A Study of Cross-Platform, Cross-Compiler Performance and Portability

Reuben D. Budiardja, ORNL
Sunita Chandrasekaran, U. Delaware
Josh Davis, U. Delaware
Johannes Doerfert, ANL
Alaina Edwards, ORNL
Jeff Larkin, NVIDIA
Verónica G. Melesse Vergara, ORNL

ORNL is managed by UT-Battelle LLC for the US Department of Energy

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725 and resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

Outline

- Motivation
- Benchmarks and mini-apps featured
  - SPEC ACCEL
  - GenASiS
  - su3
- How to get more efficient compilers?
  - Three (short) stories of application problems and better compilers
- Conclusions

Motivation

- Past works have focused primarily on the performance and portability of codes across differing hardware architectures
- Performance portability is also required between compilers on the same architecture
- Explore performance portability using widely available benchmarks and mini-apps
- Provide feedback to vendors on issues identified

OAK RIDGE
National Laboratory

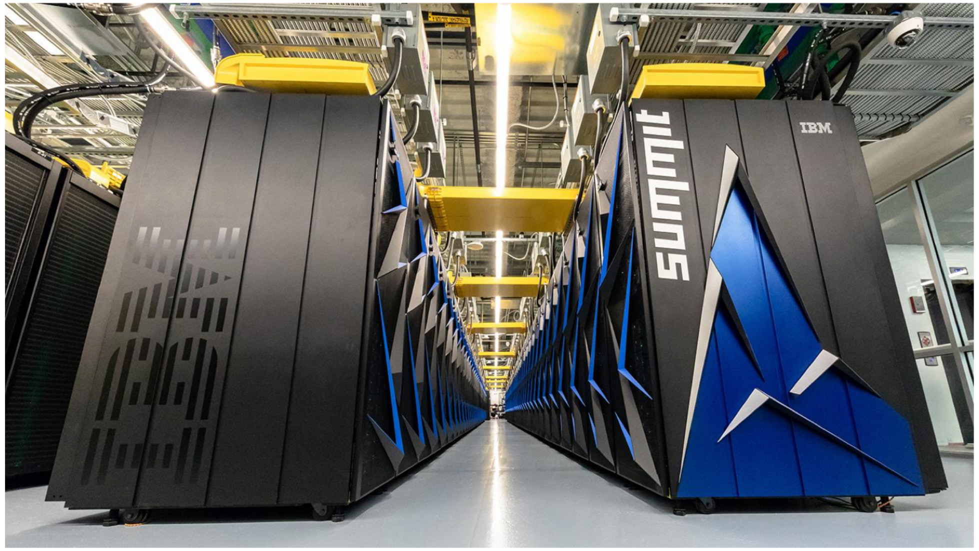
Benchmarks & Mini-applications

Benchmarks & Mini-Apps

- SPEC ACCEL (**S**tandard **P**erformance **E**valuation **C**orporation **Accel**erator suite)
  - Collection of computationally intensive parallel applications in C and Fortran
  - Includes OpenACC, OpenMP target offload, and OpenCL versions
- GenASiS (**Gen**eral **A**strophysics **Si**mulation **S**ystem)
  - A simulation system (a collection of solvers, manifolds, physics) for astrophysics simulation
  - A subdivision, `GenASiS Basics`, contains simplified proxy-applications
- su3
  - mini-application extracted from MILC (Lattice QCD code)
  - matrix-matrix multiply with complex numbers

Experimental Setup: SPEC ACCEL

- SPEC ACCEL (https://www.spec.org/accel)
  - Three benchmark sets: OpenCL, OpenACC, and OpenMP
    - Here we focus on OpenACC and OpenMP sets
    - ACC: contains OpenACC ports
    - OMP: contains OpenMP ports
  - Consists of 15 benchmarks in a wide range of domains: thermodynamics, molecular dynamics, image processing, and more!
    - 7 in C; 6 in Fortran; 2 use C & Fortran
- Target system: Summit
  - ACC experiments:
    - PGI 20.1, GCC 9.2.0
  - OMP experiments:
    - XL 16.1.1 PTF8, GCC 9.2.0, LLVM 11.0.0-rc1 (C-only)
  - Single node jobs offloading to a single NVIDIA V100 GPU



