Porting Linear Algebra Libraries to the AMD Ecosystem

ICL Friday Lunch Talk
06/12/2020

Hartwig Anzt, Terry Cojean, Fritz Goebel, Thomas Gruetzmacher, Pratik Nayak, Tobias Ribizel, Yuhsiang M. Tsai
Steinbuch Centre for Computing (SCC)


Terry Cojean


Fritz Göbel


Thomas Grützmacher


Pratik Nayak

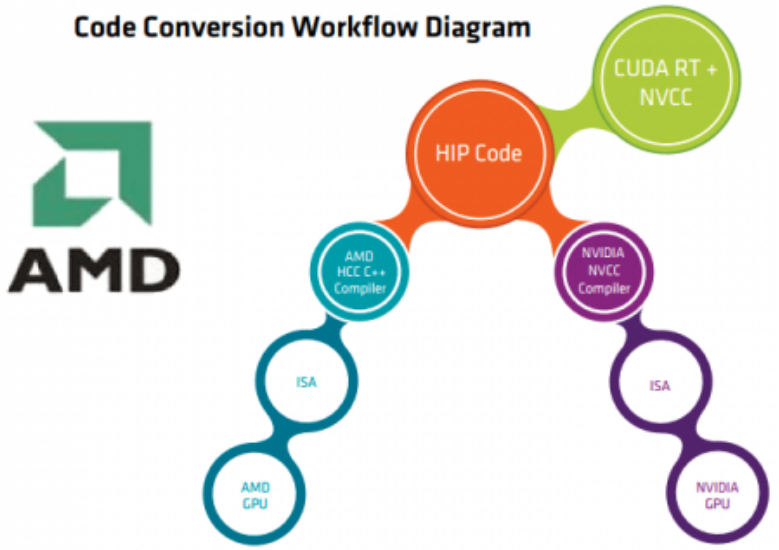

Tobias Ribizel


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.

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
- …

ECP

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

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;

Extend Ginkgo's hardware scope to AMD GPUs

Ginkgo

Core

Library Infrastructure

Library core contains architecture-agnostic

```cuda
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

```hip
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

Core

Library Infrastructure

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;
  kernel::set_value<value>
    <<<dim3(grid_size), dim3(block_size)>>> (
        num, array);
  return 0;
}
```

IDENTICAL

DIFFERENT

HIP

```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;
}
```

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

Optimized architecture-specific kernels;

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;

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

HIP code generation via Cuda2HIP script

1. Generate target filename;
2. Use AMD's *hipify* script to convert CUDA code to HIP code;
3. Changes all CUDA-related headers, namespaces, function names to HIP-specific names;
4. Correct namespace definitions;

