Evaluating Portable Parallelization Strategies for Heterogeneous Architectures

Mohammad Atif[1], Meghna Battacharya[1], Paolo Calafiura[3], Taylor Childers[4], Mark Dewing[2], Zhihua Dong[1], Oliver Gutsche[2], Salman Habib[4], Kyle Knoepfel[2], Matti Kortelainen[2], Ka Hei Martin Kwok[2], Charles Leggett[3], Meifeng Lin[1], Vincent Pascuzzi[1], Alexei Strelchenko[2], Vaktang Tsulaia[3], Brett Viren[1], Tianle Wang[1], Beomki Yeo[3], Haiwang Yu[1]

[1]Brookhaven National Laboratory, Upton, NY 11973, USA
[2]Fermi National Accelerator Laboratory, Batavia, IL 60510, USA
[3]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
[4]Argonne National Laboratory, Lemont, IL 60439, USA

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

Portable Parallelization APIs and Languages

- **Kokkos:** A C++ abstraction layer (library) that supports parallel execution of the code and data management for different host and accelerator architectures.
- **SYCL:** A specification for a cross-platform C++ abstraction layer. Implementations are provided by different vendors/organizations to support different architectures.
- **OpenMP:** Compiler directive-based programming model for parallel execution on different host and accelerator architectures.
- **alpaka:** Header only parallel abstraction library that provides low level control of hardware, targeting CPUs, GPUs and FPGAs
- **std::execution::pararallel** C++ standards based approach to launching parallel tasks. Still under development by standards bodies, with possible full integration with C++26.

	CUDA	HIP	OpenMP Offload	Kokkos	dpc++ / SYCL	alpaka	std::par
NVidia GPU					codeplay and intel/llvm		nvc++
AMD GPU				feature complete for select GPUs	via hipSYCL and intel/llvm		
Intel GPU		HIPLZ: early prototype		native and via OpenMP target offload		prototype	oneAPI::dpl
multicore CPU							g++ & tbb
FPGA						via SYCL	

Experiment Testbeds

- WCT – WireCell Toolkit (DUNE): Liquid Argon TPC Simulation
- FCS – FastCaloSim (ATLAS): Parametrized LAr Calorimeter Simulation
- Patatrack (CMS): Silicon pixel tracker reconstruction
- p2r (CMS): Propagate to R track follower
- ACTS tracking workflow (ACTS): multistage track finder and following

	Kokkos	SYCL	OpenMP	std::par	alpaka
WireCell			WIP	desired	stretch
FastCaloSim			WIP		WIP
Patatrack		WIP	WIP	WIP	
p2R					
ACTS	partial	partial		partial	

Metrics

- Ease of Learning
  - novices, C++ developers, GPU experts
- Code conversion
  - CPU → GPU, API → API
- Extent of modifications to existing code
  - Control of main, threading/execution model
- Extent of modifications to EDM / Data
- Extent of modifications to build rules / system
- Hardware Mapping
  - current and promised future support of hardware
- Feature Availability
  - reductions, kernel chaining, callbacks, concurrency
- Address needs of large and small workflows
- Long term sustainability and code stability
  - backward/forward compatibility of API and eg CUDA
- Compilation time
- Run time
  - what happens to original CPU code
- Ease of Debugging
- Aesthetics
  - beauty is in the eye of the beholder
- Interoperability
  - interaction with externals, thread pools, c++ standards