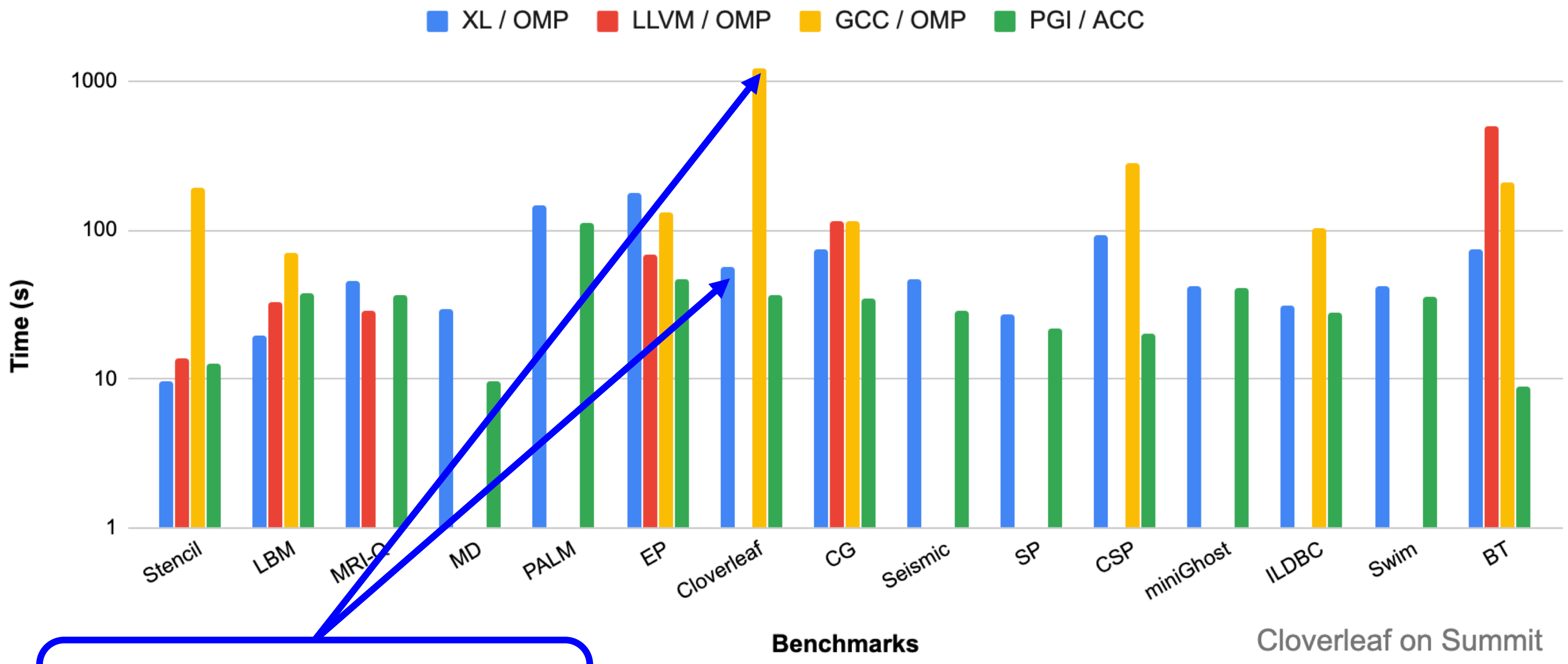
Summit supercomputer (Source: ORNL)

Results: SPEC ACCEL

- XL compiler and PGI compiler can compile all benchmarks
  - Improvement over previous XL versions
- GCC support can compile 11 OpenACC and all OpenMP benchmarks
  - 7 OpenMP and 2 OpenACC benchmarks encounter runtime errors
- All but one of the C benchmarks compile and run successfully with LLVM
  - CSP encounters a compiler error

