Portable and Transparent Host-Device Communication Optimization for GPGPU Environments

Christos Margiolas, Michael F.P. O'Boyle

University of Edinburgh
Institute for Computing Systems Achritecture

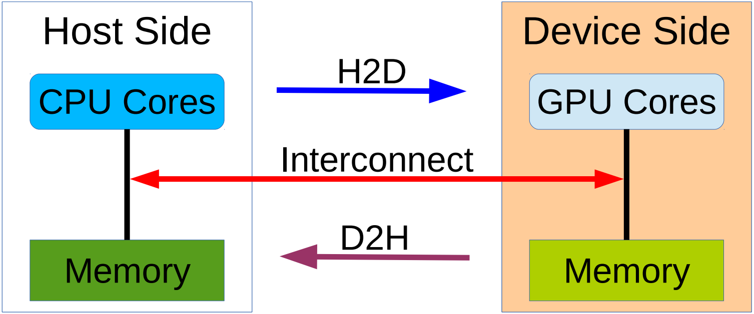
February 17, 2014

Host-Device Communication

What is Host-Device Communication?

Host-Device Communication

What is Host-Device Communication?



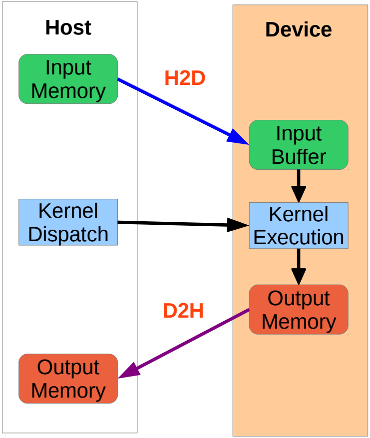
- H2D: Host to Device Communication
- D2H: Device to Host Communication

Computation Offloading

Why Host-Device Communication is required?

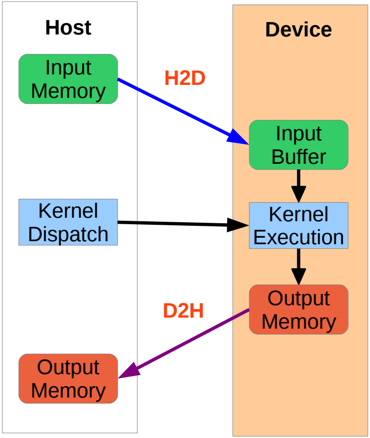
Computation Offloading

Why Host-Device Communication is required?



Computation Offloading

Why Host-Device Communication is required?

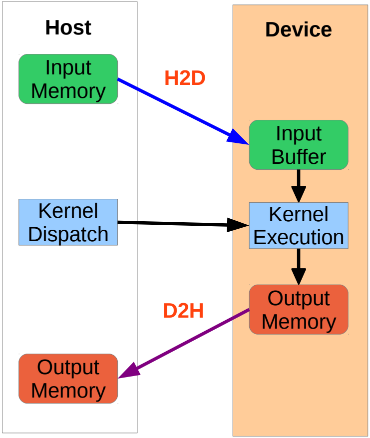


Computation offloading requires:

- **H2D** Transfer of Input Data
- **D2H** Transfer of Output Data

Computation Offloading

Why Host-Device Communication is required?



Computation offloading requires:

- **H2D** Transfer of Input Data
- **D2H** Transfer of Output Data

Communication Overhead

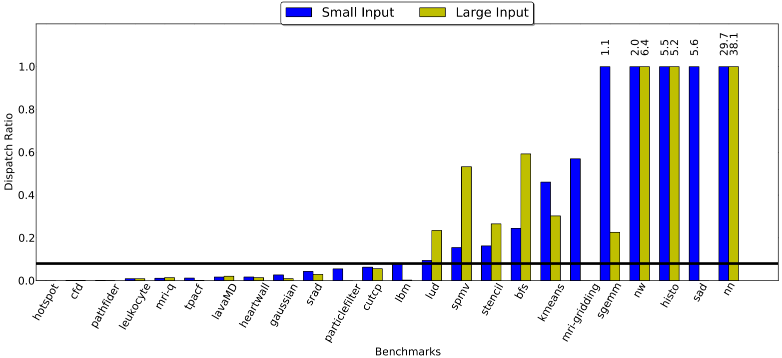
What is the impact of communication overhead on application execution?

Quantifying the communication overhead

$$\text{Dispatch Ratio} = \frac{\text{Cumulative Host-Device Communication Time}}{\text{Cumulative Device Computation Time}}$$

Quantifying the communication overhead (2)

Dispatch Ratio across Parboil and Rodinia benchmarks

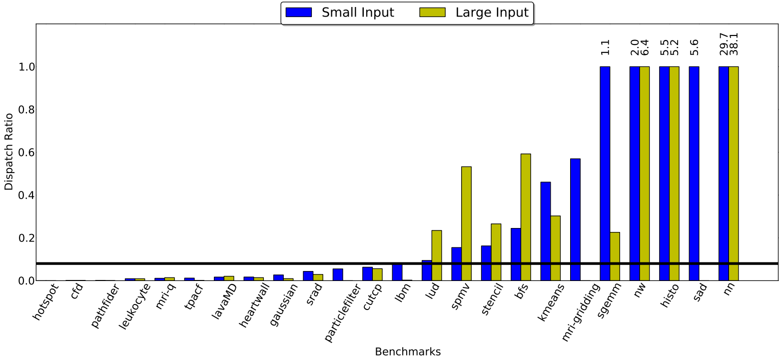


Communication Overhead

Significant to extremely high overhead for 12 benchmarks in total.

Quantifying the communication overhead (2)

Dispatch Ratio across Parboil and Rodinia benchmarks



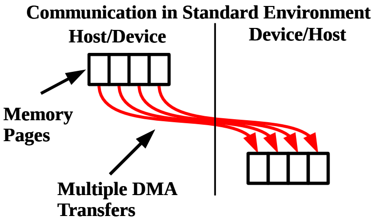
Communication Overhead

Significant to extremely high overhead for 12 benchmarks in total.

Reducing Communication Overhead

Can we reduce the communication overhead?

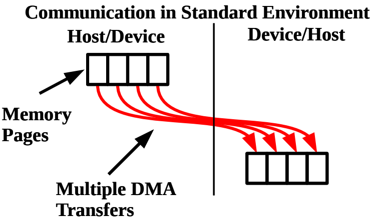
Memory Allocation affects Communication Performance

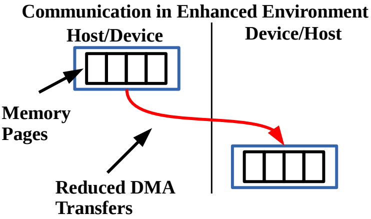


Standard Allocation:

- Mem. Pages Swappable
- Transfer per page
- Unsteady Performance

Memory Allocation affects Communication Performance





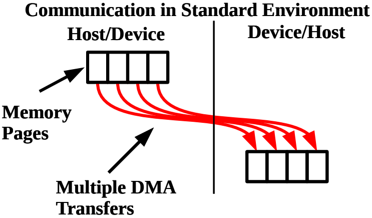
Standard Allocation:

- Mem. Pages Swappable
- Transfer per page
- Unsteady Performance

Allocation with Mem. Locking:

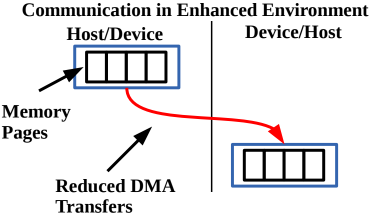
- Mem. Pages pinned in RAM
- Transfer per Max DMA size
- Improved performance

Memory Allocation affects Communication Performance



Standard Allocation:

- Mem. Pages Swappable
- Transfer per page
- Unsteady Performance



Allocation with Mem. Locking:

- Mem. Pages pinned in RAM
- Transfer per Max DMA size
- Improved performance

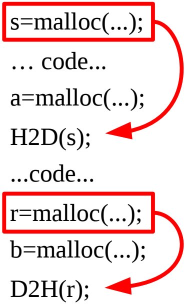
Issue 1: Platform capabilities

Multiple allocation policies available and affect Host-Device Communication. Need to quantify and compare them.

