The Sparse Matrix Vector Product on High-End GPUs

SIAM Conference on Parallel Processing for Scientific Computing (PP20)
February 12 - 15, 2020
Hyatt Regency Seattle | Seattle, Washington, U.S.

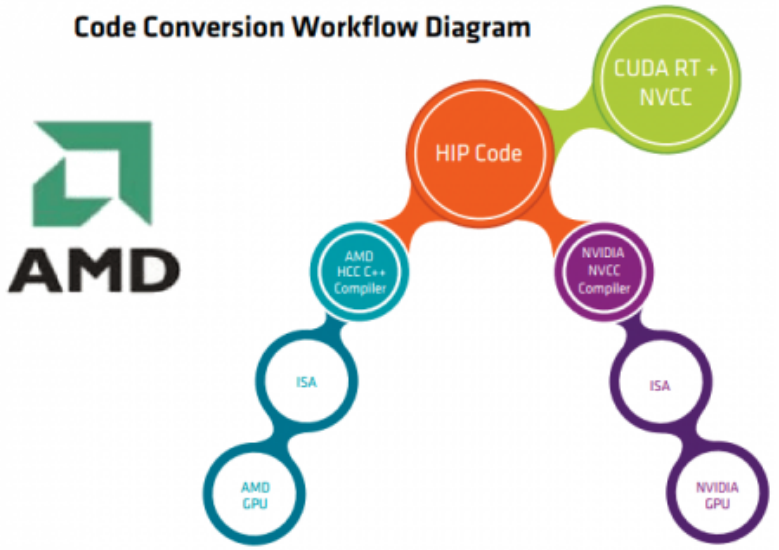
Hartwig Anzt, Terry Cojean, Yuhsiang M. Tsai
Steinbuch Centre for Computing (SCC)

Mike Tsai

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration and the Helmholtz Impuls und VernetzungsfondVH-NG-1241.

SpMV on GPUs – Moving away from the NVIDIA hegemony

- In the past, NVIDIA GPUs were dominating the GPGPU market;
- We see an increasing adoption of AMD GPUs in leadership supercomputers:
  - *Frontier system in OakRidge (2021)*
  - *El Capitan in Lawrence Livermore National Lab ? (2023)*
- AMD is heavily investing in the HIP software development ecosystem;
  - *HIP programming similar to CUDA programming;*
  - *HIP libraries similar to cuBLAS, cuSPARSE, …*
- ***The Race is on!***



- *How can we prepare the Ginkgo sparse linear algebra library for cross-platform portability?*
- *Are the CUDA-optimized kernels suitable for AMD GPUs?*
- *How does the performance compare across different GPUs?*

Extend Ginkgo's hardware scope to AMD GPUs

Ginkgo

https://github.com/ginkgo-project/ginkgo

Library core contains architecture-agnostic algorithm implementation;

Runtime polymorphism selects the right kernel depending on the target architecture;

Architecture-specific kernels execute the algorithm on target architecture;

Core

Library Infrastructure

Algorithm Implementations

- Iterative Solvers
- Preconditioners
- …

EXASCALE COMPUTING PROJECT

Part of

https://xsdk.info/

Kernels

Reference

Reference kernels

- SpMV
- Solver kernels
- Precond kernels
- …

OpenMP

OpenMP-kernels

- SpMV
- Solver kernels
- Precond kernels
- …

CUDA

CUDA-GPU kernels

- SpMV
- Solver kernels
- Precond kernels
- …

Reference are sequential kernels to check correctness of algorithm design and optimized kernels;

Optimized architecture-specific kernels;

googletest
Google C++ Testing Framework

Extend Ginkgo's hardware scope to AMD GPUs

Ginkgo

Library core contains architecture-agnostic algorithm implementation;

Runtime polymorphism selects the right kernel depending on the target architecture;

Architecture-specific kernels execute the algorithm on target architecture;

Core

Library Infrastructure

Algorithm Implementations

- Iterative Solvers
- Preconditioners
- …

Kernels

Reference

Reference kernels

- SpMV
- Solver kernels
- Precond kernels
- …

OpenMP

OpenMP-kernels

- SpMV
- Solver kernels
- Precond kernels
- …

CUDA

HIP-GPU kernels

- SpMV
- Solver kernels
- Precond kernels
- …

HIP

HIP-GPU kernels

- SpMV
- Solver kernels
- Precond kernels
- …

Reference are sequential kernels to check correctness of algorithm design and optimized kernels;

Optimized architecture-specific kernels;

googletest

Google C++ Testing Framework

Extend Ginkgo's hardware scope to AMD GPUs

Ginkgo

Library core contains architecture-agnostic

Core

Library Infrastructure

```cpp
namespace kernel {
  template <int value>
  __global__ void set_value(
      const int num, int *__restrict__ array) {
    auto tid = blockDim.x * blockIdx.x + threadIdx.x;
    if (tid < num) {
      array[tid] = value;
    }
  }
}
int main() {
  // allocation of memory
  // calculation of grid/block_size
  constexpr int value = 3;
  kernel::set_value<value>
    <<<dim3(grid_size), dim3(block_size)>>> (
        num, array);
  return 0;
}
```

CUDA

```cpp
namespace kernel {
  template <int value>
  __global__ void set_value(
      const int num, int *__restrict__ array) {
    auto tid = blockDim.x * blockIdx.x + threadIdx.x;
    if (tid < num) {
      array[tid] = value;
    }
  }
}
int main() {
  // allocation of memory
  // calculation of grid/block_size
  constexpr int value = 3;
  hipLaunchKernelGGL(
    HIP_KERNEL_NAME(kernel::set_value<value>),
    dim3(grid_size), dim3(block_size), 0, 0,
    num, array);
  return 0;
}
```

HIP

Reference are sequential kernels to check correctness of algorithm design and optimized kernels;

Optimized architecture-specific kernels;

googletest
Google C++ Testing Framework

Extend Ginkgo's hardware scope to AMD GPUs

Ginkgo

Library core contains architecture-agnostic algorithm implementation;

Runtime polymorphism selects the right kernel depending on the target architecture;

Architecture-specific kernels execute the algorithm on target architecture;

Core

Library Infrastructure

Algorithm Implementations

- Iterative Solvers
- Preconditioners
- …

Kernels

Reference

Reference kernels

- SpMV
- Solver kernels
- Precond kernels
- …

OpenMP

OpenMP-kernels

- SpMV
- Solver kernels
- Precond kernels
- …

CUDA

CUDA-GPU kernels

- SpMV
- Solver kernels
- Precond kernels
- …

HIP

HIP-GPU kernels

- SpMV
- Solver kernels
- Precond kernels
- …

Reference are sequential kernels to check correctness of algorithm design and optimized kernels;

Optimized architecture-specific kernels;

googletest

Google C++ Testing Framework

Extend Ginkgo's hardware scope to AMD GPUs

Ginkgo

Library core contains architecture-agnostic algorithm implementation;

Runtime polymorphism selects the right kernel depending on the target architecture;

Architecture-specific kernels execute the algorithm on target architecture;

Core

Library Infrastructure

Algorithm Implementations

- Iterative Solvers
- Preconditioners
- …

To avoid code duplication, common contains kernels shared between CUDA and HIP (upon parameter configs)

Common

- Shared kernels

Kernels

Reference

Reference kernels

- SpMV
- Solver kernels
- Precond kernels
- …

OpenMP

OpenMP-kernels

- SpMV
- Solver kernels
- Precond kernels
- …

CUDA

CUDA-GPU kernels

- SpMV
- Solver kernels
- Precond kernels
- …

HIP

HIP-GPU kernels

- SpMV
- Solver kernels
- Precond kernels
- …

Reference are sequential kernels to check correctness of algorithm design and optimized kernels;

Optimized architecture-specific kernels;