	XL 16.1.1 PTF8	PGI 20.1	GCC 9.2.0		LLVM 11.0.0rc1
Benchmark	OMP	ACC	OMP	ACC	OMP
Stencil	✓	✓	✓	✓	✓
LBM	✓	✓	✓	✓	✓
MRI-Q	✓	✓	!	✗	✓
MD	✓	✓	!	!	
PALM	✓	✓	!	✗	
EP	✓	✓	✓	✗	✓
Cloverleaf	✓	✓	✓	✗	
CG	✓	✓	✓	✓	✓
Seismic	✓	✓	!	✓	
SP	✓	✓	!	!	
CSP	✓	✓	✓	✓	✗
miniGhost	✓	✓	!	✓	
ILDBC	✓	✓	✓	✓	
Swim	✓	✓	!	✓	
BT	✓	✓	✓	✓	✓

Results: SPEC ACCEL

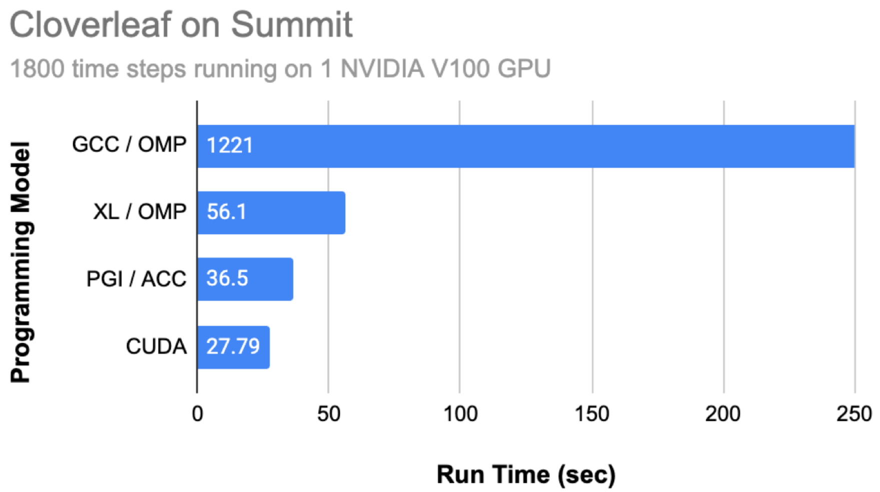


Cloverleaf OpenACC / PGI:

- **~33x faster than OpenMP / GCC**
- **~1.5x faster than OpenMP / XL**

Cloverleaf CUDA:

- **~43x and ~2x faster than OpenMP with GCC and XL, respectively**
- **~1.3x faster than PGI OpenACC**