Performance Studies

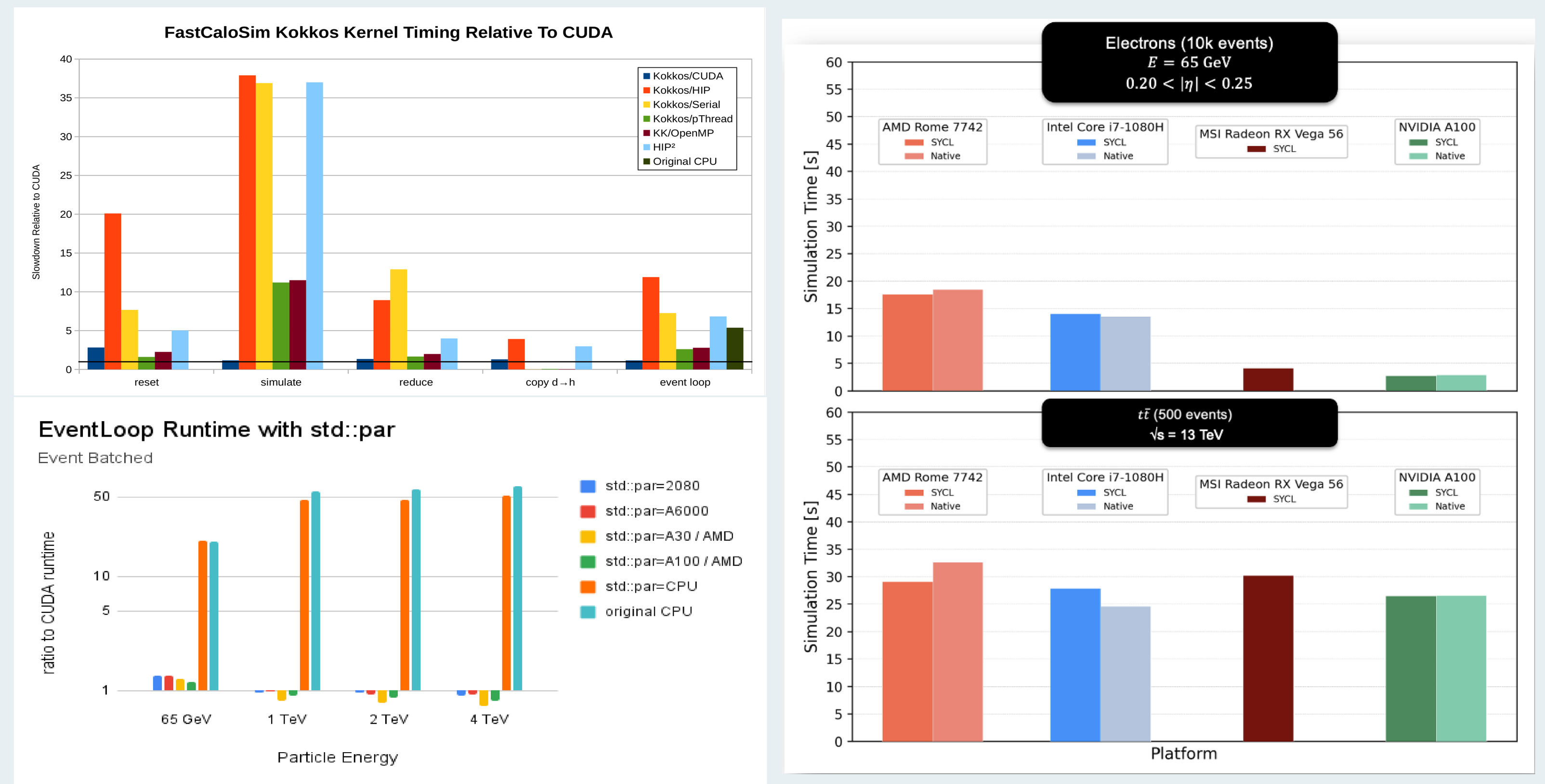

Figure 1: FastCaloSim Timings.

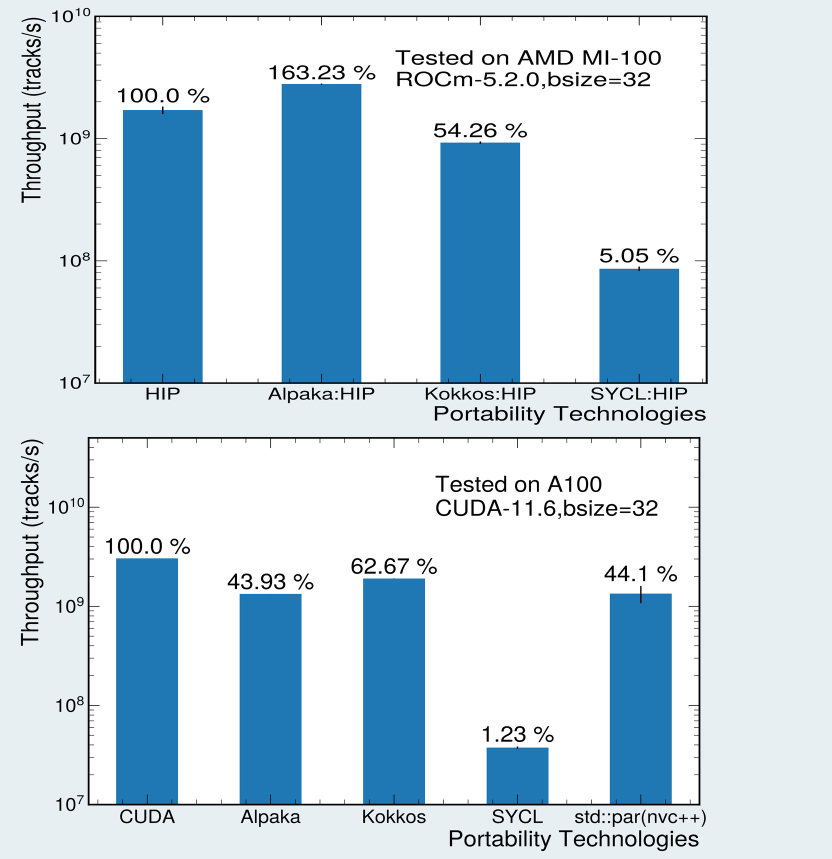

Figure 2: p2r Timings.

