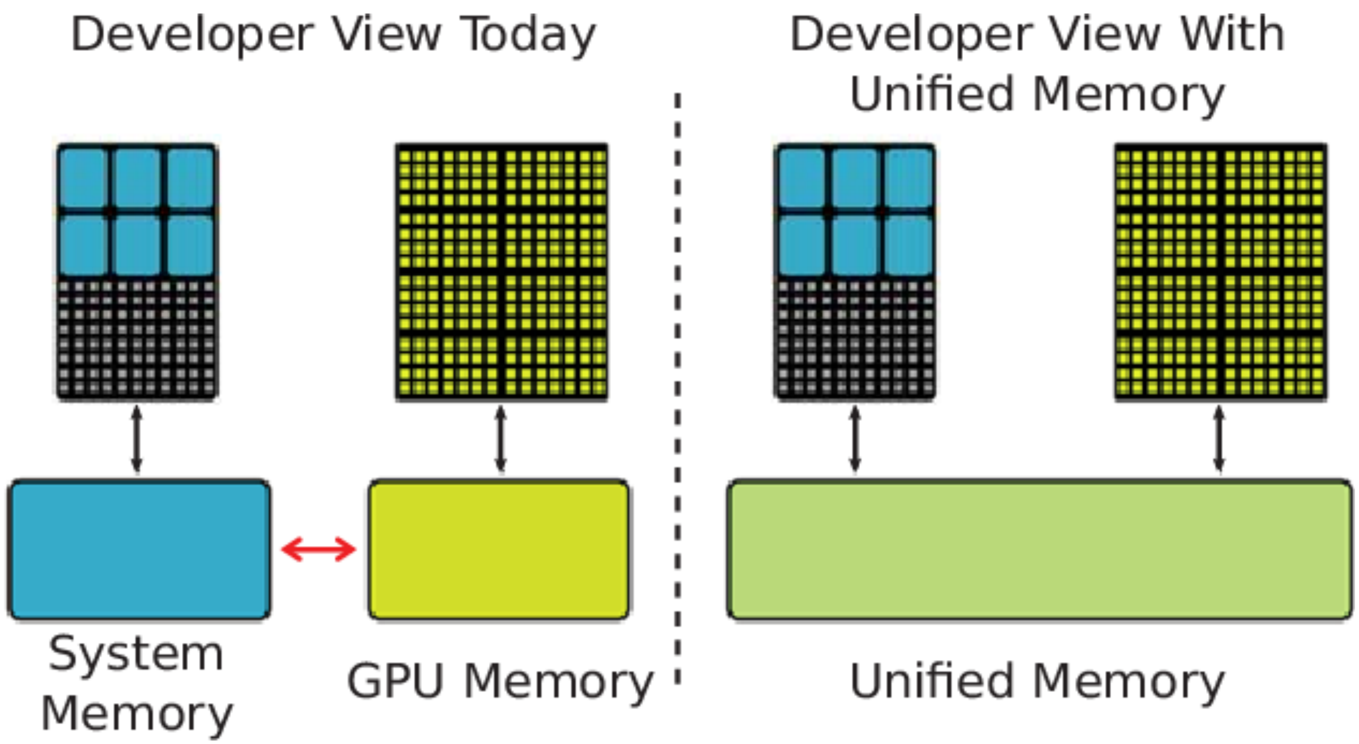
Machine Learning Guided Optimal Use of GPU Unified Memory

Hailu Xu, **Murali Emani***, Pei-Hung Lin**,
Liting Hu, Chunhua Liao**

Florida International University
Argonne National Laboratory*
Lawrence Livermore National Laboratory**

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and supported by LLNL-LDRD 18-ERD-006. LLNL-PRES-796963. This research was funded in part by the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357

Background – Unified Memory



Benefits of Unified Memory:

- combines the advantages of explicit copies and zero-copy access
- eliminates manual management of data migration across host and device

Background – Unified Memory

Deep Copy

Explicit Memory Management

```
char **data;
// allocate and initialize data on the CPU

char **d_data;
char **h_data = (char**)malloc(N*sizeof(char*));
for (int i = 0; i < N; i++) {
  cudaMalloc(&h_data[i], N);
  cudaMemcpy(h_data[i], data[i], N, ...);
}
cudaMalloc(&d_data, N*sizeof(char*));
cudaMemcpy(d_data, h_data, N*sizeof(char*), ...);

gpu_func<<<...>>>(d_data, N);
```