```
namespace::kernel<<< ...>>> (...)   -->   namespace::hipLaunchKernelGGL(kernel, ...)
                                          hipLaunchKernelGGL(namespace::kernel, ...)
```

5. Modify the hip *config* file;
6. Check correctness of HIP kernel using unit tests;

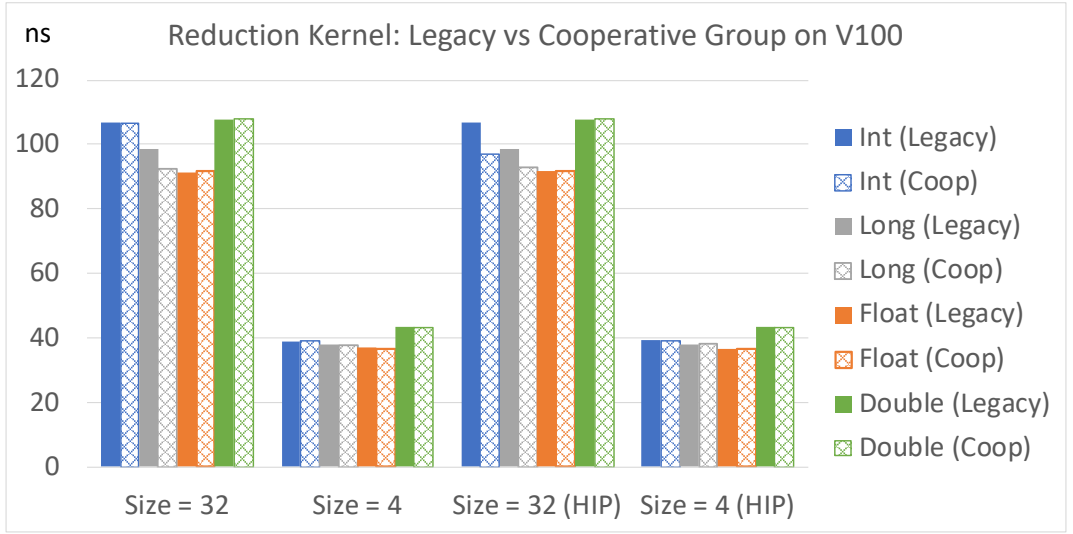
Workflow generating a ”Common” codebase

1. Introduce variables representing the architecture-specific parameters;
2. Move all shared code into a “*common*” file;
3. In the backends, set the architecture-specific parameters;
4. Include the “*common*” file;
5. Use the Cuda2Hip script for converting the code;
6. Modify the hip file *config* to support different architectures;

Some functionality can not be converted with the Cuda2Hip script.
This includes, e.g., some intrinsic functions, cooperative group functions.

Porting Cooperative Group Functionality

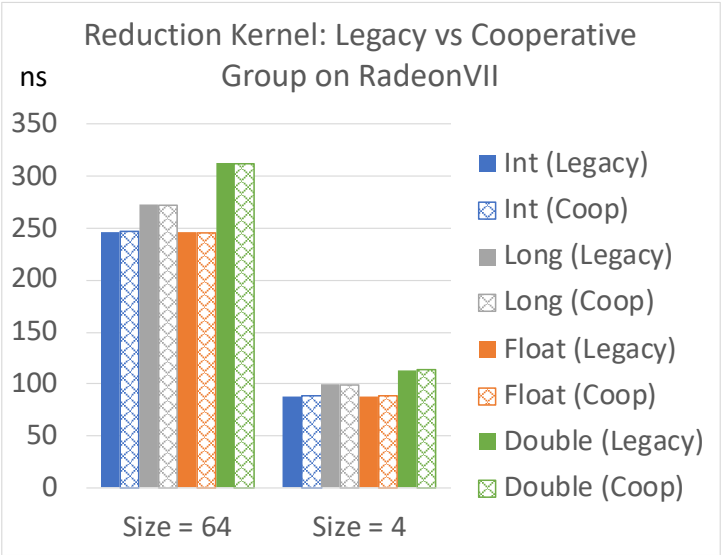
CUDA 9 introduced cooperative groups for flexible thread programming. Cooperative groups provide an interface to handle thread block and warp groups and apply the shuffle. HIP only supports block and grid groups with thread rank() , size() and sync() , but no subwarp-wide group operations like shuffles and vote operations.



Ginkgo's Cooperative Groups

$$\begin{aligned}
