Diogenes: A tool for exposing Hidden GPU Performance Opportunities

Benjamin Welton and Barton Miller

2019 Performance Tools Workshop

July 29th, Tahoe, CA.

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

Paradyn™

Overview of Diogenes

Automatically detect performance issues with CPU-GPU interactions (synchronizations, memory transfers)

- Unnecessary interactions
- Misplaced interactions
- **We do not do** GPU kernel profiling, general CPU profiling, etc

Output is a list of unnecessary or misplaced interactions

- Including an estimate of potential benefit (in terms of application runtime) of fixing these issues.

Features of Diogenes

Binary instrumentation of the application and CUDA user space driver for data collection

- Collect information not available from other methods
  - Use (or non-use) of data from the GPU by the CPU
  - Identify hidden interactions
    - Conditional interactions (ex. a synchronous cuMemcpyAsync call).
    - Detect and measure interactions on the private API.
  - Directly measure synchronization time
  - Look at the contents of memory transfers

Analysis method to show only problematic interactions.

Current Status of Diogenes

Prototype is working on Power 8/9 architectures

- Including on the current GPU driver versions used on LLNL/ORNL machines

What Works:

- Identifying unnecessary transfers
  - non-unified memory transfers only
- Identifying unnecessary/misplaced synchronizations that occur at a single point (type 1 & 2 below)

Type 1: No use of GPU Computed Data

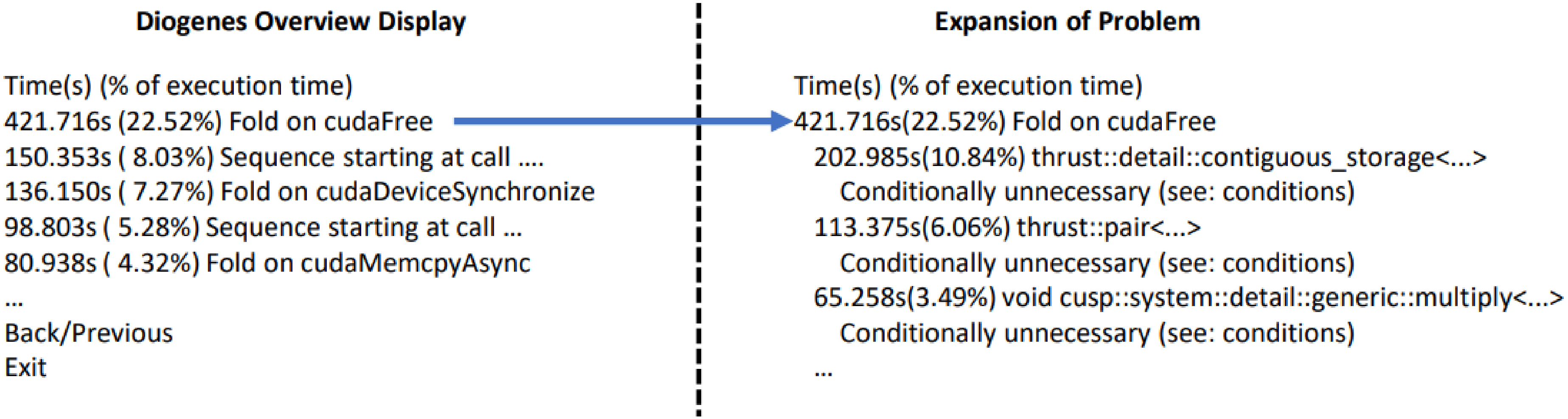
```
Synchronization();
for(…) {
   // Work with no GPU dependencies
}
Synchronization();
```

Type 2: Misplaced Synchronization

```
Synchronization();
for(…) {
   // Work with no GPU dependencies
}
result = GPUData[0] + …
```

Current Status of Diogenes

Ncurses interface for exploring Diogenes analysis



Diogenes Overview Display

```
Time(s) (% of execution time)
421.716s (22.52%) Fold on cudaFree
150.353s ( 8.03%) Sequence starting at call ....
136.150s ( 7.27%) Fold on cudaDeviceSynchronize
98.803s ( 5.28%) Sequence starting at call ...
80.938s ( 4.32%) Fold on cudaMemcpyAsync
...
Back/Previous
Exit
```

Expansion of Problem

```
Time(s) (% of execution time)
421.716s(22.52%) Fold on cudaFree
  202.985s(10.84%) thrust::detail::contiguous_storage<...>
     Conditionally unnecessary (see: conditions)
  113.375s(6.06%) thrust::pair<...>
     Conditionally unnecessary (see: conditions)
  65.258s(3.49%) void cusp::system::detail::generic::multiply<...>
     Conditionally unnecessary (see: conditions)
  ...
```

Diogenes Predictive Accuracy Overview

App Name	App Type	Diogenes Estimated Benefit (Top N, % of Exec)	Actual Benefit by Manual Fix (Top N,% of Exec)
cumf_als	Matrix Factorization	10.0%	8.3%
AMG	Algebraic Solver	6.8%	5.8%
Rodinia	Gaussian Benchmark	2.2%	2.1%
cuIBM	CFD	10.8%	17.6%

- Estimates for the top 1-3 most prominent problems in each application.
  - Tried to be as careful as possible to alter only the problematic operation

Diogenes Collection and Analysis Techniques

1. Identify and time interactions
   - Including hidden synchronizations and memory transfers

   Binary Instrumentation of libcuda to identify and time calls performing synchronizations and/or data transfers

2. Determine the necessity of the interaction
   - If the interaction is necessary for correctness, is it placed in an efficient location?

   Synchronizations: A combination of memory tracing, CPU profiling, and program slicing

   Duplicate Data Transfers: Content based data deduplication approach.

3. Provide an estimate of the fixing the bad interactions

   Diogenes uses a new Feed Forward Instrumentation workflow for data collection combined with a new model to produce the estimate

Diogenes – Workflow

Diogenes uses a newly developed technique called feed forward instrumentation:

- The results of previous instrumentation guides the insertion of new instrumentation.

Diogenes performs each step automatically (via a launcher)

Diogenes – Workflow

Diogenes uses a newly developed technique called feed forward instrumentation:

- The results of previous instrumentation guides the insertion of new instrumentation.