GPU code w/ Unified Memory

```
char **data;
// allocate and initialize data on the CPU

gpu_func<<<...>>>(data, N);
```

*http://on-demand.gputechconf.com/gtc/2018/presentation/s8430-everything-you-need-to-know-about-unified-memory.pdf

Background – Unified Memory

NVIDIA provides the **cudaMemAdvise()** API to advise the UM driver

```
cudaMemAdvise(const void *,
              size_t,
              enum cudaMemoryAdvise,
              int)
```

Background – Unified Memory

NVIDIA provides the **cudaMemAdvise()** API to advise the UM driver

```
cudaMemAdvise(const void *,        → data object
              size_t,
              enum cudaMemoryAdvise,  → Choice of memory advice
              int)                 → device
```

Background – Unified Memory

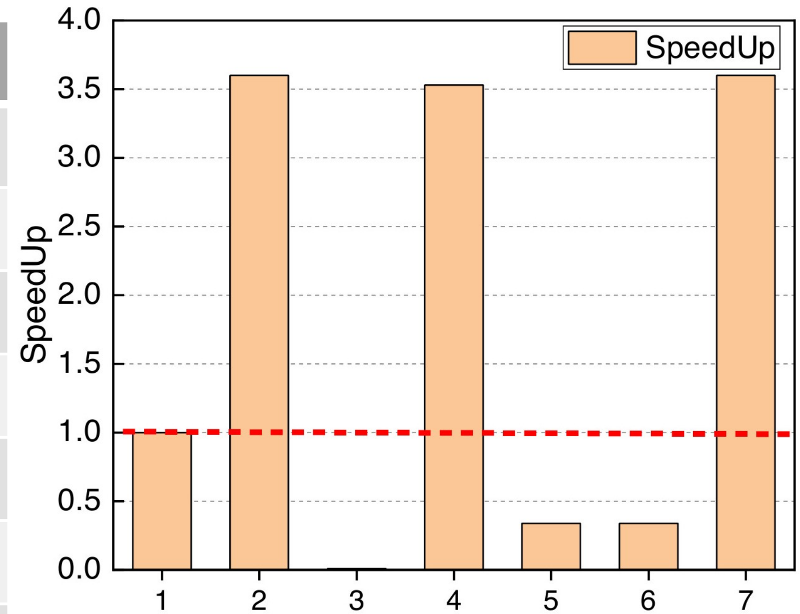
Different choices for Unified Memory:

- ***Default:*** default on-demand page migration to accessing processor, using the first-touch policy
- ***cudaMemAdviseSetReadMostly:*** Data will mostly be read and only occasionally be written to
- ***cudaMemAdviseSetPreferredLocation:*** Set the preferred location for the data as the specified device
- ***cudaMemAdviseSetAccessedBy:*** Data will be accessed by the specified device, so prevent page faults as much as possible

Impact of different choices

Table1: Code variants in the `gaussian` benchmark

Var	Description
1	baseline using discrete memory for all objects
2	modified to use unified memory for all objects
3	set array ***a*** with the ***ReadMostly*** advice
4	set array ***a*** with the ***PreferredLocation*** advice on GPU
5	set array ***a*** with the ***AccessedBy*** advice on GPU
6	set array ***a*** with the ***PreferredLocation*** advice on CPU
7	set array ***a*** with the ***AccessedBy*** advice on CPU



rent code

Different choices of advice lead to **3.5 times speed up** or **200x degradation**.

Problem

- Extremely challenging for programmers to decide when and how to efficiently use UM for various kinds of applications.
- For a given memory object, there is a wide range of choices

Problem

- Extremely challenging for programmers to decide when and how to efficiently use UM for various kinds of applications.

- For a given memory object, there is a wide range of choices

Whether and how to use unified memory?

