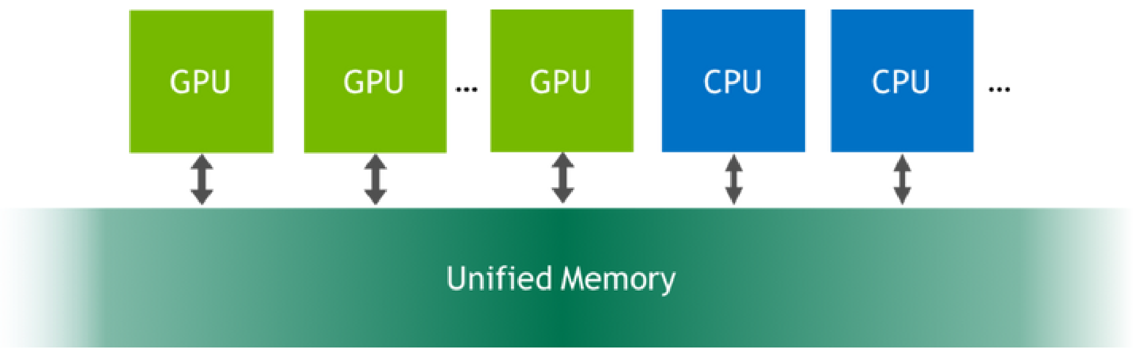
MonteCarlo Benchmarking

Andrea Paparella - 10701904
Andrea Piras - 10725972

Professor: Marco Domenico Santambrogio
Advisor: Ian Di Dio Lavore

The Problem

Investigating the performance impact of NVIDIA's Unified Memory paradigm.



Traditional

Allocate memory on CPU and GPU and manually copy data between them.

Unified Memory

Single memory space that is accessible by both the CPU and GPU.

```
checkCudaErrors(cudaMalloc(&plan->d_CallValue, sizeof(__TOptionValue)*(plan->optionCount)));
checkCudaErrors(cudaMallocHost(&plan->h_OptionData, sizeof(__TOptionData)*(plan->optionCount)));
//Allocate internal device memory
checkCudaErrors(cudaMallocHost(&plan->h_CallValue, sizeof(__TOptionValue)*(plan->optionCount)));
```

Original Implementation

From the Tartan benchmark suite written in C++

- 2 Modes: Threaded & Streamed
- 2 Input Scalings: Weak & Strong
- Allocation: cudaMalloc & cudaMallocHost
- Data Migration: cudaMemcpyAsync

Li, Ang, et al. "Tartan: evaluating modern GPU interconnect via a multi-GPU benchmark suite." 2018

```
checkCudaErrors(cudaMallocManaged(&plan->um_OptionData, sizeof(__TOptionData) * (plan->optionCount
checkCudaErrors(cudaMallocManaged(&plan->um_CallValue, sizeof(__TOptionValue) * (plan->optionCount)
// Allocate internal device memory
// Allocate states for pseudo random number generators
```

Unified Memory Implementation

From the original implementation

- From cudaMalloc & cudaMallocHost to cudaMallocManaged
- Removal of cudaMemcpyAsync
- Optimizations through different versions