\texttt{Size} &= \text{Given subwarp size} \\
\texttt{Rank} &= \texttt{tid \% Size} \\
\texttt{LaneOffset} &= \lfloor \texttt{tid \% warpsize / Size} \rfloor \times \texttt{Size} \\
\texttt{Mask} &= \sim 0 \texttt{ >> (warpsize - Size) << LaneOffset}
\end{aligned}$$

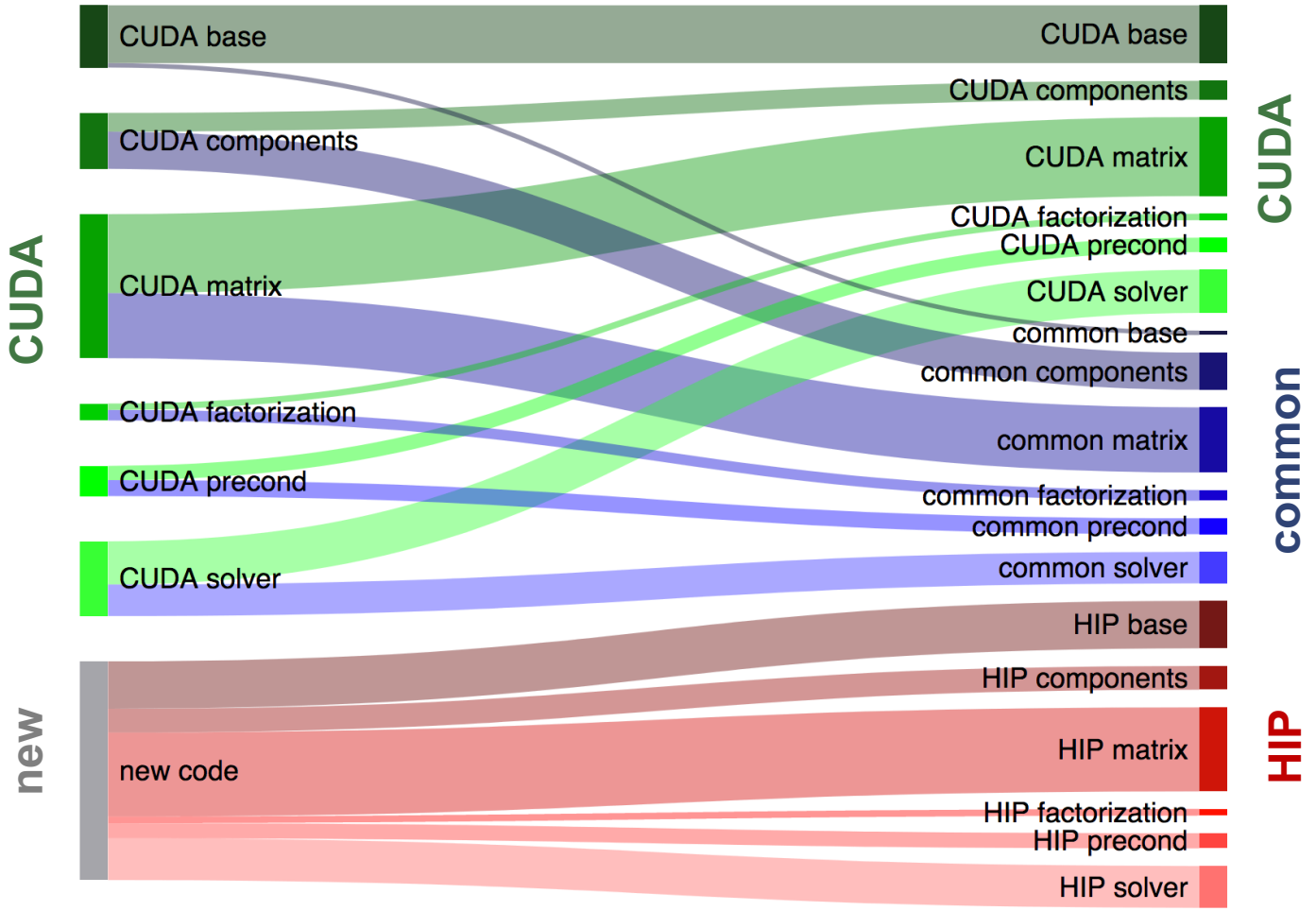
```
subwarp.shfl_xor(data, bitmask) = __shfl_xor(data, bitmask, Size)
     subwarp.ballot(predicate) = (__ballot(predicate) & Mask) >> LaneOffset
        subwarp.any(predicate) = (__ballot(predicate) & Mask) != 0
        subwarp.all(predicate) = (__ballot(predicate) & Mask) == Mask
```



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).

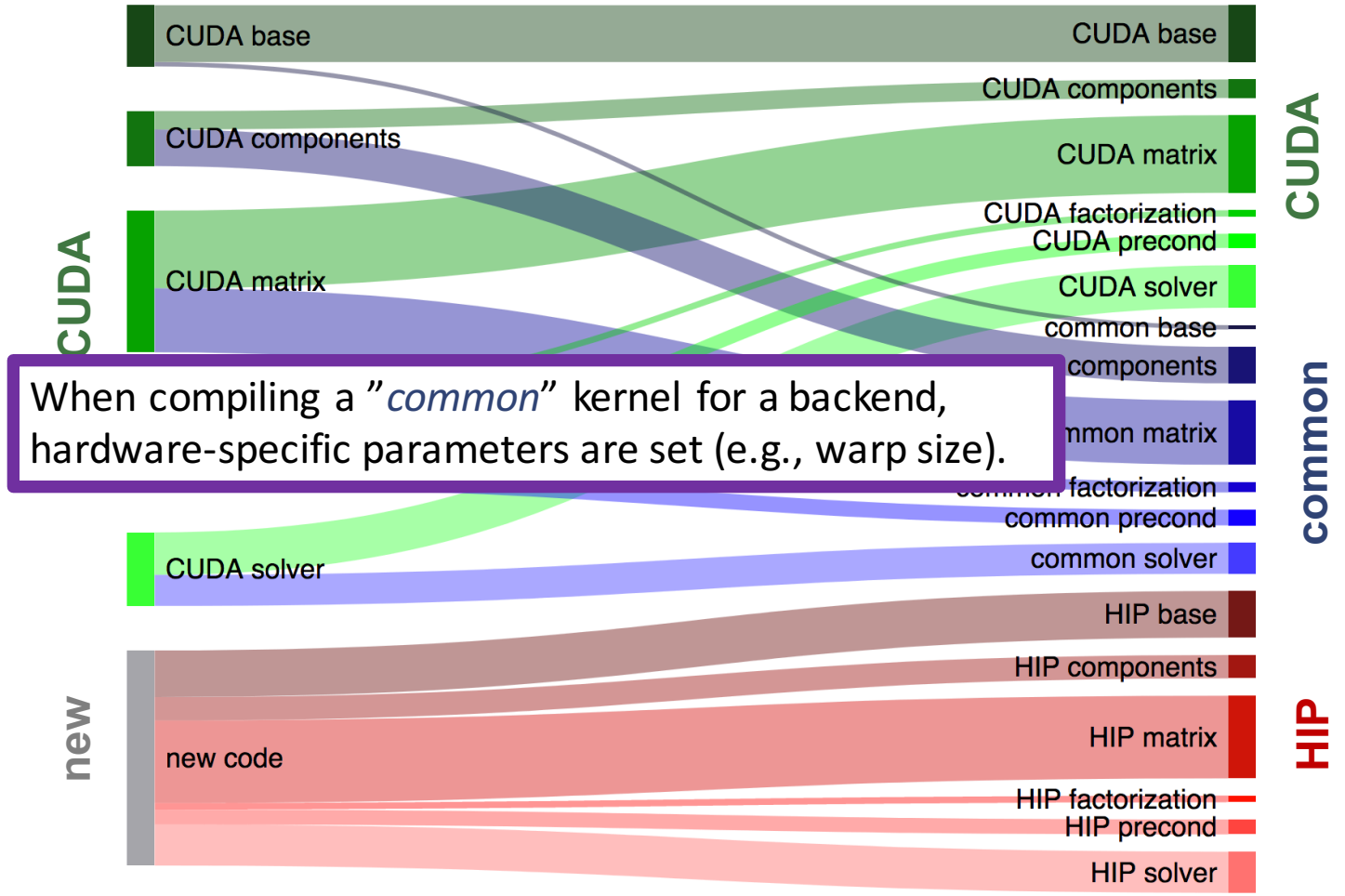
Module	common	cuda	hip
base	112	1435	1176
component	919	467	589
matrix	1617	1908	2048
factorization	262	159	165
preconditioner	395	356	375
solver	780	1071	1038



Extend Ginkgo's hardware scope to AMD GPUs

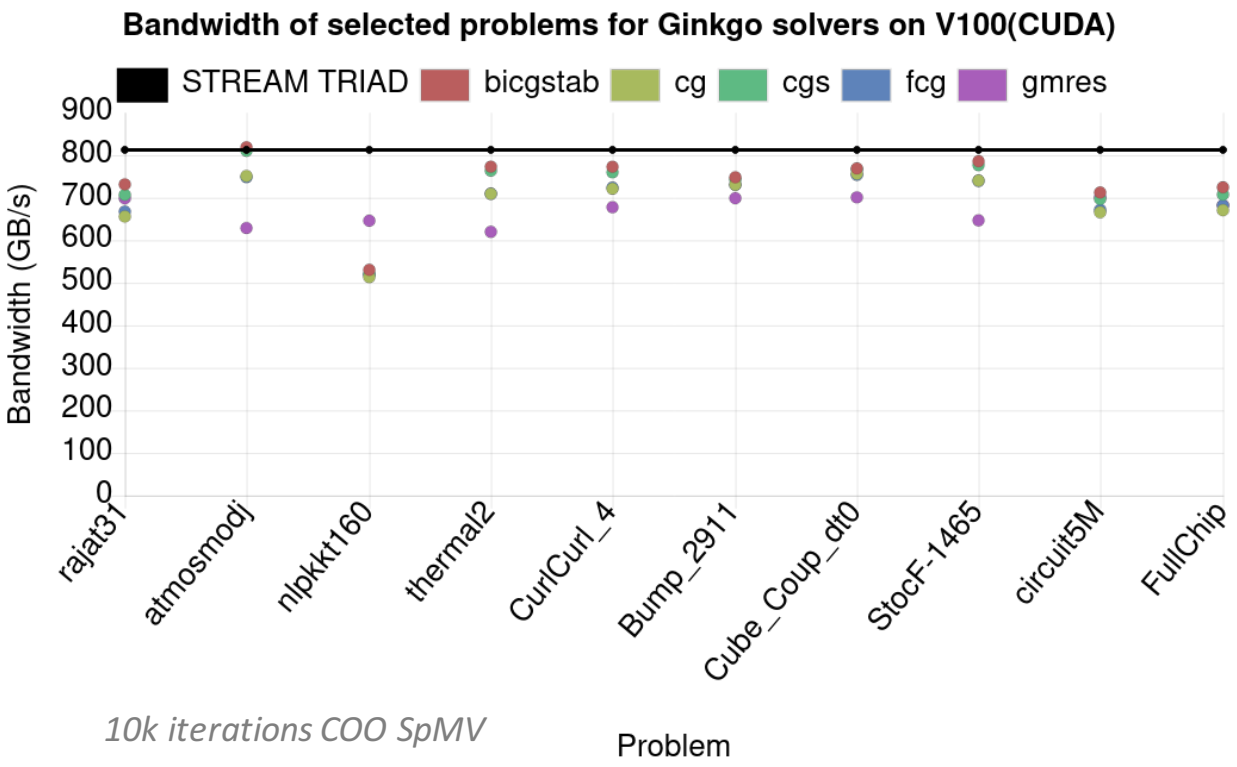
- Kernels shared between CUDA and AMD backends (upon parameter setting) are relocated in the ``common'' module.
- New code necessary for HIP-specific optimizations and for implementing functionality currently missing in the HIP ecosystem (e.g. cooperative groups).

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