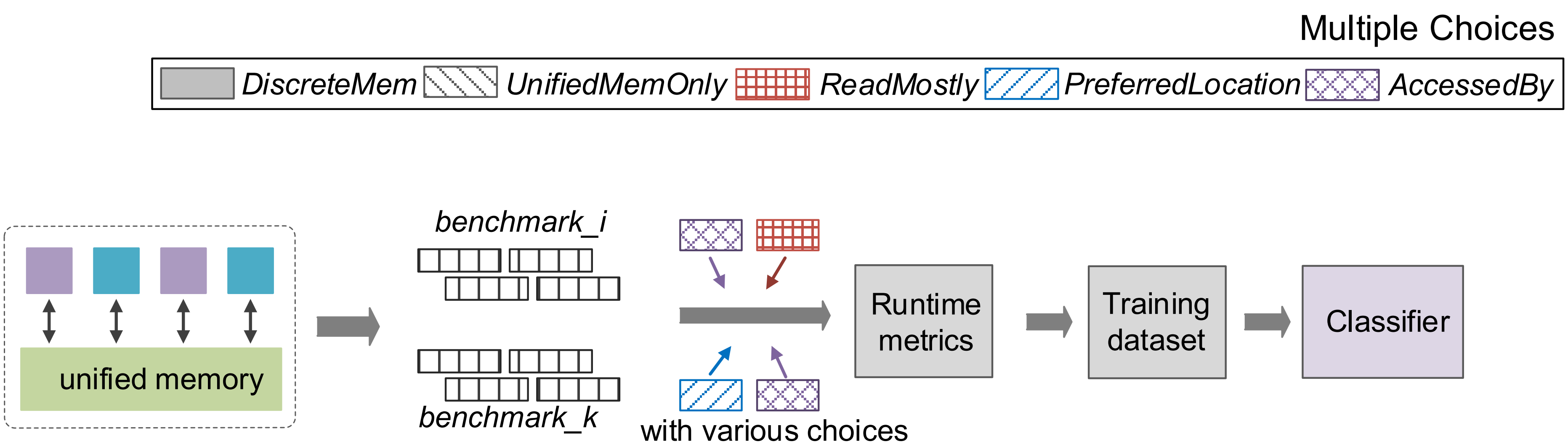
Proposed Approach

- Use machine learning-based model to guide the memory advice choice

- Offline training and online inference phases

Offline Training

- Benchmarks with different advice; runtime metrics collection; format to training dataset; build the classifier



Feature Engineering

- `Nvidia Nsight Compute` command line profiler to fetch detailed runtime performance metrics of the benchmarks

- The default profiling phase contains 8 sections such as
  - Compute Workload Analysis,
  - Memory Workload Analysis,
  - Scheduler Statistics,
  - Warp State Statistics,
  - Instruction Statistics,
  - Launch Statistics,
  - Occupancy