The Methodology

Performance Metrics

Initialization & Execution Time
Page Faults

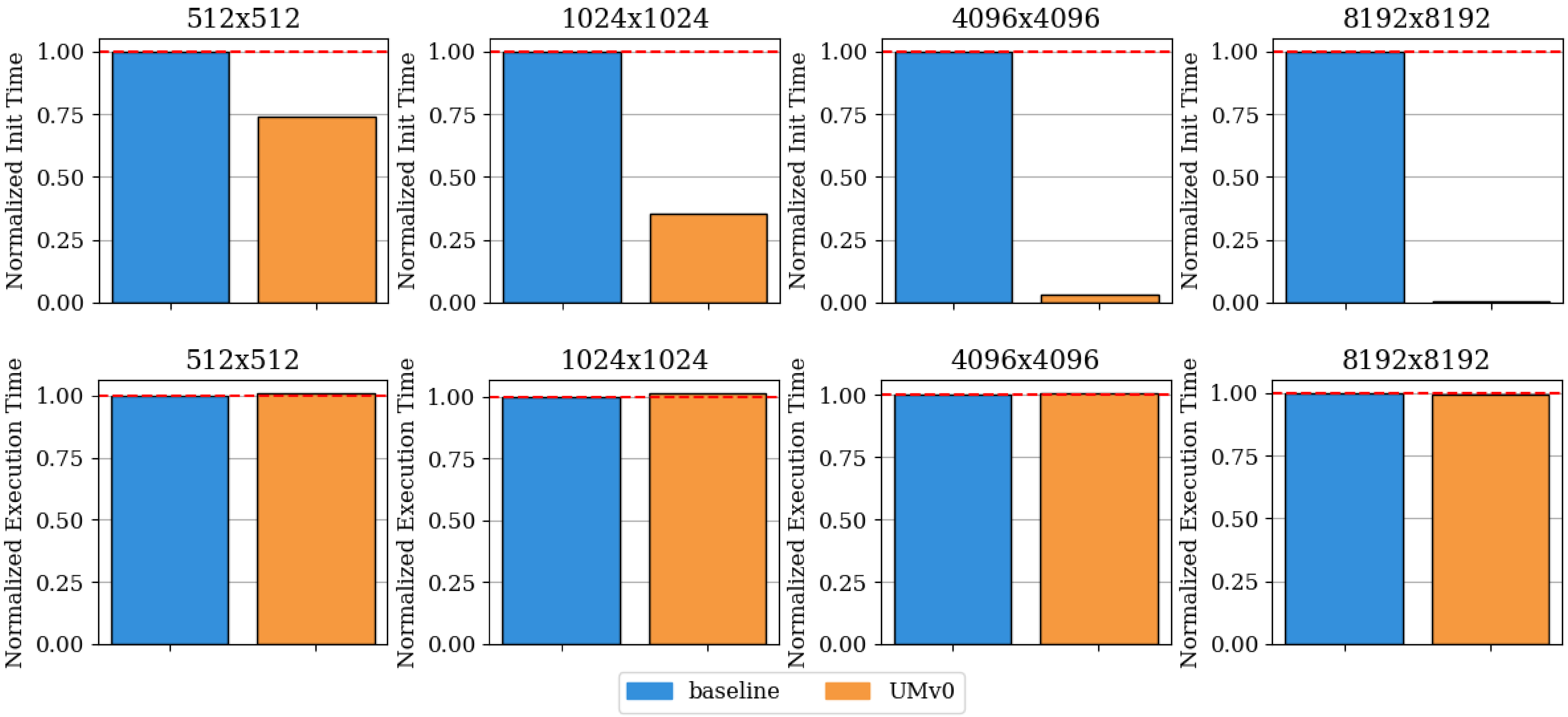
Data Collection

Nvprof & NVIDIA Nsight

Testing

Different Input Sizes
Different GPUs Number
Average Time of 5-10 runs

Results - Unified Memory



Streamed Strong Scaling

Optimizing through prefetching

Version 1

cudaMemPrefetchAsync

Proactively allocate memory on a specific location before it is actually accessed

Intuition

Migrate data to device requesting it to avoid page faults caused by attempts to retrieve data that is not present on memory

Optimizing through advising

Version 2

cudaMemAdvise

Advise the Unified Memory subsystem about the memory usage pattern

- cudaMemAdviseSetPreferredLocation
- cudaMemAdviseSetReadMostly
- cudaMemAdviseSetAccessedBy

Intuition

Reduce page faults and memory migrations through automatic optimization guided by the hinted policies