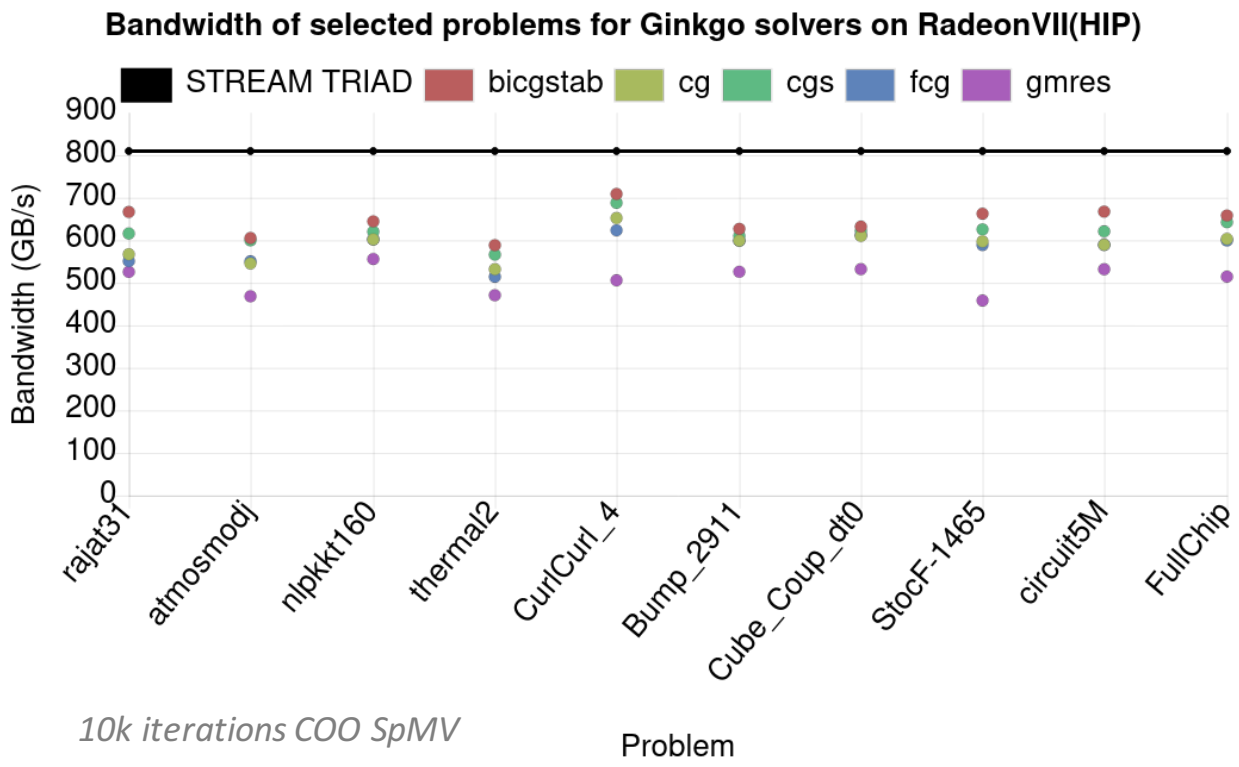


Bandwidth & Performance of the Ginkgo's Krylov solvers

	CUDA V100	AMD RVII
Copy	790.475 GB/s	841.668 GB/s
Scal	787.301 GB/s	841.933 GB/s
Add	811.311 GB/s	806.632 GB/s
Axpy	812.617 GB/s	809.753 GB/s
Dot	844.321 GB/s	635.677 GB/s

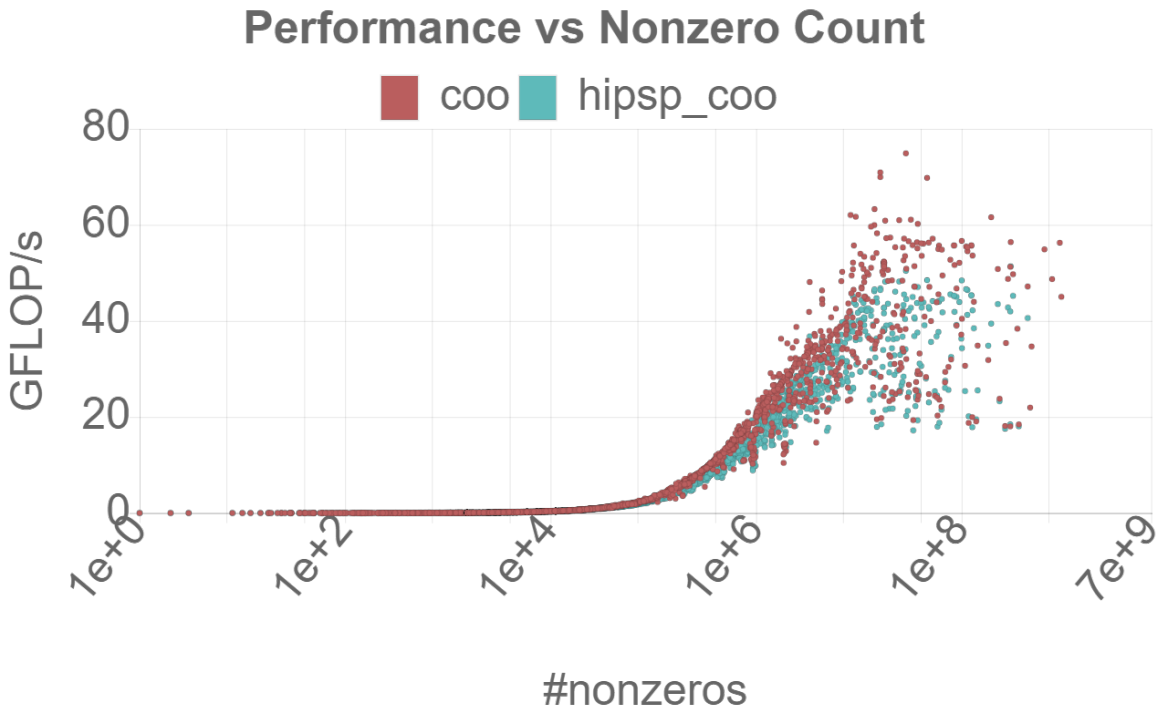


10k iterations COO SpMV



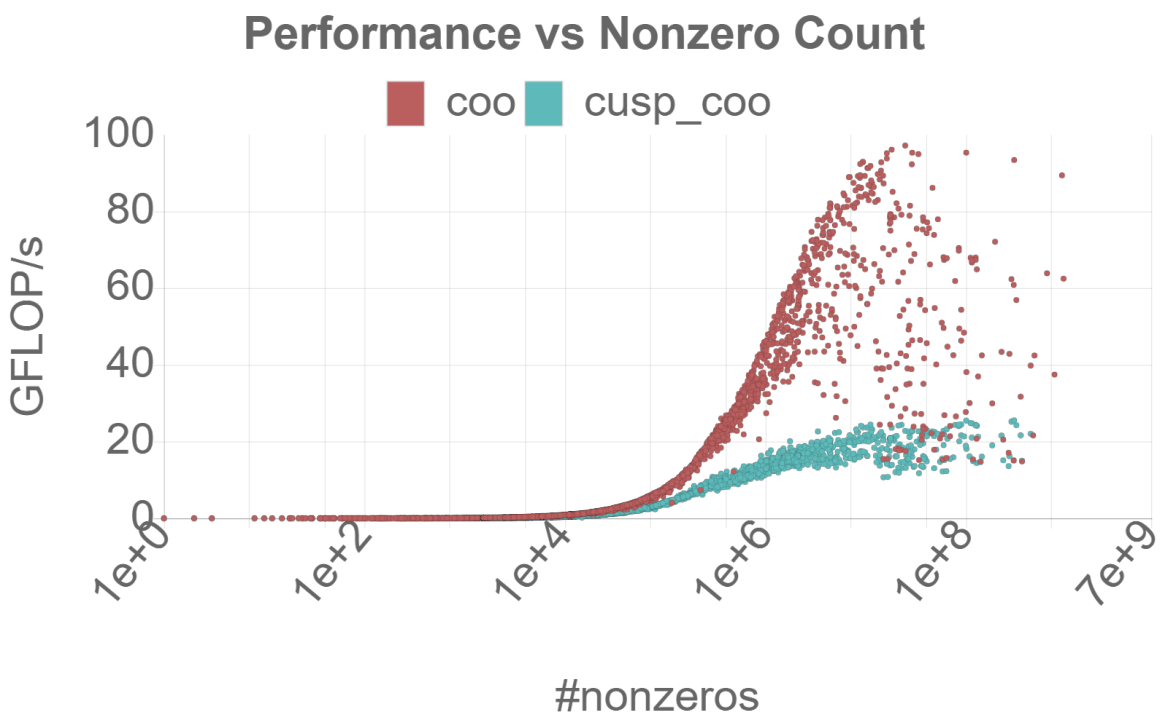
10k iterations COO SpMV

How does Ginkgo compare to the vendor libraries - COO SpMV



Ginkgo vs HIPsparse on RadeonVII

HIP v.3.1.20044-3684ef8

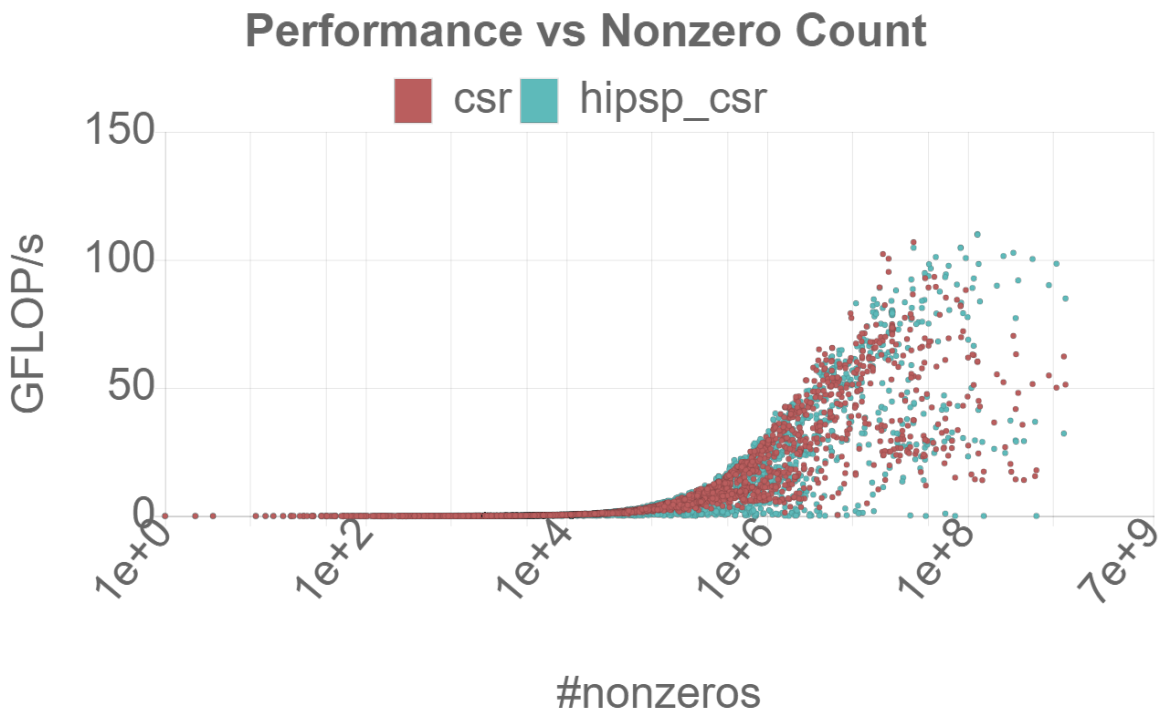


Ginkgo vs cuSPARSE on V100

Cuda v.9.2.214

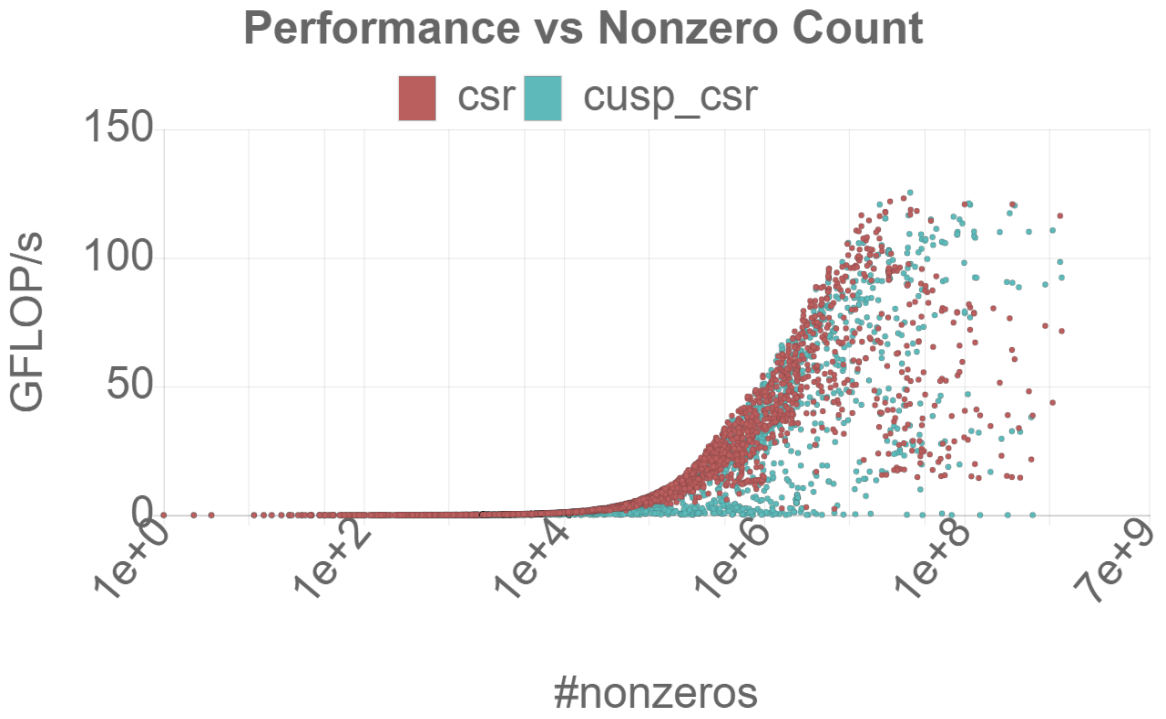
Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

How does Ginkgo compare to the vendor libraries - CSR SpMV



Ginkgo vs HIPsparse on RadeonVII

HIP v.3.1.20044-3684ef8

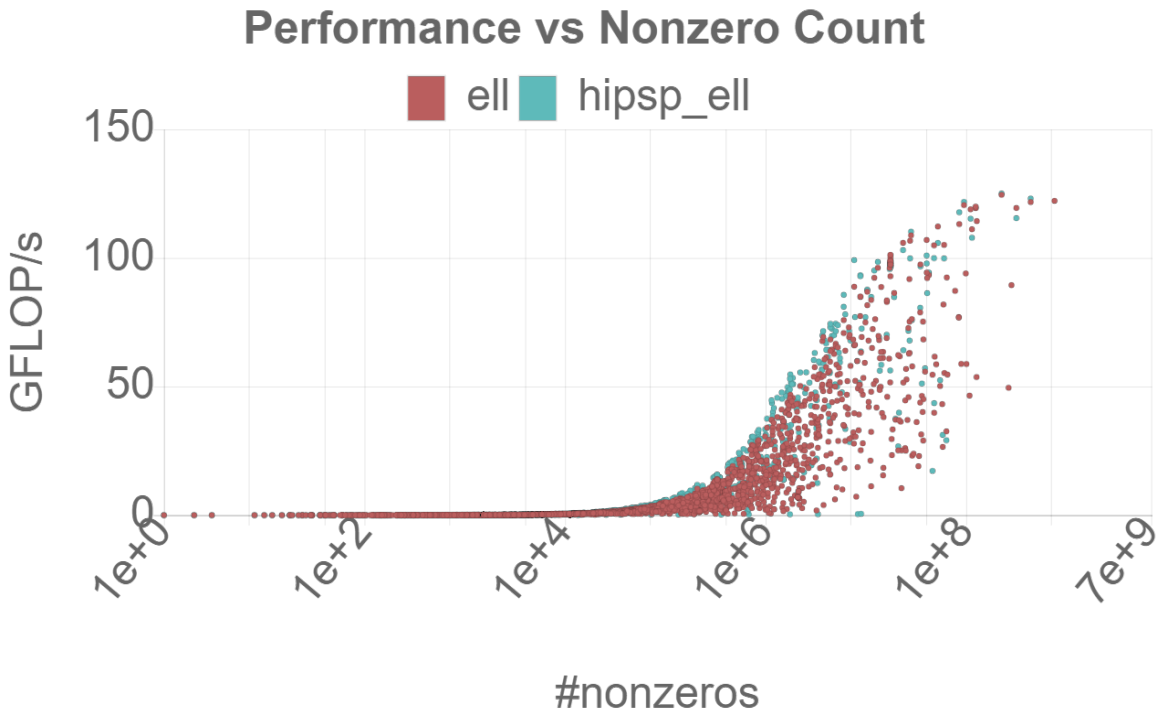


Ginkgo vs cuSPARSE on V100

Cuda v.9.2.214

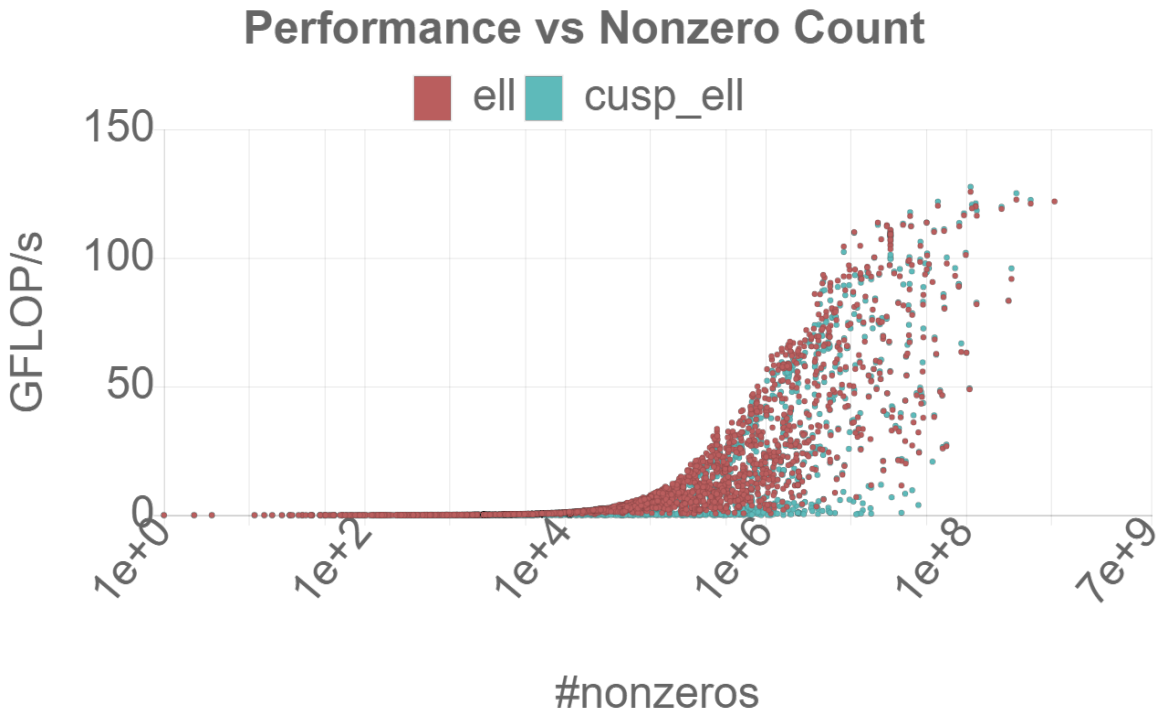
Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

How does Ginkgo compare to the vendor libraries - ELL SpMV



Ginkgo vs HIPsparse on RadeonVII

HIP v.3.1.20044-3684ef8

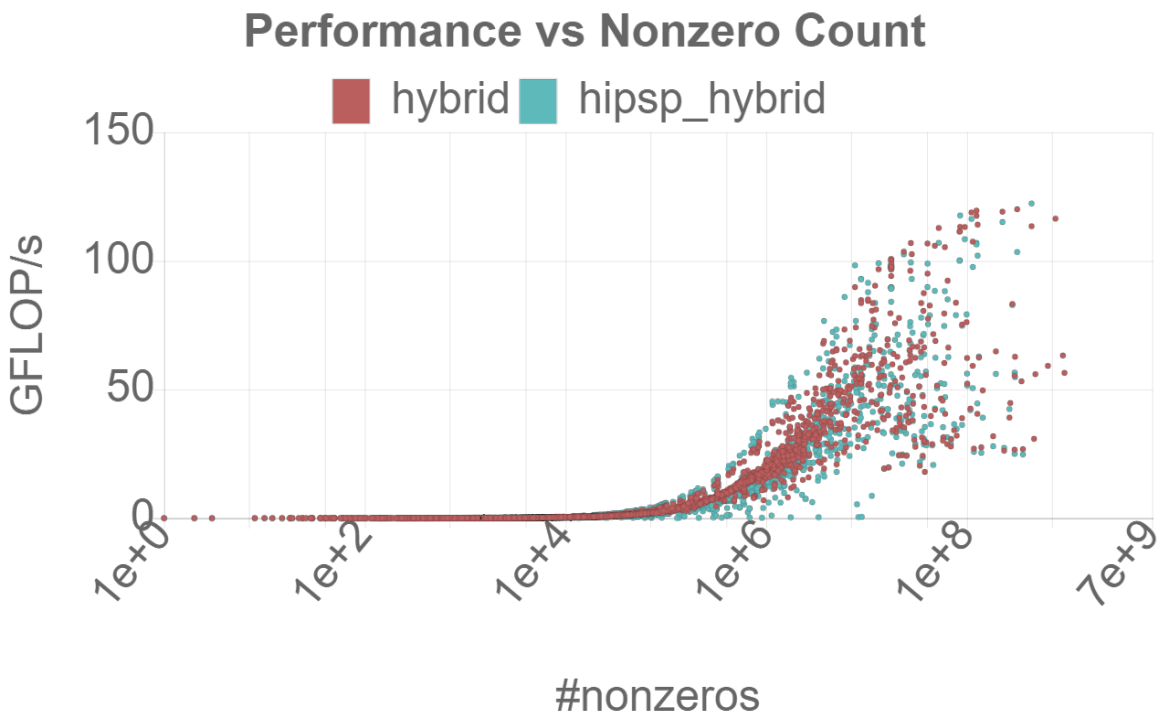


Ginkgo vs cuSPARSE on V100

Cuda v.9.2.214

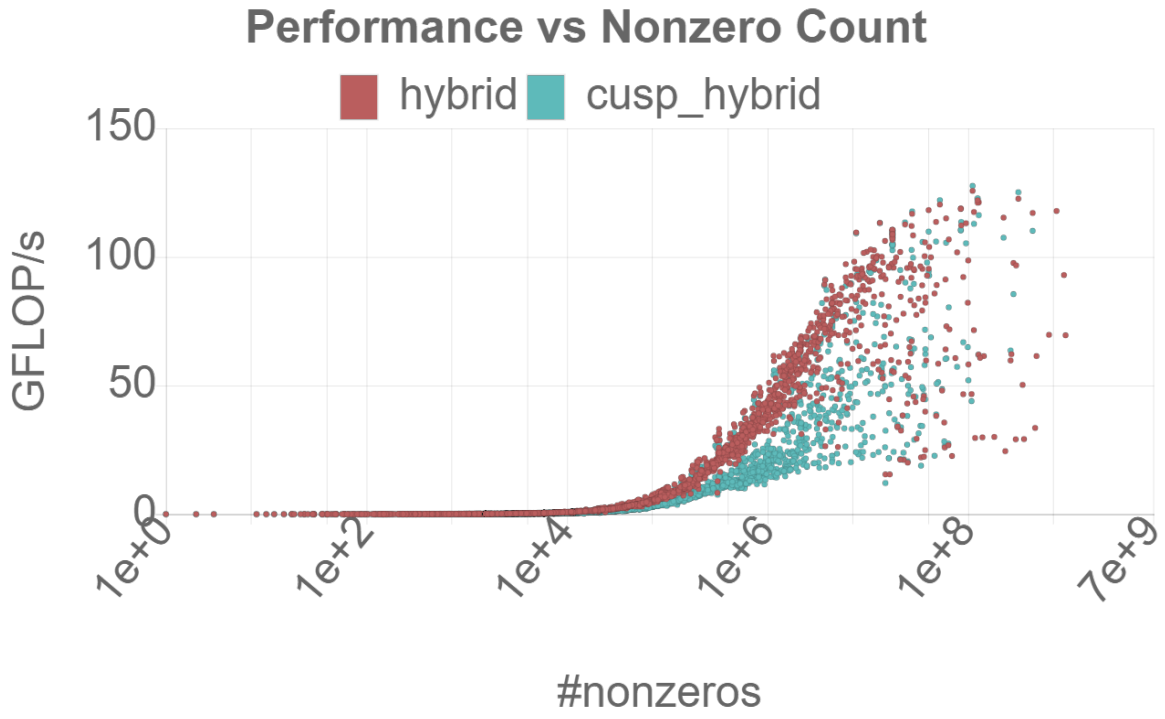
Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