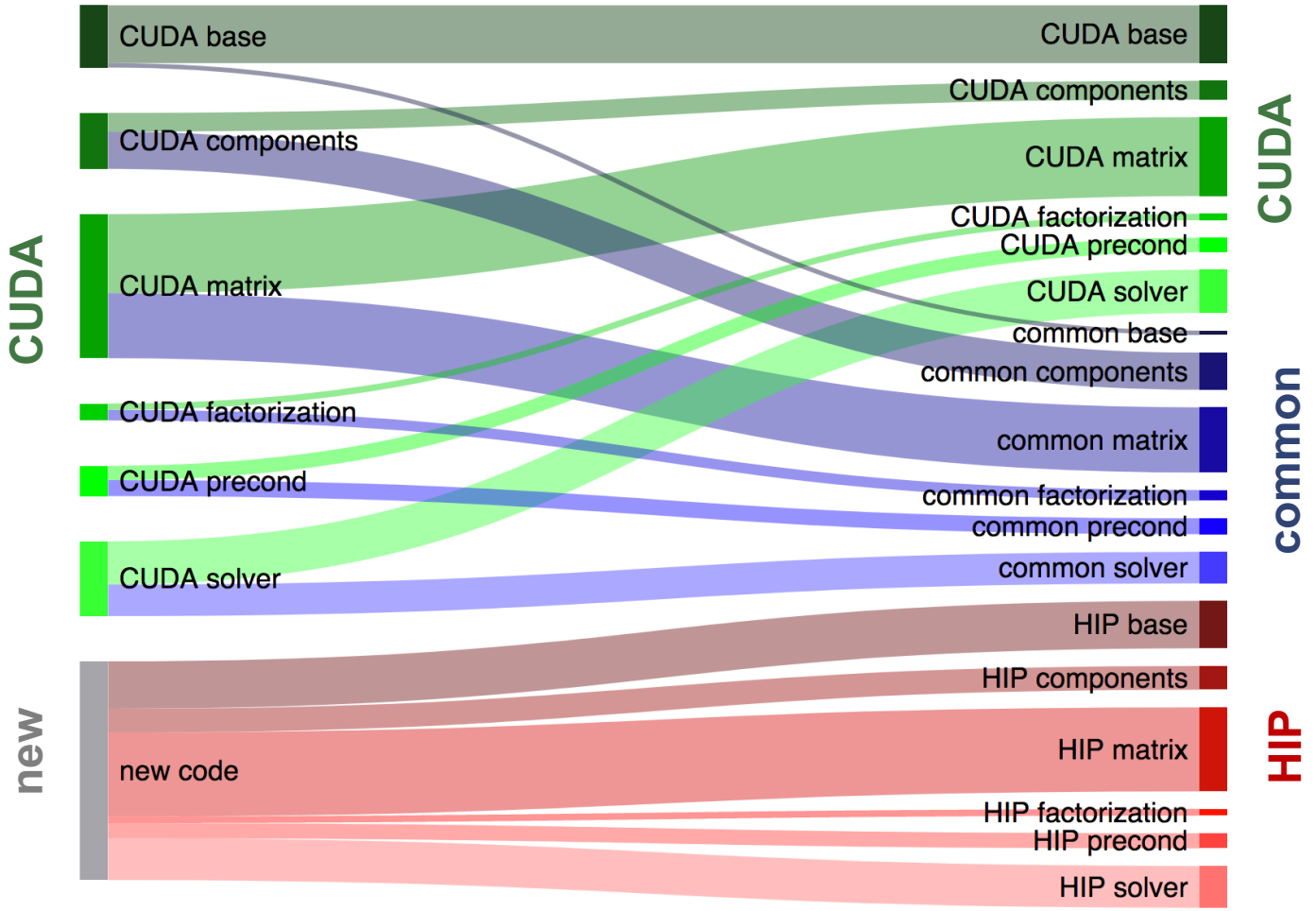
googletest

Google C++ Testing Framework

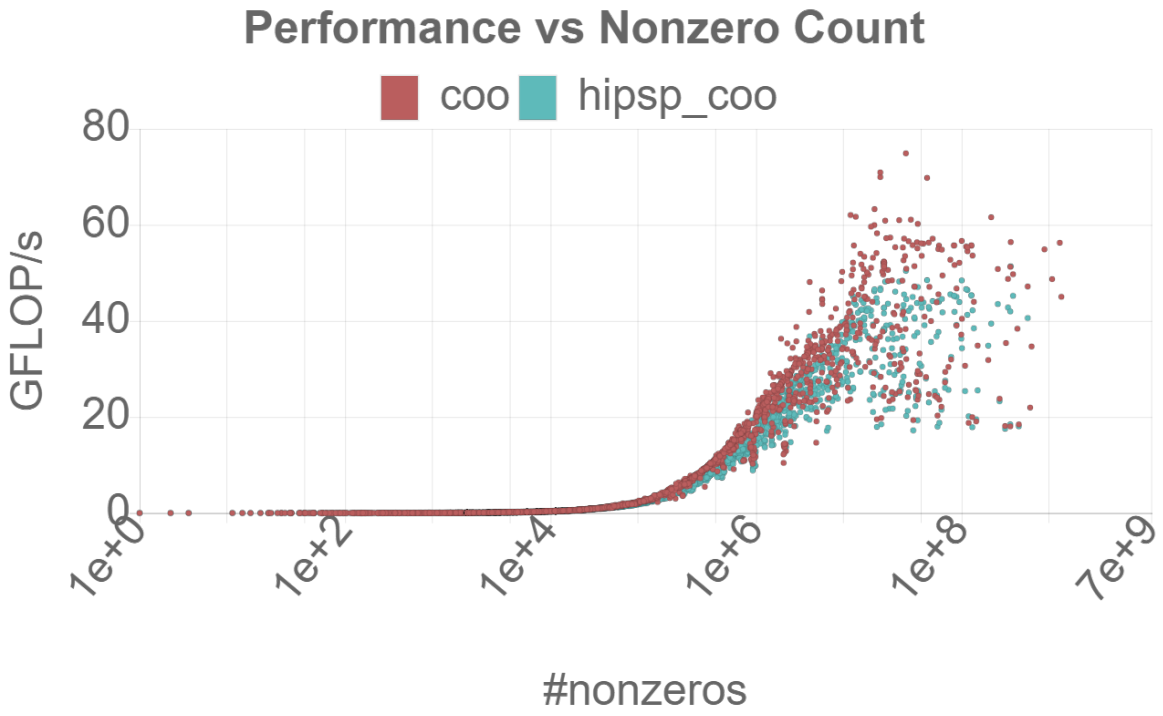
Extend Ginkgo's hardware scope to AMD GPUs

- Kernels shared between CUDA and AMD backends (upon parameter setting) are relocated in the ``common'' module.
- New code necessary for HIP-specific optimizations and for implementing functionality currently missing in the HIP ecosystem (e.g. cooperative groups).

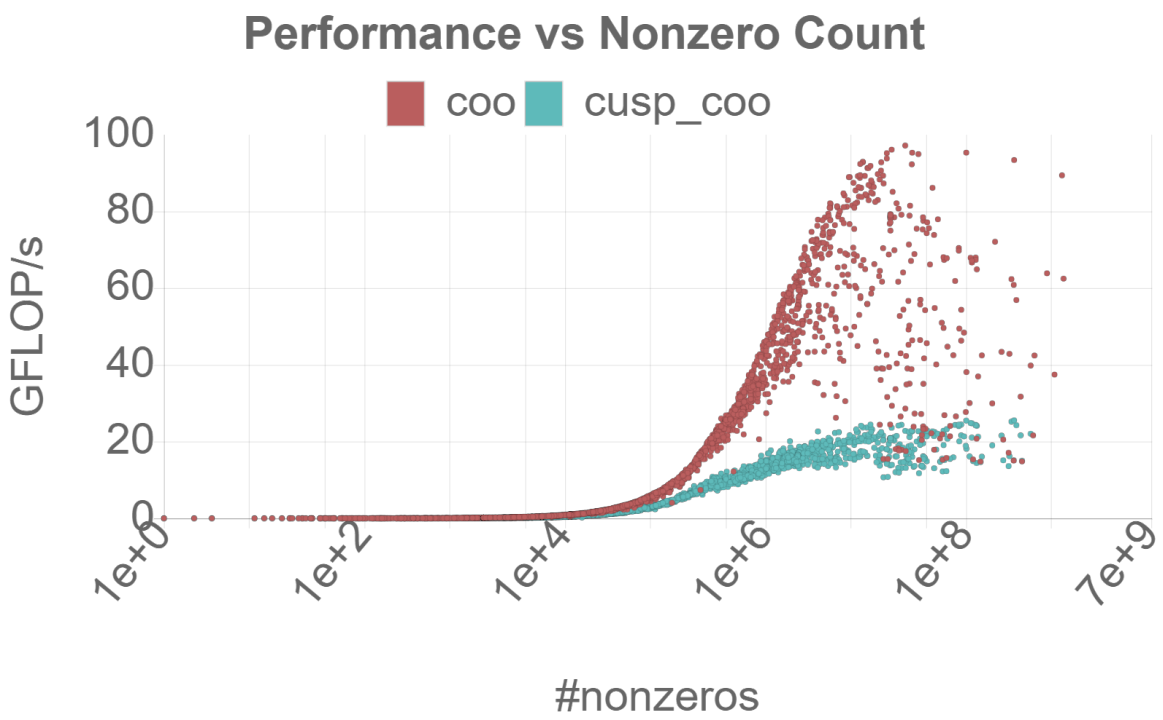
Module	common	cuda	hip
base	112	1435	1176
component	919	467	589
matrix	1617	1908	2048
factorization	262	159	165
preconditioner	395	356	375
solver	780	1071	1038