Combining optimizations together

Version 3

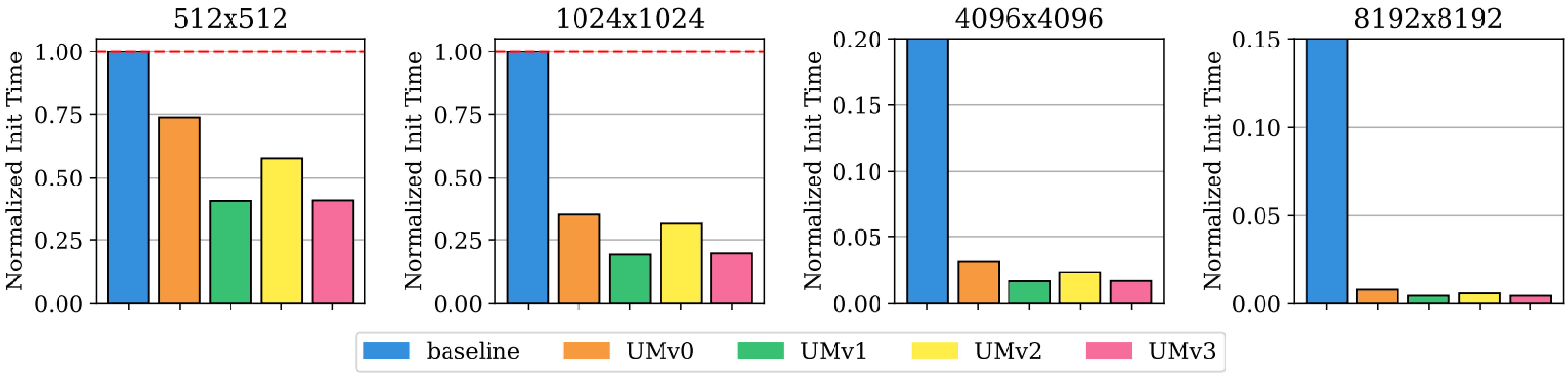
Objective

Leverage the optimal features of both versions

Intuition

Prefetching eliminates page faults but can be expensive: replace prefetches with advices to mitigate the occurrence of page faults and enhance performance