Few Mem. Allocations used in Host-Device Communication

```
s=malloc(...);
… code...
a=malloc(...);
H2D(s);
...code...
r=malloc(...);
b=malloc(...);
D2H(r);
```

Few Mem. Allocations used in Host-Device Communication



- Dozens of memory allocations performed by an application.
- Only few are used for Host-Device Communication.

Few Mem. Allocations used in Host-Device Communication

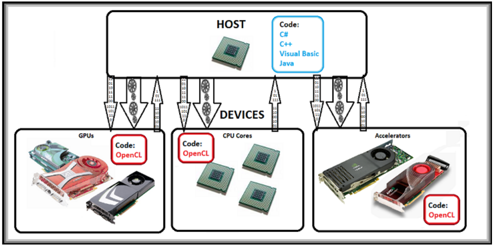
```
s=malloc(...);
... code...
a=malloc(...);
H2D(s);
...code...
r=malloc(...);
b=malloc(...);
D2H(r);
```

- Dozens of memory allocations performed by an application.
- Only few are used for Host-Device Communication.

Issue 2: Application Behavior

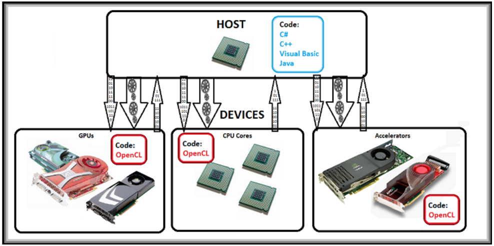
Need to **detect the memory allocations that are used in Host-Device communication**. The goal is to serve them with the allocation policy that leads to the highest communication rates.

Portability and Transparency



Processor Types:
CPUs,GPUs,HSA,DSPs,FPGAs
Programming Interface:
OpenCL

Portability and Transparency

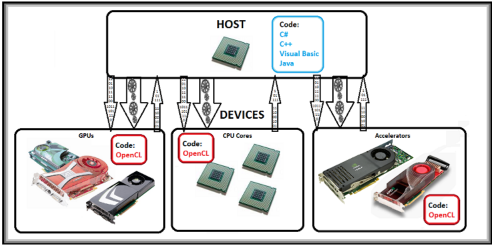


Processor Types:
CPUs,GPUs,HSA,DSPs,FPGAs
Programming Interface:
OpenCL

Matter of Fact

Target Platform remains unknown until the execution time.

Portability and Transparency



Processor Types:
CPUs,GPUs,HSA,DSPs,FPGAs
Programming Interface:
OpenCL

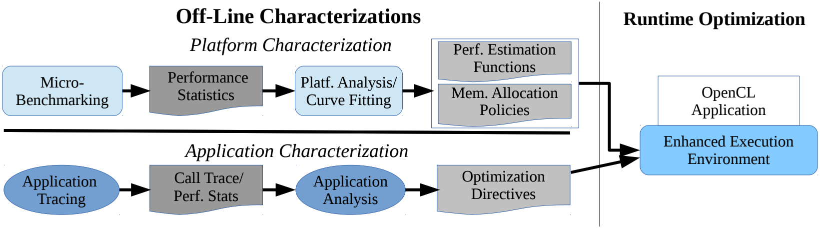
Matter of Fact

Target Platform remains unknown until the execution time.

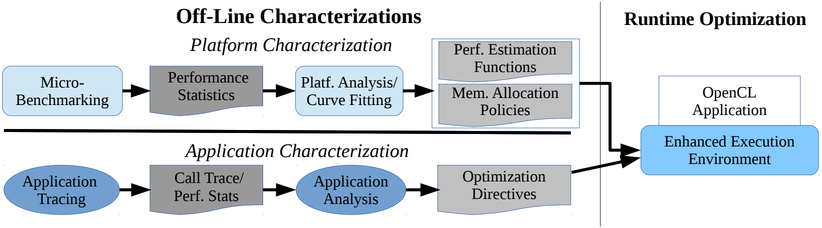
Issue 3: Portability and Transparency

Need for optimizations portable across platforms and transparent to applications and runtime libraries.

Optimization Overview

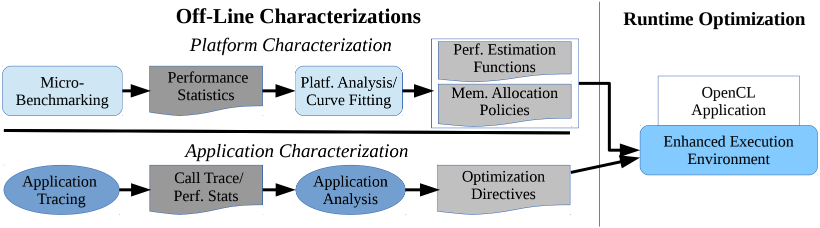


Optimization Overview



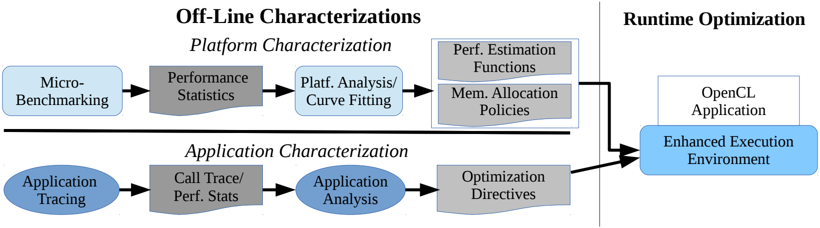
- **Platform Characterization** discovers the memory allocation and host-device communication capabilities of the platform.

Optimization Overview



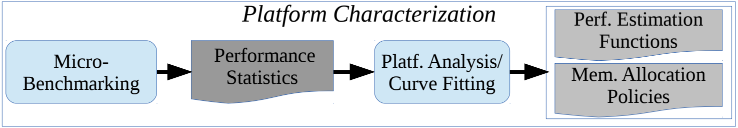
- **Platform Characterization** discovers the memory allocation and host-device communication capabilities of the platform.
- **Application Characterization** detects the memory allocations that are used in host-device communication.

Optimization Overview

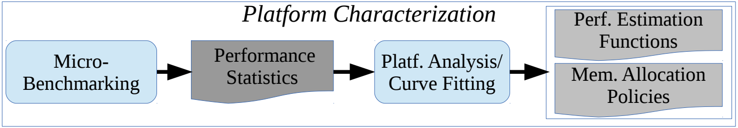


- **Platform Characterization** discovers the memory allocation and host-device communication capabilities of the platform.
- **Application Characterization** detects the memory allocations that are used in host-device communication.
- **Runtime Optimization** uses both characterizations for the runtime optimization of the application.

Platform Characterization / Micro-benchmarking

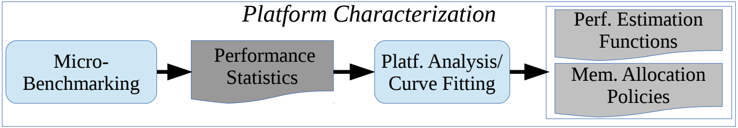


Platform Characterization / Micro-benchmarking



Two micro-benchmarks provide **allocation** and **communication** overhead statistics.

