Portable and Transparent Host-Device Communication Optimization for GPGPU Environments

Christos Margiolas, Michael F.P. O’Boyle

University of Edinburgh
Institute for Computing Systems Architecture

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 requires:
• H2D Transfer of Input Data
• D2H Transfer of Output Data

Communication Overhead

What is the impact of communication overhead on application execution?
Computation Offloading

Why Host-Device Communication is required?

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

Communication Overhead

What is the impact of communication overhead on application execution?
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?
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.

Christos Margiolas CGO 2014

www.inf.ed.ac.uk
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?

Christos Margiolas CGO 2014
Memory Allocation affects Communication Performance

**Communication in Standard Environment**

- Host/Device
- Device/Host
- Memory Pages
- Multiple DMA Transfers

**Standard Allocation:**
- Mem. Pages Swappable
- Transfer per page
- Unsteady Performance

**Issue 1: Platform capabilities**

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

**Communication in Standard Environment**
- Host/Device
  - Memory Pages
  - Multiple DMA Transfers
- Device/Host

**Communication in Enhanced Environment**
- Host/Device
  - Memory Pages
  - Reduced DMA Transfers
- Device/Host

**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);

• 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.
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

- Dozens of memory allocations performed by an application.

- Only few are used for Host-Device Communication.

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

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

Issue 3: Portability and Transparency
Need for optimizations portable across platforms and transparent to applications and runtime libraries.

Christos Margiolas CGO 2014
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.
• **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.

---

**Optimization Overview**

**Off-Line Characterizations**

- **Platform Characterization**
  - Micro-Benchmarking
  - Performance Statistics
  - Platf. Analysis/ Curve Fitting
  - Perf. Estimation Functions
  - Mem. Allocation Policies

- **Application Characterization**
  - Application Tracing
  - Call Trace/ Perf. Stats
  - Application Analysis
  - Optimization Directives

---

**Runtime Optimization**

- **OpenCL Application**
- **Enhanced Execution Environment**
• **Platform Characterization** discovers the memory allocation and host-device communication capabilities of the platform.
• **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.
• **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.
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.
Two micro-benchmarks provide allocation and communication overhead statistics.
Two micro-benchmarks provide allocation and communication overhead statistics.

We consider the following memory allocation policies:
Two micro-benchmarks provide allocation and communication overhead statistics.
We consider the following memory allocation policies:

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

Allocation Overhead

- Std
- Std(Lock)
- OpenCL
- Hybrid

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);
Remark:
Policies with high allocation overhead lead to low communication overhead. Curve fitting is performed on the collected statistics and generates performance estimation functions.
Remark: Policies with high allocation overhead lead to low communication overhead.
Remark: Policies with high allocation overhead lead to low communication overhead.
Curve fitting is performed on the collected statistics and generates performance estimation 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 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 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 tracing generates a **Compressed Trace** with:

- Every call to **OpenCL** and **memory allocation** functions.
Application tracing generates a **Compressed Trace** with:

- Every call to **OpenCL** and **memory allocation** functions.
- **Dependences** between the calls and data objects.
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 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 Call Flow**
  - Read Arguments
  - Function Call
  - Write Arguments
  - OpenCL/Mem. Allocation Library

- 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.
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.
• Tracing is performed via a wrapping library.
• 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.
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.
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:

\[
\text{Dispatch Ratio} \geq 0.1
\]
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

Application Call Flow

Read Arguments

Function Call

Write Arguments

OpenCL/ Mem. Allocation Library

• 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.
Runtime Optimization

Standard Function Call

Application Call Flow

Read Arguments

Function Call

Write Arguments

OpenCL/Mem. Allocation Library

Call with Optimization

Application Call Flow

Read Arguments

Function Call

Write Arguments

Wrapping Library

Read Arguments Monitoring

Call Forwarding

Write Arguments Monitoring

Default Functions

Redirection

Allocation Manager

• 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.
Runtime Optimization

Standard Function Call

Call with Optimization

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

Standard Function Call

- Read Arguments
- Function Call
- Write Arguments

OpenCL/ Mem. Allocation Library

Application Call Flow

Call with Optimization

- Read Arguments
- Function Call
- Write Arguments

Wrapping Library
- Read Arguments Monitoring
- Call Forwarding
- Write Arguments Monitoring

application

Default Functions
- Redirection
- Allocation Manager

Application Call Flow

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

Christos Margiolas CGO 2014
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

Gmean(Small)=1.49
Gmean(Large)=1.51
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.