How does Ginkgo compare to the vendor libraries - hybrid SpMV



Ginkgo vs HIPsparse on RadeonVII

HIP v.3.1.20044-3684ef8



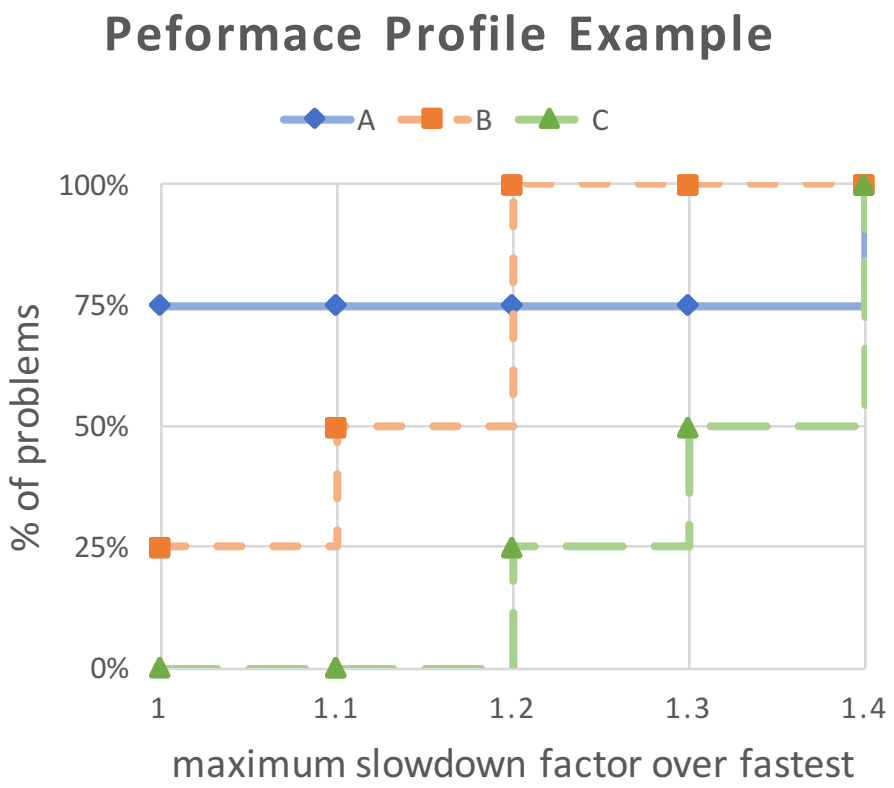
Ginkgo vs cuSPARSE on V100

Cuda v.9.2.214

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

Performance Profile

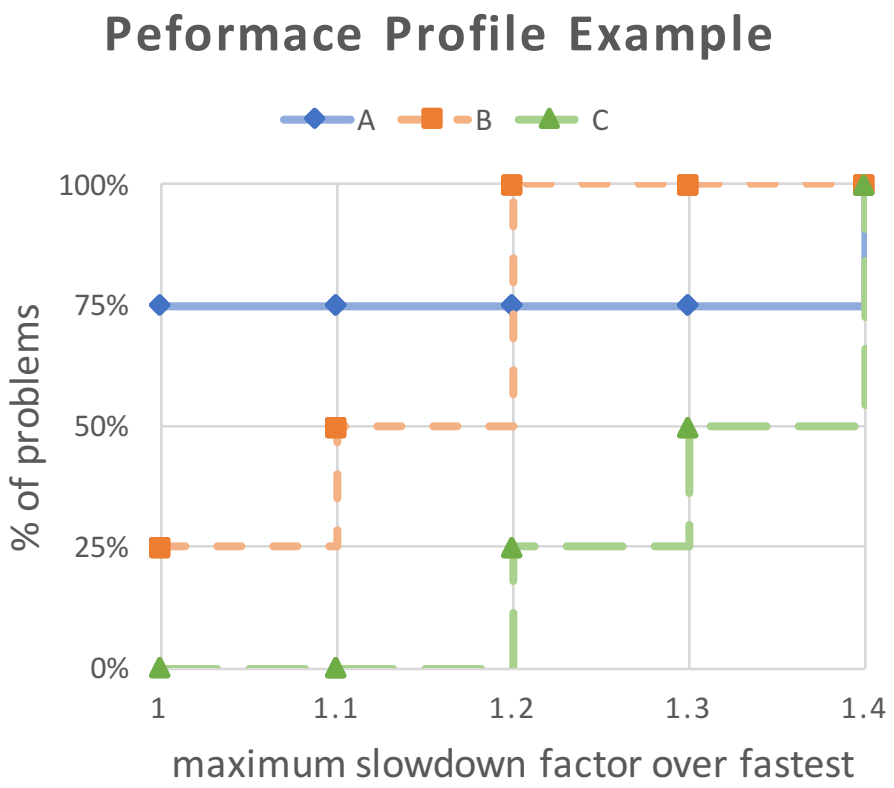
time	A	B	C
Prob1	1	1.1	1.25
Prob2	1.2	1.38	1.68
Prob3	1.5	1.8	1.74
Prob4	2.7	2	2.8



Performance Profile

time	A	B	C
Prob1	1	1.1	1.25
Prob2	1.2	1.38	1.68
Prob3	1.5	1.8	1.74
Prob4	2.7	2	2.8

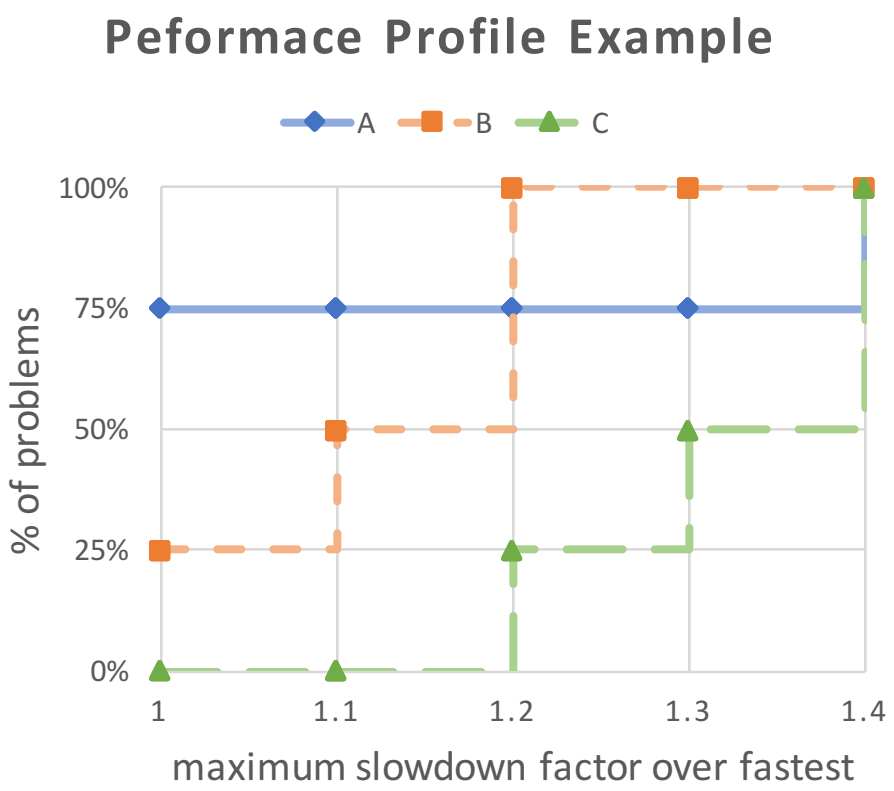
Use the fastest to normalize time get the relative ratio



Performance Profile

ratio	A	B	C
Prob1	1	1.1	1.25
Prob2	1	1.15	1.4
Prob3	1	1.2	1.16
Prob4	1.35	1	1.4

Use the fastest to normalize time get the relative ratio



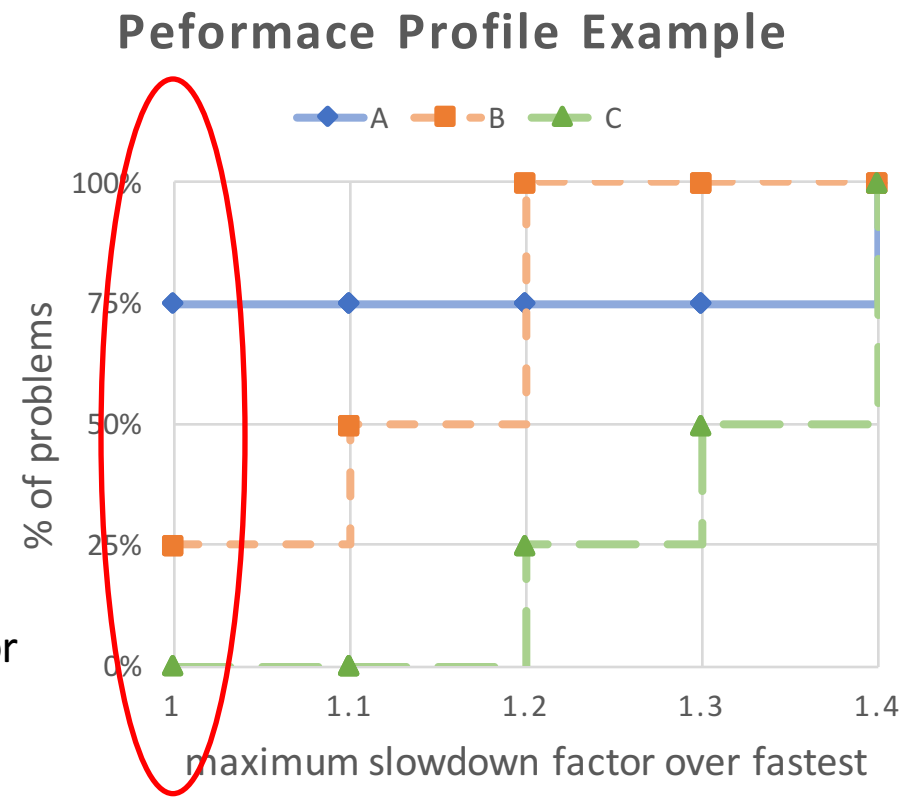
Performance Profile

ratio	A	B	C
Prob1	1	1.1	1.25
Prob2	1	1.15	1.4
Prob3	1	1.2	1.16
Prob4	1.35	1	1.4

Get the item whose ratio smaller than (<=) the slowdown factor and count the percentage of total problems

slowdown factor = 1

	75%	25%	0%



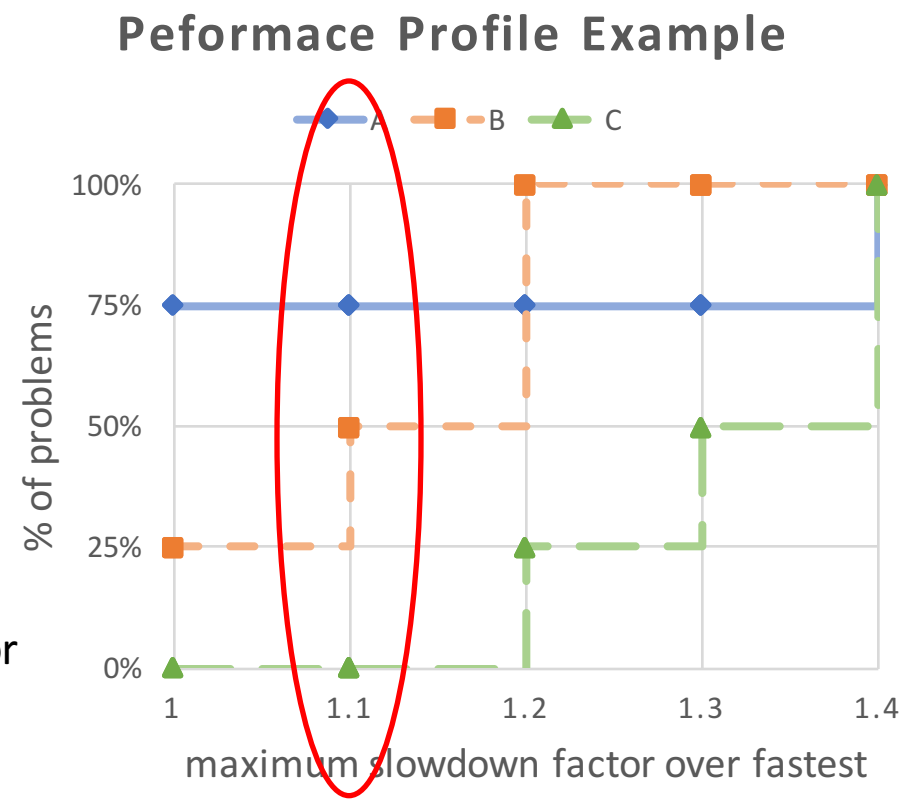
Performance Profile

ratio	A	B	C
Prob1	1	1.1	1.25
Prob2	1	1.15	1.4
Prob3	1	1.2	1.16
Prob4	1.35	1	1.4

Get the item whose ratio smaller than (<=) the slowdown factor and count the percentage of total problems