How does Ginkgo compare to the vendor libraries - COO SpMV



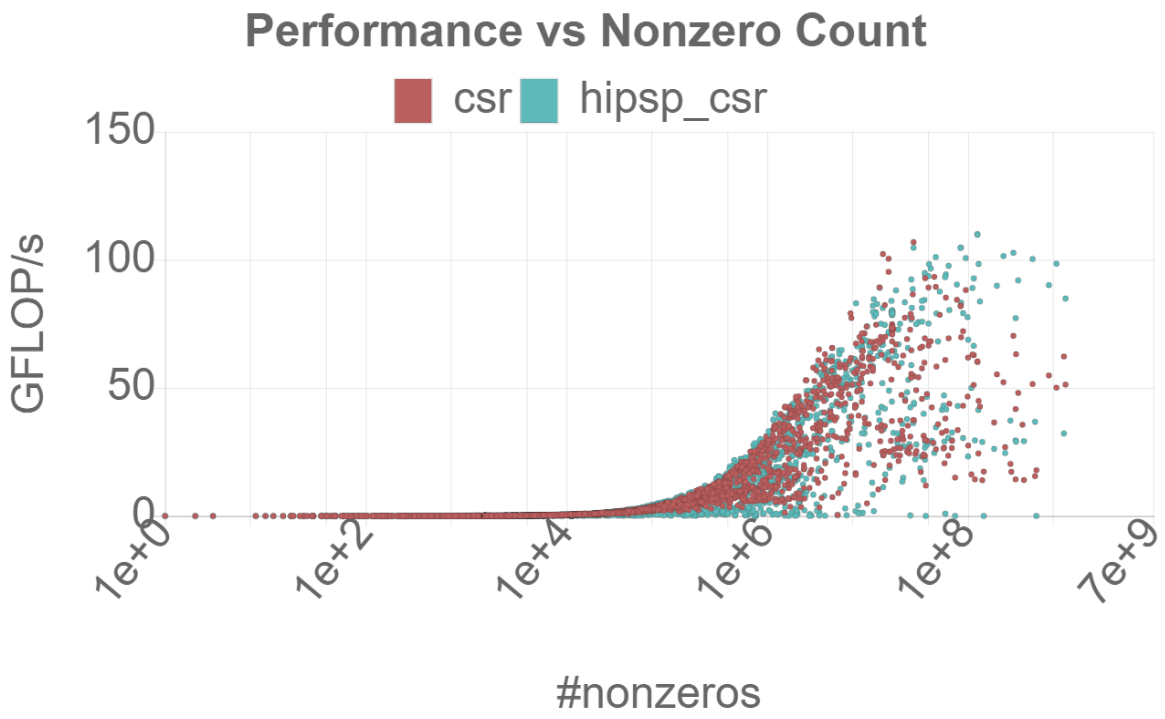
Ginkgo vs HIPsparse on RadeonVII



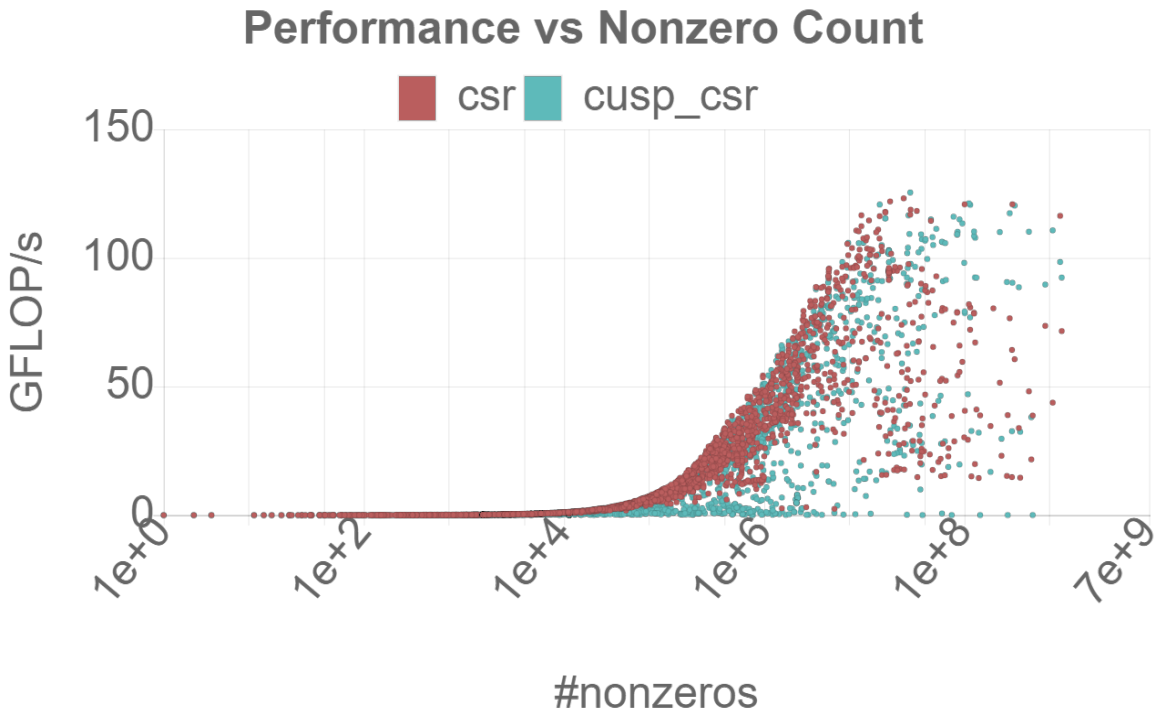
Ginkgo vs cuSPARSE on V100

Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

How does Ginkgo compare to the vendor libraries - CSR SpMV



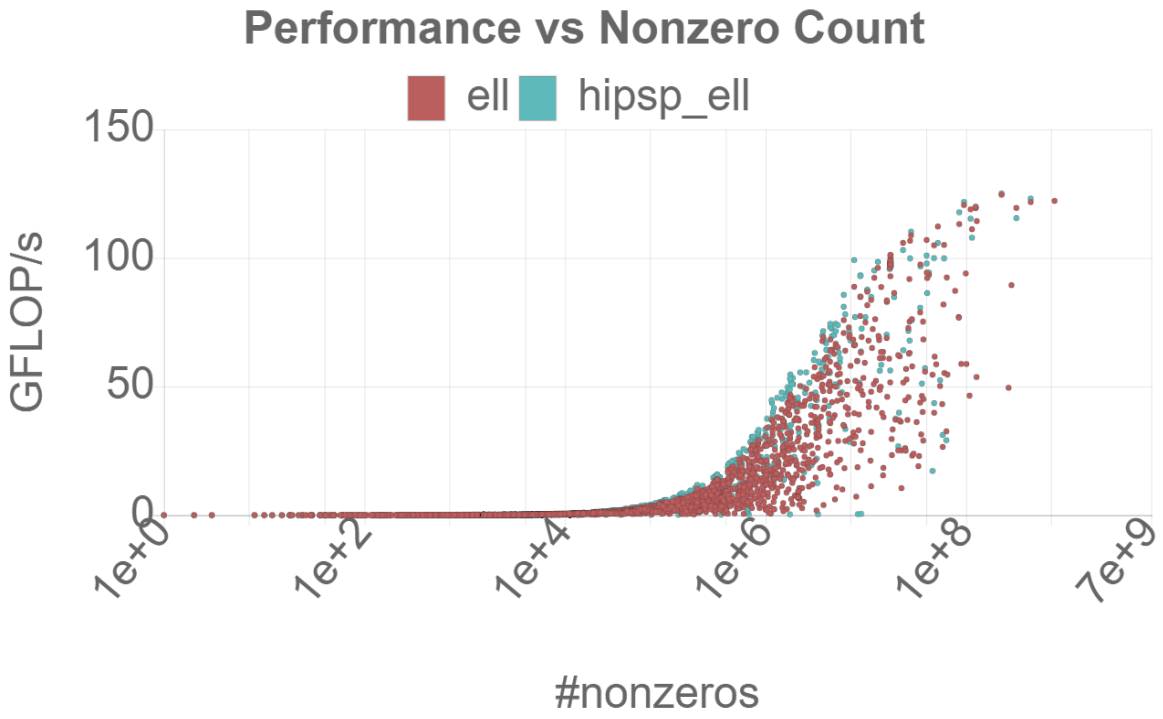
Ginkgo vs HIPsparse on RadeonVII



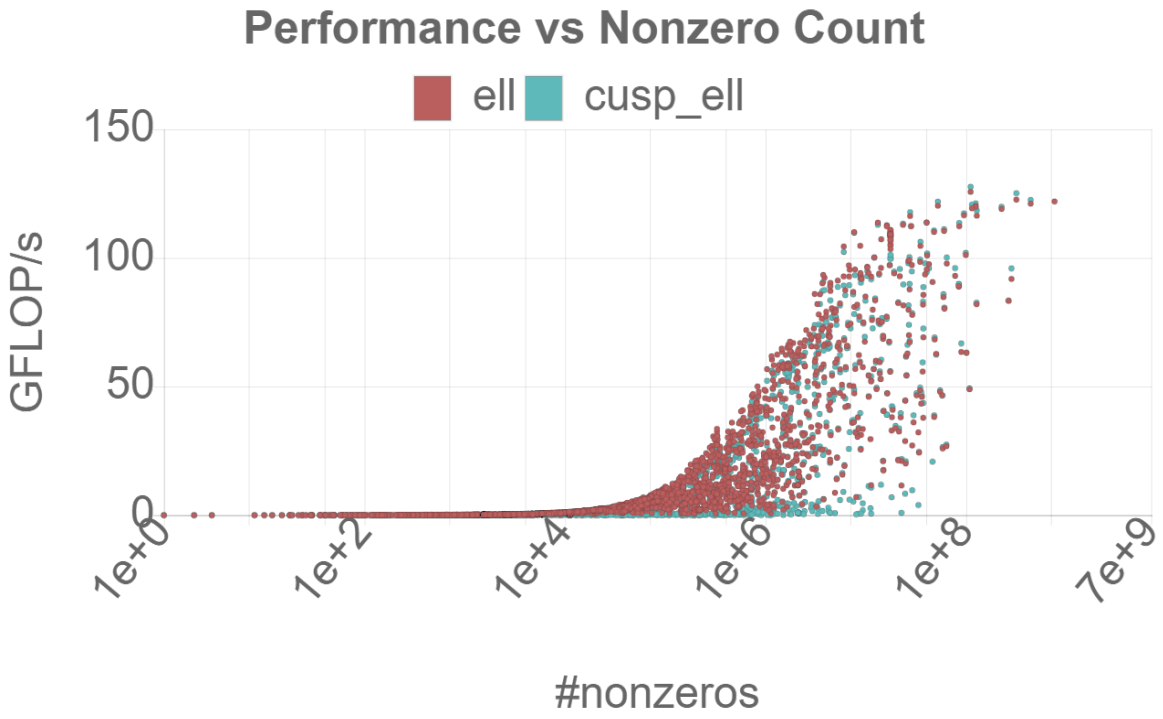
Ginkgo vs cuSPARSE on V100

Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

