Packaging HEP heterogeneous mini-apps for portable benchmarking and facility evaluation on modern HPCs

Mohammad Atif[1], Pengfei Ding[2], Ka Hei Martin Kwok[3], Charles Leggett[2]

[1]Brookhaven National Laboratory, Upton, NY 11973, USA
[2]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
[3]Fermi National Accelerator Laboratory, Batavia, IL 60510, USA

High Energy Physics - Center for Computational Excellence
HEP-CCE

https://www.anl.gov/hep-cce

Introduction

We present two methods of deploying turn-key test applications, where by means of containerization and automated configuration and build techniques such as spack, we are able to quickly test new hardware, software, environments and entire facilities with minimal user intervention, and then track performance metrics over time.

Spack

- Spack is a package manager that supports multiple package versions, platforms and compilers
- Designed to handle software distribution in the complex HPC eco-system
  - Heavy use of system packages, optimized for particular HPC system
  - Often build many variants of the same package
- Large user-base with detailed documentation available

Containerization

- Containerization using technologies such as docker or podman allows projects to be compiled against pre-built dependencies
- By providing several base container images based on different GPU programming models, such as CUDA, HIP, Kokkos and SYCL, and GPU architectures, such as AMD's MI100 and NVIDIA's A100 GPU, we can rapidly deploy the same code to run on different platforms.

P2R with Spack

- p2r implementations
  - Advantage: p2r has various implementations that covers broad range of technologies, and all 3 main GPU types.
  - Light-weight program with minimal dependencies makes it suitable for spack.
- Spack implementation
  - Implemented as a CMake package
  - Spack variants
    Used 2 multi-valued variant options: `impl={cuda,kokkos,alpaka,stdpar,sycl}` and `backend={nvidia}` to specify implementation and backends
  - p2r options
    - Allow control of program duration by exposing controls over `nevts`, `ntrks`, `bsize` and `NITER`
    - Detailed compiler options in each `impl` and `backend` combination are hidden from for simplicity and reproduciblilty.
- External packages
  - p2r depends on vendor software (e.g. CUDA, `nvc++`) or libraries (e.g. Kokkos, Alpaka) for specific implementations
  - Can provide these external packages in system locations to avoid installing from source
  - Instructions on github lists the required external packages for each versions of p2r
- Example installation commands:

```
spack install p2r-tests@kokkos impl=kokkos backend=nvidia %gcc@9.2.0
spack install p2r-tests@main impl=stdpar backend=nvidia %nvhpc@22.7
```