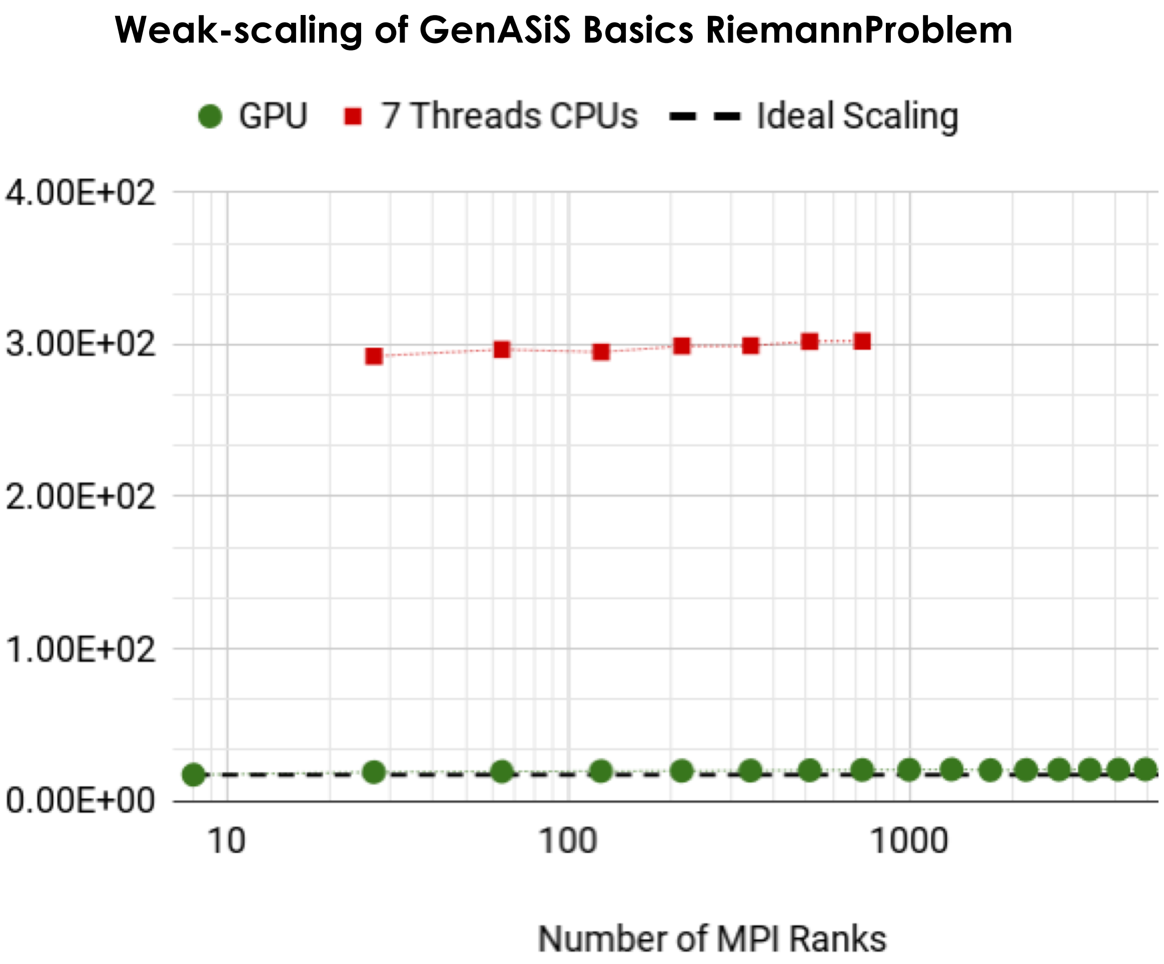
Cloverleaf CUDA: https://uk-mac.github.io/

Lessons Learned: SPEC ACCEL

- More OpenMP implementations can now successfully compile the SPEC ACCEL suite
- Can achieve good performance with OpenMP without significant changes to the code
  - More work is needed to match optimized CUDA versions for some of the benchmarks
- GCC offloading has improved but performance gaps remain
- Need to work with implementation providers to report issues
  - Iterative improvements resulted from collaboration with XL team
  - Submitting issues for GCC