How does Ginkgo compare to the vendor libraries - ELL SpMV



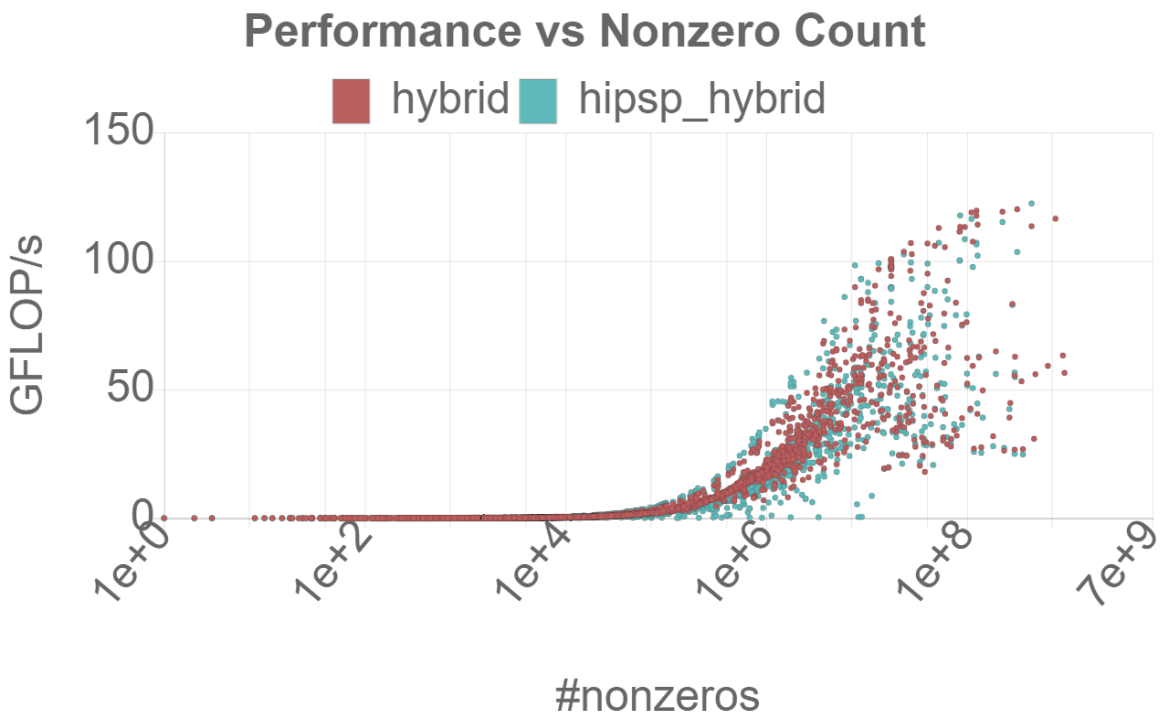
Ginkgo vs HIPsparse on RadeonVII



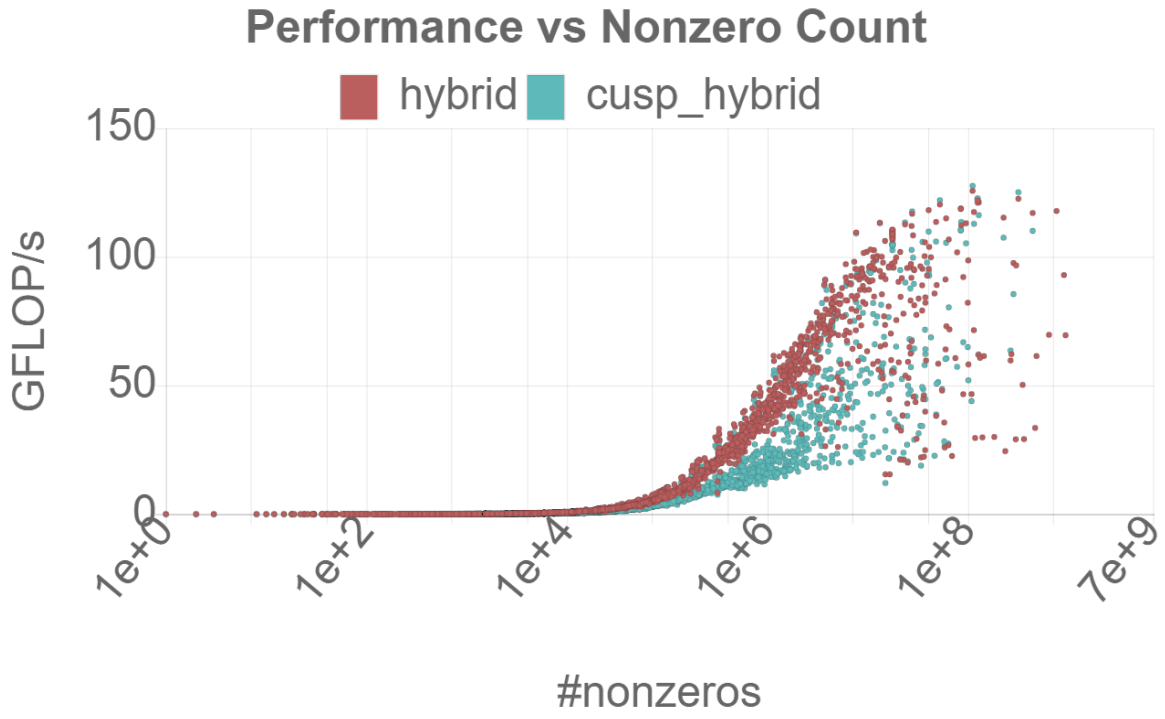
Ginkgo vs cuSPARSE on V100

Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

How does Ginkgo compare to the vendor libraries - hybrid SpMV



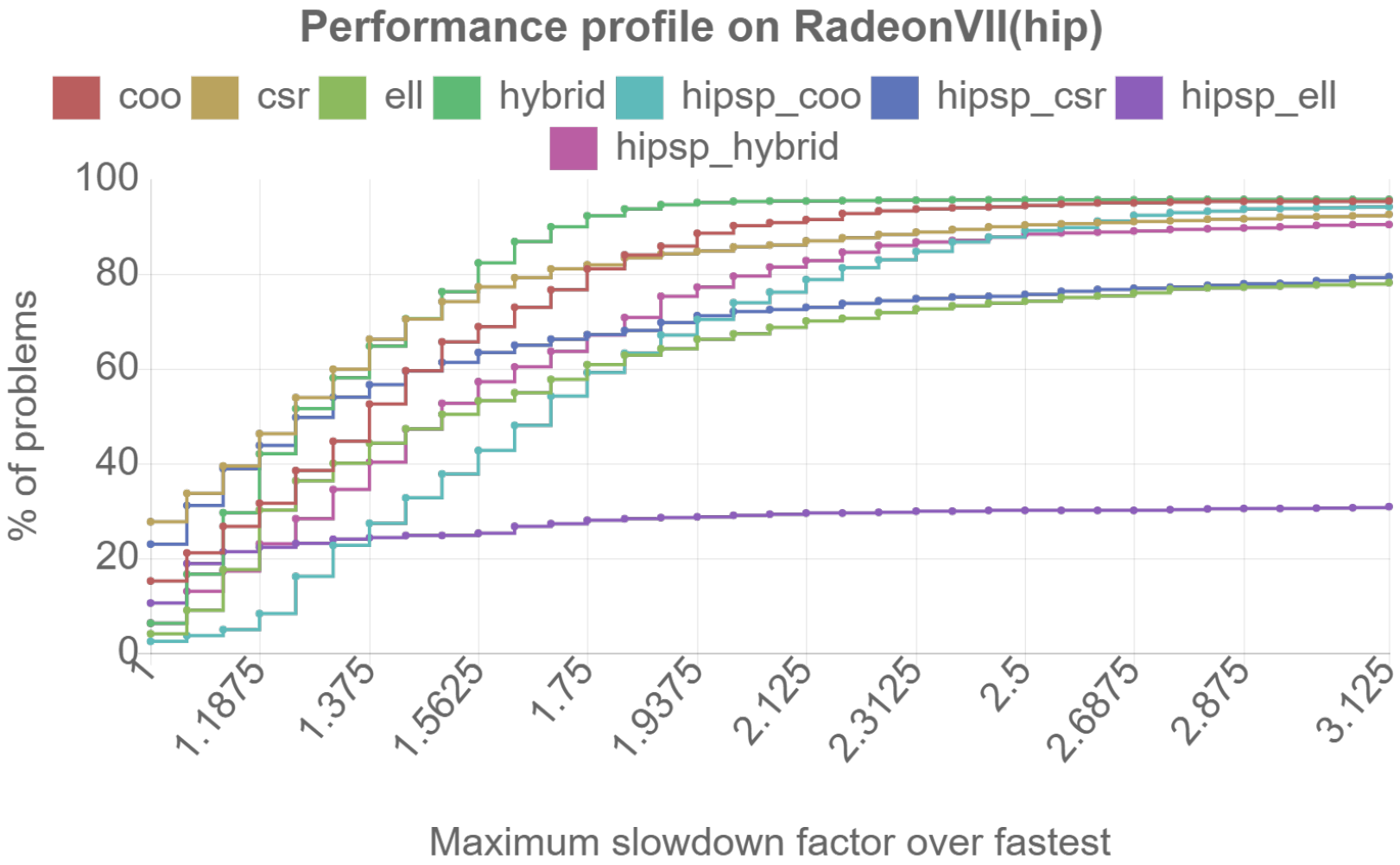
Ginkgo vs HIPsparse on RadeonVII



Ginkgo vs cuSPARSE on V100

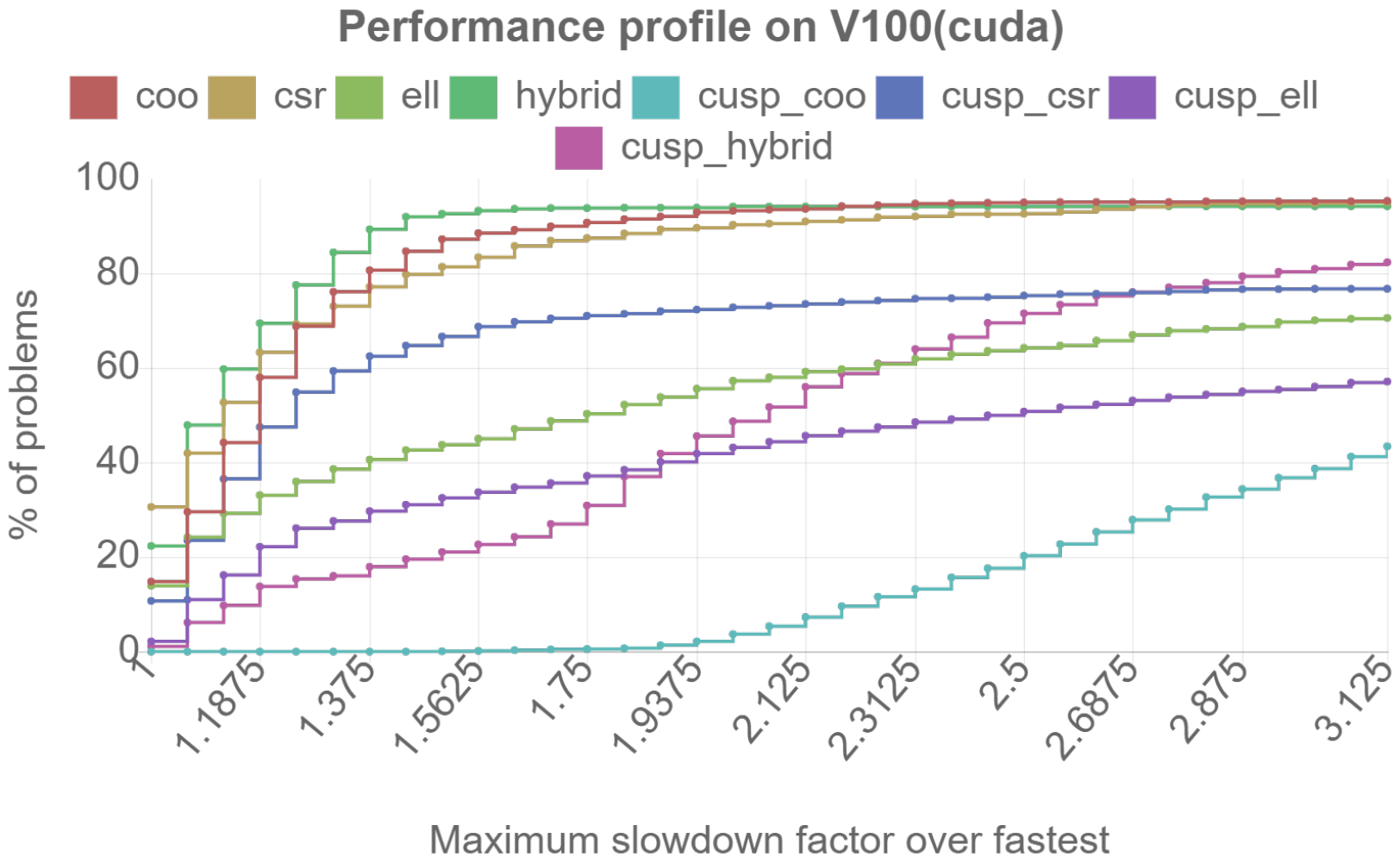
Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

