Testing a Library that Supports Different Target Architectures with GitLab CI

CASUS · FWUS · Simeon Ehrig · s.ehrig@hzdr.de · www.hzdr.de

Who am I?

- Simeon Ehrig
- CASUS Professional Support
- Computer Scientist
  - Specialized in GPU development
- Working with FWKT on alpaka

What is my Framework?

- The alpaka library is a header-only C++17 abstraction library for accelerator development.
- Writes code one time, execute it on different processors.
- Uses vendor SDKs to compile C++ code to target platform (OpenMP, Intel TBB, Nvidia CUDA, AMD HIP ...)

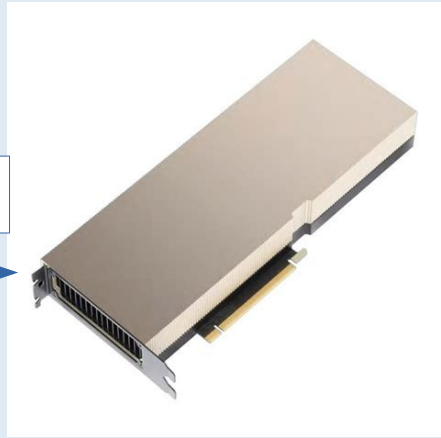
CPU

FPGA

```
struct InitRandomKernel
{
    template<typename TAcc, typename TExtent, typename TRandEngine>
    ALPAKA_FN_ACC auto operator()(
        TAcc const& acc,
        TExtent const extent,
        TRandEngine* const states,
        std::size_t pitchRand) const -> void
    {
        auto const idx = alpaka::getIdx<alpaka::Grid, alpaka::Threads>(acc);

        if(idx[0] < NUM_Y && idx[1] < NUM_X)
        {
            auto const linearIdx = alpaka::mapIdx<1u>(idx, extent)[0];
            auto const memoryLocationIdx = idx[0] * pitchRand + idx[1];
            TRandEngine engine(42, static_cast<std::uint32_t>(linearIdx));
            states[memoryLocationIdx] = engine;
        }
    }
};
```

GPU



What is my Challenge?

- Support differnt software SDKs for the different target architectures
- Only a subset of test parameter
  - 4 GCC versions
  - 6 Clang versions
  - 10 CUDA SDK version
  - 4 CMake versions
  - 7 Boost Versions
- 2800 combinations
- 6 Minutes per job
- 280h test time or 9,5h if we run 30 jobs in paralell

Solution – Job Generator

```yaml
stages:
  - generator
  - run-test-jobs

generate:
  stage: generator
  image: alpine:latest
  script:
    - apk update && apk add python3~=3.10 py3-pip
    - pip3 install pyaml
    - python generator.py conf1 > child.yml
  artifacts:
    paths:
      - child.yml
    expire_in: 1 week

run-child:
  stage: run-test-jobs
  trigger:
    include:
      - artifact: child.yml
        job: generate
    strategy: depend
```

```yaml
linux_nvcc11.4-gcc11_compile_only_cmake3.22.6_boost1.82.0
  image: registry.hzdr.de/crp/alpaka-group-container/alpa
  script:
  - source ./script/gitlabci/print_env.sh
  - source ./script/gitlab_ci_run.sh
  tags:
  - x86_64
  - cpuonly
  variables:
    ALPAKA_BOOST_VERSION: 1.82.0
    ALPAKA_CI_CMAKE_VER: 3.22.6
    CMAKE_CUDA_ARCHITECTURES: '61'
    CMAKE_CUDA_COMPILER: nvcc
    CXX: g++
    alpaka_ACC_CPU_B_TBB_T_SEQ_ENABLE: 'OFF'
    alpaka_ACC_GPU_CUDA_ENABLE: 'ON'


linux_hipcc5.5_compile_only_cmake3.26.4_boost1.80.0_ubunt
  image: registry.hzdr.de/crp/alpaka-group-container/alpa
  script:
  - source ./script/gitlabci/print_env.sh
  - source ./script/gitlab_ci_run.sh
  tags:
  - x86_64
  - cpuonly
  variables:
    alpaka_ACC_GPU_CUDA_ENABLE: 'OFF'
    alpaka_ACC_GPU_CUDA_ONLY_MODE: 'OFF'
    alpaka_ACC_GPU_HIP_ENABLE: 'ON'
    alpaka_ACC_GPU_HIP_ONLY_MODE: 'OFF'
```

Features of the job generator

- Use pair-wise combination to create spare test-job-matrix
- Job scheduling
  - Distribute jobs in waves (GitLab CI stages) to not fully utilize the CI and gave other PR/projects a chance to be executed
  - Run only a few jobs in the GPU runner -> most of the jobs compile tests on the CPU runner
- Select prebuild images from the registry
- Filter and reorder jobs via Git message
- Add special jobs (sanitizer jobs, integration jobs)

Questions

- Does anyone have similar problems with their project?
- Does anyone know existing solutions?
- Is anyone interested in integrating the CI functions into their project? -> Share work :-)