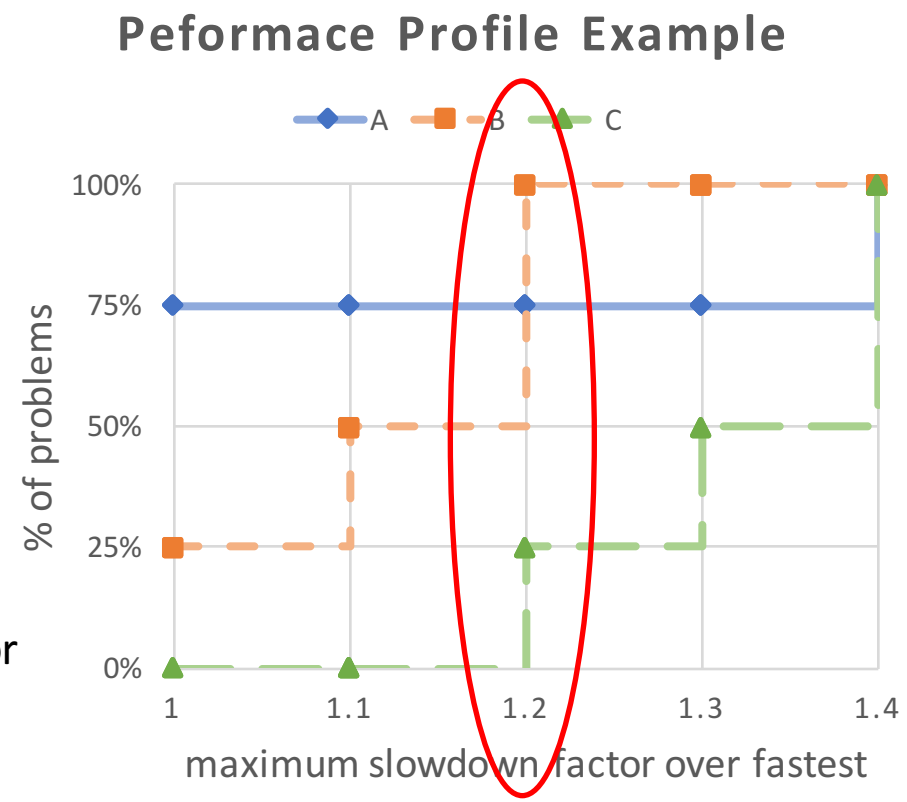
slowdown factor = 1.1

	75%	50%	0%



Performance Profile

ratio	A	B	C
Prob1	1	1.1	1.25
Prob2	1	1.15	1.4
Prob3	1	1.2	1.16
Prob4	1.35	1	1.4



Get the item whose ratio smaller than (<=) the slowdown factor and count the percentage of total problems

slowdown factor = 1.2

	75%	100%	25%

Performance Profile

ratio	A	B	C
Prob1	1	1.1	1.25
Prob2	1	1.15	1.4
Prob3	1	1.2	1.16
Prob4	1.35	1	1.4

Get the item whose ratio smaller than (<=) the slowdown factor and count the percentage of total problems

slowdown factor = 1.3

	75%	100%	50%

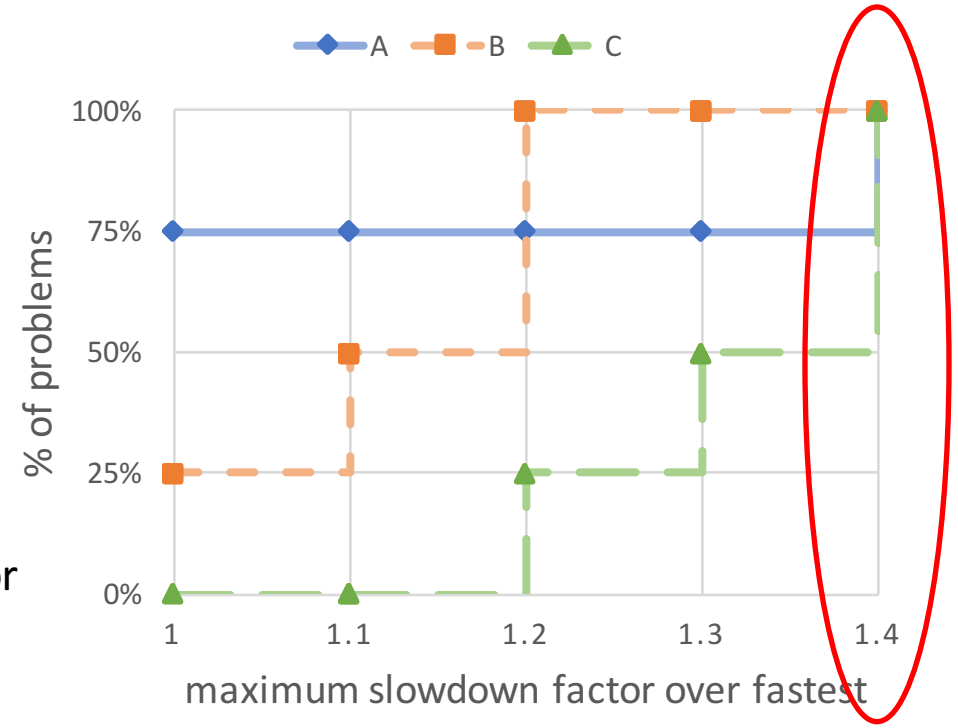
Performance Profile

ratio	A	B	C
Prob1	1	1.1	1.25
Prob2	1	1.15	1.4
Prob3	1	1.2	1.16
Prob4	1.35	1	1.4

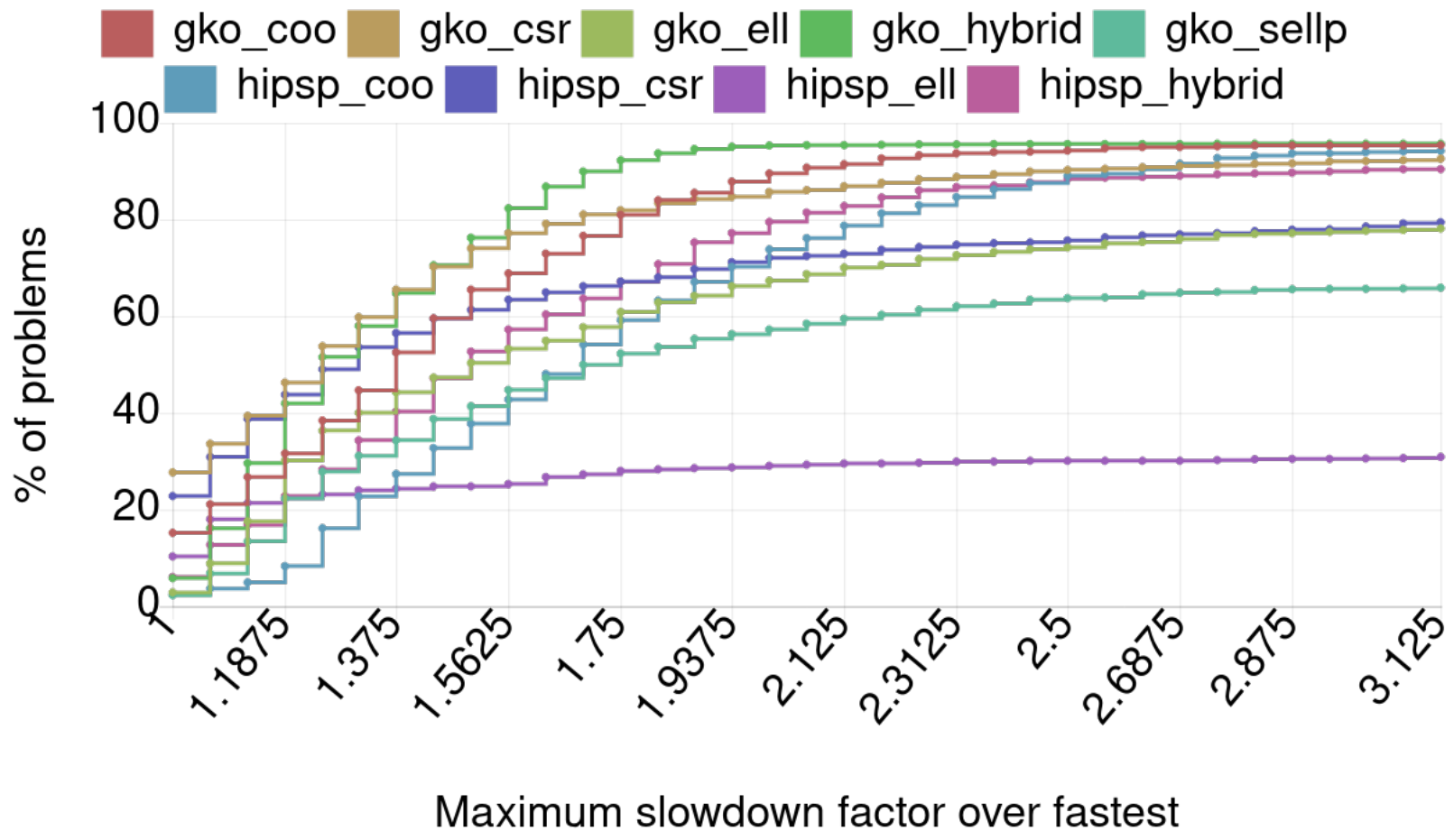
Get the item whose ratio smaller than (<=) the slowdown factor and count the percentage of total problems

slowdown factor = 1.4

	100%	**100%**	**100%**

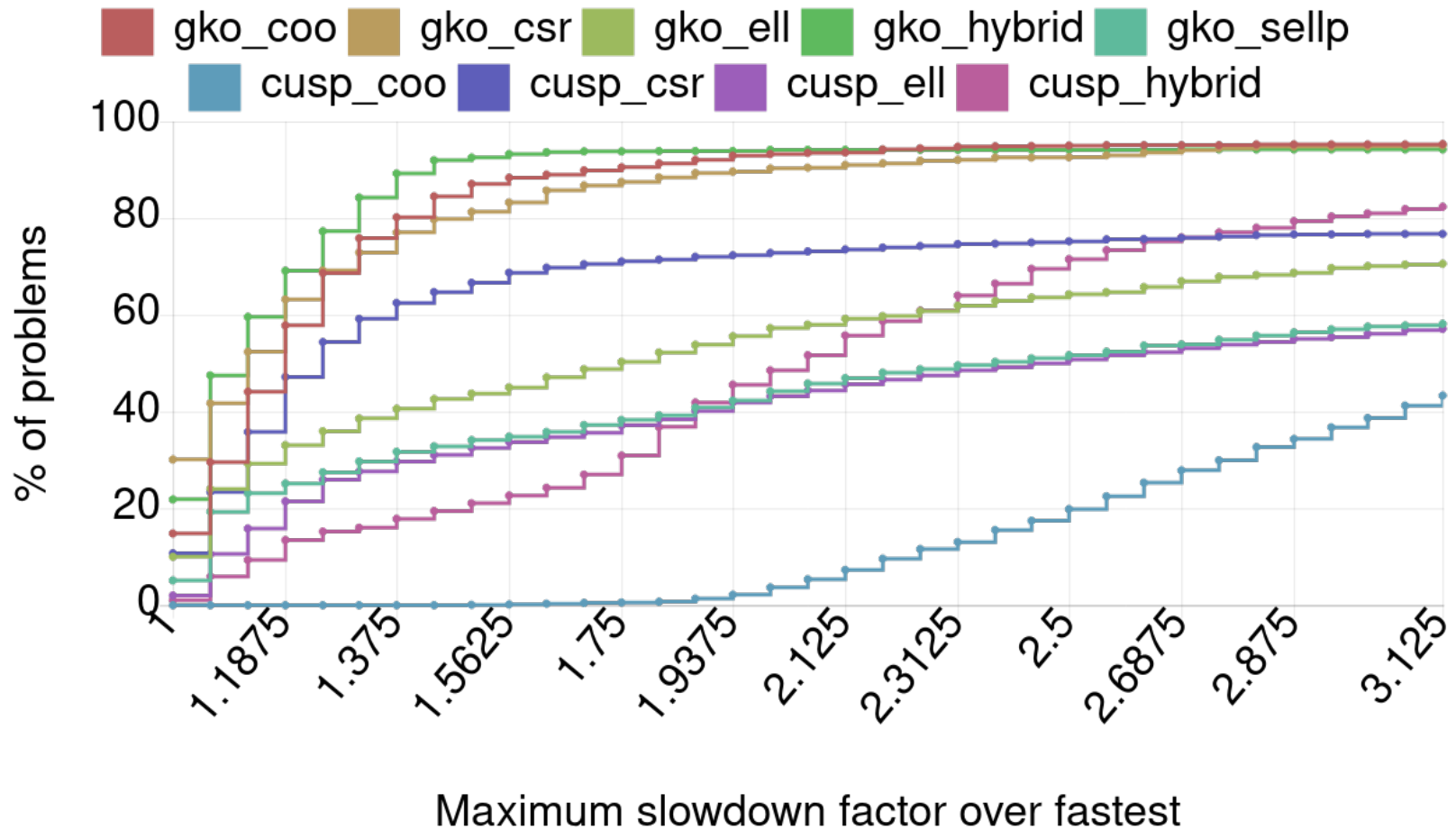


Performance Profile on AMD's RadeonVII



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

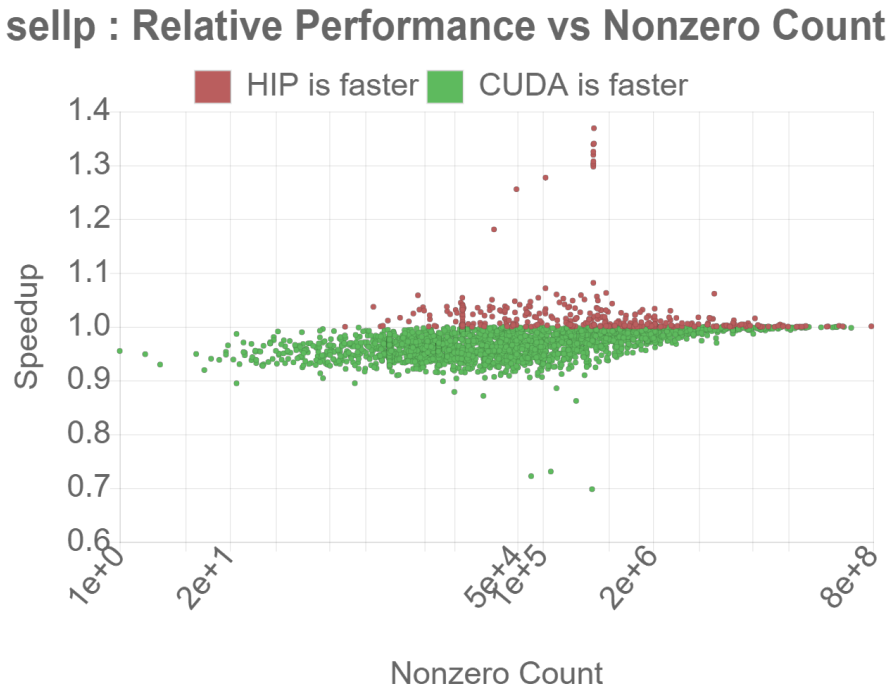
Performance Profile on NVIDIA's V100

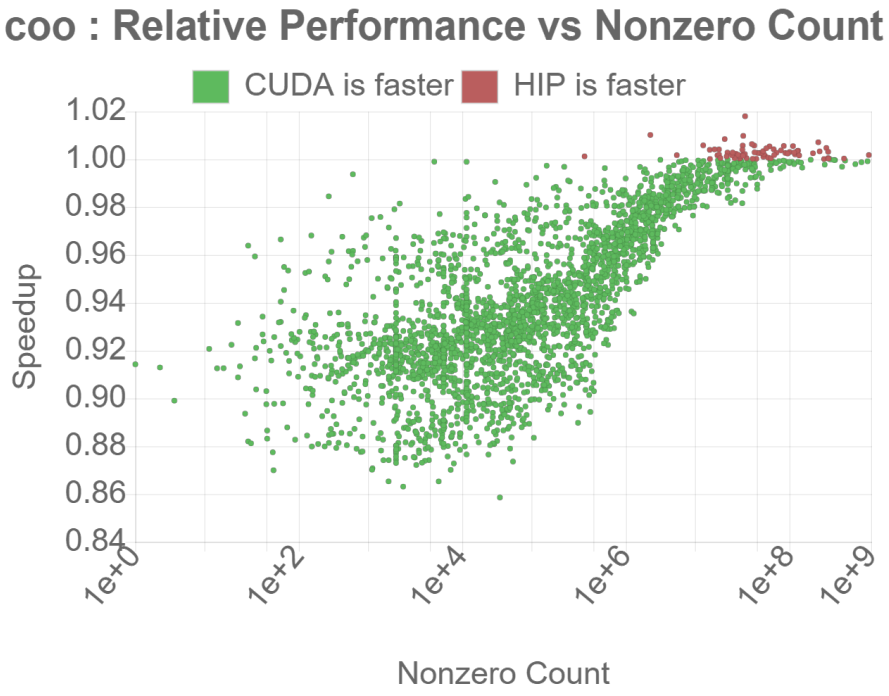


Results and interactive performance explorer available at: https://ginkgo-project.github.io/gpe/