Performance Profile on AMD's RadeonVII



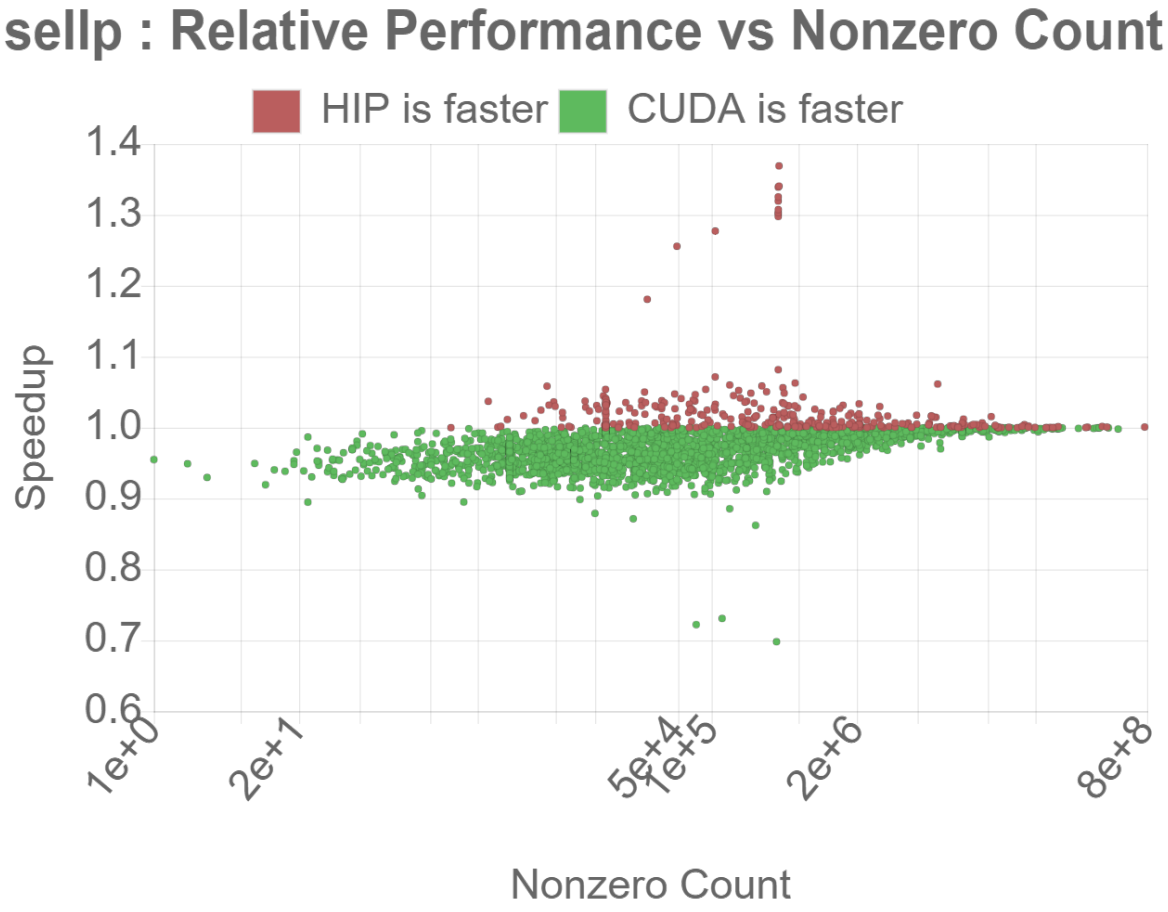
Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

Performance Profile on NVIDIA's V100



Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

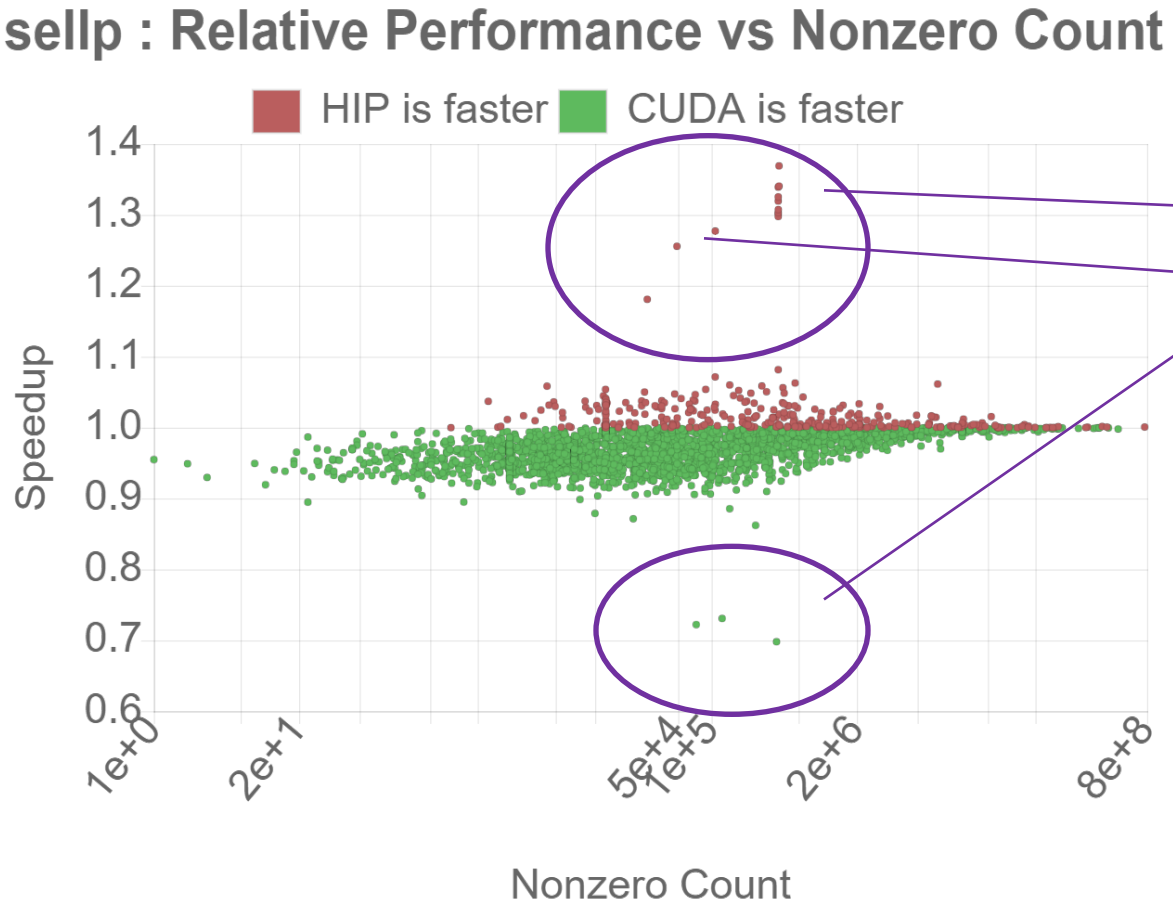
Compiling HIP code for NVIDIA GPUs – comparison against native CUDA code



- Native CUDA vs. HIP compiled for NVIDIA GPUs
- Same kernel
- All tests on NVIDIA V100 (Summit)
- We expect CUDA to be slightly faster