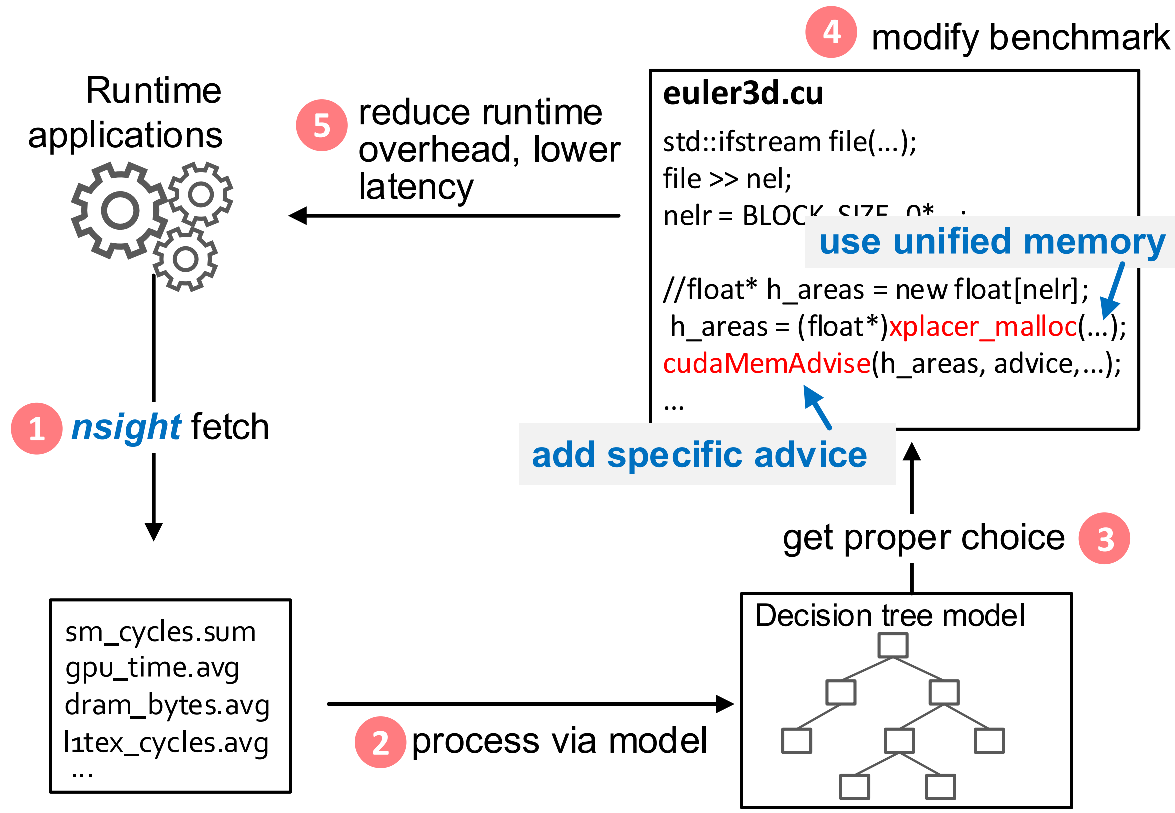
- Select important features using correlation and information gain metrics

Feature Engineering

No.	Feature Name
1	Elapsed Cycles
2	Duration
3	SM Active Cycles
4	Memory Throughput
5	Max Bandwidth
6	Avg. Execute Instructions Per Scheduler
7	Grid Size
8	Number of Threads
9	Achieved Active Warps Per SM

Table 2: List of selected features in the model.

Online Inference

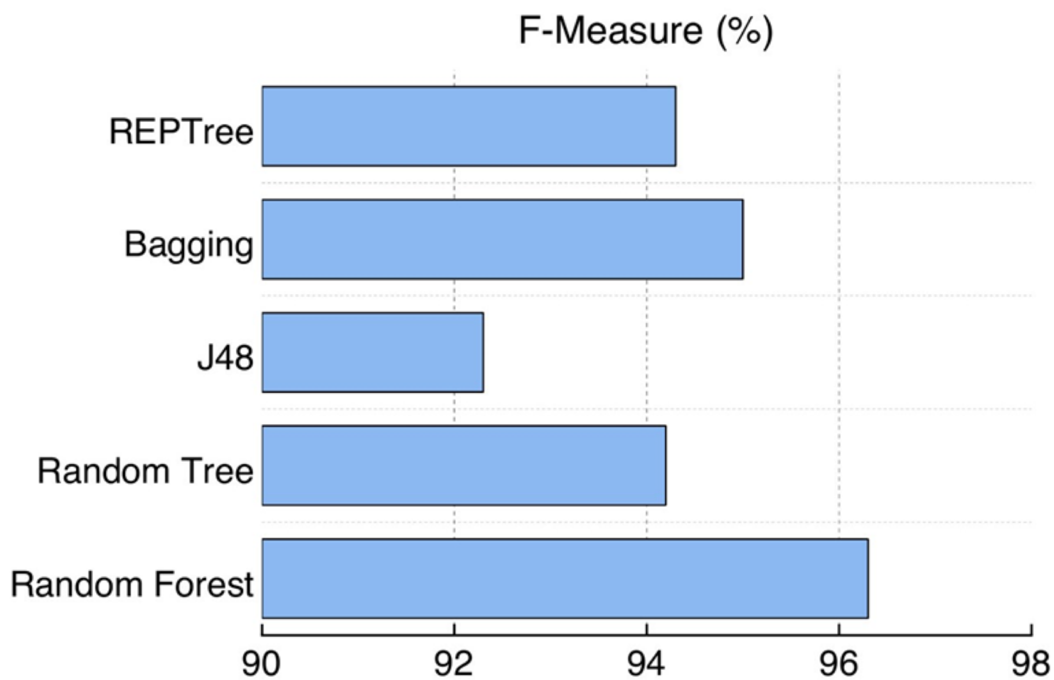


Evaluation-Testbeds and benchmarks

- Multiple benchmarks from Rodinia on the Lassen supercomputer at Livermore Computing.
- Each compute node: two IBM Power9 CPUs and four Tesla V100 GPUs
- 2,753 instances for training data