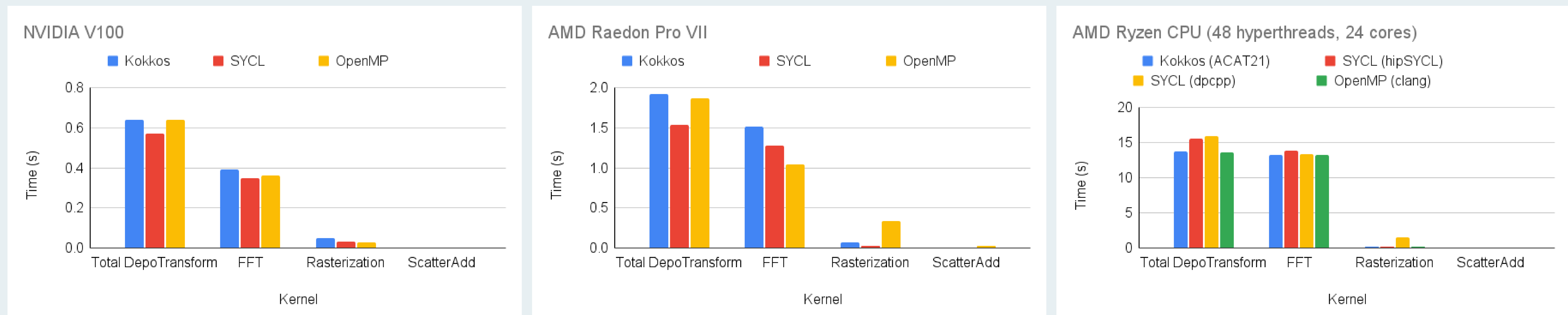

Figure 3: WCT Timings.

Test case	Throughput (events/s)
CPU version, 1 thread	$\mathbf{13.5 \pm 0.2}$
Kokkos version, Serial execution space	$\mathbf{8.5 \pm 0.2}$
CPU version, 40 threads	$\mathbf{539 \pm 9}$
Kokkos version, Threads execution space, peak (18 threads)	$\mathbf{28 \pm 1}$
CUDA version, peak (9 concurrent events and CPU threads)	$\mathbf{1840 \pm 20}$
CUDA version, 1 concurrent event	$\mathbf{720 \pm 20}$
CUDA version, 1 concurrent event, memory pool disabled	$\mathbf{159 \pm 1}$
Kokkos version, CUDA execution space	$\mathbf{115.7 \pm 0.3}$

Table 1: Patatrack Timings

Kokkos

- similar learning curve to CUDA
- needs explicit init/finalize
- crafting SoAs with views is tedious
- no support for jagged arrays
- long templates make debugging hard
- need to explicitly define backends at compilation
- no generic use of concurrent kernels
- well established
- strong developer community and prompt backend support

SYCL

- can target all hardware backends from same source, though recompilation or different compiler versions required
- near native performance
- more verbose than CUDA, but similar to Kokkos for memory management when using buffers
- callbacks may not be supported in future, no concurrent kernels
- good cmake integration
- strong support by Intel, pushing towards integration in C++ standards

OpenMP

- easy to implement, does not require major changes to the C++ code
- performance varies from compiler to compiler
- specialized operations (e.g. atomic) less performant than CUDA
- does not support GPU scan operation
- under active development
- architecture agnostic compiler directives can offload to multiple GPUs, FPGAs

std::par

- plain C++ - low entry bar for developers
- cannot access low level GPU features
- memory transfers restricted to USM
- unless kernels have a direct thrust counterpart, not as performant as CUDA
- no asynchronous operations
- no ability to specify kernel execution parameters (block/grid size)
- compilers still under development, can be buggy
- C++ standards compliant

alpaka

- Header-only C++ library
- Single-source programming model (kernels are embedded in application code)
- At compilation alpaka kernels are transformed into native kernels. Achieves compatibility with the vendor ecosystem (e.g., debugging tools)
- Low-level and powerful API. Not always super-intuitive
- Heavy usage of C++ template meta-programming. The application code tends to be quite verbose

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, Fermilab and Lawrence Berkeley National Laboratory.

Argonne National Laboratory · Brookhaven National Laboratory · Fermilab · Berkeley Lab

ACAT2022, 10/23/2022-10/28/2022