Step 1
Measure execution time of the application (without instrumentation)

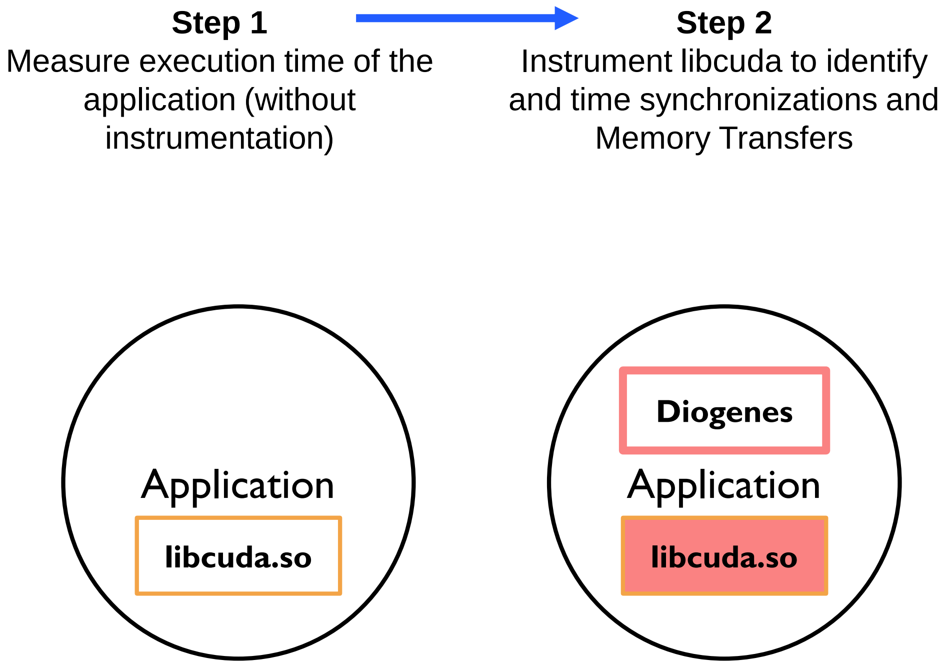
Application
libcuda.so

Diogenes performs each step automatically (via a launcher)

Diogenes – Workflow

Diogenes uses a newly developed technique called feed forward instrumentation:

- The results of previous instrumentation guides the insertion of new instrumentation.



Step 1
Measure execution time of the application (without instrumentation)

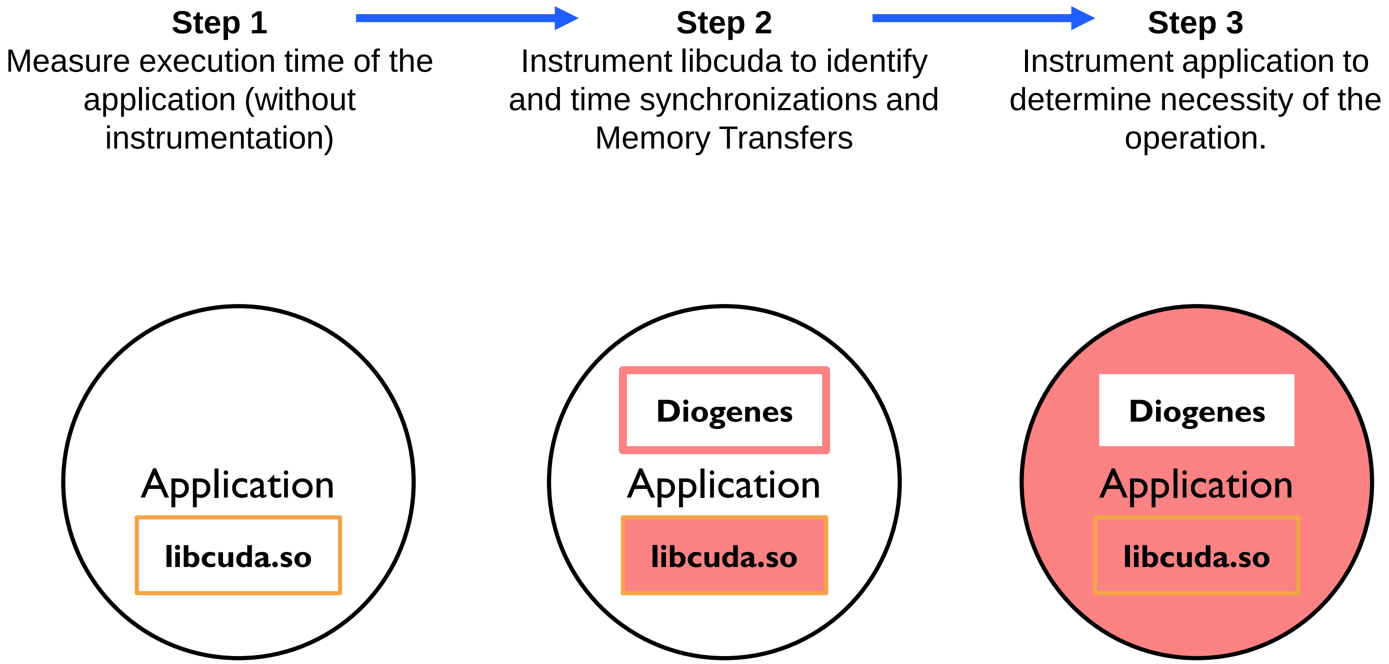
Step 2
Instrument libcuda to identify and time synchronizations and Memory Transfers

Diogenes performs each step automatically (via a launcher)

Diogenes – Workflow

Diogenes uses a newly developed technique called feed forward instrumentation:

- The results of previous instrumentation guides the insertion of new instrumentation.



Step 1
Measure execution time of the application (without instrumentation)

Step 2
Instrument libcuda to identify and time synchronizations and Memory Transfers