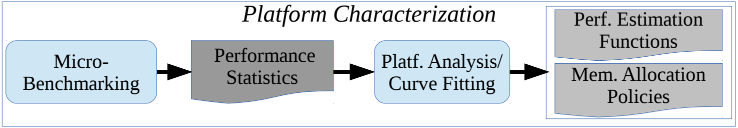
Platform Characterization / Micro-benchmarking



Two micro-benchmarks provide **allocation** and **communication** overhead statistics.

We consider the following **memory allocation policies**:

Platform Characterization / Micro-benchmarking

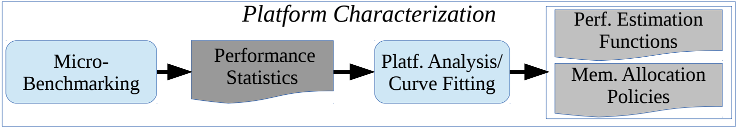


Two micro-benchmarks provide **allocation** and **communication** overhead statistics.

We consider the following **memory allocation policies**:

- **Standard**, the standard memory allocator.

Platform Characterization / Micro-benchmarking

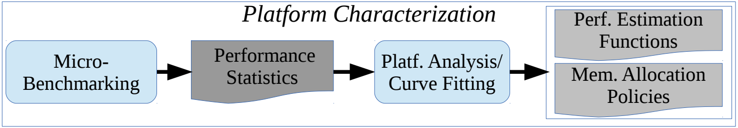


Two micro-benchmarks provide **allocation** and **communication** overhead statistics.

We consider the following **memory allocation policies**:

- **Standard**, the standard memory allocator.
- **OpenCL**, allocation via OpenCL library.

Platform Characterization / Micro-benchmarking

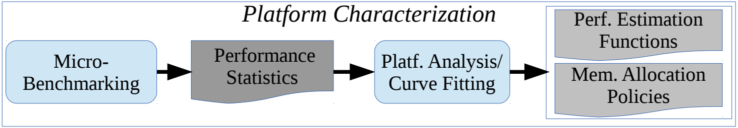


Two micro-benchmarks provide **allocation** and **communication** overhead statistics.

We consider the following **memory allocation policies**:

- **Standard**, the standard memory allocator.
- **OpenCL**, allocation via OpenCL library.
- **Standard with Locking**, allocation with memory locking via POSIX.

Platform Characterization / Micro-benchmarking



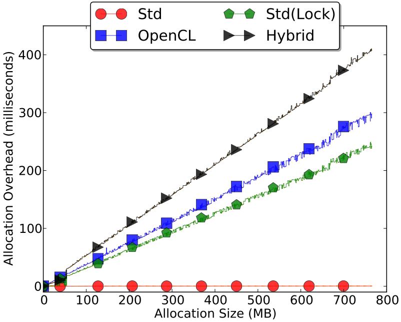
Two micro-benchmarks provide **allocation** and **communication** overhead statistics.

We consider the following **memory allocation policies**:

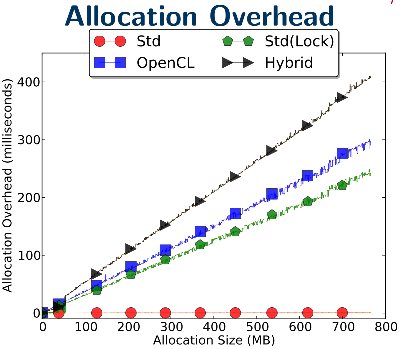
- **Standard**, the standard memory allocator.
- **OpenCL**, allocation via OpenCL library.
- **Standard with Locking**, allocation with memory locking via POSIX.
- **Hybrid**, combination of OpenCL and POSIX policies.

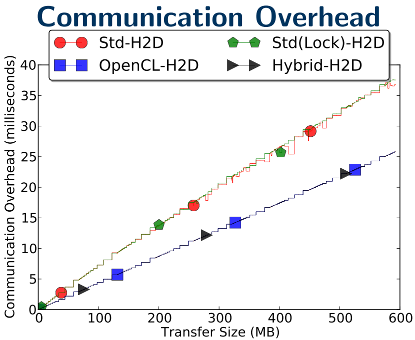
Platform Characterization / Performance Estim. Functions

Allocation Overhead

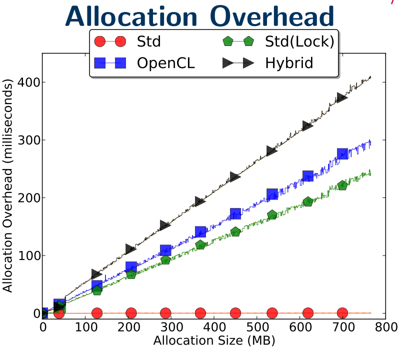


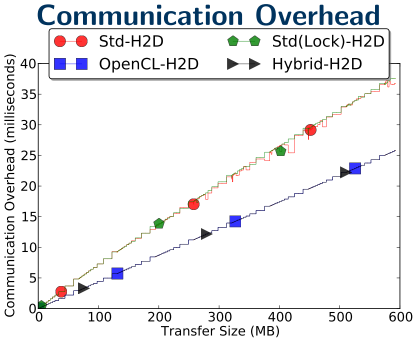
Platform Characterization / Performance Estim. Functions





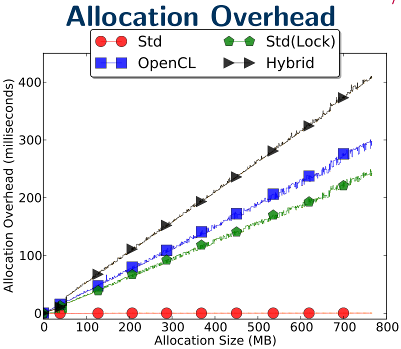
Platform Characterization / Performance Estim. Functions

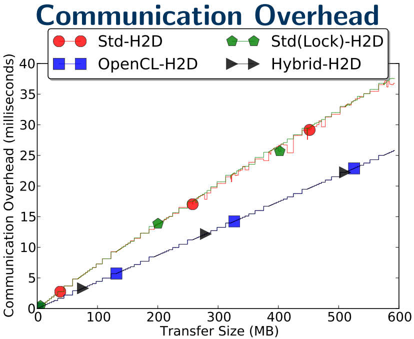




Remark: Policies with high allocation overhead lead to low communication overhead.

Platform Characterization / Performance Estim. Functions

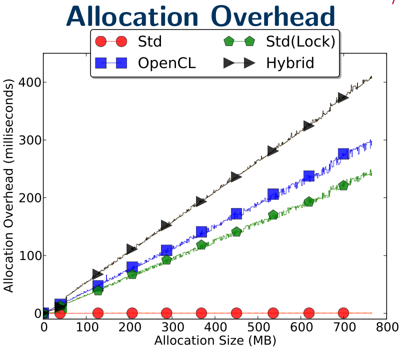


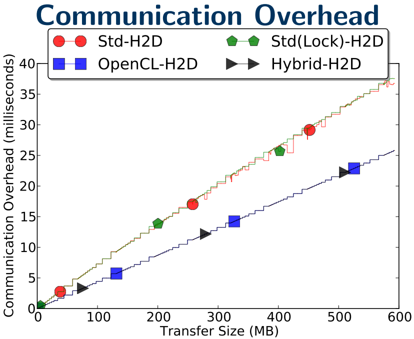


Remark: Policies with high allocation overhead lead to low communication overhead.

Curve fitting is performed on the collected statistics and generates **performance estimation functions**.