Results - Initialization Time



Streamed Strong Scaling

230x

Best SpeedUp

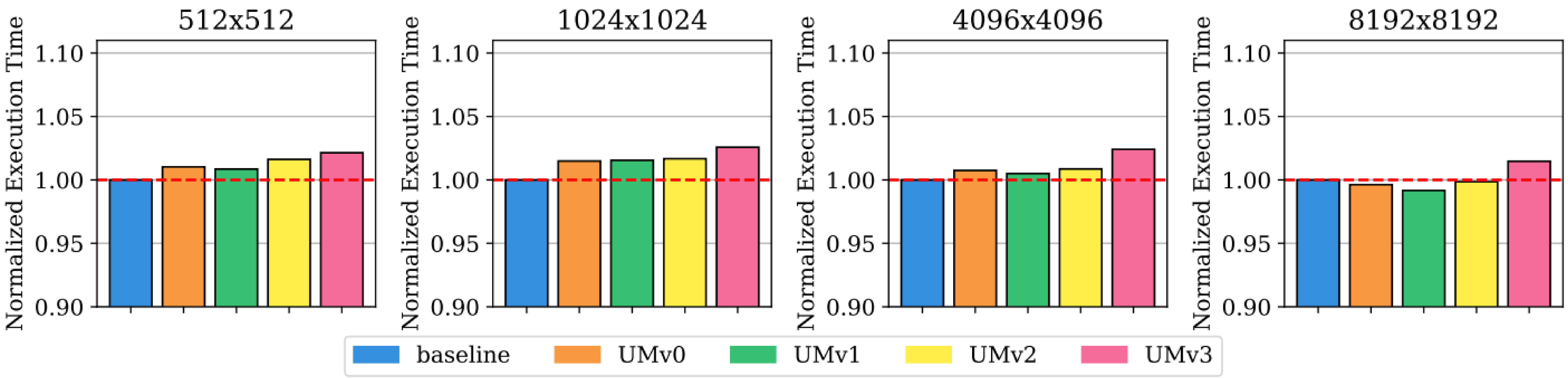
100%

Page Fault Reduction

% 0.03

Overall Occupancy

Results - Execution Time



Streamed Strong Scaling

<2.5%
Execution Time Variation

100%
Page Fault Reduction

% 99.97
Overall Occupancy

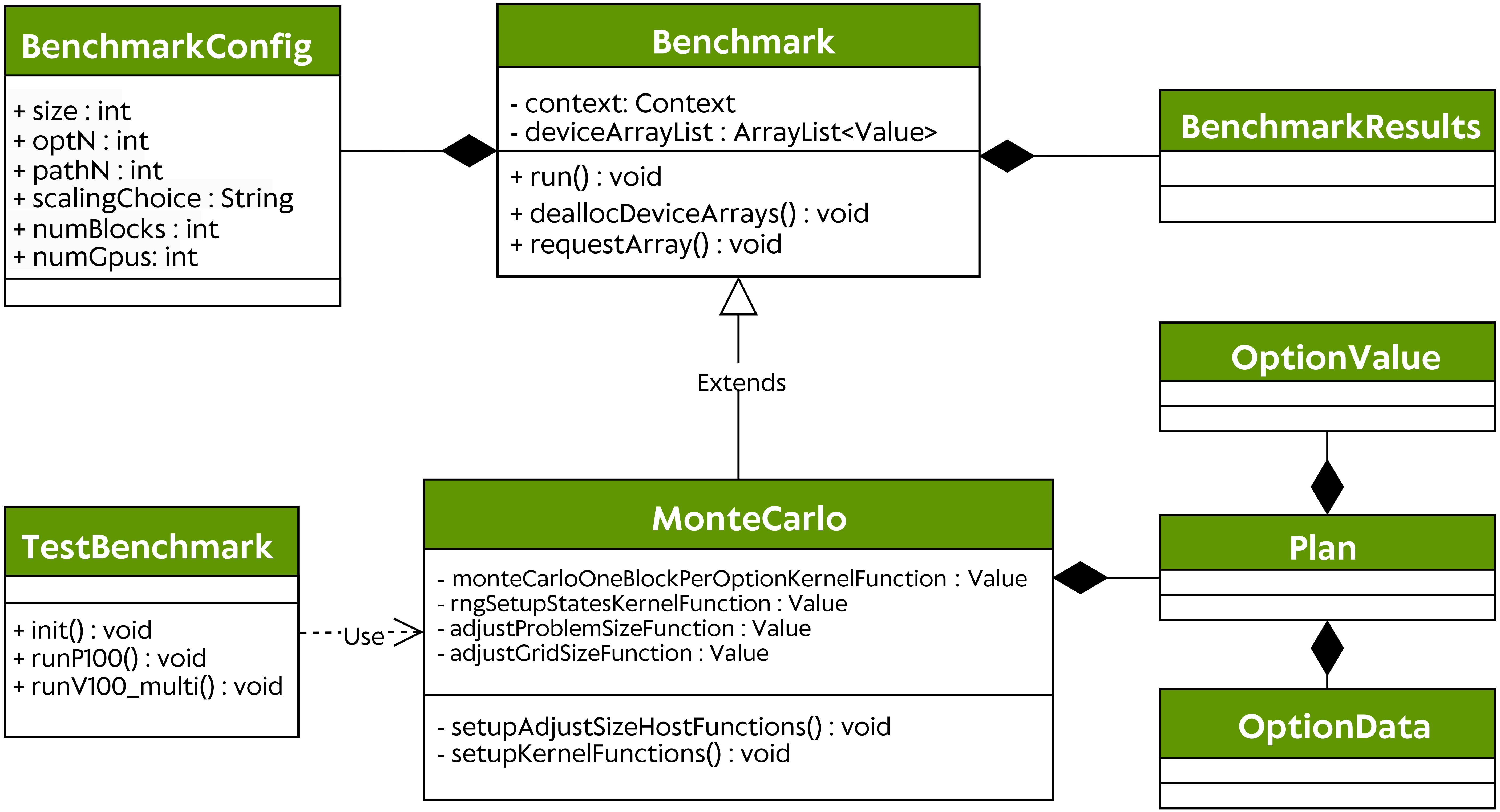
```
monteCarloOneBlockPerOptionKernelFunction = cu.invokeMember("bindkernel", path_to_binded_kernels,
    "cxx MonteCarloOneBlockPerOption( " +
        "d: out pointer uint32, " +
        "v: out pointer uint32, " +
```