Compiling HIP code for NVIDIA GPUs – comparison against native CUDA code

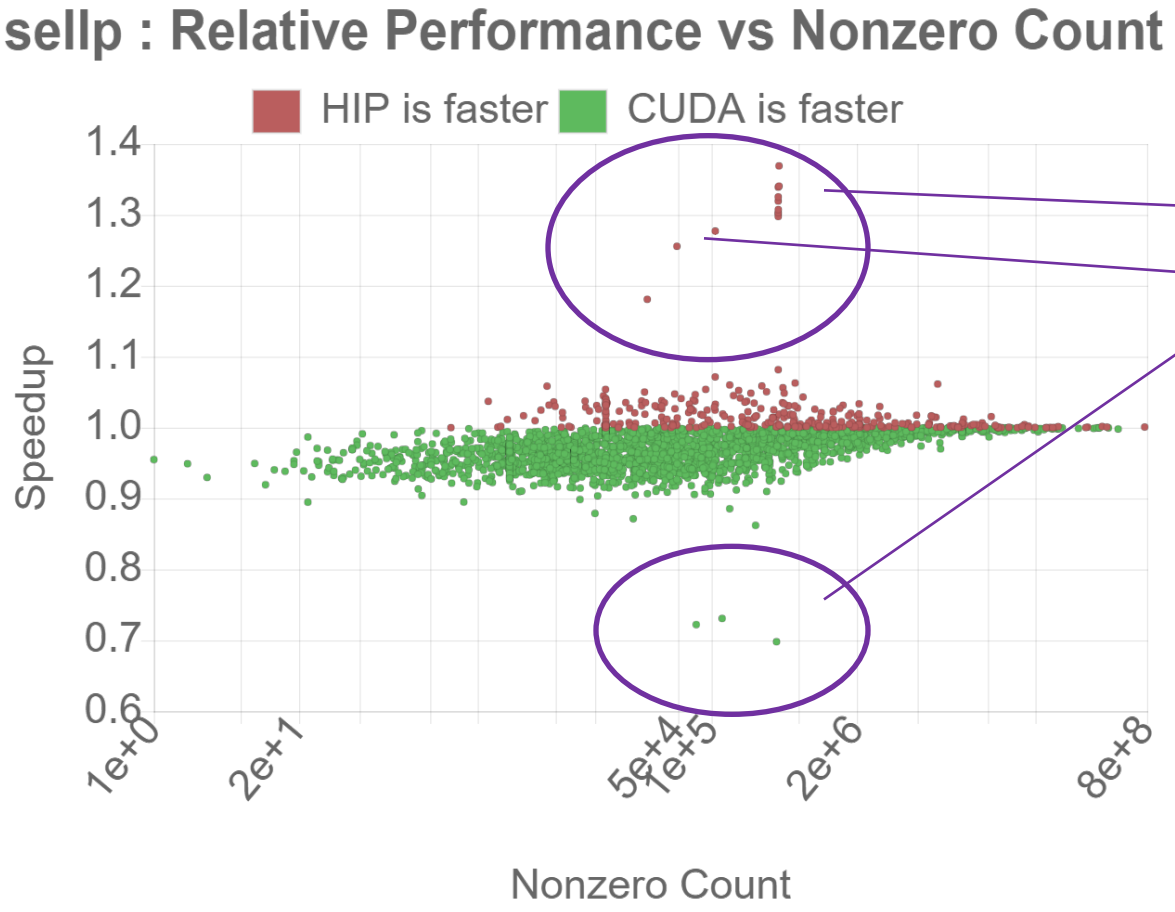
- Native CUDA vs. HIP compiled for NVIDIA GPUs
- Same kernel
- All tests on NVIDIA V100 (Summit)
- We expect CUDA to be slightly faster



HIP faster than CUDA on NVIDIA GPU?

outliers? machine noise?

Compiling HIP code for NVIDIA GPUs – comparison against native CUDA code

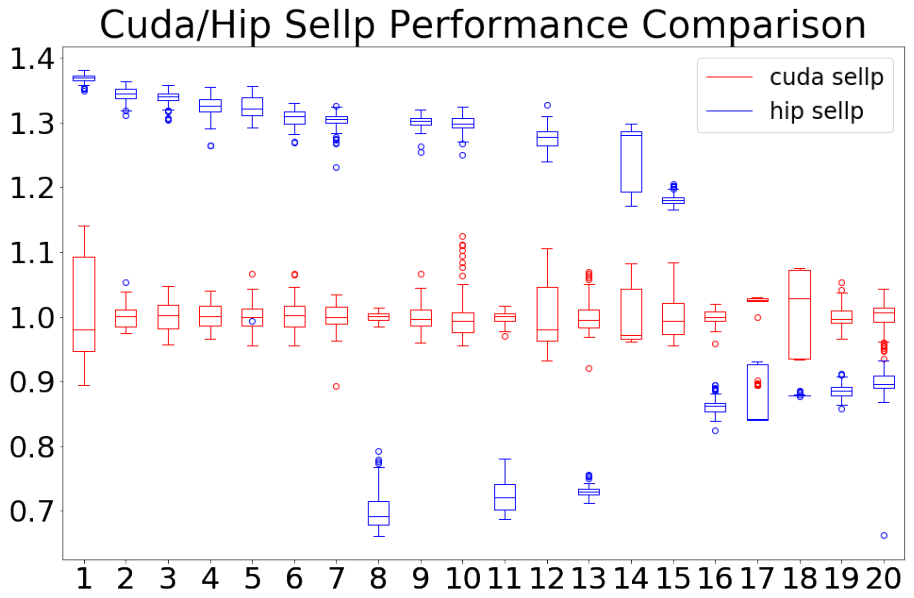


- Native CUDA vs. HIP compiled for NVIDIA GPUs
- Same kernel
- All tests on NVIDIA V100 (Summit)
- We expect CUDA to be slightly faster

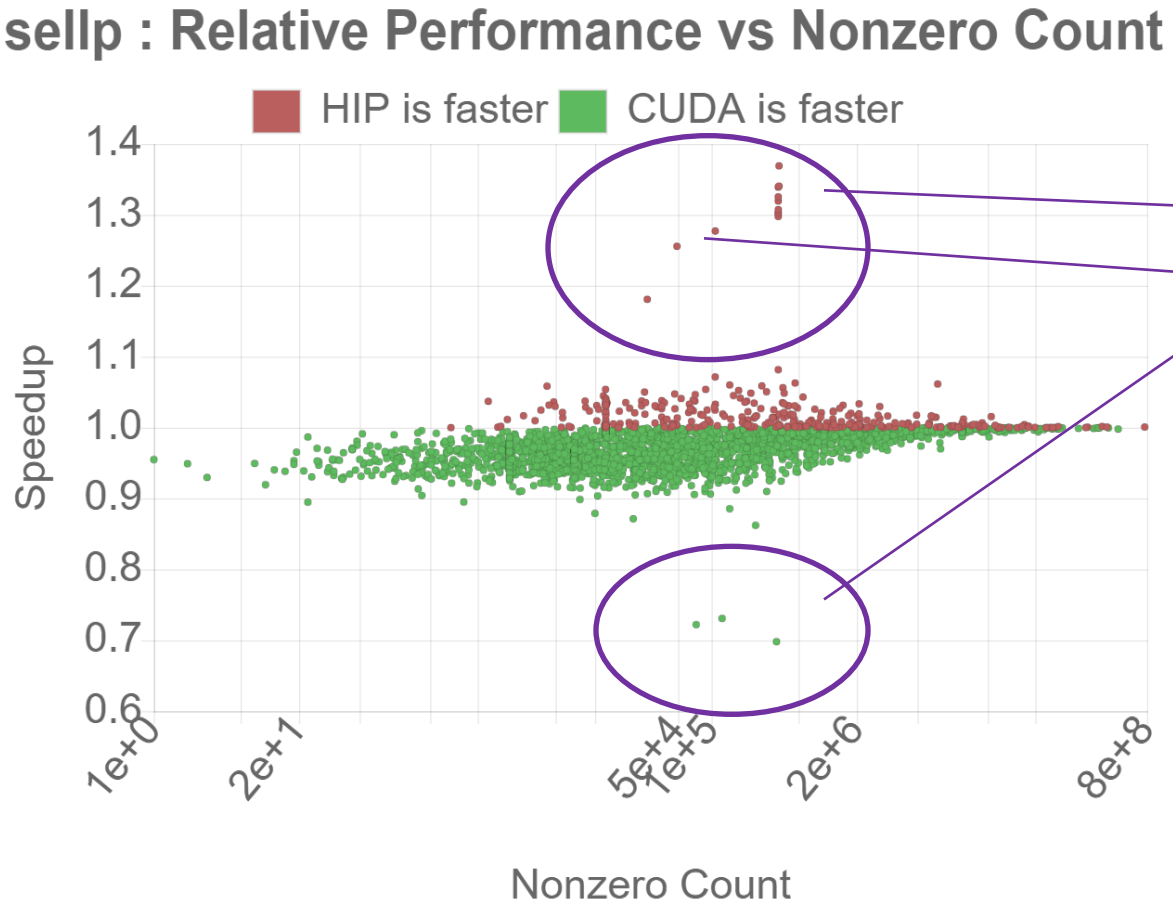
HIP faster than CUDA on NVIDIA GPU?

outliers? machine noise?

Outlier stats on 100 runs a 20 reps:



Compiling HIP code for NVIDIA GPUs – comparison against native CUDA code

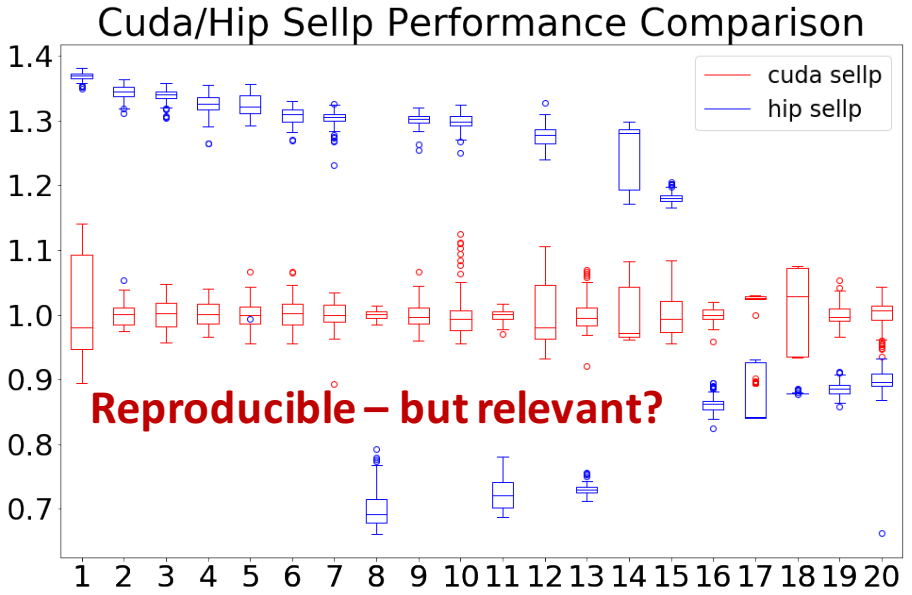


- Native CUDA vs. HIP compiled for NVIDIA GPUs
- Same kernel
- All tests on NVIDIA V100 (Summit)
- We expect CUDA to be slightly faster

HIP faster than CUDA on NVIDIA GPU?