Platform Characterization / Performance Estim. Functions



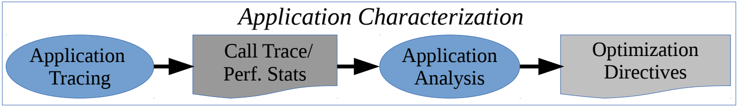


Remark: Policies with high allocation overhead lead to low communication overhead.

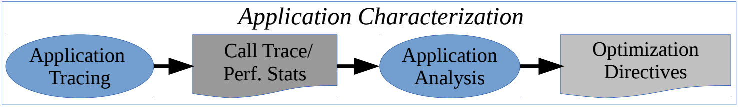
Curve fitting is performed on the collected statistics and generates **performance estimation functions**.

- **allocation_overhead(alloc_policy, size);**
- **communication_overhead(alloc_policy, size);**

Application Characterization / Tracing (1)

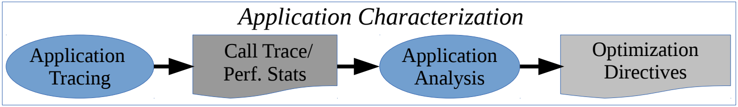


Application Characterization / Tracing (1)



Application tracing generates a **Compressed Trace** with:

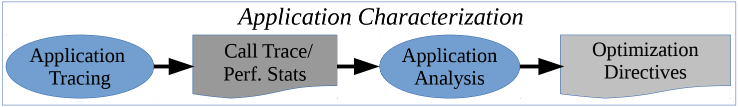
Application Characterization / Tracing (1)



Application tracing generates a **Compressed Trace** with:

- Every call to **OpenCL** and **memory allocation** functions.

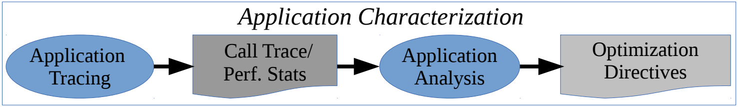
Application Characterization / Tracing (1)



Application tracing generates a **Compressed Trace** with:

- Every call to **OpenCL** and **memory allocation** functions.
- **Dependences** between the calls and data objects.

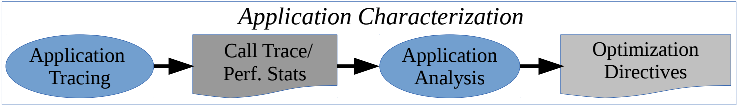
Application Characterization / Tracing (1)



Application tracing generates a **Compressed Trace** with:

- Every call to **OpenCL** and **memory allocation** functions.
- **Dependences** between the calls and data objects.
- Performance statistics for **host-device communication** and **kernel execution** operations.

Application Characterization / Tracing (1)



Application tracing generates a **Compressed Trace** with:

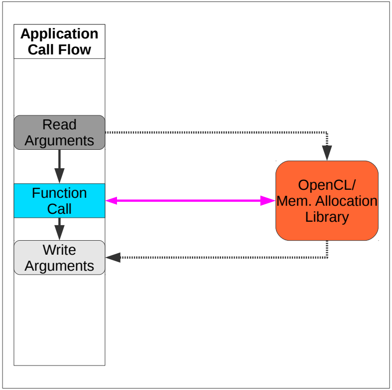
- Every call to **OpenCL** and **memory allocation** functions.
- **Dependences** between the calls and data objects.
- Performance statistics for **host-device communication** and **kernel execution** operations.

Important Feature

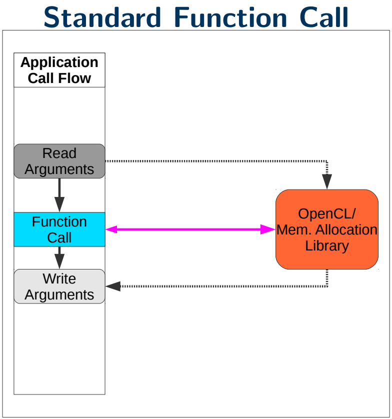
Trace compression guarantees that the trace remains the same regardless of the input size.

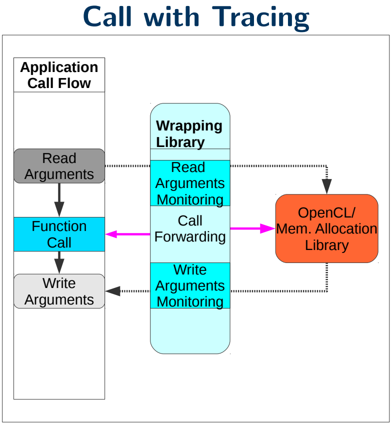
Application Characterization / Tracing (2)

Standard Function Call

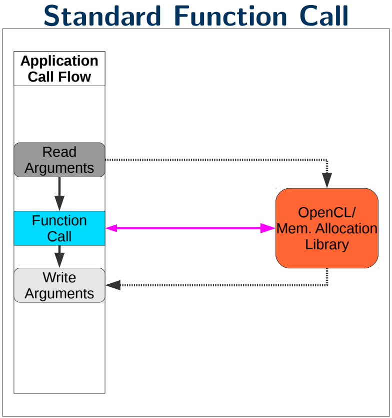


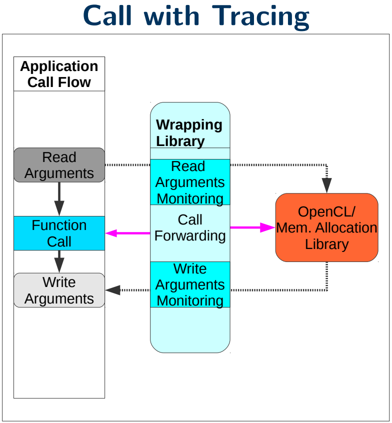
Application Characterization / Tracing (2)





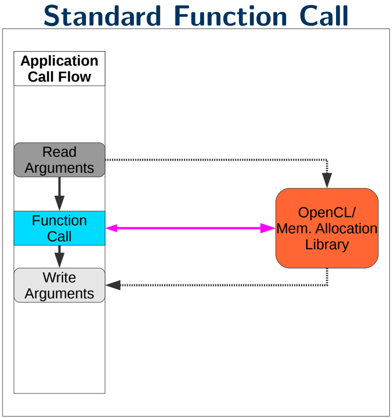
Application Characterization / Tracing (2)

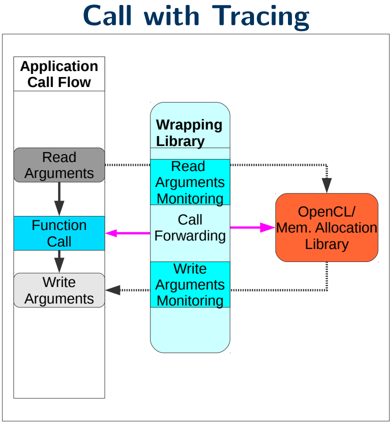




- Tracing is performed via a wrapping library.

Application Characterization / Tracing (2)





- Tracing is performed via a wrapping library.
- An SSA scheme is used for tracking the updates of non-scalar data objects, such as OpenCL Memory Buffers.

Application Characterization / Application Analysis

The analysis operates on the compressed trace in two stages.

Application Characterization / Application Analysis

The analysis operates on the compressed trace in two stages.

1: Optimization Eligibility Heuristic:

$$\text{Dispatch Ratio} = \frac{\text{Cumulative Host-Device Communication Time}}{\text{Cumulative Device Computation Time}}$$

An application is eligible if:

$$\text{Dispatch Ratio} \geq 0.1$$

Application Characterization / Application Analysis

The analysis operates on the compressed trace in two stages.