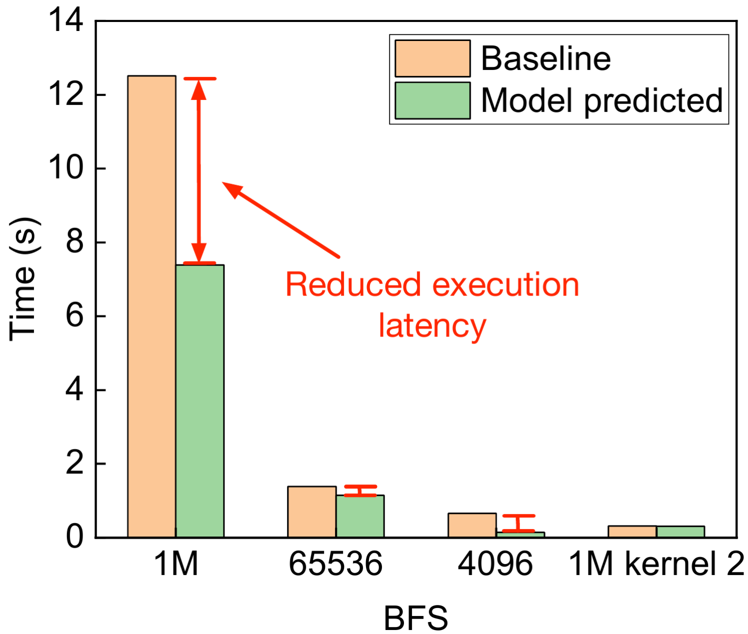
Benchmarks	Kernels	Arrays	Variants	Input dataset
CFD	4	3	(2x6x6x6)	3
BFS	2	6	(2x6x6)	3
Gaussian	2	3	(2x6x6x6)	67
Hotspot	1	2	(2x6x6)	8

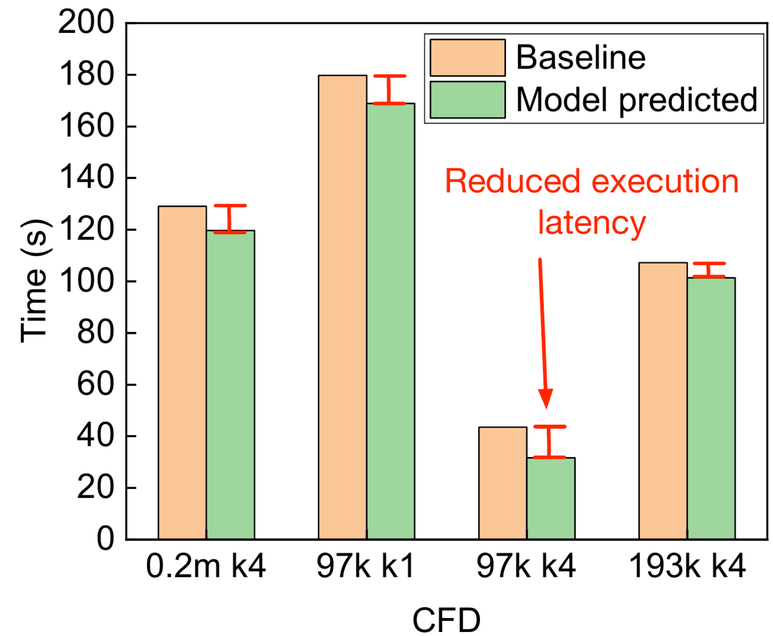
Results - Accuracy



- Random Forest classifier achieves the best performance with F-measure up to **96.3%**
- Effective and optimal predictions for the benchmarks