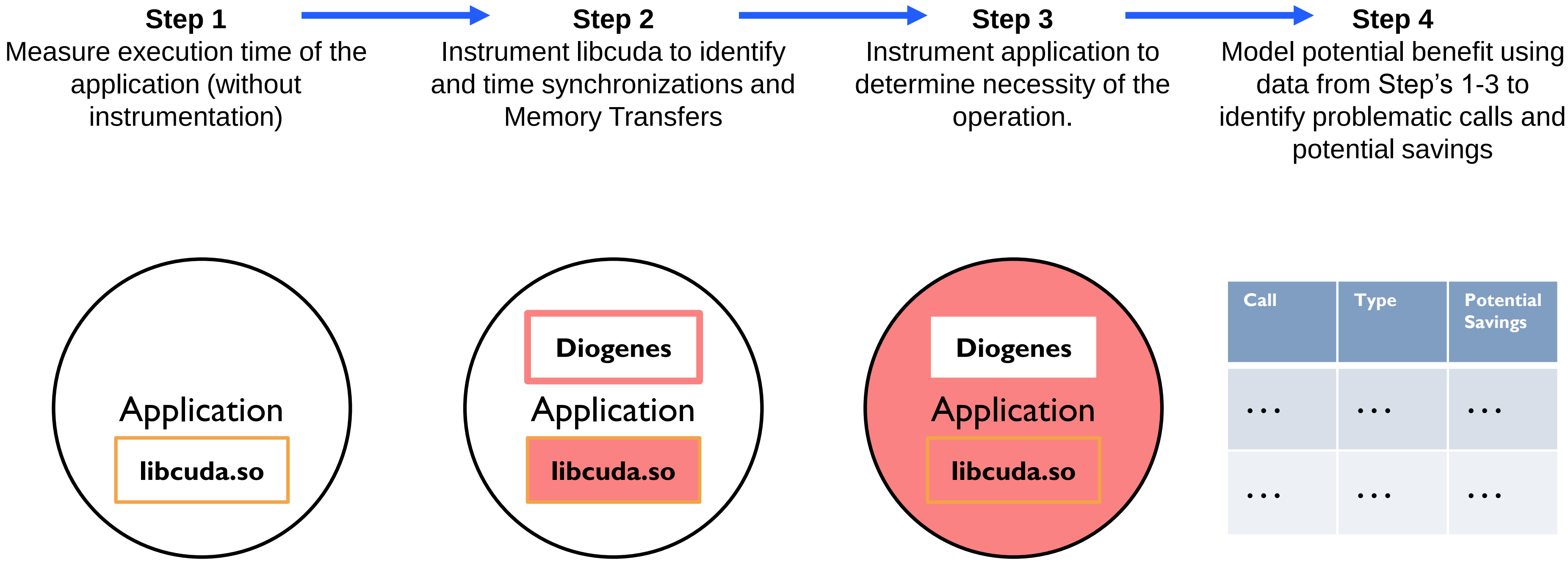
Step 3
Instrument application to determine necessity of the operation.

Diogenes performs each step automatically (via a launcher)

Diogenes – Workflow

Diogenes uses a newly developed technique called feed forward instrumentation:

- The results of previous instrumentation guides the insertion of new instrumentation.



Step 1
Measure execution time of the application (without instrumentation)

Step 2
Instrument libcuda to identify and time synchronizations and Memory Transfers

Step 3
Instrument application to determine necessity of the operation.

Step 4
Model potential benefit using data from Step's 1-3 to identify problematic calls and potential savings

Call	Type	Potential Savings
…	…	…
…	…	…

Diogenes performs each step automatically (via a launcher)

Diogenes – Overhead/Limitations

Overhead:

- ~~30-70x~~ 6x-20x application run time
- ~~Dyninst parsing overhead on really large binaries (e.g. >40 minutes for 1.5 GB binary)~~
  - Parse overhead now in the few minute range for parsing large binaries thanks to parallel parsing.

Limited to single user threaded programs

The Gap In Performance Tools

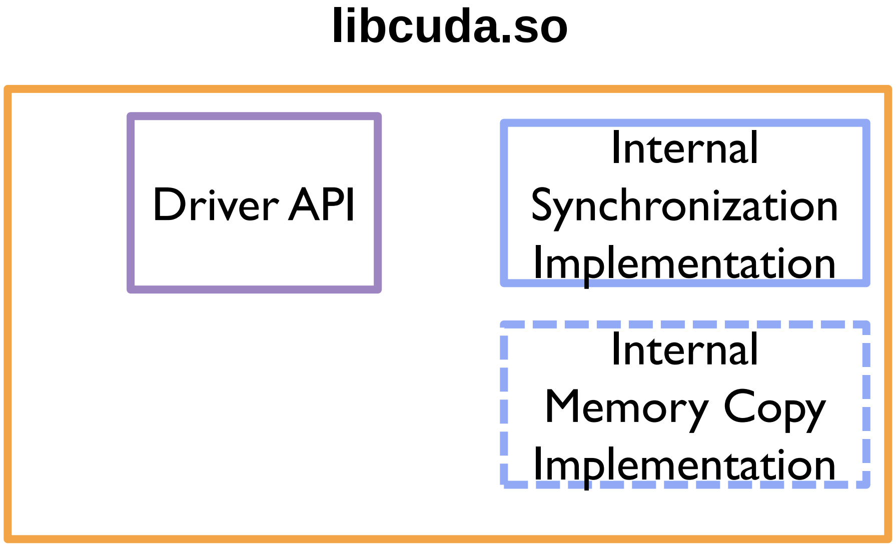
Existing Tools (CUPTI, etc.) have collection and analysis gaps preventing detection of issues

- Don't collect performance data on hidden interactions
  - Conditional Interactions
  - Implicitly synchronizing API calls
  - Private API calls

Conditional Interaction

Conditional Interactions are unreported (and undocumented) synchronizations/transfers performed by a CUDA call.

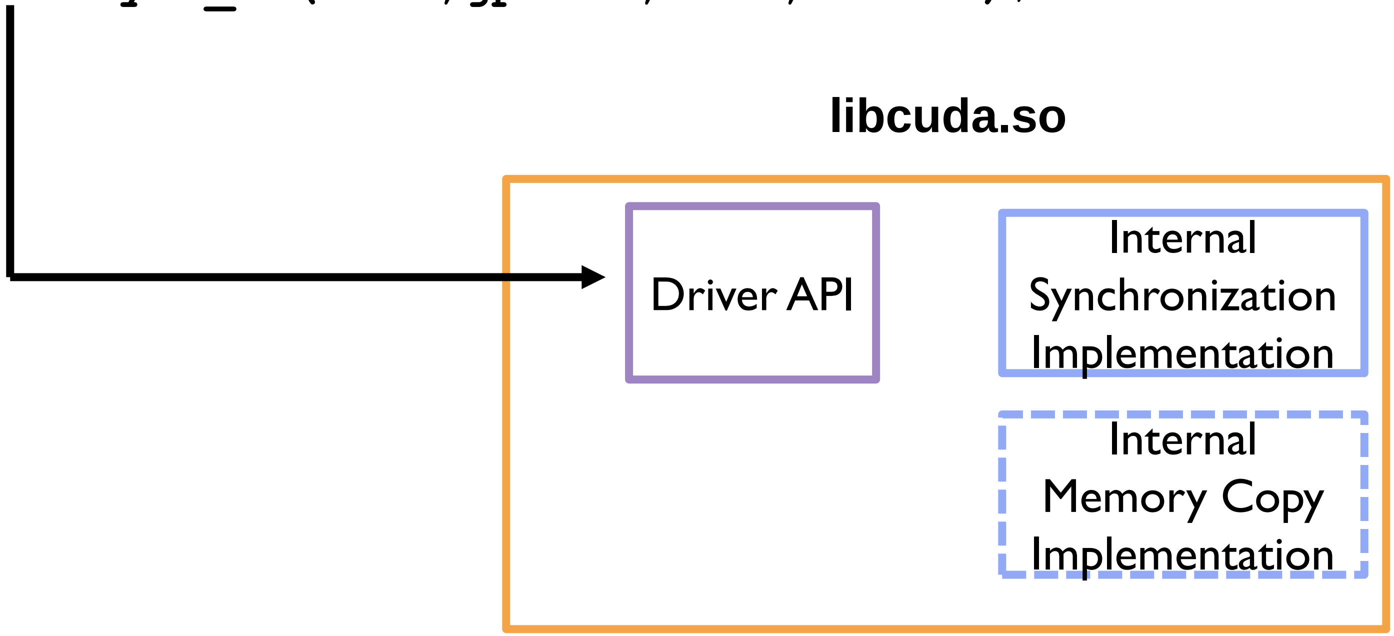
```
dest = malloc(size);
cuMemcpyDtoHAsync_v2(dest,gpuMem,size,stream);
```