1: Optimization Eligibility Heuristic:

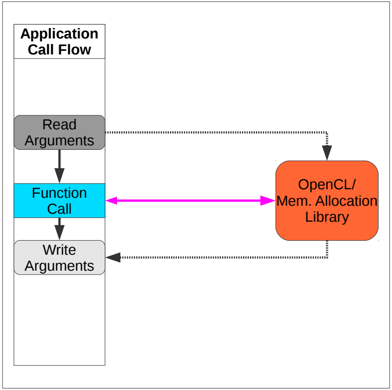
$$\text{Dispatch Ratio} = \frac{\text{Cumulative Host-Device Communication Time}}{\text{Cumulative Device Computation Time}}$$

An application is eligible if:

$$\text{Dispatch Ratio} \geq 0.1$$

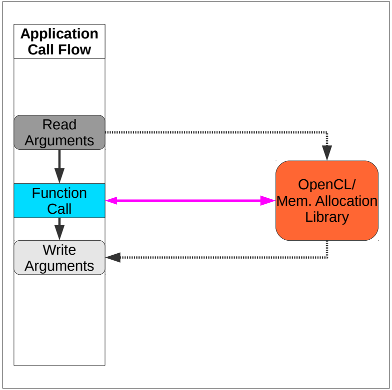
2: Memory Allocation Detection:
If the application is eligible, **the detection of memory allocations used in host-device communication** takes place.

Runtime Optimization
Standard Function Call

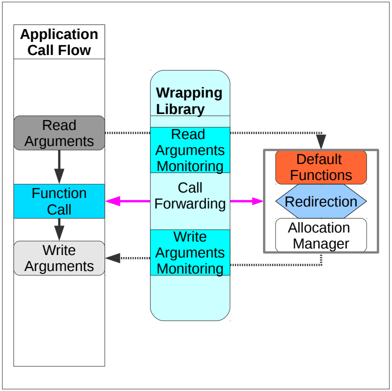


Runtime Optimization

Standard Function Call

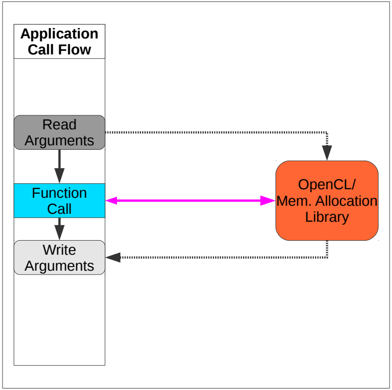


Call with Optimization

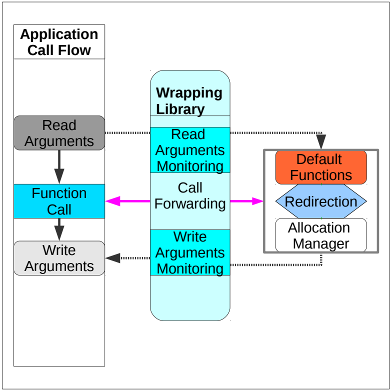


Runtime Optimization

Standard Function Call



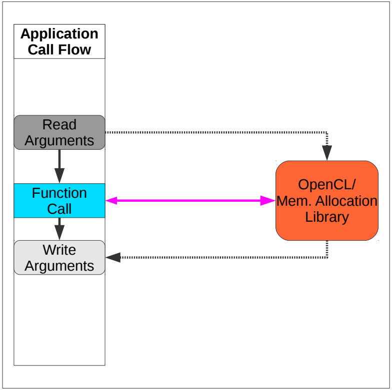
Call with Optimization



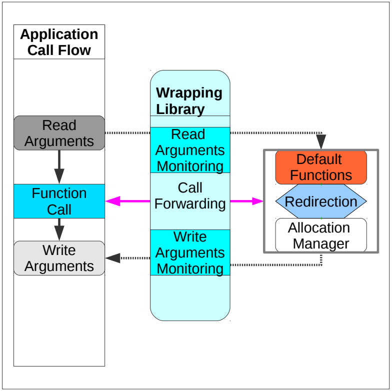
- Redirection of the annotated allocations to the best policy.

Runtime Optimization

Standard Function Call



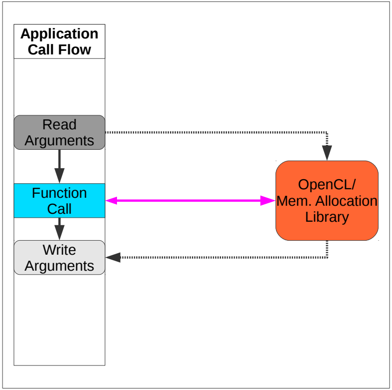
Call with Optimization



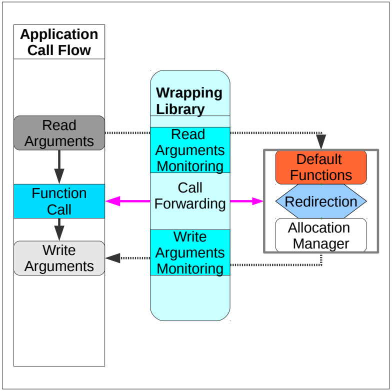
- Redirection of the annotated allocations to the best policy.
- Both Platform and Application characterizations required.

Runtime Optimization

Standard Function Call

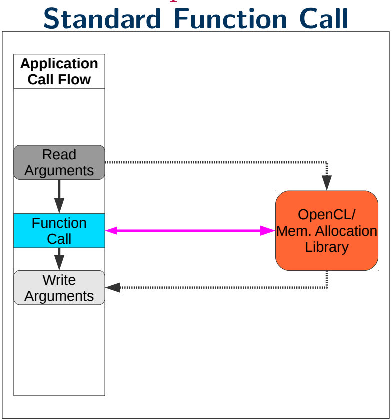


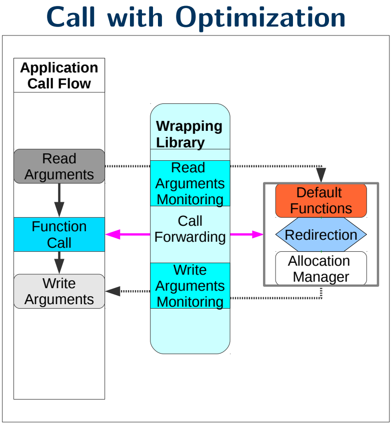
Call with Optimization



- Redirection of the annotated allocations to the best policy.
- Both Platform and Application characterizations required.
- User-space memory allocators for policies with high overhead.

Runtime Optimization





- Redirection of the annotated allocations to the best policy.
- Both Platform and Application characterizations required.
- User-space memory allocators for policies with high overhead.
- Safety. If an application presents an unexpected behavior, the optimization falls back to the default behavior.

Experimental Setup

Platforms:

We evaluate on **three** platform configurations.

Experimental Setup

Platforms:

We evaluate on **three** platform configurations.

GTX Platform Intel i7 X990 3.47GHz, 12GB 1333MHz,
NVIDIA GTX 580

Experimental Setup

Platforms:

We evaluate on **three** platform configurations.

GTX Platform Intel i7 X990 3.47GHz, 12GB 1333MHz, **NVIDIA GTX 580**

AMD Platform Intel i7 X990 3.47GHz, 8GB 1333MHz, **ATI Radeon HD 5970**

Experimental Setup

Platforms:

We evaluate on **three** platform configurations.

GTX Platform Intel i7 X990 3.47GHz, 12GB 1333MHz,
NVIDIA GTX 580

AMD Platform Intel i7 X990 3.47GHz, 8GB 1333MHz,
ATI Radeon HD 5970

K20 Platform Intel i7 3820 3.60GHz, 8GB 1333MHz,
NVIDIA Tesla K20c

Experimental Setup

Platforms:

We evaluate on **three** platform configurations.