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

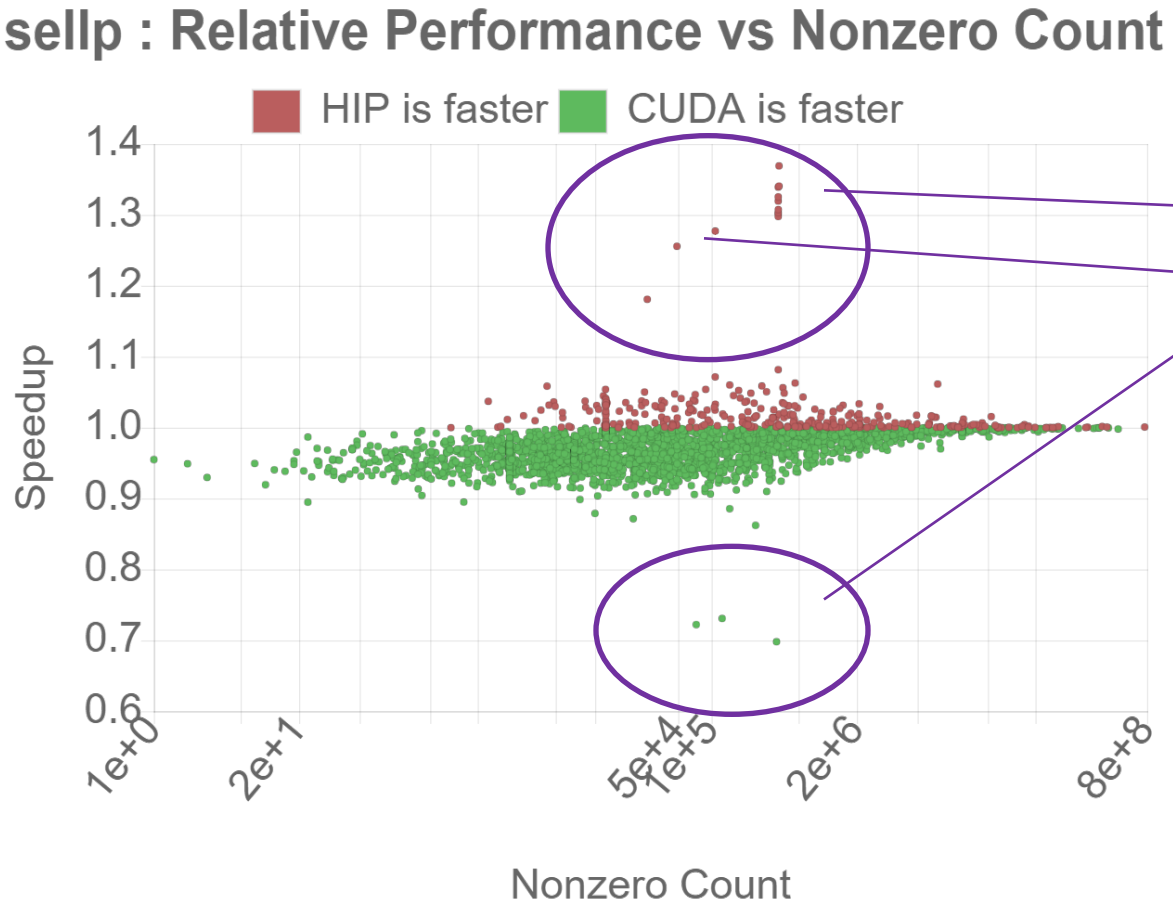
- Native CUDA vs. HIP compiled for NVIDIA GPUs
- Same kernel
- All tests on NVIDIA V100 (Summit)
- We expect CUDA to be slightly faster





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

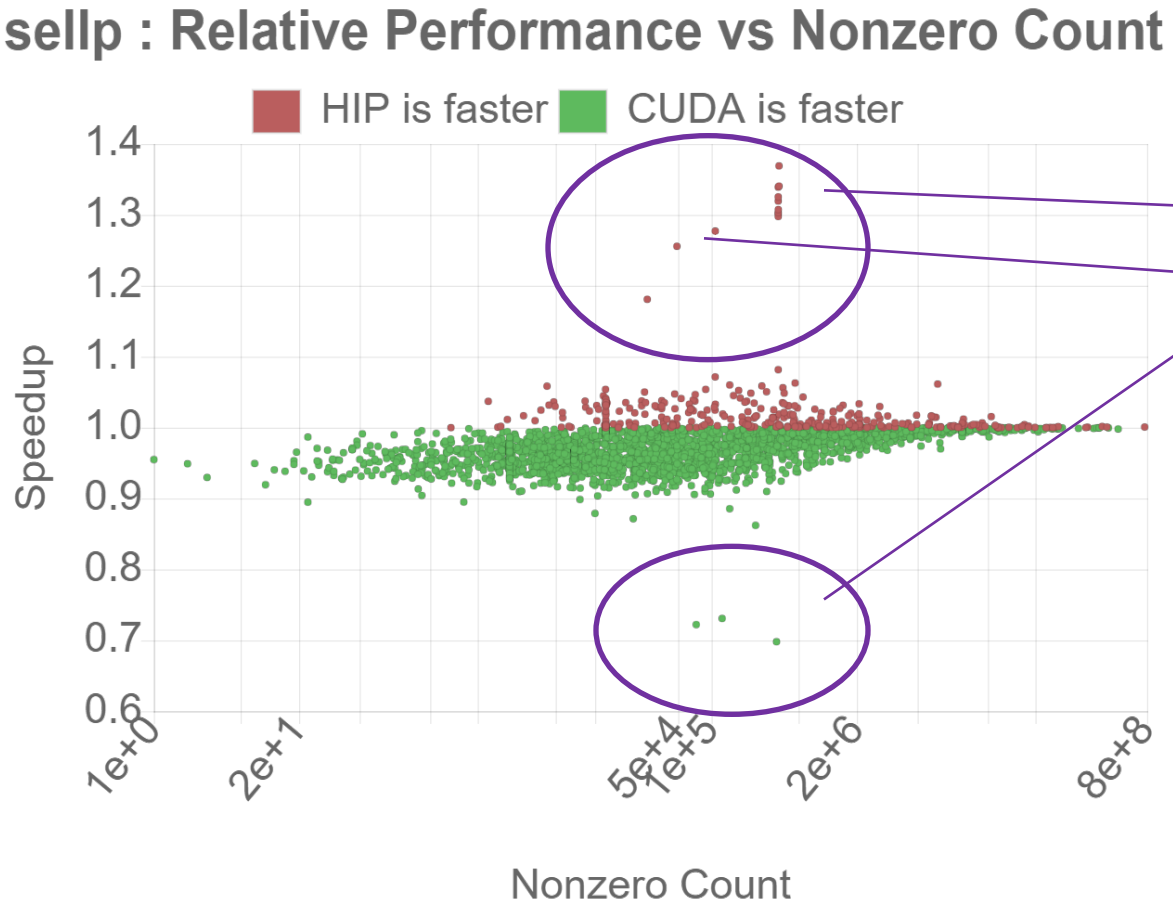
- Native CUDA vs. HIP compiled for NVIDIA GPUs
- Same kernel
- All tests on NVIDIA V100 (Summit)
- We expect CUDA to be slightly faster



HIP faster than CUDA on NVIDIA GPU?

outliers? machine noise?

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

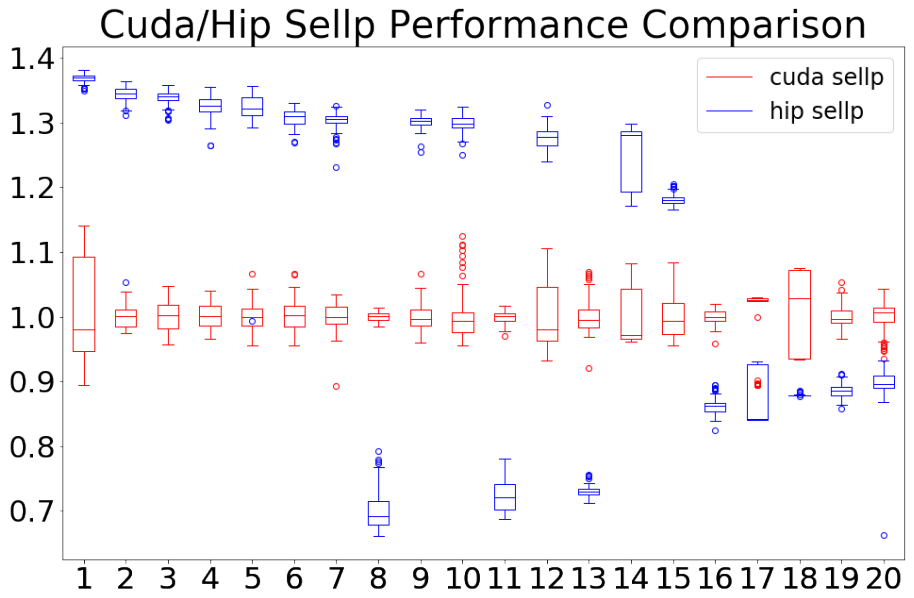


- Native CUDA vs. HIP compiled for NVIDIA GPUs
- Same kernel
- All tests on NVIDIA V100 (Summit)
- We expect CUDA to be slightly faster

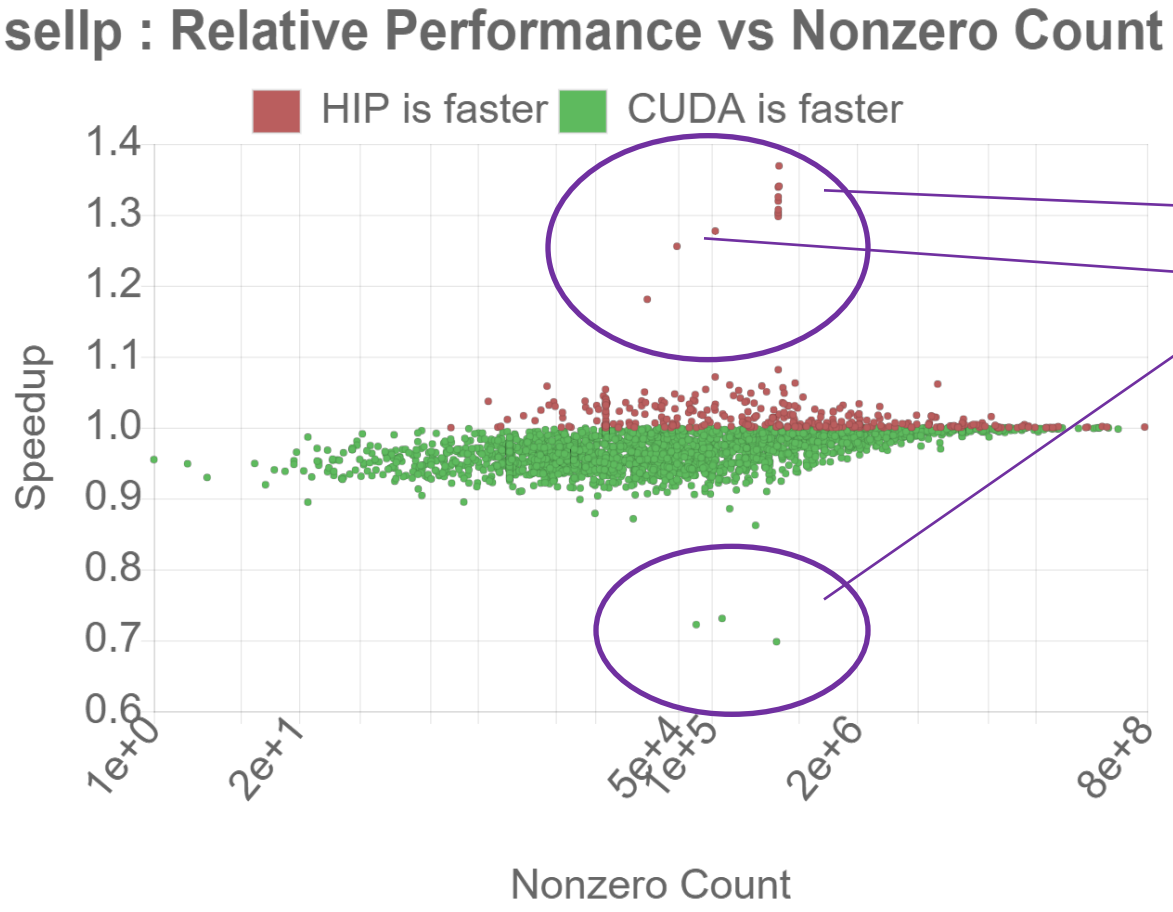
HIP faster than CUDA on NVIDIA GPU?

outliers? machine noise?

Outlier stats on 100 runs a 20 reps:



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

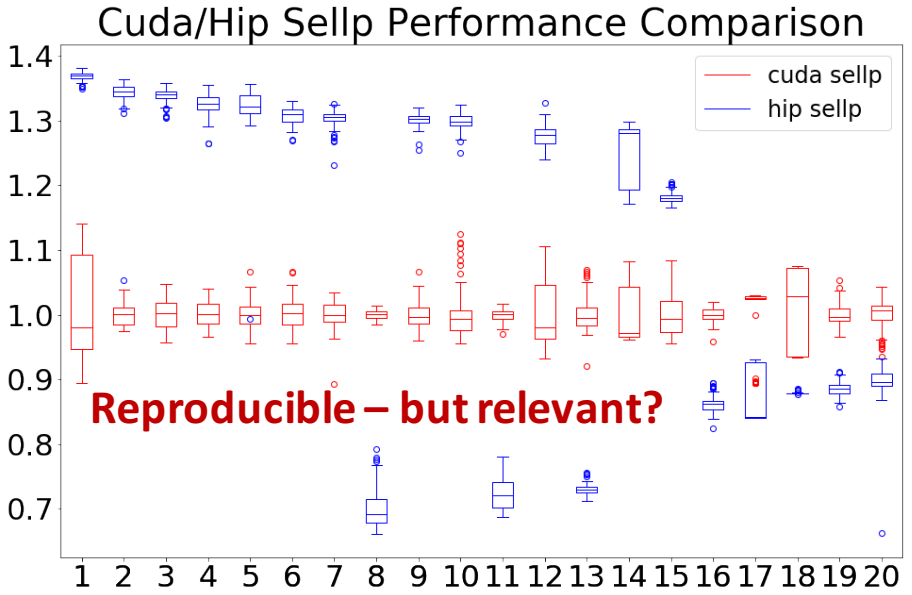


- Native CUDA vs. HIP compiled for NVIDIA GPUs
- Same kernel
- All tests on NVIDIA V100 (Summit)
- We expect CUDA to be slightly faster

HIP faster than CUDA on NVIDIA GPU?

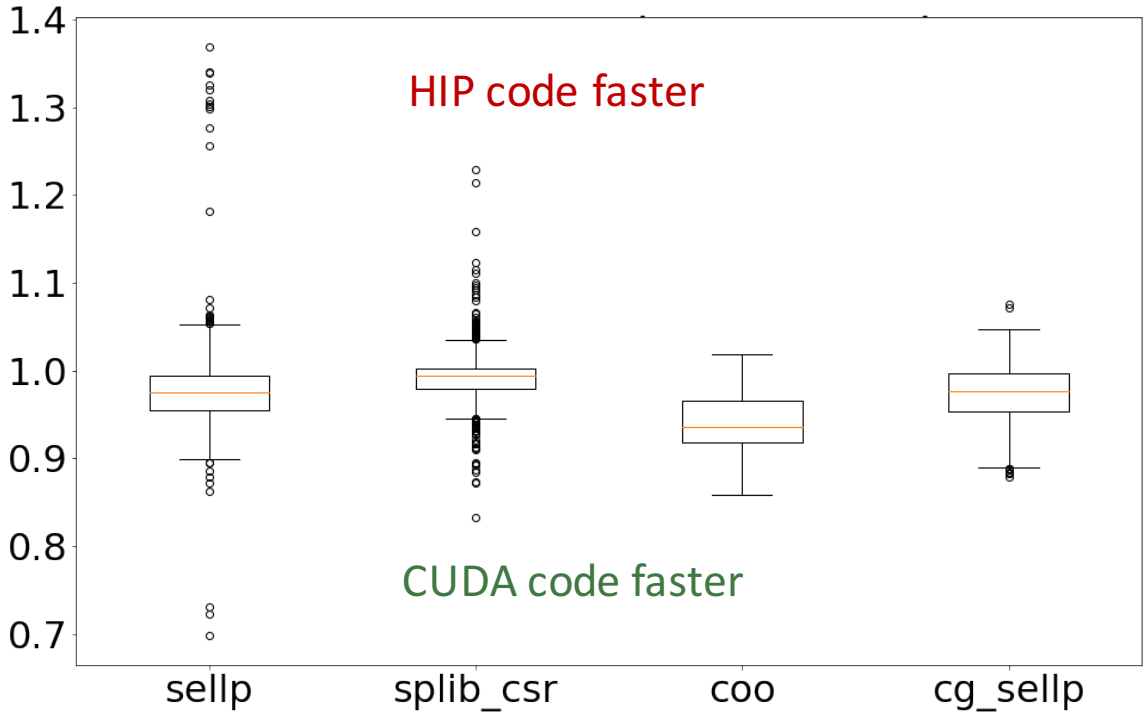
outliers? machine noise?