libcuda.so
Driver API
Internal Synchronization Implementation
Internal Memory Copy Implementation

Conditional Interaction

Conditional Interactions are unreported (and undocumented) synchronizations/transfers performed by a CUDA call.

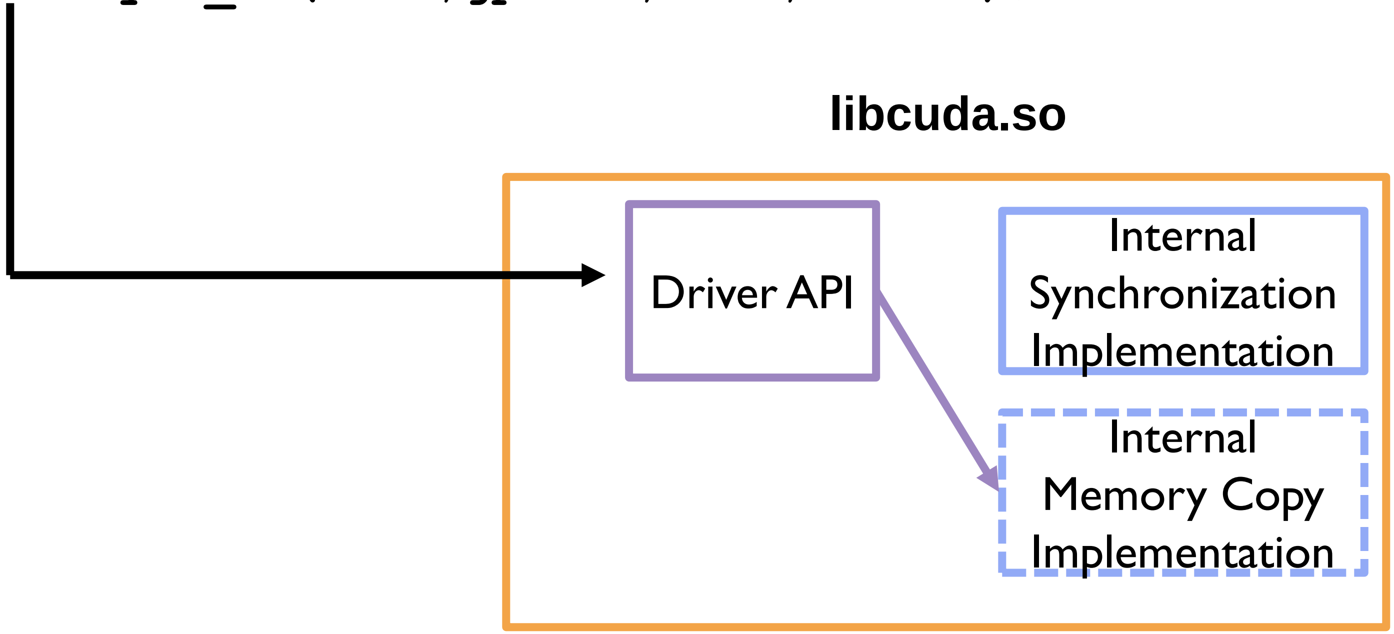
```
dest = malloc(size);
cuMemcpyDtoHAsync_v2(dest,gpuMem,size,stream);
```


libcuda.so
Driver API
Internal Synchronization Implementation
Internal Memory Copy Implementation

Conditional Interaction

Conditional Interactions are unreported (and undocumented) synchronizations/transfers performed by a CUDA call.

```
dest = malloc(size);
cuMemcpyDtoHAsync_v2(dest,gpuMem,size,stream);
```