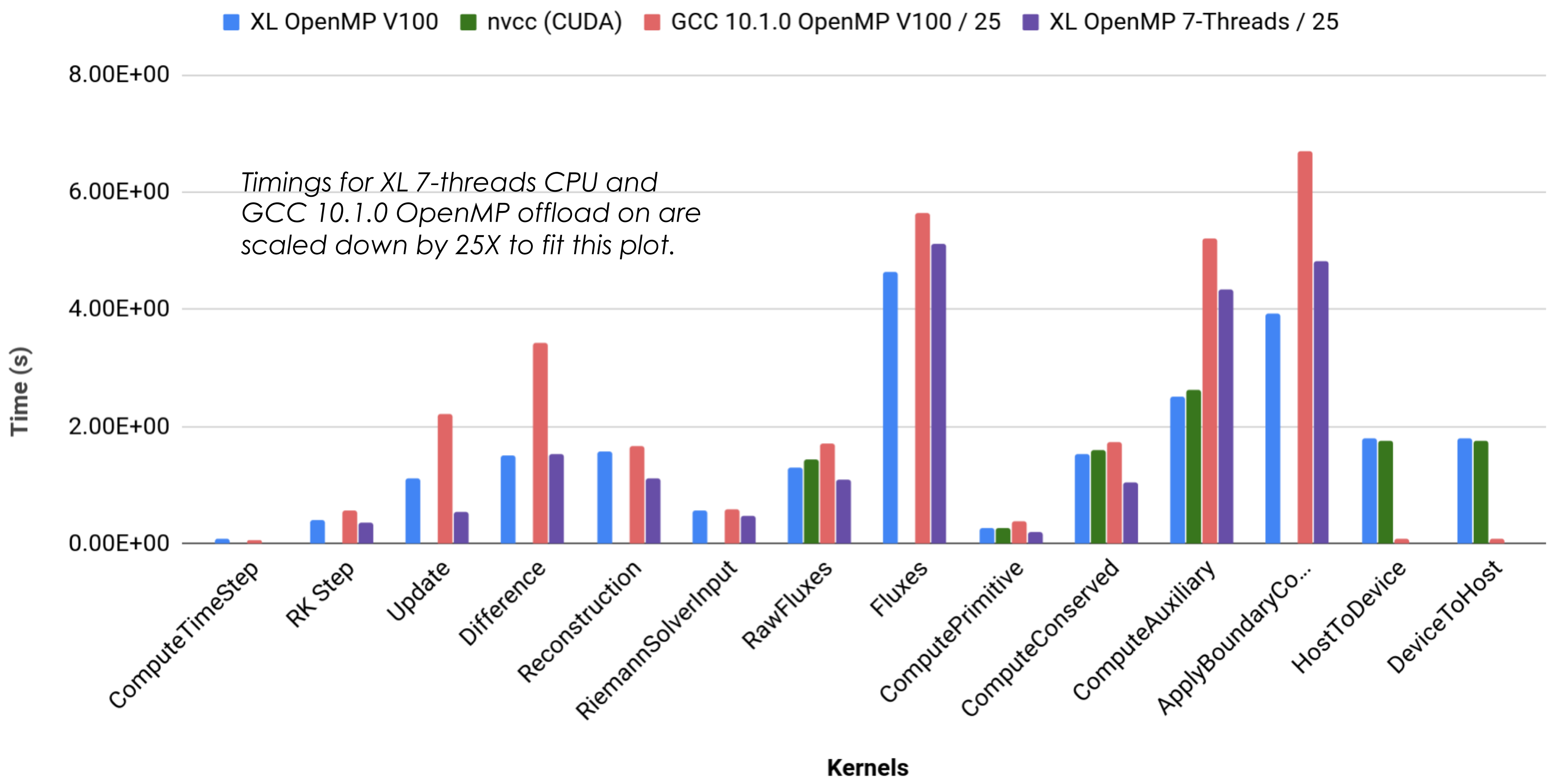
Experimental Setup: GenASiS

- GenASiS
  - Target system: Summit, Frontier (future)
  - Programming model:
    - OpenMP (CPU + target offload)
    - CUDA / HIP library routines, to supplement missing functionality in OpenMP 4.5
    - CUDA / HIP kernels (experimental only)
  - Compilers:
    - XL Fortran 16.1.1 PTF 8, nvcc / hipcc, GCC
- 1 GPU + n CPU threads per MPI task
  - "proportional resource tests" comparisons on Summit:
    7 CPU threads + 1 GPU per MPI task:
    `jsrun -r6 -c7 -g1 -a1 -bpacked:7`



Results: GenASiS Basics RiemannProblem

Kernel timings for 50 cycles, 3D - 256^3 cells per GPU (**lower is better/faster**)



Lesson Learned from GenASiS

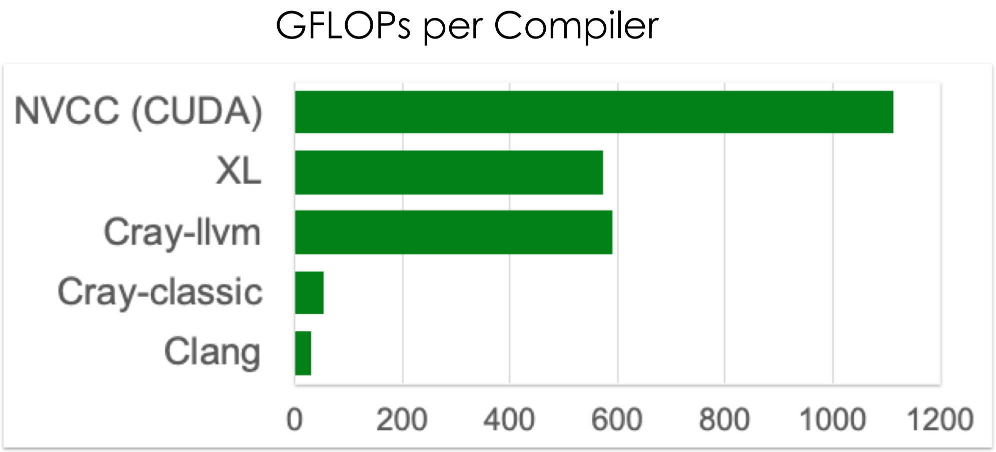
- Explicit memory management and control of data movement is essential for performance
  - persistent memory allocation on the device
  - explicitly map data location on GPU → avoid (implicit) allocation and transfer