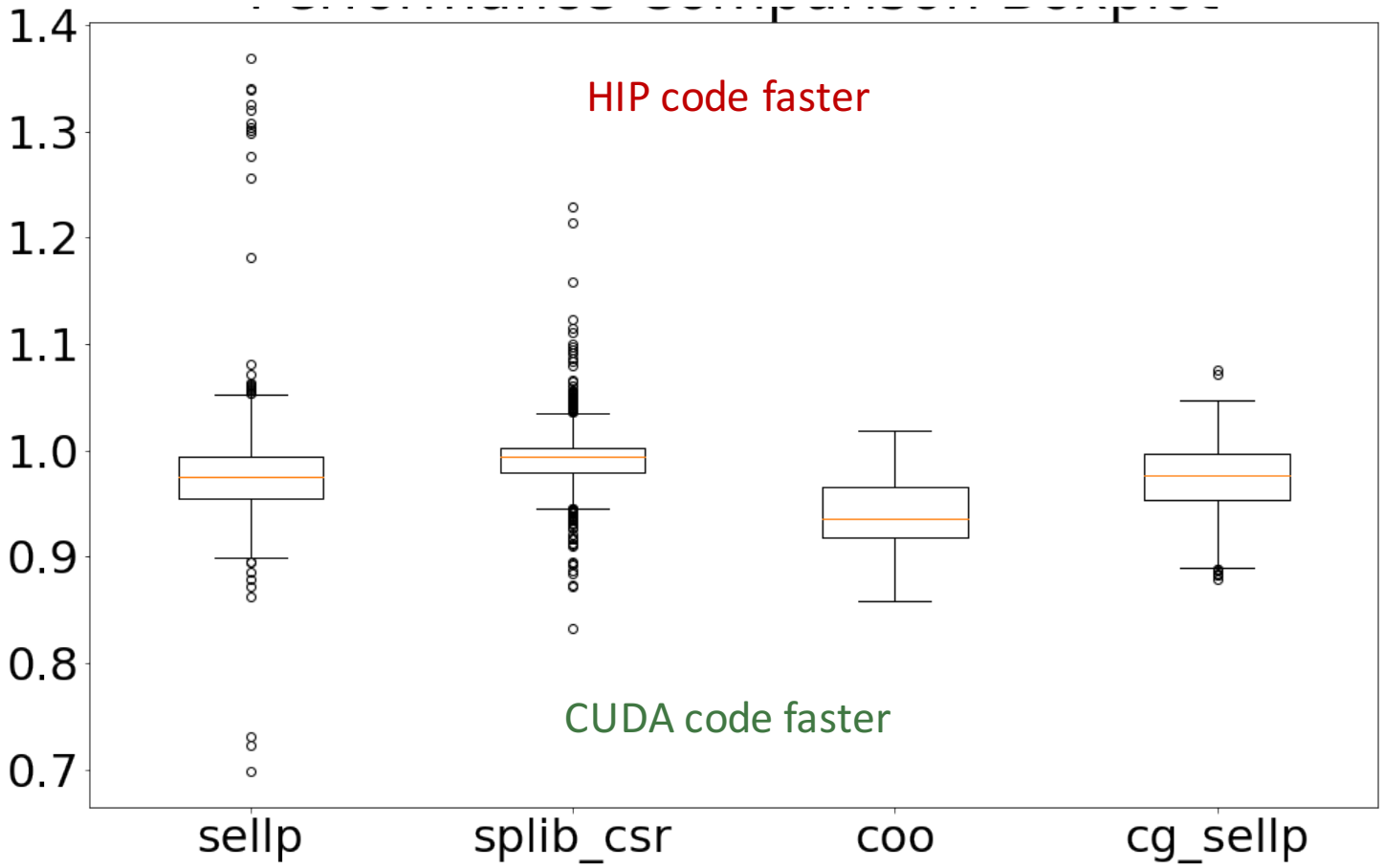
outliers? machine noise?

Outlier stats on 100 runs a 20 reps:



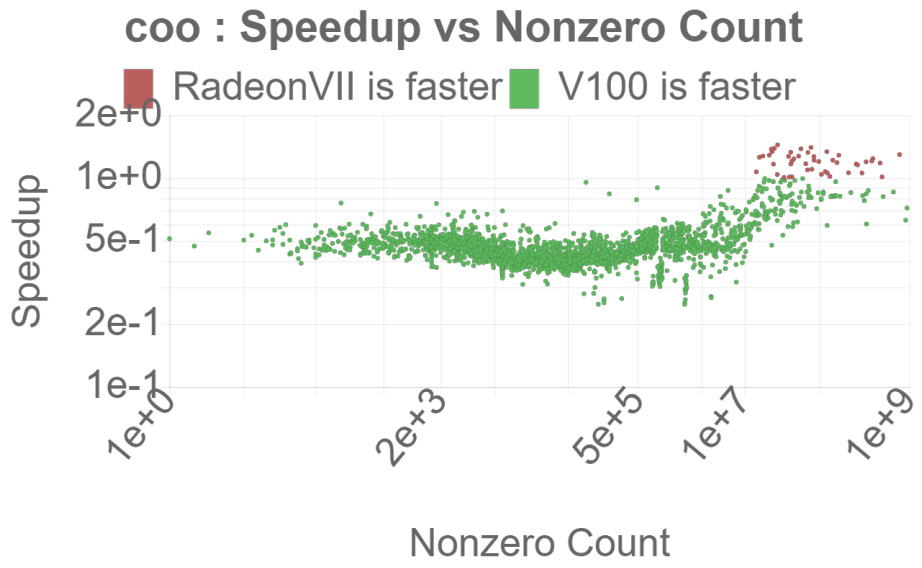
Compiling HIP code for NVIDIA GPUs – comparison against native CUDA code

- Running on V100 GPU
- 2,800 test matrices
- Compare key functionality
  - Ginkgo Sellp SpMV
  - Ginkgo Coo SpMV
  - Vendor's Csr SpMV
  - Ginkgo's CG solver

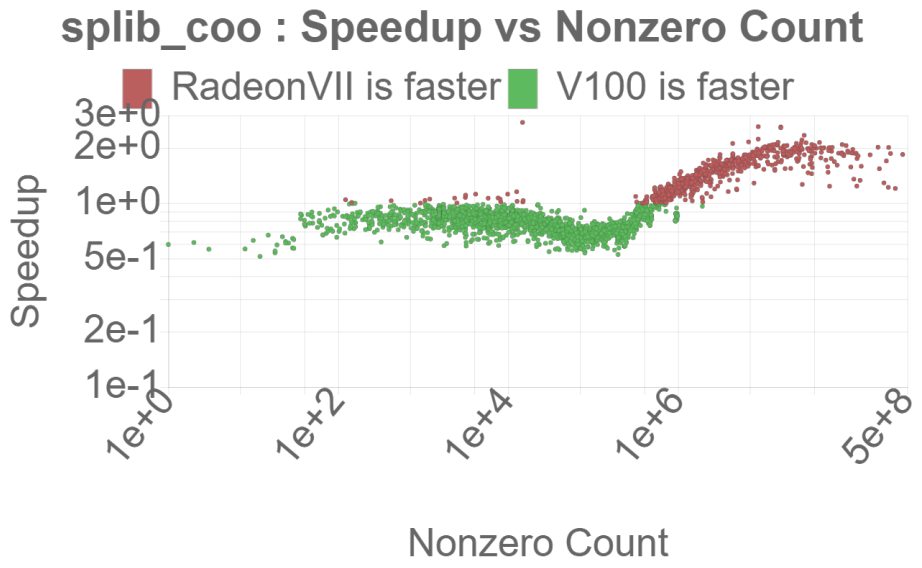


Slight advantages on the CUDA side, but usually <5%.

How do GPU architectures compare in terms of SpMV performance?



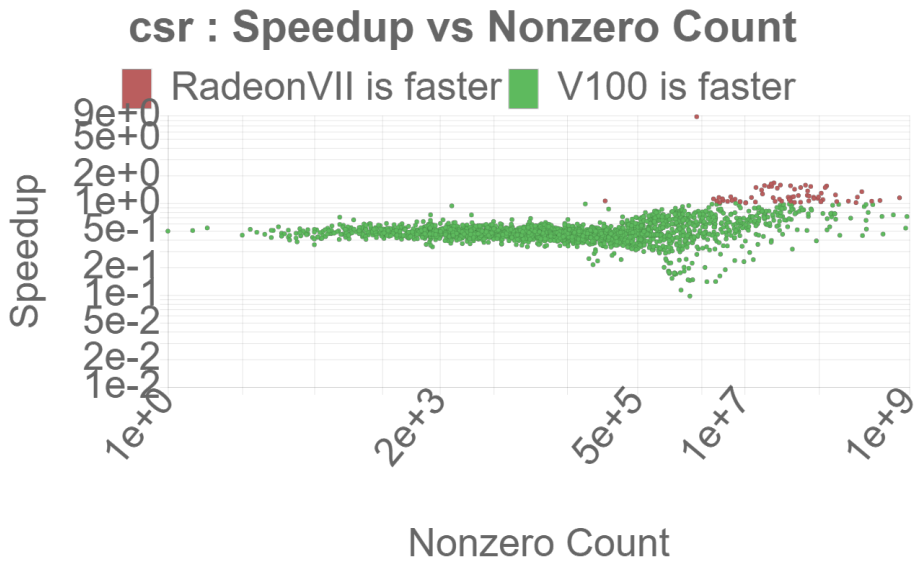
Ginkgo COO SpMV



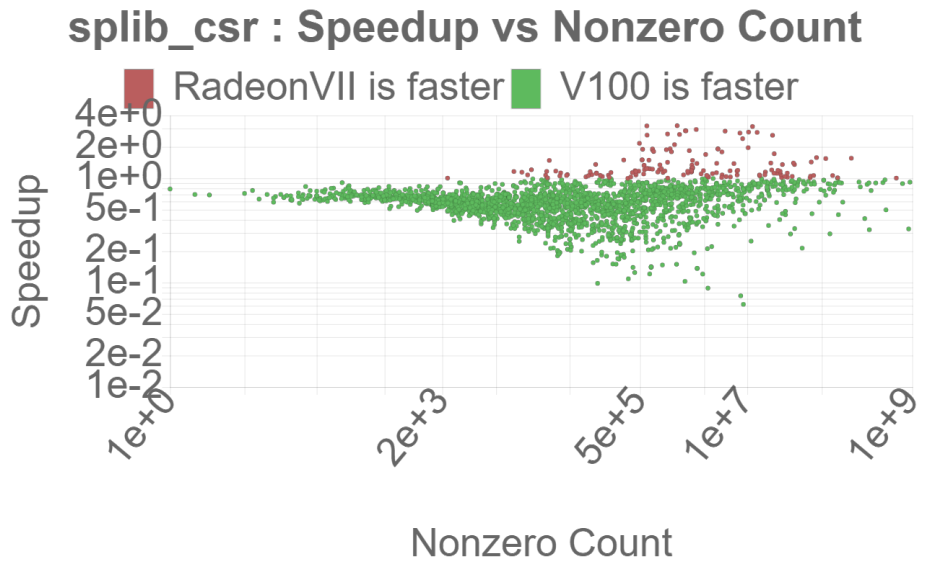
Vendor library COO SpMV

Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

How do GPU architectures compare in terms of SpMV performance?