GTX Platform Intel i7 X990 3.47GHz, 12GB 1333MHz,
NVIDIA GTX 580

AMD Platform Intel i7 X990 3.47GHz, 8GB 1333MHz,
ATI Radeon HD 5970

K20 Platform Intel i7 3820 3.60GHz, 8GB 1333MHz,
NVIDIA Tesla K20c

Benchmarks:

Experimental Setup

Platforms:

We evaluate on **three** platform configurations.

GTX Platform Intel i7 X990 3.47GHz, 12GB 1333MHz,
NVIDIA GTX 580

AMD Platform Intel i7 X990 3.47GHz, 8GB 1333MHz,
ATI Radeon HD 5970

K20 Platform Intel i7 3820 3.60GHz, 8GB 1333MHz,
NVIDIA Tesla K20c

Benchmarks:

Two Benchmark suites, **Rodinia** and **Parboil**.

Experimental Setup

Platforms:

We evaluate on **three** platform configurations.

GTX Platform Intel i7 X990 3.47GHz, 12GB 1333MHz,
NVIDIA GTX 580

AMD Platform Intel i7 X990 3.47GHz, 8GB 1333MHz,
ATI Radeon HD 5970

K20 Platform Intel i7 3820 3.60GHz, 8GB 1333MHz,
NVIDIA Tesla K20c

Benchmarks:

Two Benchmark suites, **Rodinia** and **Parboil**.
Evaluation with **12 optimization eligible** benchmarks.

Experimental Setup

Platforms:

We evaluate on **three** platform configurations.

GTX Platform Intel i7 X990 3.47GHz, 12GB 1333MHz, **NVIDIA GTX 580**

AMD Platform Intel i7 X990 3.47GHz, 8GB 1333MHz, **ATI Radeon HD 5970**

K20 Platform Intel i7 3820 3.60GHz, 8GB 1333MHz, **NVIDIA Tesla K20c**

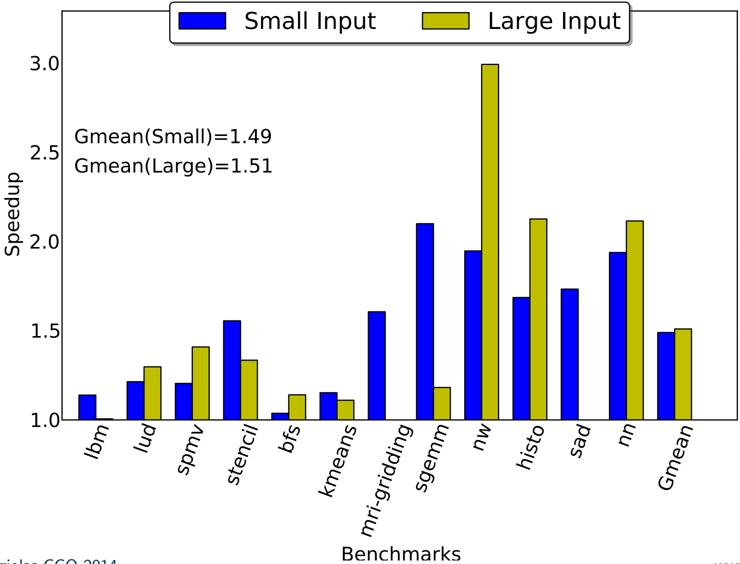
Benchmarks:

Two Benchmark suites, **Rodinia** and **Parboil**.

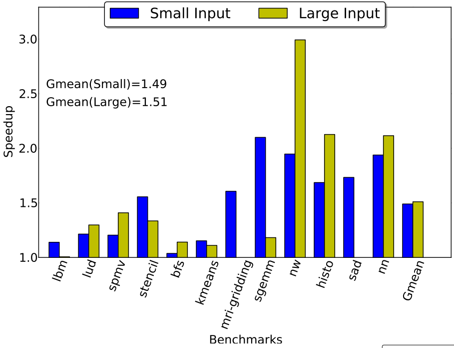
Evaluation with **12 optimization eligible** benchmarks.

Use of the smallest and largest available datasets.

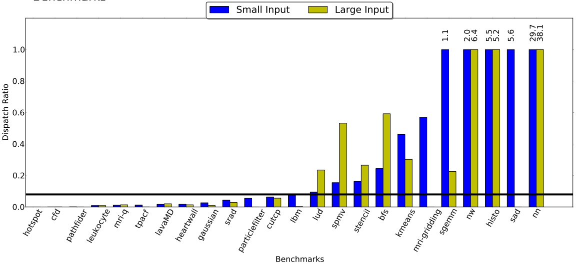
Performance Evaluation on GTX Platform



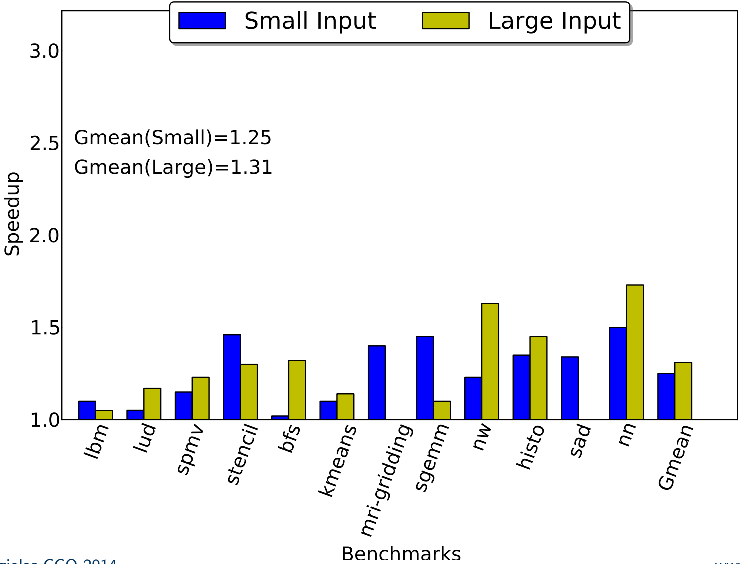
Performance Evaluation on GTX Platform



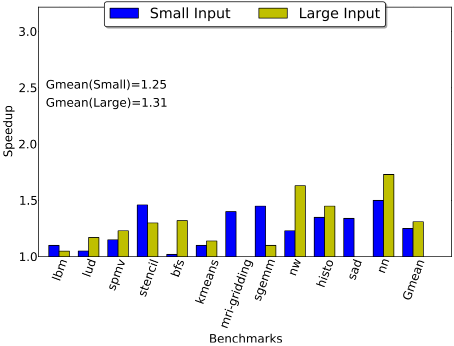
- Speedups from 1.05x to 3.0x.
- Gmean speedup remains roughly stable across datasets.
- Speedup gains proportional to dispatch ratio.



Performance Evaluation on AMD Platform



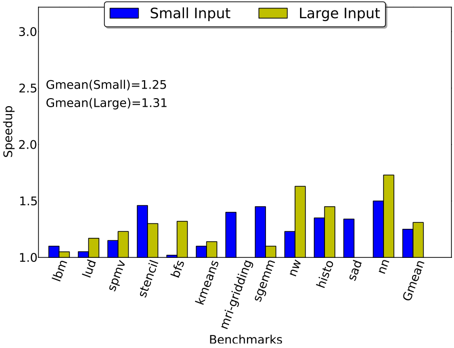
Performance Evaluation on AMD Platform



- Speedups ranging from 1.05x to 1.7x.
- Similar gmean speedup for both datasets.