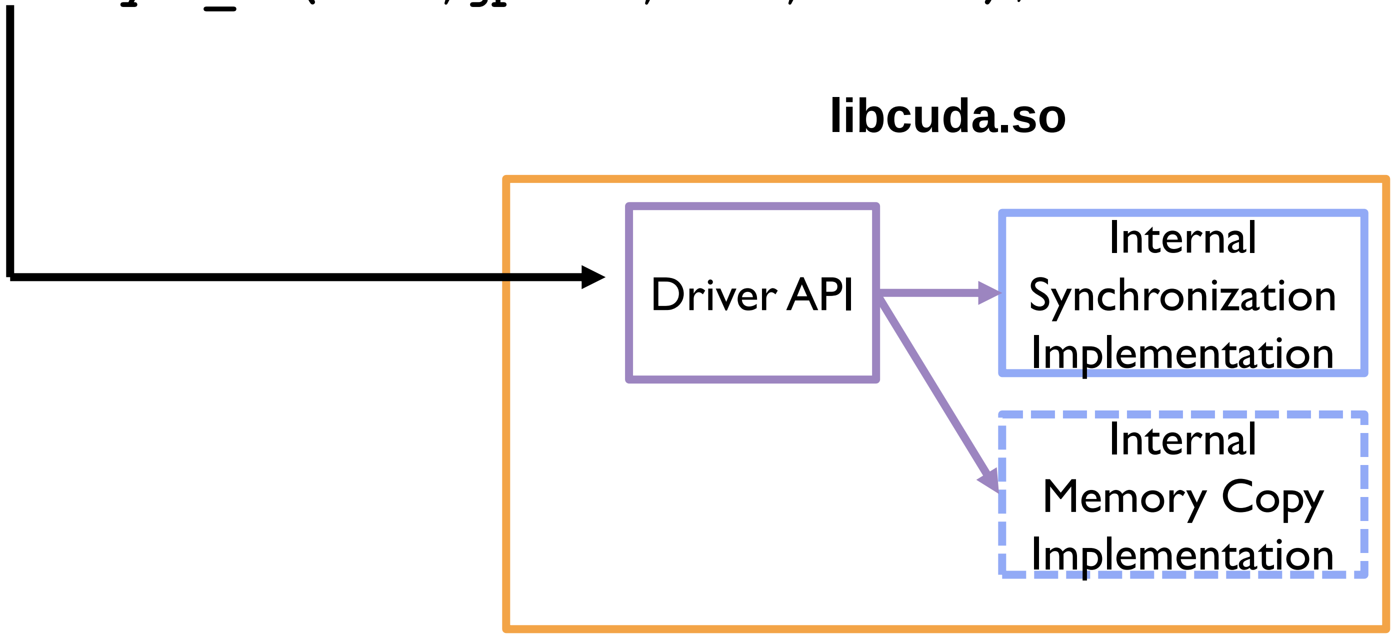


libcuda.so
Driver API
Internal Synchronization Implementation
Internal Memory Copy Implementation

Conditional Interaction

Conditional Interactions are unreported (and undocumented) synchronizations/transfers performed by a CUDA call.

```
dest = malloc(size);
cuMemcpyDtoHAsync_v2(dest,gpuMem,size,stream);
```

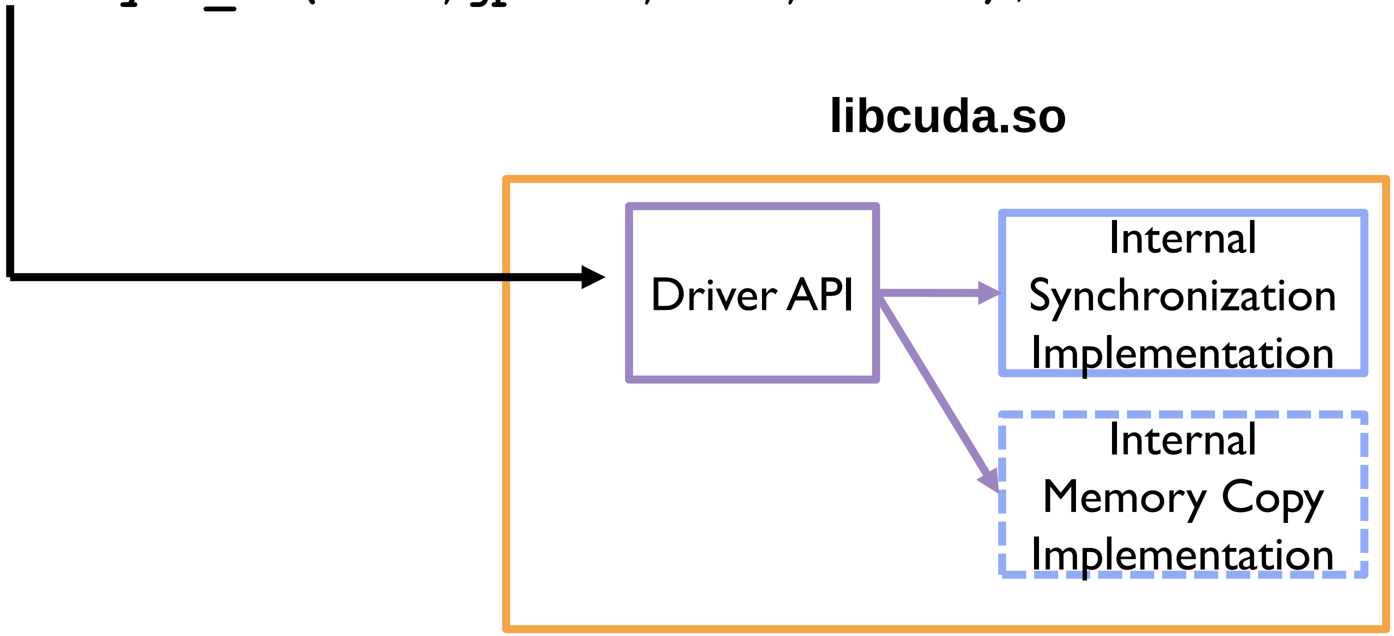

libcuda.so
Driver API
Internal Synchronization Implementation
Internal Memory Copy Implementation

Conditional Interaction

Conditional Interactions are unreported (and undocumented) synchronizations/transfers performed by a CUDA call.

```
dest = malloc(size);
cuMemcpyDtoHAsync_v2(dest,gpuMem,size,stream);
```