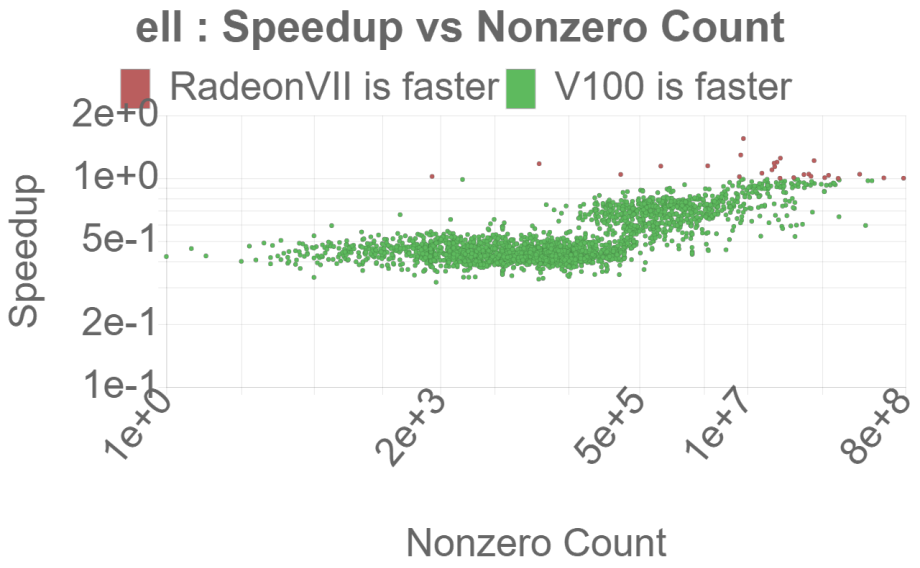
Ginkgo CSR SpMV



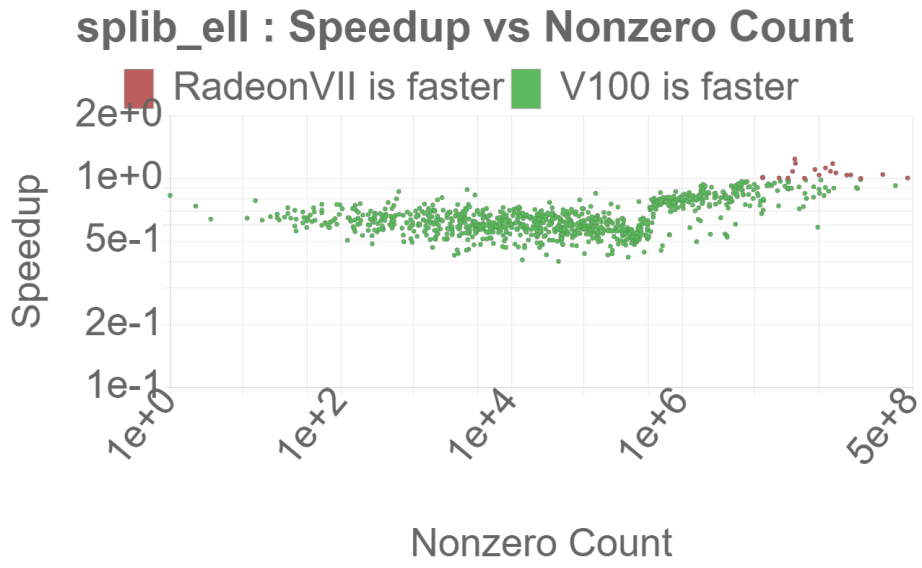
Vendor library CSR SpMV

Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

How do GPU architectures compare in terms of SpMV performance?



Ginkgo ELL SpMV



Vendor library ELL SpMV

Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

Summary: The Sparse Matrix Vector Product on High-End GPUs

- AMD and its **HIP software ecosystem** becomes a relevant alternative to NVIDIA CUDA;
- **Significant similarities** in the languages (HIP/CUDA) allows for shared kernel implementations;
- **HIP allows to compile for NVIDIA GPUs**
  - in most cases with moderate performance loss compared to native CUDA code;
- **AMD** GPUs and **NVIDIA** GPUs **comparable in sparse linear algebra performance**;
- We deployed comprehensive **cross-platform SpMV functionality** in the **Ginkgo** library;
- We provide a comprehensive **SpMV performance study** for **interactive exploration**:

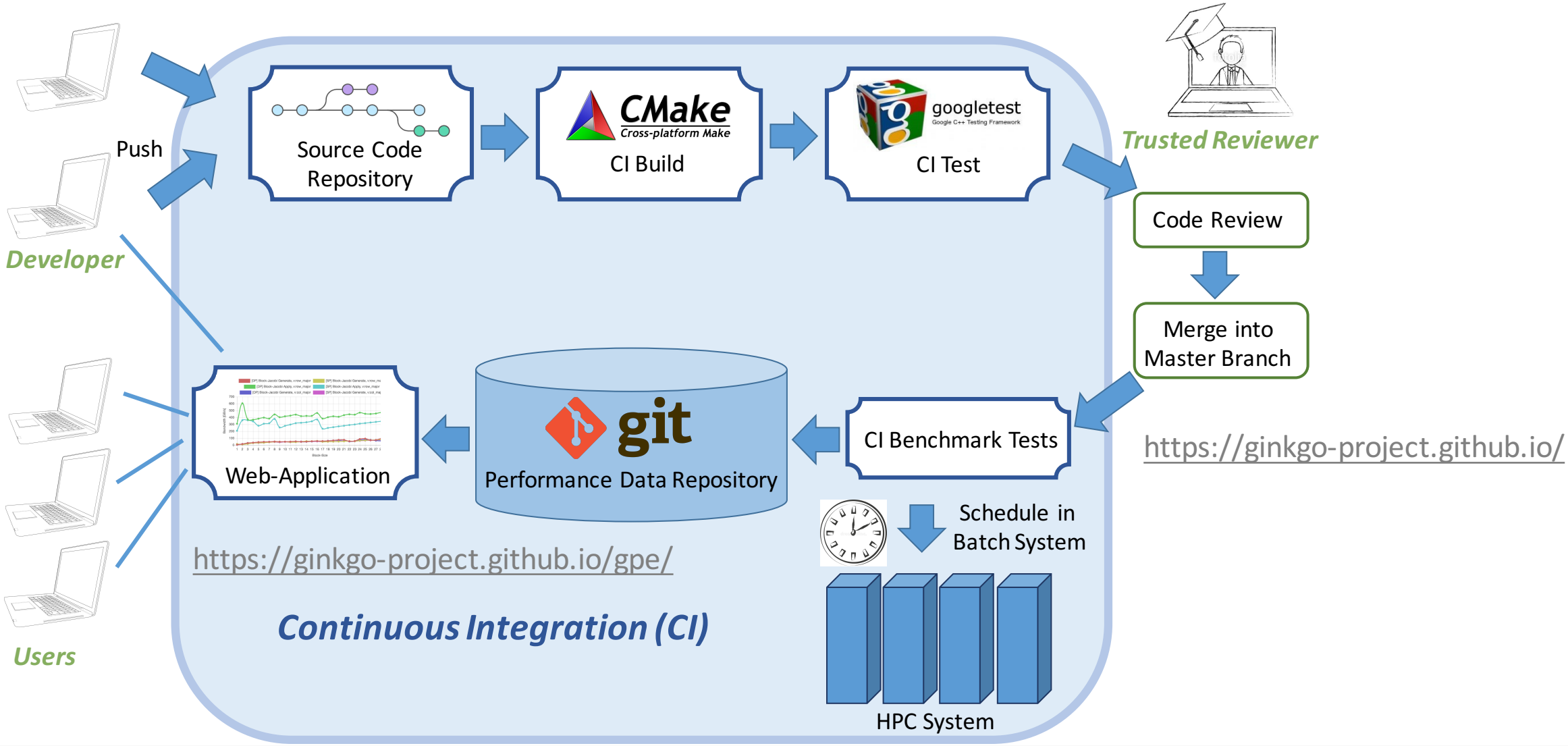
https://ginkgo-project.github.io/gpe/

https://xsdk.info/

Check out our poster on Ginkgo at the poster session tonight in PP2 at 6pm, 5th floor:
Ginkgo - a Node-Level Sparse Linear Algebra Library for High Performance Computing

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration and the Helmholtz Impuls und VernetzungsfondVH-NG-1241.

Backup: Ginkgo development workflow



Backup: Ginkgo Design

- Open-source C++ framework for sparse linear algebra.
- Sparse linear solvers, preconditioners, SpMV etc.
- Focused on Multicore and Manycore accelerators;
- Software quality and sustainability efforts guided by xSDK community policies:

https://xsdk.info/

- 


- 

- 
GitLab CI


- Static polymorphism for templating precisions
  - ValueType (default: Z,C,D,S), IndexType (int32/64)
- Smart pointers to avoid memory leaks
- Runtime polymorphism for operators and kernels
  - Kernels have the same signature for different architectures
  - **Executor** determines which kernel is used

Determines where Data lives & operations are executed

  - **LinOp** class for any linear operator:
    - Matrices
    - Solvers
    - Preconditioners
    - …

generate
apply
…

Backup: GPU format conversion