AMD Platform presents lower speedups than the NVIDIA one.
Two are the main reasons:

Performance Evaluation on AMD Platform

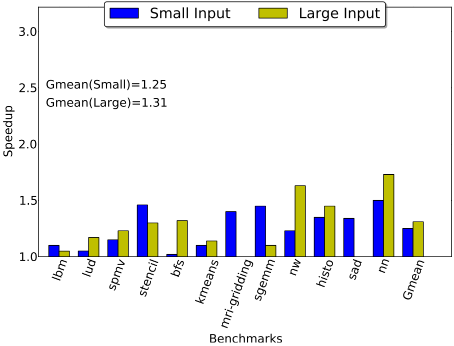


- Speedups ranging from 1.05x to 1.7x.
- Similar gmean speedup for both datasets.

AMD Platform presents lower speedups than the NVIDIA one. Two are the main reasons:

- AMD OpenCL uses intermediate data buffers allocated with special policies as part of its implementation.

Performance Evaluation on AMD Platform

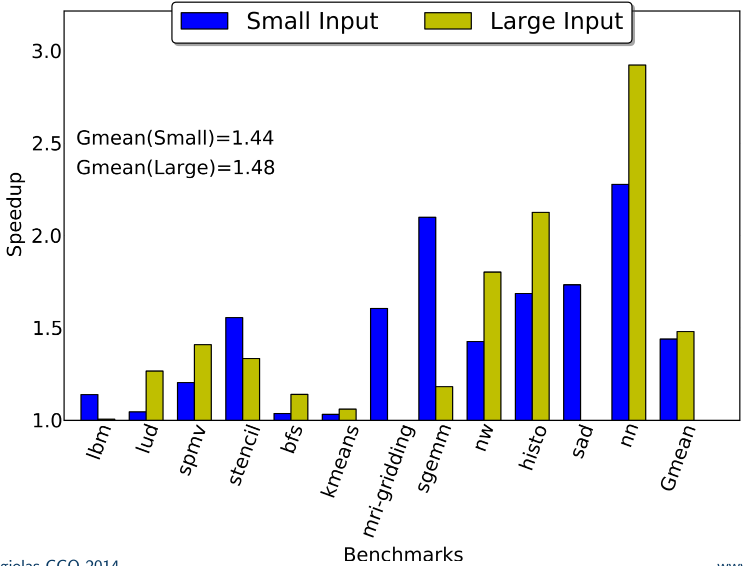


- Speedups ranging from 1.05x to 1.7x.
- Similar gmean speedup for both datasets.

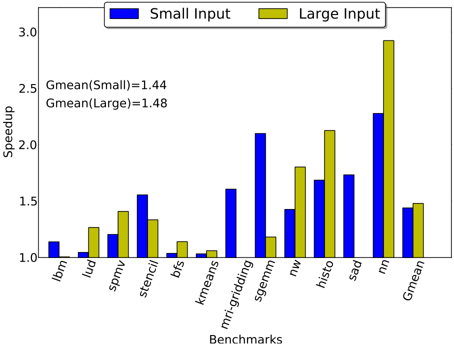
AMD Platform presents lower speedups than the NVIDIA one. Two are the main reasons:

- AMD OpenCL uses intermediate data buffers allocated with special policies as part of its implementation.
- AMD OpenCL restricts allocations with memory locking to low sizes.

Performance Evaluation on K20 Platform

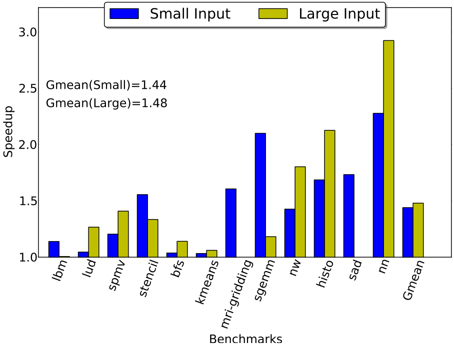


Performance Evaluation on K20 Platform



- Speedups from 1.1x to 2.9x.
- Gmean speedup roughly stable for both datasets.

Performance Evaluation on K20 Platform



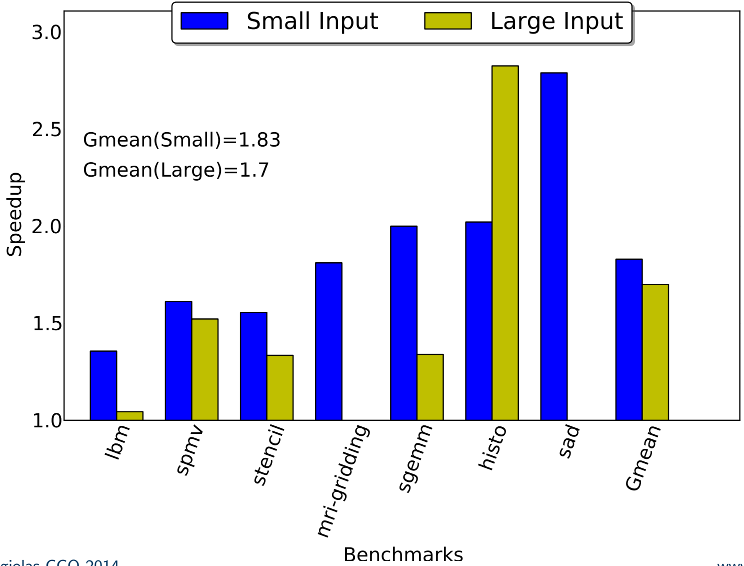
- Speedups from 1.1x to 2.9x.
- Gmean speedup roughly stable for both datasets.

The performance gains are similar to the one of GTX platform. We notice only significant difference for **nn** and **nw** benchmarks.

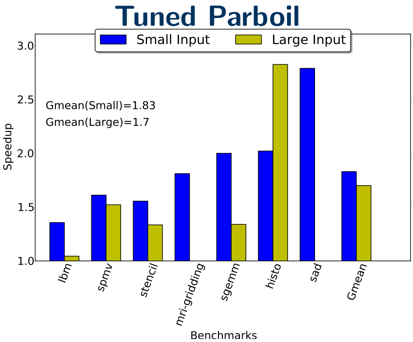
Performance Evaluation with Tuned Benchmarks

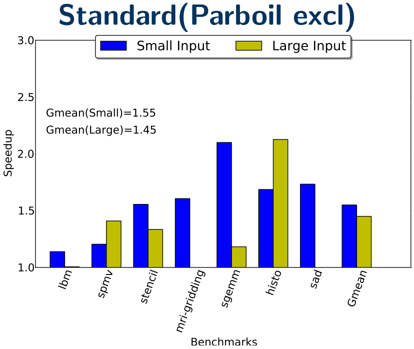
- Parboil provides tuned versions of its benchmarks for NVIDIA.
- The benchmarks now have faster kernels for NVIDIA GPUs.
- We evaluate our optimization on GTX Platform.

Performance Evaluation on GTX Platform (Tuned Parboil)



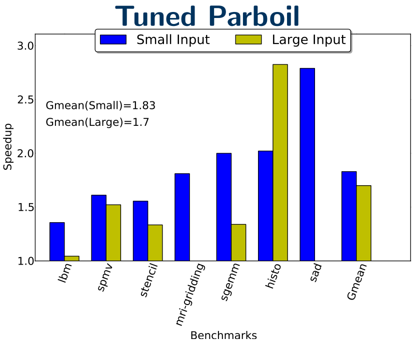
Performance Evaluation on GTX Platform (Tuned Parboil)

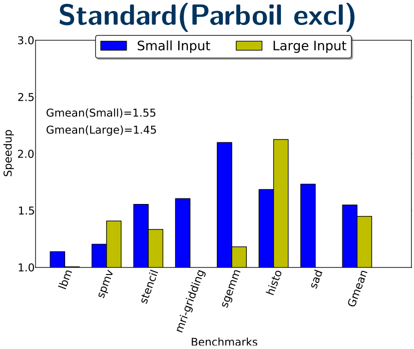




- Our optimization now delivers about 25% higher speedups.