GrCUDA Implementation

From the C++ implementation to Java

- Abstraction of CUDA code to other languages
- Unified Memory automatically managed by the underlying system
- Memory regions stored into DeviceArrays

Our Benchmark Design



https://github.com/necst/grcuda/tree/master/projects/resources/java/grcuda-benchmark

Best Practice

Binding kernels and functions from files

Binding from files

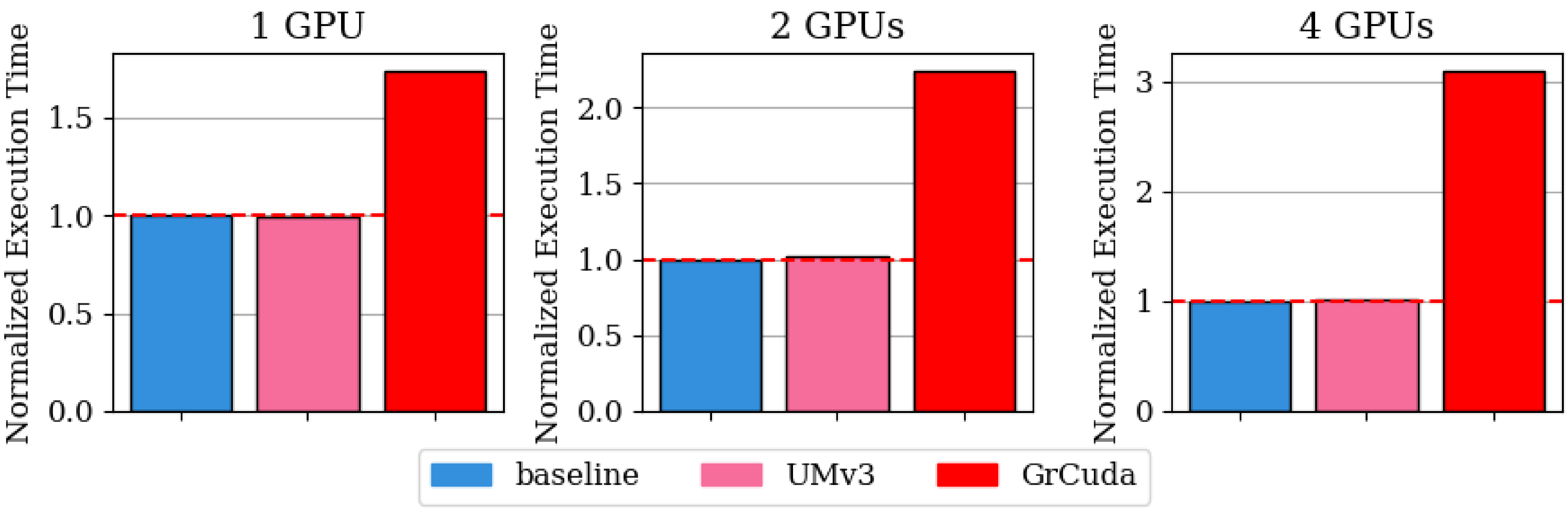
CUDA kernels are written separately, compiled into a separate file, and bound using a binding function.

Intuition

Data structures and functions of external libraries are not supported by the standard building from Java strings.

Binding from files is used to overcome library import constraints.