- **Performance parity** between OpenMP offload and CUDA is **achievable**
  - OpenMP is more portable
  - OpenMP has benefits of being more “natural” to the application
  - compilers need to do a good job of optimization
  - no need to rewrite kernels, simpler to port from multi-threading to GPU offload
  - can continue to exploit base language feature (Fortran)

Experimental Setup: su3

- su3 mini-app
  - Target system: Cori GPU Nodes, Summit
  - Programming model: OpenMP, CUDA
  - Compilers:
    - Clang 11 [Cori]
    - CCE 9.0.0 (Classic) [Cori]
    - CCE 10.0.0 (LLVM) [Cori]
    - NVCC 10.2.89 [Cori, CUDA]
    - XL 16.0.0-5 [Summit]
  - Mini-app: matrix-matrix multiply with complex numbers, proxy for MILC

Results: su3 mini-app




```
#pragma omp target teams distribute
for(int i = 0; i < total_sites; ++i) {
    #pragma omp parallel for collapse(3)
    // for loops...
```

```
#pragma omp target teams {
    #pragma omp parallel {
        // compute istart, iend...
        for (int i = istart; i < iend; ++i) {
            #pragma omp for collapse(3)
            // for loops...
```

- Excess DRAM transactions responsible for Clang slowdown
- Cray-classic slowdown associated with small grid size, low utilization
  - Can get **4.6x speedup** by increasing num_teams
- Unnecessary memory flushes in Clang caused by splitting directives, interleaving code
- Restructuring directives gives an **18x speedup** with Clang

Lessons Learned: su3 mini-app

- Tuning is necessary for best performance across compilers
    - Particularly of launch parameters: compiler defaults can be quite poor
- Splitting constructs can impair performance
    - Avoid interleaving code between teams and parallel directives
- Some implementations are more robust than others
- Guide tuning and changes with profiling

How to get Efficient Compilers?

Three (short) stories of application problems and better compilers

How To Get Efficient Compilers? --- Talk to the compiler people ...

Three (short) stories of application problems and better compilers

1. HPGMG

Application people: "One performance limiter I see with LLVM/Clang is a huge amount of time spent in cuMemAlloc and cuMemFree calls.**"**

Compiler People (as I recall it): Thanks for reporting this, we'll look into it!

Compiler People (as I recall it): Can you try this version?

Application people: "I now only see 1 cuMemAlloc [...] The net effect is that this particular version of HPGMG runs 2.4x faster"

How To Get Efficient Compilers? --- Talk to the compiler people ...

Three (short) stories of application problems and better compilers

2. (Mini)QMCPack

- Observe a problem and file a bug:

  Bug 46450 - More registers are used when multiple target regions are compiled together

(paraphrased)

- **Compiler people:** How to reproduce the problem?
- **Application people:** Here is a reproducer!
- **Compiler people:** I see, it's a weird implementation detail, can you try this version please?
- **Application people:** It works for me, there are still issues but this solves my problem for now.

How To Get Efficient Compilers? --- Talk to the compiler people...