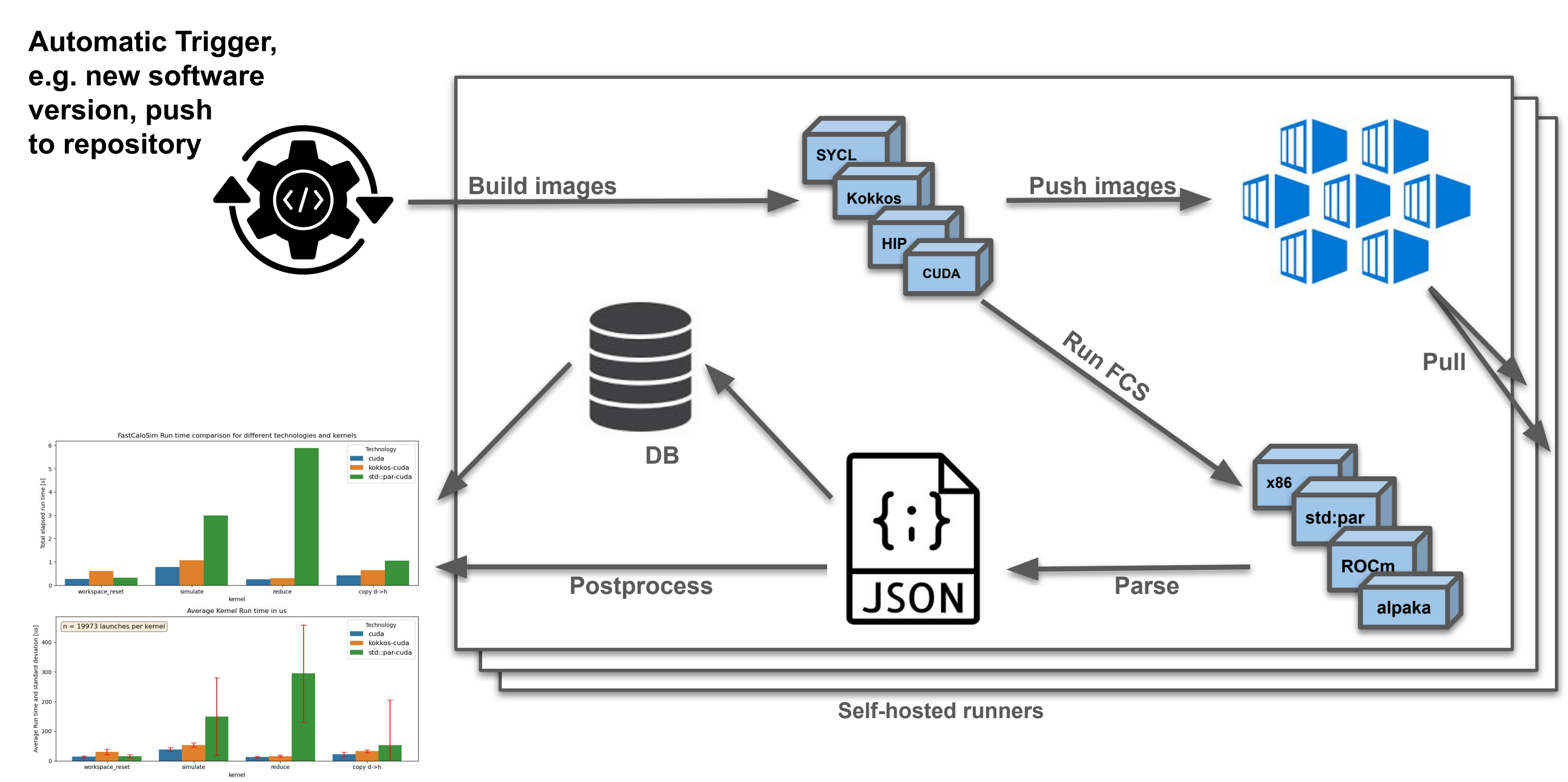
	p2r implementations						
	TBB	CUDA	HIP	Kokkos	Alpaka	Std::par	SYCL
NVDIA	⬜	🟩	🟥	🟩	🟩	🟩 nvc++	🟩
AMD	⬜	⬜	🟩	🟩	🟩	🟥	🟩
Intel	⬜	⬜	⬜	🟩	🟥	🟩	🟩 dpl
CPU	🟩	⬜	⬜	🟩 OMP	🟩 TBB	🟩	🟩

Available p2r implementations on different execution backends (Green solid cells). This work implements the spack installation method of the NVIDIA backends for all p2r implementations (Blue boarded cells).

- Further development
  - Near term extend to other GPU/CPU backends
  - Test on different HPC systems
  - Integrate into official spack github as a package

FastCaloSim with Docker Containerization and Automated Git CI Workflows on HPCs

- Docker containers for FastCaloSim are built on base NVIDIA and AMD GPU images. Kokkos & SYCL added as needed
- ROOT v6.32.02 added to each container built with appropriate compiler
- Continuous Integration workflows are triggered from git actions
- A message is sent to a HPC gateway by GitHub webhook each time when a CI workflow is triggered
- The gateway checks if the CI job is:
  - triggered in an allowed repository and branch
  - triggered by a GitHub user who already has an account with the HPC facility
- The gateway sends the job to the HPC
  - Start a self-hosted runner remotely on the HPC
  - Self-hosted runner takes only the admitted CI job, and tear itself down immediately after job completion.
- Job downloads the appropriate base docker image then:
  - Builds the repository against the base image dependencies
  - Tags and pushes the new image to an image registry
  - Executes the FastCaloSim binary with various particle & energy configurations
  - Job outputs are parsed to extract performance metrics



- Performance metrics and platform specifications are uploaded to a database
- Plots are automatically generated to track performance

Work supported by US Department of Energy, Office of Science, Office of High Energy Physics under the High Energy Physics Center for Computational Excellence (HEP-CCE), a collaboration between Argonne National Laboratory, Brookhaven National Laboratory, Fermi National Accelerator Laboratory and Lawrence Berkeley National Laboratory.