPI and iprof

Aurelio A. Vivas Meza; Solomon Bekele; Thomas Applencourt; **Brice Videau**

October 10, 2023

Table of Contents

Intro

We work with HPC applications that are highly parallel, distributed, but that also leverage accelerators such as GPUs. Programming languages and models to implement these HPC applications have never been more diverse:

Languages

- FORTRAN
- C
- C++
- Python

Prospective languages

- Julia
- Lua
- PGAS approaches

Programming models

- MPI
- OpenMP
- CUDA, L0, ROCm, HIP, OpenCL
- SYCL, Kokkos, Raja

Domain Based Programming Models

- Linear algebra: BLAS/LAPACK
- FFTs: cuFFT, FFTWx, mkl FFT
- Low level AI: cuDNN, clDNN, Intel DNNL
- AI/ML: TensorFlow/Caffe/PyTorch

Problematic

This plethora of alternatives are entwined, especially since heterogeneous computing is the norm.

Possible Dependencies

- SYCL:
  - HIP
  - OpenCL
  - L0
- OpenMP:
  - OpenCL
  - CUDA
  - L0
- OpenCL:
  - L0
  - CUDA
- HIP:
  - CUDA
  - OpenCL
  - ROCm
  - L0
- Kokkos
  - OpenMP
  - CUDA
  - SYCL
- ...

Objectives

Introspect Applications and programming models

- Analyze applications based on those models;
- Understand application performances;
- Understand interactions between applications / compilers / run-times / system / hardware;
- Influence/optimize application at any point:
  - writing,
  - optimization,
  - execution.

Examples

- How programming models are implemented on top of each other?
  - How OpenMP nowait are implemented in LLVM?
- How applications are using programming models?
  - What is the maximum memory allocated by my program on the GPU?

Solution: Tracing

- Trace as many programming models as possible
  - Trace should capture as much context as possible, and be lightweight as possible
- Develop tools to analyze traces
  - Summary, timeline, etc...
- Modular code architecture
  - Ease the implementation of new “front-end”
- Solution needs to be efficient, robust, and scalable, but maybe not all at the same time :)
  - Capture millions of events per second per node;
  - Run for hours;
  - Up to 10,624 nodes.

THAPI: Tracing Heterogeneous APIs

Programming-Model Centric Debugging / Tracing

Traces should contain enough information to reconstruct the programming model state.

Traces can be:

- Tallied to give high-level summary
- Used to generate flame-graphs
- Used to check valid usage of programming model
  - Check for error code
  - Correct synchronization
  - API semantics
- Analyzed using dedicated tools
- Input for simulation frameworks

THAPI Principles

- Programming-Model centric tracing
  - Save arguments and results of each runtime entry points

```
18:56:59.677295870 - arc03 - vpid: 37040, vtid: 37040
   - lttng_ust_ze:zeKernelSetIndirectAccess_entry:
      { hKernel: 0x0000000002cd2b20, flags: [ ZE_KERNEL_INDIRECT_ACCESS_FLAG_DEVICE ] }
18:56:59.677296042 - arc03 - vpid: 37040, vtid: 37040
   - lttng_ust_ze:zeKernelSetIndirectAccess_exit:
      { zeResult: ZE_RESULT_SUCCESS }
```

- Flexible
  - Fine granularity, you can enable/disable individual events tracing,
  - Trace can be read programmatically (C, Python, Ruby),
  - We provide tools calibrated to our needs as starting-blocks.
- Low/Reasonable overhead

THAPI Consists of Two Big Components

Open source at: *https://github.com/argonne-lcf/THAPI*

- The tracing of events
  - Use low level tracing: Linux Tracing Toolkit Next Generation (LTTng):
  - Tracepoints are generated from APIs' headers
- The parsing of the trace
  - Use Babeltrace2 library and tools
  - Pretty Printer, Tally, Timeline/Flamegraph, ...

Supported APIs

- OpenCL, Level Zero, Cuda Runtime/Driver, HIP
- OMPT

iprof is an orchestrator around THAPI, lttng, and babeltrace2.

Automatic LTTng Tracepoint Generation

- We trace all APIs entry points (OpenCL, CUDA, Level Zero, HIP) or OMPT callbacks
  - Tracing using interception library
  - We also support sampling of user-events
- Tedious, error prone, and hard to maintain by hand
- Automatic generation from headers or API description (OpenCL)
  - C99 parser => YAML intermediary representation
  - YAML + user provided meta information + user provided tracepoints => wrapper functions + Trace Model
  - Trace Model => tracepoints