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



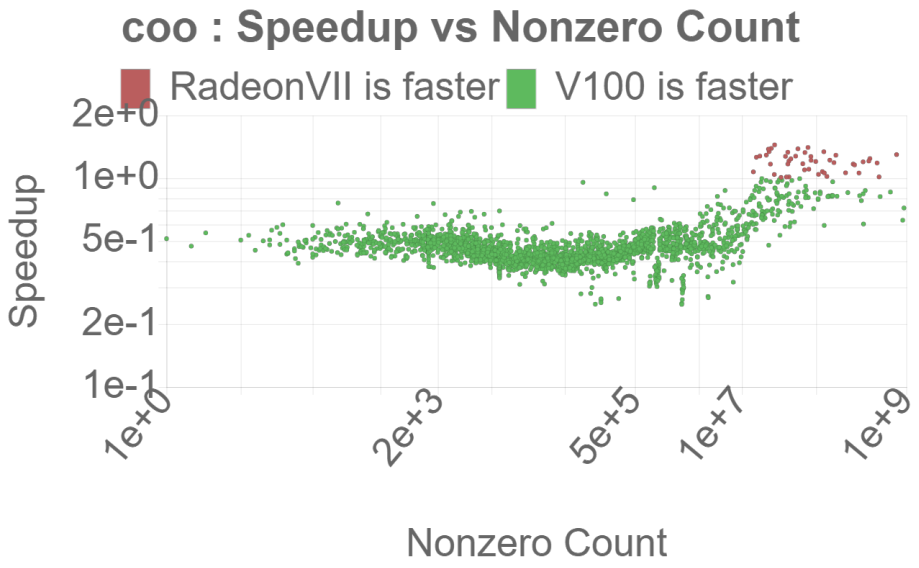
Slight advantages on the CUDA side, but usually <5%.

Summary: Porting Linear Algebra Libraries to the AMD Ecosystem

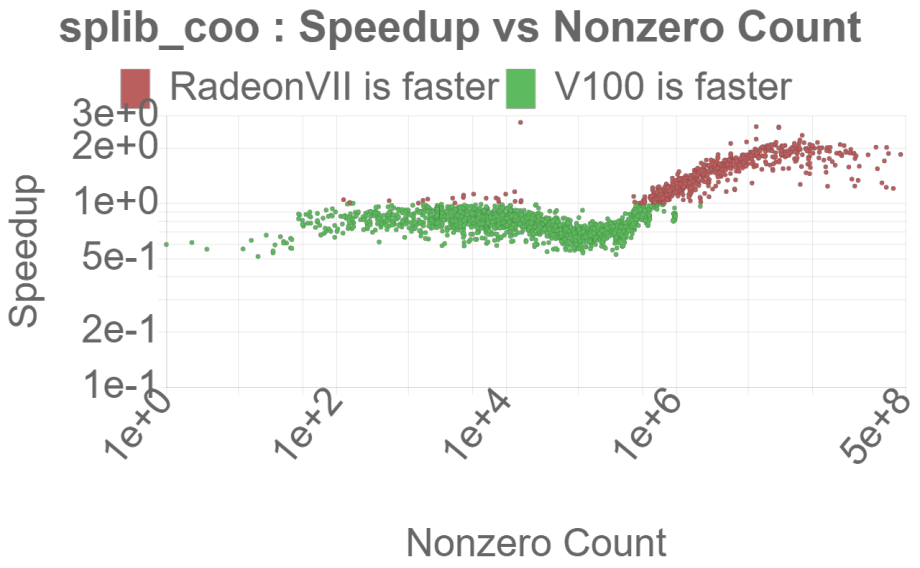
- AMD with its **HIP software ecosystem** becomes a relevant alternative to NVIDIA CUDA;
- **Significant similarities** in the languages (HIP/CUDA) allow for shared kernel implementations;
- **HIP allows to compile for NVIDIA GPUs**
  - in most cases with moderate performance loss compared to native CUDA code;
- **AMD** GPUs and **NVIDIA** GPUs **comparable in sparse linear algebra performance**;
- Adding CMake-support for both AMD and NVIDIA extremely challenging;

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.

How do GPU architectures compare in terms of SpMV performance?



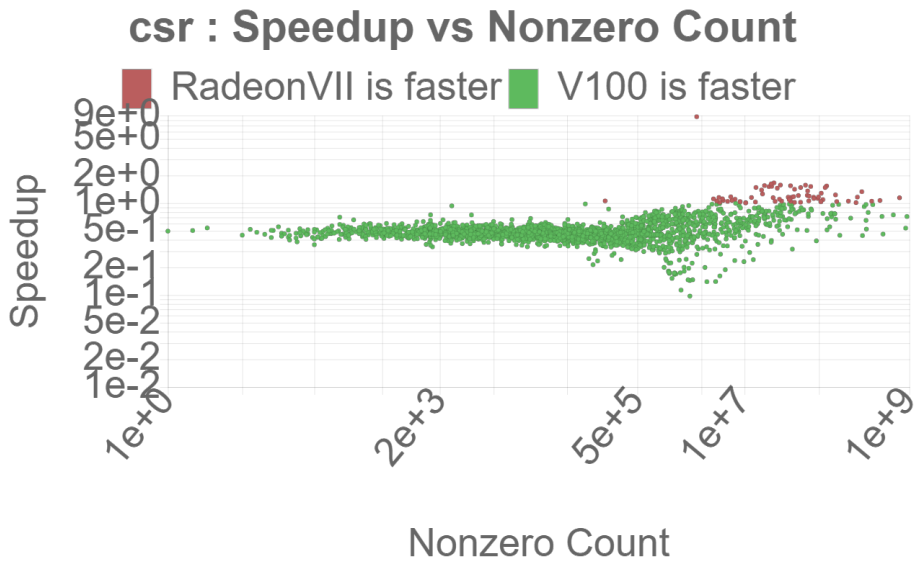
Ginkgo COO SpMV



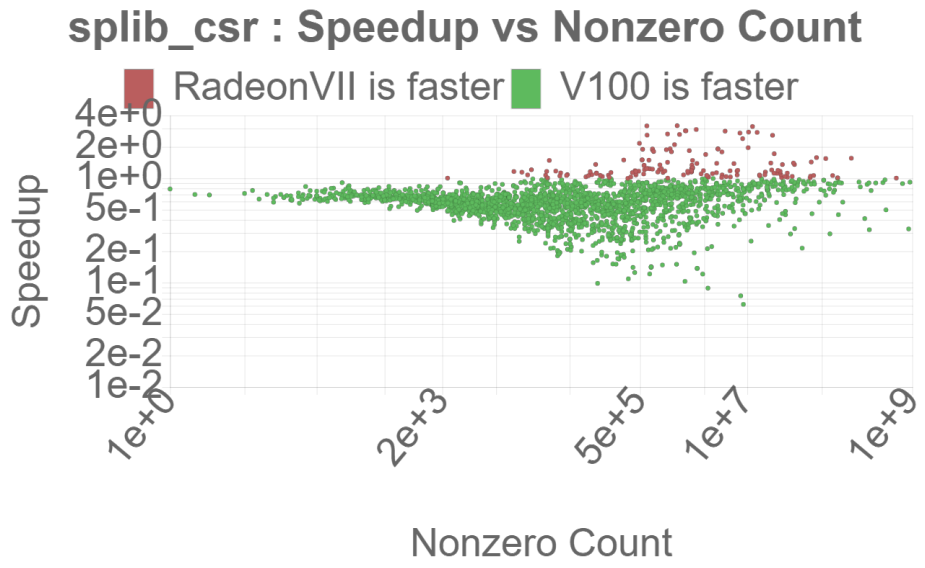
Vendor library COO SpMV

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

How do GPU architectures compare in terms of SpMV performance?



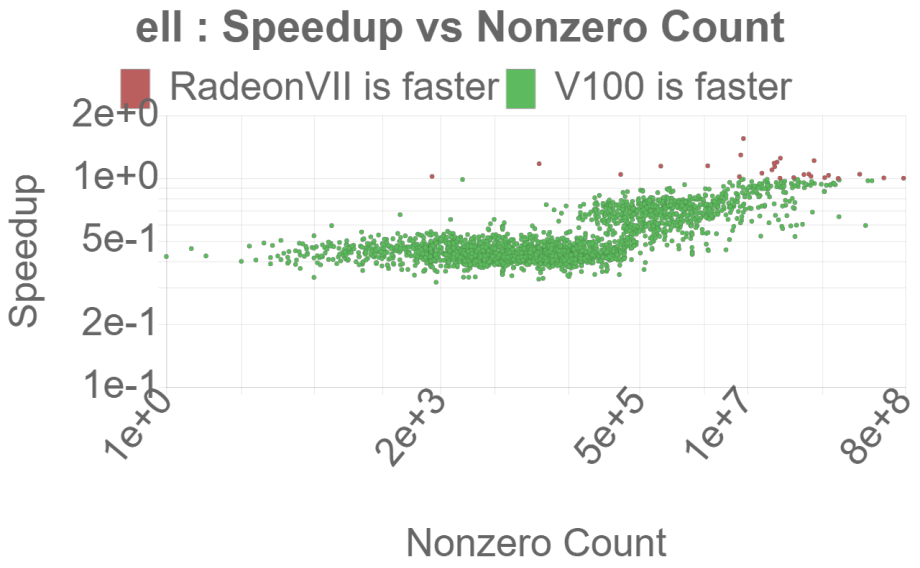
Ginkgo CSR SpMV



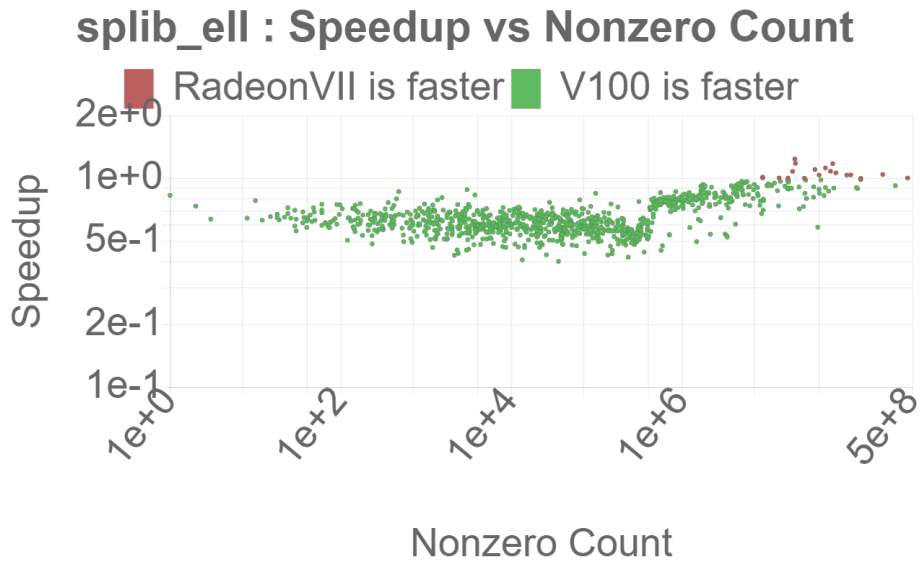
Vendor library CSR SpMV

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/