Performance Evaluation on GTX Platform (Tuned Parboil)





- Our optimization now delivers about 25% higher speedups.

In fact, optimized kernels expose further the communication overhead and our optimization has additional effect.

Future Work

We consider future applications for our optimization. We focus on:

Future Work

We consider future applications for our optimization. We focus on:

- Virtually Unified Address Spaces

Future Work

We consider future applications for our optimization. We focus on:

- Virtually Unified Address Spaces
  - Data transfers still take place.

Future Work

We consider future applications for our optimization. We focus on:

- Virtually Unified Address Spaces
    - Data transfers still take place.
    - Need for Efficient Data-Prefetching.

Future Work

We consider future applications for our optimization. We focus on:

- Virtually Unified Address Spaces
  - Data transfers still take place.
  - Need for Efficient Data-Prefetching.
- HSA Architecture

Future Work

We consider future applications for our optimization. We focus on:

- Virtually Unified Address Spaces
  - Data transfers still take place.
  - Need for Efficient Data-Prefetching.
- HSA Architecture
  - CPU/GPU cores share single memory.

Future Work

We consider future applications for our optimization. We focus on:

- Virtually Unified Address Spaces
  - Data transfers still take place.
  - Need for Efficient Data-Prefetching.
- HSA Architecture
  - CPU/GPU cores share single memory.
  - Need for data placement that reduces memory contention.

Future Work

We consider future applications for our optimization. We focus on:

- Virtually Unified Address Spaces
  - Data transfers still take place.
  - Need for Efficient Data-Prefetching.
- HSA Architecture
  - CPU/GPU cores share single memory.
  - Need for data placement that reduces memory contention.
  - Need for efficent Memory Allocation.

Future Work

We consider future applications for our optimization. We focus on:

- Virtually Unified Address Spaces
  - Data transfers still take place.
  - Need for Efficient Data-Prefetching.
- HSA Architecture
  - CPU/GPU cores share single memory.
  - Need for data placement that reduces memory contention.
  - Need for efficent Memory Allocation.
  - Special Memory Allocation policies targeting data placement.

Summary

We build a host-device communication optimization for GPGPU environments which:

Summary

We build a host-device communication optimization for GPGPU environments which:

- Leads to significant speedups (1.51x, 1.31x, 1.48x)

Summary

We build a host-device communication optimization for GPGPU environments which:

- Leads to significant speedups (1.51x, 1.31x, 1.48x)
- Is portable across platforms.

Summary

We build a host-device communication optimization for GPGPU environments which:

- Leads to significant speedups (1.51x, 1.31x, 1.48x)
- Is portable across platforms.
- Is transparent to applications, Runtime and OS.

Summary

We build a host-device communication optimization for GPGPU environments which:

- Leads to significant speedups (1.51x, 1.31x, 1.48x)
- Is portable across platforms.
- Is transparent to applications, Runtime and OS.
- Automatically detects platform capabilities and application behavior.

Summary

We build a host-device communication optimization for GPGPU environments which:

- Leads to significant speedups (1.51x, 1.31x, 1.48x)
- Is portable across platforms.
- Is transparent to applications, Runtime and OS.
- Automatically detects platform capabilities and application behavior.
- Requires no application code modification or recompilation.

Thank you!

Special Forces (1) / Tracing

Code Sample

```
segm=malloc(SIZE);
clSetKernelArg(kernel, ...., &buf);
for(i=0; i<n; i++)
{
   do_smth(&segm);
   clEnqueueWriteBuffer(..., buf, segm, ...);
   clEnqueueNDRangeKernel(..., kernel, ...);
   clEnqueueReadBuffer(..., buf, ..., segm);
}
```

Special Forces (2) / Tracing

Performed Function Calls

Call	Def	Use
malloc	s0	
karg	k1	b0,k0
wbuffer	b1	b0,s0,q0
kexec	b2	b1,k1,q0
rbuffer	s1	b2,s0,q0
wbuffer	b3	b2,s1,q0
kexec	b4	b3,k1,q0
rbuffer	s2	b4,s0,q0
...	n-3 loop iterations	...
wbuffer	b2n+1	b2n,sn,q0
kexec	b2n+2	b2n+1,k1,q0
rbuffer	sn+1	b2n+2,s0,q0

Compressed Call Trace

Call	Def	Use
malloc	s0	
karg	k1	b0,k0
wbuffer(#n)	b1	b0,s0,q0
kexec(#n)	b2	b1,k1,q0
rbuffer(#n)	s1	b2,s0,q0

Special Forces (3) / Analysis Algorithm

Allocation Detection Algorithm

```
for each c in the Call Trace
 if c is a host-device communication that involves a memory segment s
  retrieve s', the first state of s (through SSA)
  retrieve co, the creator (allocation call) of s'
  annotate co as optimization candidate
```

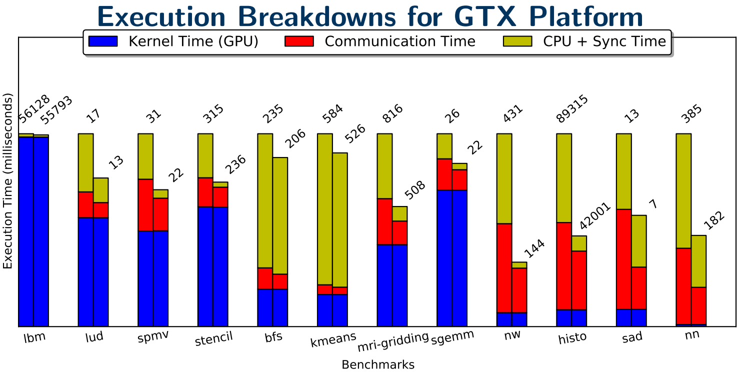
Special Forces (3) / Analysis Algorithm

Allocation Detection Algorithm

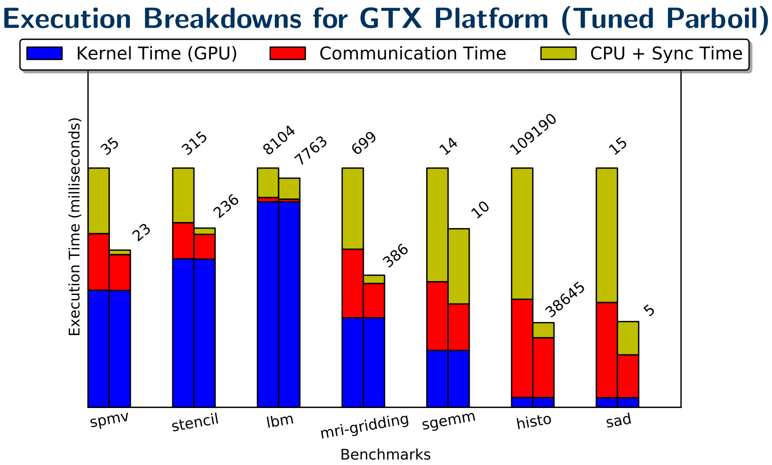
```
for each c in the Call Trace
 if c is a host-device communication that involves a memory segment s
  retrieve s', the first state of s (through SSA)
  retrieve co, the creator (allocation call) of s'
  annotate co as optimization candidate
```

- The compressed trace is given as input.
- It detects the memory allocations that are used for host-device communication.
- It outputs the annotated allocations for the runtime optimizer.

Special Forces (4) / Execution Breakdowns



Special Forces (5) / Execution Breakdowns



Special Forces (6) / Tuned Parboil Ratio Comp