Synchronous due to the way dest was allocated



libcuda.so

Driver API

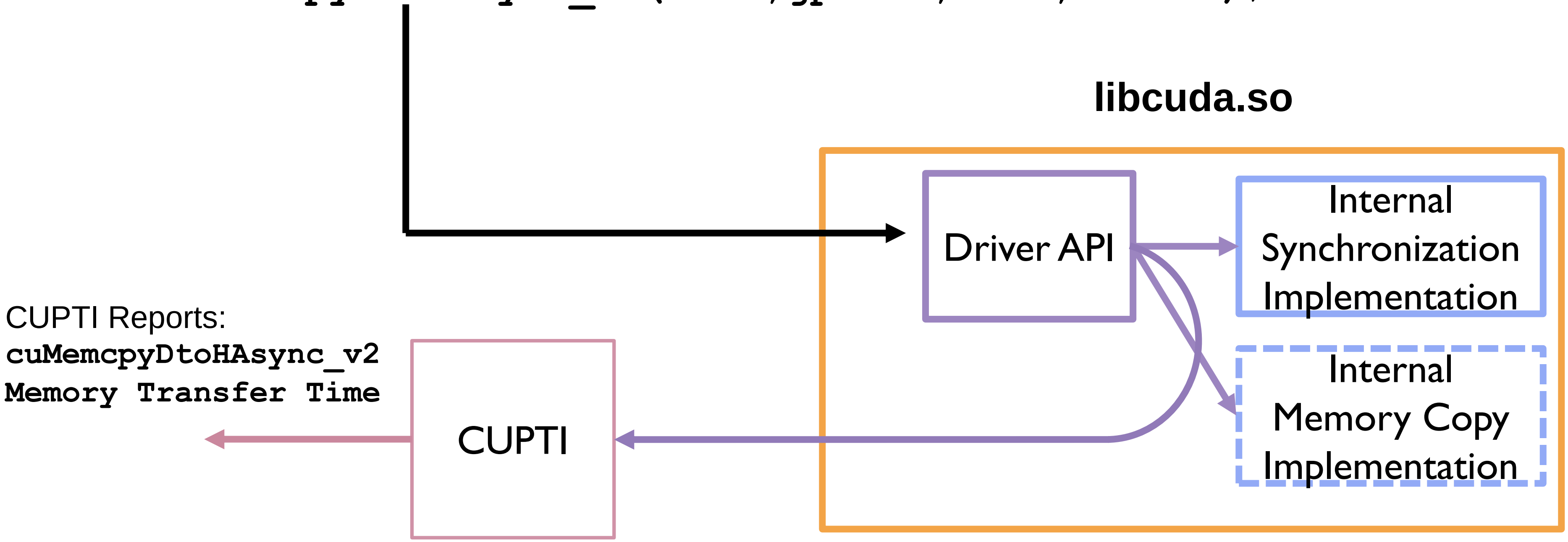
Internal Synchronization Implementation

Internal Memory Copy Implementation

Conditional Interaction Collection Gap

CUPTI doesn't report when undocumented interactions are performed by a call.

```
dest = malloc(size);
cuMemcpyDtoHAsync_v2(dest,gpuMem,size,stream);
```



libcuda.so
Driver API
Internal Synchronization Implementation
Internal Memory Copy Implementation
CUPTI

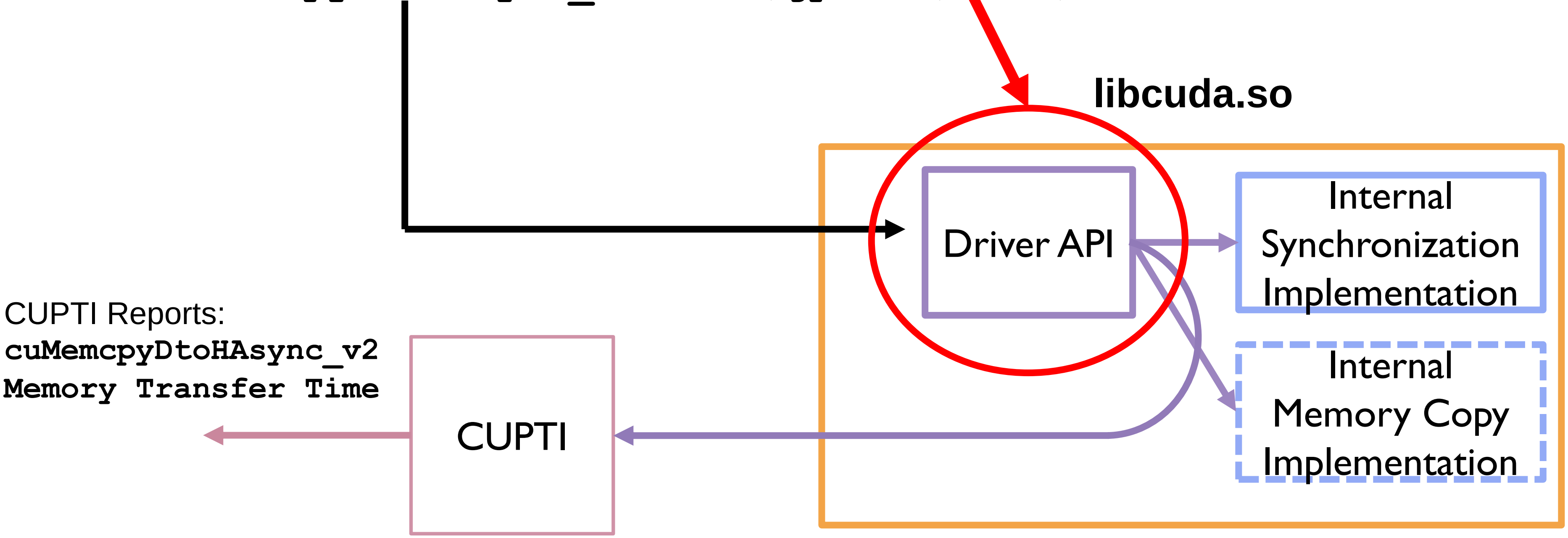
CUPTI Reports:
`cuMemcpyDtoHAsync_v2`
`Memory Transfer Time`

Conditional Interaction Collection Gap

CUPTI doesn’t report when undocumented interactions are performed by a call.

Call back to CUPTI does not contain information about whether a synchrounization occurred.

```
dest = malloc(size);
cuMemcpyDtoHAsync_v2(dest,gpuMem,size,stream);
```



libcuda.so

Driver API

Internal Synchronization Implementation

Internal Memory Copy Implementation

CUPTI

CUPTI Reports:
`cuMemcpyDtoHAsync_v2`
`Memory Transfer Time`

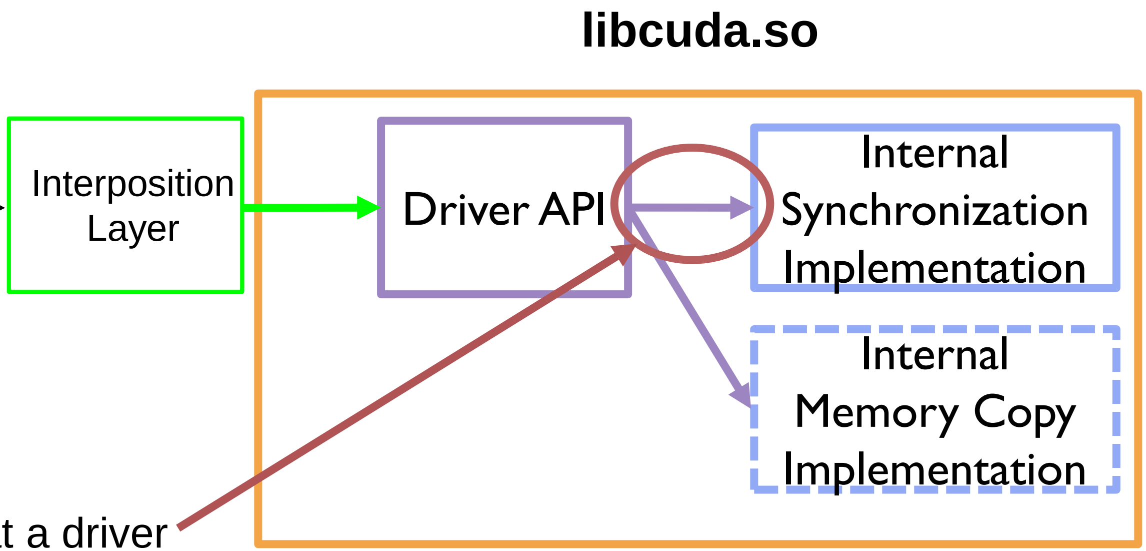
Conditional Interaction Collection Gap

Hard to detect with library interposition approaches due to:

1. Need to know under what undocumented conditions a call can perform an interaction.

```
dest = malloc(size);
cuMemcpyDtoHAsync_v2(dest,gpuMem,size,stream);
```

2. Need to capture operations potentially unrelated to CUDA to see if the call meets those conditions.


libcuda.so
Interposition Layer
Driver API
Internal Synchronization Implementation
Internal Memory Copy Implementation

3. Hope that a driver update doesn’t change behavior.

Implicit Synchronization Collection Gap

CUPTI does not collect synchronization performance data for implicitly synchronizing CUDA calls

- Examples include cudaMemcpy, cudaFree, etc

We believe CUPTI collects performance data for synchronizations only for the following calls

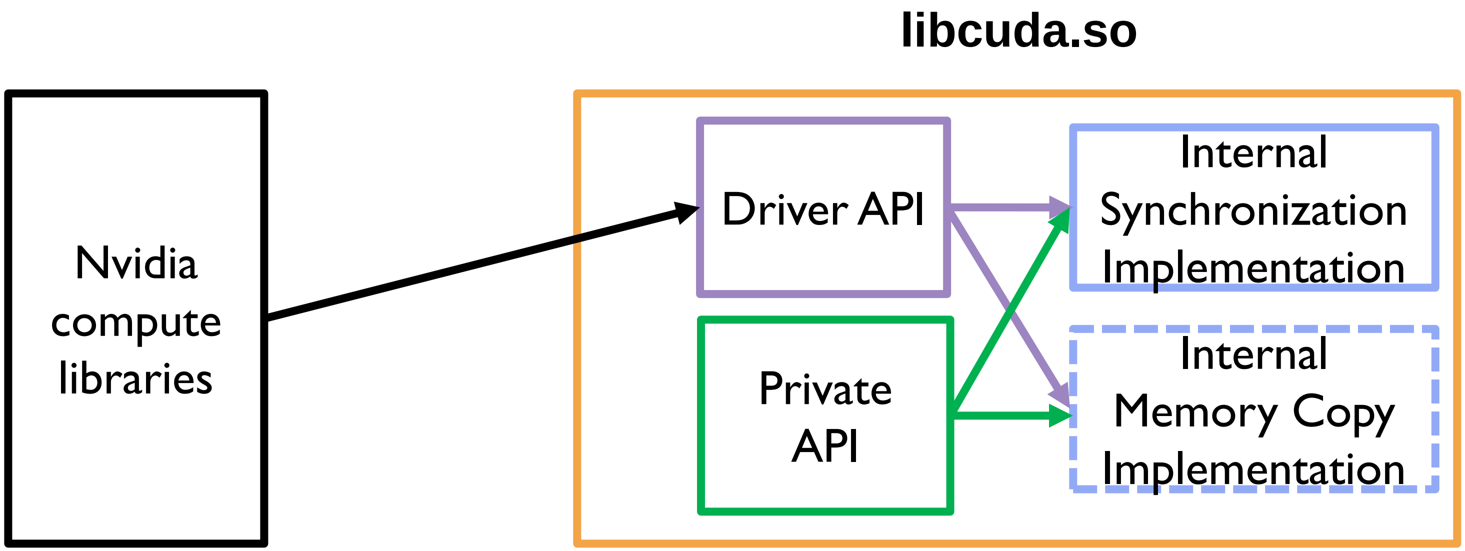
- cudaDeviceSynchronize
- cudaStreamSynchronize.

[Unconfirmed] Change in the way synchronizations are performed in CUDA 10 that effect all CUDA calls.

- It now appears all calls check to see if a synchronization should be performed
- Change from previous behavior of only potentially synchronous calls performing this check

The Private API

Large private API used by Nvidia compute libraries (cufft, cublas, cudnn, etc) which has all the capabilities of the public API (and many more).

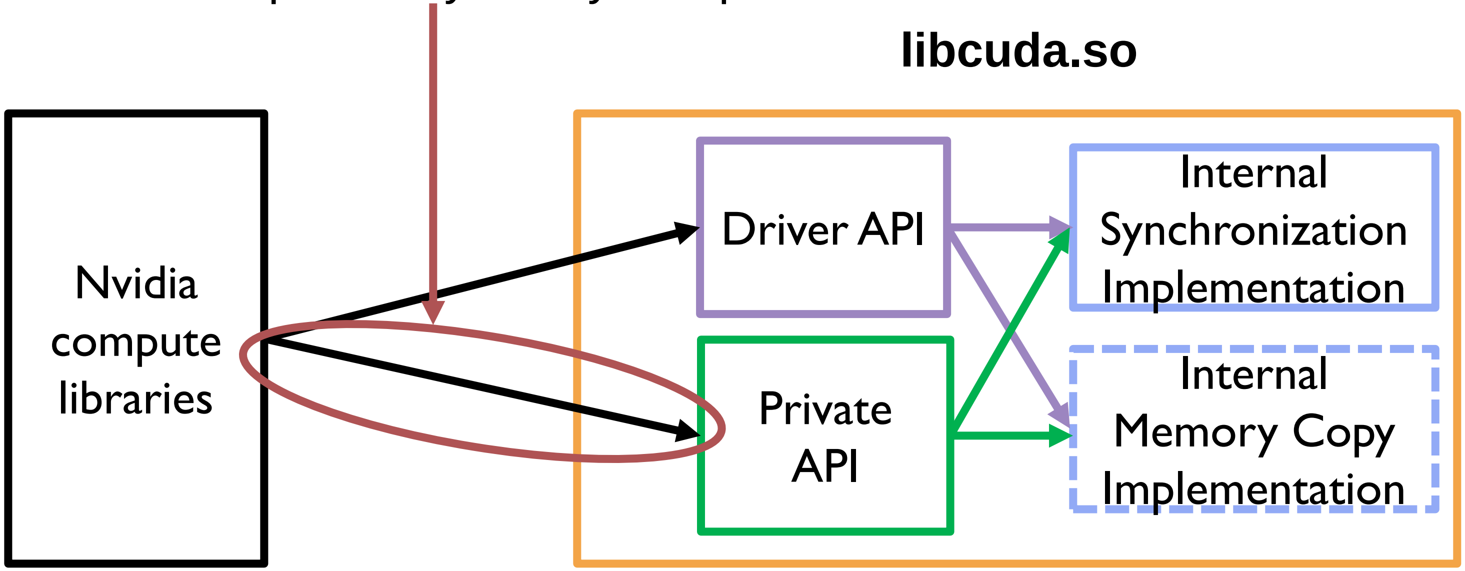

libcuda.so
Nvidia compute libraries
Driver API
Private API
Internal Synchronization Implementation
Internal Memory Copy Implementation

*Fun Fact: CUPTI sets its callbacks through the Private API

The Private API

Large private API used by Nvidia compute libraries (cufft, cublas, cudnn, etc) which has all the capabilities of the public API (and many more).