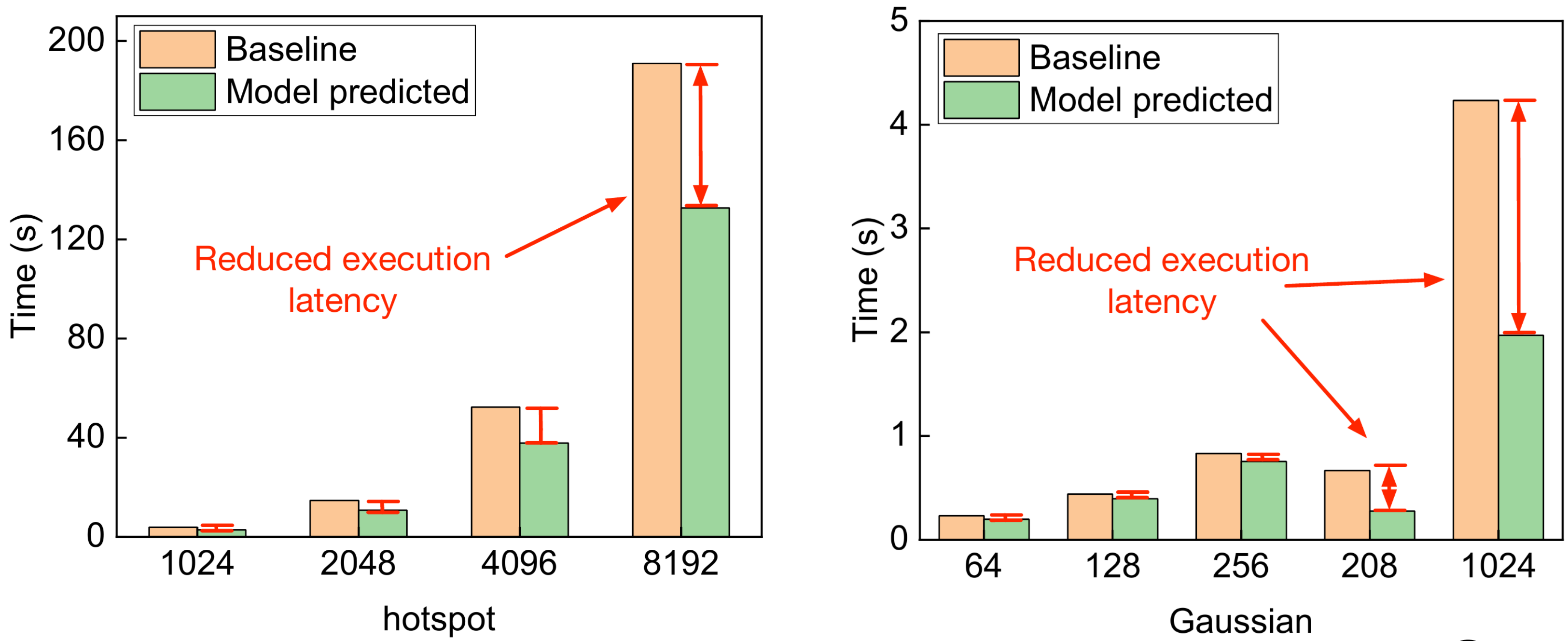
Results – Reduced latency





BFS has near **<u>40%</u>** deduction at most; CFD has average **<u>8%</u>** deduction in execution latency.

Results – Reduced latency



Hotspot benchmark has around **35%** deduction; Gaussian has at most **60%** deduction in execution latency.

Conclusion & Future work

- We study the hybrid use of both discrete and unified memory APIs on GPUs, with additional consideration for selecting different memory advice choices.
- A machine learning-based approach is proposed to guide optimal use of GPU unified memory
- Design code transformation to enable runtime adaptation of CUDA programs leveraging online inference decisions

Future work:

- extend to evaluate the advice choices at a finer granularity considering calling context.
- employ runtime code generation and/or adaptation techniques to automatically generate codes using suggested optimal memory choices
- evaluate the overhead for collecting training data and investigate how to reduce the overhead

Thank you!

Problem

Whether and how to use unified memory?

Whether? ⇨ **Decide to use unified memory or not**

How? ⇨ **Decide which advice should be used**