iprof Showcase

THAPI Examples: `iprof -t ./a.out`

Wrapping the API entry points to be able to reconstruct the context.

```
./iprof -t ./a.out
```

```
ompt_callback_target:
  { kind: ompt_target, endpoint: ompt_scope_end, device_num: 0, task_data: 0x0000000000000000,
    target_id: 1, codeptr_ra: 0x00007f5b26fa47e0 }
# ...
ompt_callback_target_data_op_intel:
  { endpoint: ompt_scope_begin, target_id: 1, host_op_id: 7,
    optype: ompt_target_data_transfer_to_device,
    src_addr: 0x00007f5b20088280, src_device_num: -10, dest_addr: 0xffffc001ffd80000,
    dest_device_num: 0, bytes: 131072, codeptr_ra: 0x00007f5b26fa47e0 }
clEnqueueMemcpyINTEL_entry:
  { command_queue: 0x181a540, blocking: CL_FALSE,
    dst_ptr: 0xffffc001ffd80000, src_ptr: 0x00007f5b20088280, size: 64,
    num_events_in_wait_list: 0,
    event_wait_list: 0x0, event: 0x7ffc4ac01378, event_wait_list_vals: [] }
clEnqueueMemcpyINTEL_exit:
  { errcode_ret_val: CL_SUCCESS, event_val: 0x1dffb30 }
ompt_callback_target_data_op_intel:
  { endpoint: ompt_scope_end, target_id: 1, host_op_id: 7,
    optype: ompt_target_data_transfer_to_device,
    src_addr: 0x00007f5b20088280, src_device_num: -10, dest_addr: 0xffffc001ffd80000,
    dest_device_num: 0, bytes: 131072, codeptr_ra: 0x00007f5b26fa47e0 }
```

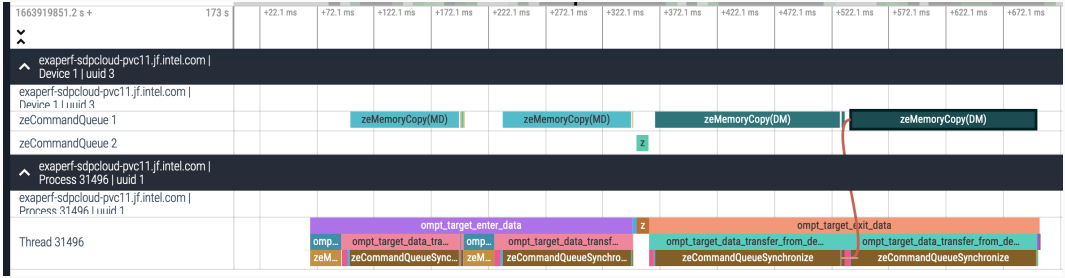
THAPI Examples: iprof

```
$iprof ./target_teams_distribute_parallel_do.out # Using Level0 backend
Trace location: /home/tapplencourt/lttng-traces/iprof-20210408-204629
BACKEND_OMP | 1 Hostnames | 1 Processes | 1 Threads |
       Name |    Time | Time(%) | Calls | Average |    Min |    Max |
ompt_target | 3.65ms | 100.00% |     1 |  3.65ms | 3.65ms | 3.65ms |
      Total | 3.65ms | 100.00% |     1 |

BACKEND_OMP_TARGET_OPERATIONS | 1 Hostnames | 1 Processes | 1 Threads |
                                  Name |     Time | Time(%) | Calls |  Average |      Min |      Max |
                 ompt_target_data_alloc |   1.97ms |  54.19% |     4 | 491.63us |    847ns |   1.12ms |
  ompt_target_data_transfer_to_device |   1.26ms |  34.63% |     5 | 251.37us | 112.60us | 460.90us |
ompt_target_data_transfer_from_device | 250.76us |   6.91% |     1 | 250.76us | 250.76us | 250.76us |
               ompt_target_submit_intel | 155.04us |   4.27% |     1 | 155.04us | 155.04us | 155.04us |
[...]
                                Total |   3.63ms | 100.00% |    11 |

BACKEND_ZE | 1 Hostnames | 1 Processes | 1 Threads |
                               Name |     Time | Time(%) | Calls |  Average |      Min |      Max |
                     zeModuleCreate | 846.26ms |  96.89% |     1 | 846.26ms | 846.26ms | 846.26ms |
      zeCommandListAppendMemoryCopy |  10.73ms |   1.23% |    12 | 893.82us |  12.96us |   5.33ms |
[...]
                              Total | 873.46ms | 100.00% |   117 |

Device profiling | 1 Hostnames | 1 Processes | 1 Threads | 1 Devices |
                                 Name |     Time | Time(%) | Calls | Average |     Min |     Max |
                        zeMemoryCopy(DM) |  64.48us |   7.14% |     1 | 64.48us | 64.48us | 64.48us |
__omp_offloading_33_7d35e996_MAIN___l9 |  27.84us |   3.08% |     1 | 27.84us | 27.84us | 27.84us |
[...]
                                Total | 902.72us | 100.00% |    13 |
```


Argonne NATIONAL LABORATORY

Timeline visualization

Use perfetto/chrome protobuf trace format



Conclusion

Conclusion

- Programming Model Centric
- Support multiple backends
  - OpenCL, Level Zero, Cuda Runtime/Driver, HIP
  - OMPT
- Trace Analysis
  - tally
  - timeline
  - pretty printing
- Deployed on ALCF systems and/or easily installation from Spack
- Open to collaboration to add new backend and new tools