Calls are not reported by CUPTI* and are not captured by library interposition


libcuda.so
Nvidia compute libraries
Driver API
Private API
Internal Synchronization Implementation
Internal Memory Copy Implementation

*Fun Fact: CUPTI sets its callbacks through the Private API

Diogenes Predictive Accuracy Overview

App Name	App Type	Diogenes Estimated Benefit (% of Exec)	Actual Benefit by Manual Fix (% of Exec)
cumf_als	Matrix Factorization	10.0%	8.3%
AMG	Algebraic Solver	6.8%	5.8%
Rodinia	Gaussian Benchmark	2.2%	2.1%
cuIBM	CFD	10.8%	17.6%

cuIBM’s and cumf_als had synchronization issues that were symptoms of larger problems

- Memory management issues (cudaMalloc/cudaFree)
- Asynchronous transfer issues (synchronous cudaMemcpyAsync)

Fixing the cause of these issues can result in much larger benefit

- Removing the malloc, using cudaMallocHost to allocate memory to be used with cudaMemcpyAsync, etc.

Identifying Larger Synchronization Problems

Extend Diogenes to determine the potential remedy of the synchronization issue:

- Remove the synchronization
- Move the synchronization
- Fix the memory management issue
- Fix the asynchronous transfer issue

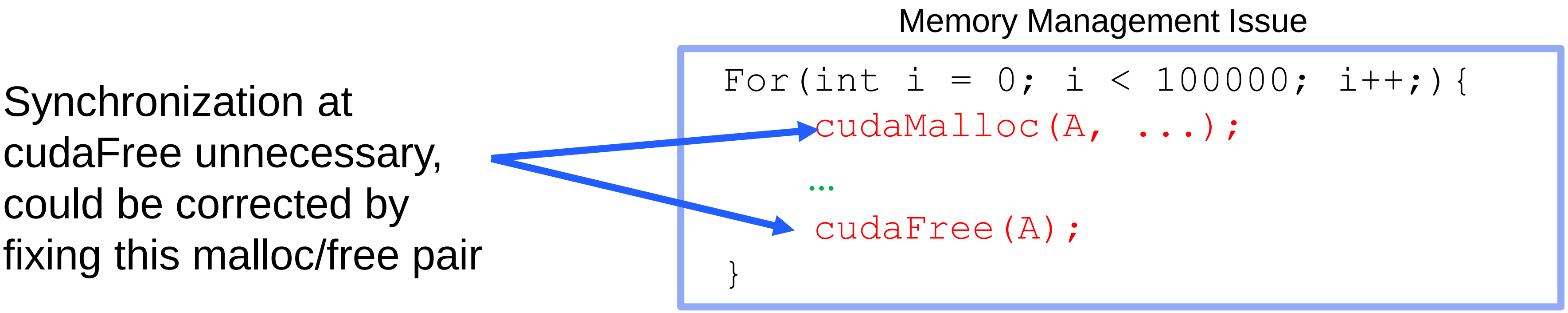
Memory Management Issue

```
For(int i = 0; i < 100000; i++;){
    cudaMalloc(A, ...);
    …
    cudaFree(A);
}
```

Identifying Larger Synchronization Problems

Extend Diogenes to determine the potential remedy of the synchronization issue:

- Remove the synchronization
- Move the synchronization
- Fix the memory management issue
- Fix the asynchronous transfer issue



Synchronization at cudaFree unnecessary, could be corrected by fixing this malloc/free pair

Memory Management Issue

```
For(int i = 0; i < 100000; i++;){
      cudaMalloc(A, ...);
      ...
      cudaFree(A);
}
```

Identifying Larger Synchronization Problems

Implemented an autocorrect feature that can apply a remedy for memory management and asynchronous transfer issues

- No modeling, the number reported is the actual benefit.

Memory Management Issue

```
For(int i = 0; i < 100000; i++;){
    cudaMalloc(A, ...);
    …
    cudaFree(A);
}
```

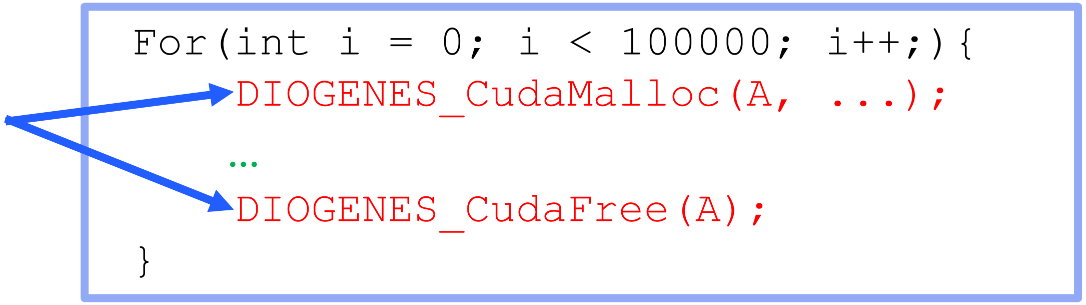
Identifying Larger Synchronization Problems

Implemented an autocorrect feature that can apply a remedy for memory management and asynchronous transfer issues

- No modeling, the number reported is the actual benefit.

Use Dyninst to rewrite cudaFree (and their associated cudaMalloc operations) with calls to a memory pool that does not synchronize

Memory Management Issue



```
For(int i = 0; i < 100000; i++;){
      DIOGENES_CudaMalloc(A, ...);
      ...
      DIOGENES_CudaFree(A);
}
```

Diogenes Autocorrect Preliminary Results

App Name	App Type	Diogenes Estimated Benefit (% of Exec)	AutoFix Reduction in Exec Time (% of Exec)
cumf_als	Matrix Factorization	17.3%	43%
culBM	CFD	22.0%	47%

Note: Still in progress research, numbers may change

Questions?

Papers:

- Diogenes: Looking For An Honest CPU/GPU Performance Measurement Tool
  - To appear at SC19, Available now on http://paradyn.org/
- Autocorrect/Remedy Identification with Diogenes
  - Available soon

Diogenes Github: http://github.com/bwelton/diogenes