Results - Comparing GPUs Number



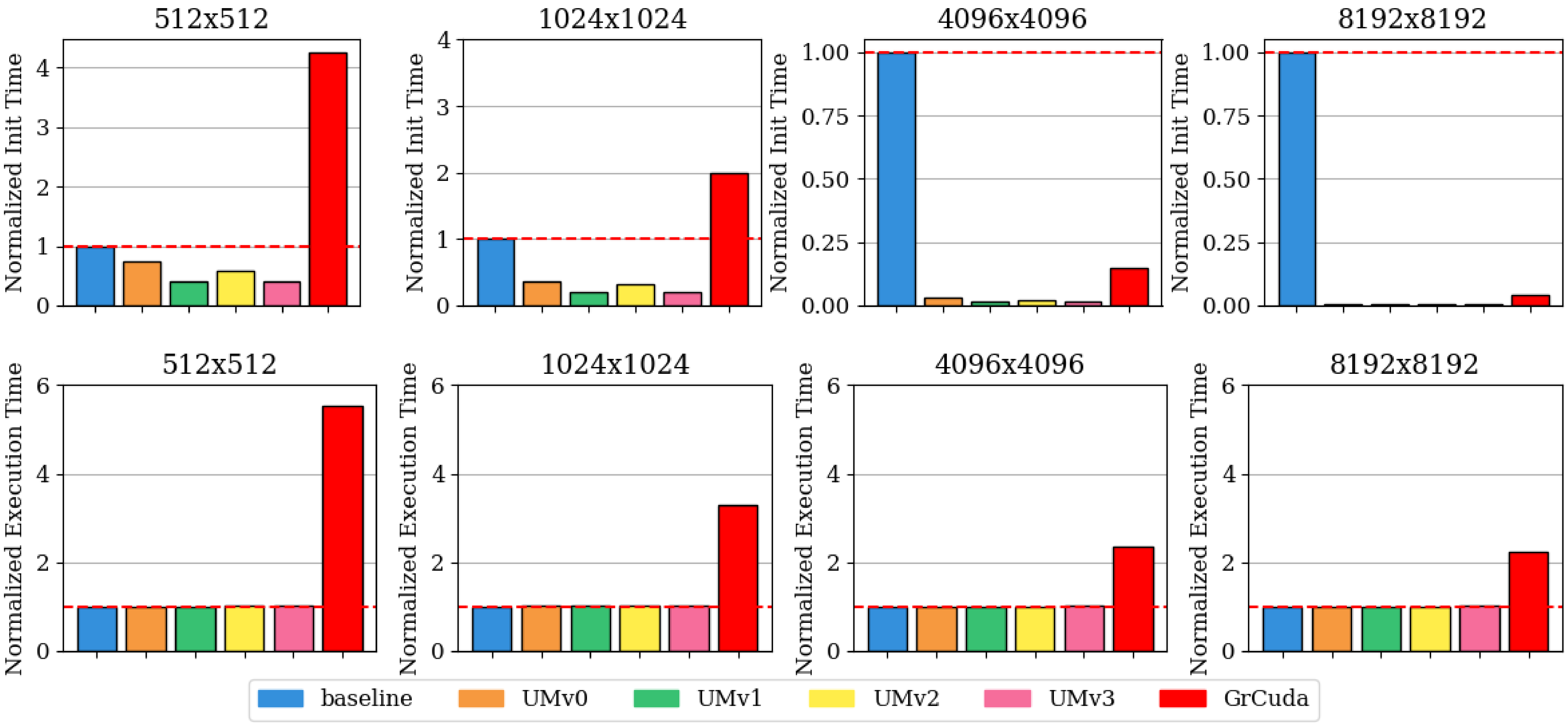
Streamed Strong Scaling

1.5x - 3x

Increased GrCuda Execution Time

As GPU number increases GrCuda performance gets worse

Results - Comparing Sizes



Streamed Strong Scaling

Performance Improvement as size increases

Our Conclusions

#1

Unified Memory optimizations reduced Initialization Time

#2

Unified Memory optimizations introduction didn't alter execution time.

#3

Benchmarks execution patterns matter when using Unified Memory

#4

GrCuda had worse performances but improved with size

Thank you for your attention!

Andrea Paparella
andrea1.paparella@polimi.mail.it

Andrea Piras
andrea6.piras@polimi.mail.it