Three (short) st

3. GridMini SU(3 ifeng Lin (BNL)]

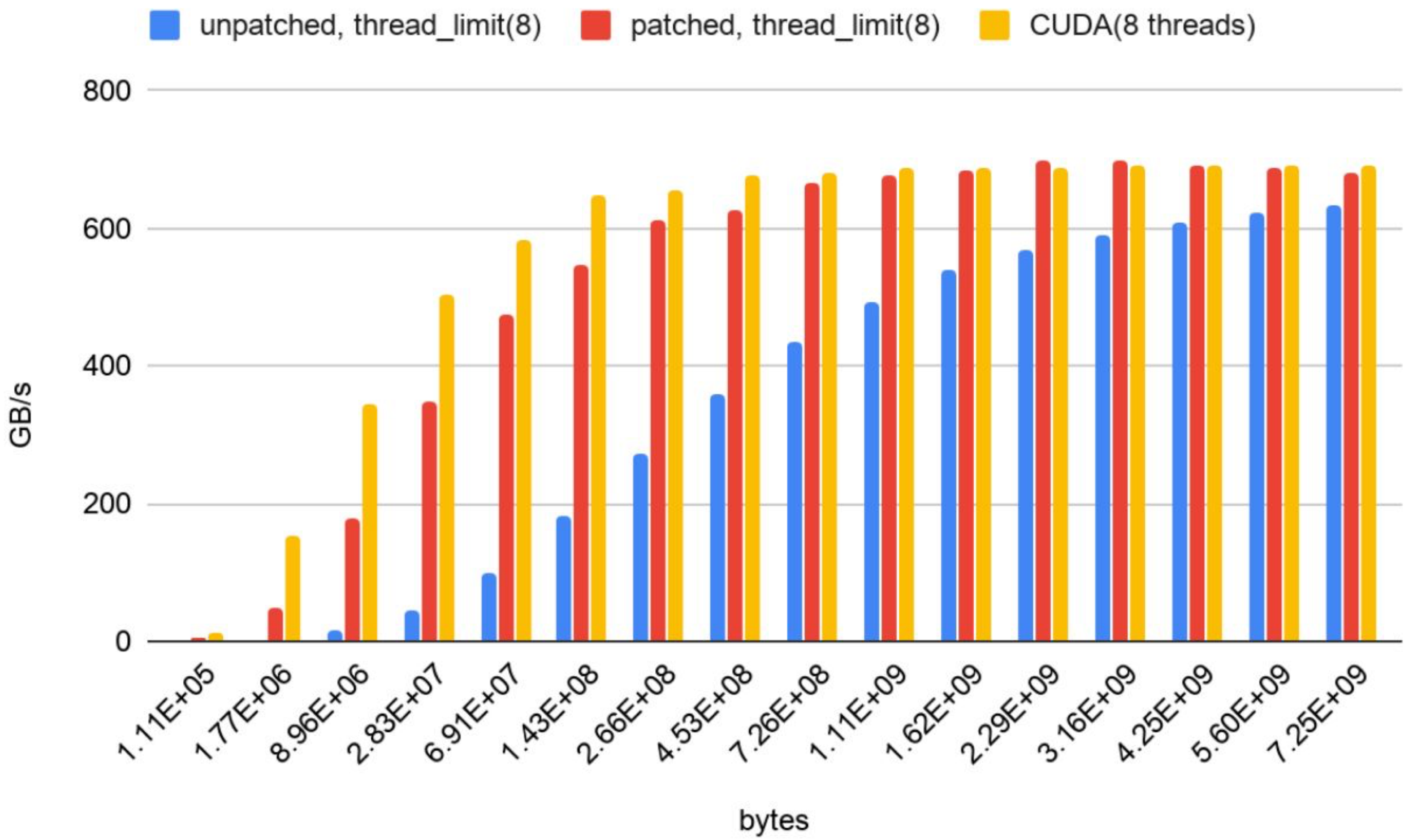
Day 1. "The p A on V100)."

Day 2. Tropica

Day 3. - talking
- trying
- turns c ficantly!"

Day 4. - Profi

"Culprit for poor perf
- repoi



"Application/Compiler Team communication is key!"

Conclusions

- Performance portability is often required not just between architectures but also between compilers on the same architecture
- As seen with GenASiS and SU(3)xSU(3), it is not necessary to sacrifice performance for portability
  - OpenMP kernels can achieve performance parity with their CUDA version
- As seen with su3, significant speedups can be attained by tuning parameters and rearranging OpenMP directives
- Talk to compiler vendors about observations, issues, problems, and ideas!
  - More implementations that perform well on various architectures benefit everyone!

Thank you! Questions?