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

Gmean(Small) = 1.25
Gmean(Large) = 1.31
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 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

Gmean(Small) = 1.44
Gmean(Large) = 1.48

Benchmarks:
- lbm
- lud
- spmv
- stencil
- bfs
- kmeans
- mri-gridding
- sgemm
- nw
- histo
- sad
- nn
- Gmean

Speedup

Small Input
Large Input
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)

- Increased speedups ranging from 1.1x to 2.8x.
- Gmean speedup has increased, 1.83x and 1.7x.

Gmean(Small) = 1.83
Gmean(Large) = 1.7
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:

• 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 efficient Memory Allocation.
• Special Memory Allocation policies targeting data placement.
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 efficient Memory Allocation.
  - Special Memory Allocation policies targeting data placement.
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.
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.
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

```c
1 segm = malloc(SIZE);
2 clSetKernelArg(kernel, ...., &buf);
3 for (i = 0; i < n; i++)
4 {
5     do_smth(&segm);
6     clEnqueueWriteBuffer(..., buf, segm, ...);
7     clEnqueueNDRangeKernel(..., kernel, ...);
8     clEnqueueReadBuffer(..., buf, ..., segm);
9 }
```
## Performed Function Calls

<table>
<thead>
<tr>
<th>Call</th>
<th>Def</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc</td>
<td>s0</td>
<td></td>
</tr>
<tr>
<td>karg</td>
<td>k1</td>
<td>b0,k0</td>
</tr>
<tr>
<td>wbuffer</td>
<td>b1</td>
<td>b0,s0,q0</td>
</tr>
<tr>
<td>kexec</td>
<td>b2</td>
<td>b1,k1,q0</td>
</tr>
<tr>
<td>rbuffer</td>
<td>s1</td>
<td>b2,s0,q0</td>
</tr>
<tr>
<td>wbuffer</td>
<td>b3</td>
<td>b2,s1,q0</td>
</tr>
<tr>
<td>kexec</td>
<td>b4</td>
<td>b3,k1,q0</td>
</tr>
<tr>
<td>rbuffer</td>
<td>s2</td>
<td>b4,s0,q0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>wbuffer</td>
<td>b2n+1</td>
<td>b2n,q0</td>
</tr>
<tr>
<td>kexec</td>
<td>b2n+2</td>
<td>b2n+1,k1,q0</td>
</tr>
<tr>
<td>rbuffer</td>
<td>sn+1</td>
<td>b2n+2,q0</td>
</tr>
</tbody>
</table>

## Compressed Call Trace

<table>
<thead>
<tr>
<th>Call</th>
<th>Def</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc</td>
<td>s0</td>
<td></td>
</tr>
<tr>
<td>karg</td>
<td>k1</td>
<td>b0,k0</td>
</tr>
<tr>
<td>wbuffer(#n)</td>
<td>b1</td>
<td>b0,s0,q0</td>
</tr>
<tr>
<td>kexec(#n)</td>
<td>b2</td>
<td>b1,k1,q0</td>
</tr>
<tr>
<td>rbuffer(#n)</td>
<td>s1</td>
<td>b2,s0,q0</td>
</tr>
</tbody>
</table>
Special Forces (3) / Analysis Algorithm

Allocation Detection Algorithm

1. for each c in the Call Trace
2. if c is a host-device communication that involves a memory segment s
3. retrieve s’, the first state of s (through SSA)
4. retrieve co, the creator (allocation call) of s’
5. annotate co as optimization candidate
### Allocation Detection Algorithm

```
1 for each c in the Call Trace
2 if c is a host-device communication that involves a memory segment s
3 retrieve s', the first state of s (through SSA)
4 retrieve co, the creator (allocation call) of s'
5 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.
Execution Breakdowns for GTX Platform

- **Kernel Time (GPU)**
- **Communication Time**
- **CPU + Sync Time**

**Benchmarks**
- `lbm`
- `lud`
- `spmv`
- `stencil`
- `bfs`
- `kmeans`
- `mri-gridding`
- `sgemm`
- `nw`
- `histo`
- `sad`
- `nn`

**Execution Time (milliseconds)**

- `56128`
- `17`
- `31`
- `315`
- `235`
- `584`
- `816`
- `26`
- `431`
- `89315`
- `13`
- `385`
- `55793`
- `13`
- `22`
- `236`
- `206`
- `526`
- `508`
- `22`
- `144`
- `42001`
- `7`
- `182`

**Execution Time (milliseconds) for Benchmarks**

- `56128`
- `17`
- `31`
- `315`
- `235`
- `584`
- `816`
- `26`
- `431`
- `89315`
- `13`
- `385`
- `55793`
- `13`
- `22`
- `236`
- `206`
- `526`
- `508`
- `22`
- `144`
- `42001`
- `7`
- `182`
Execution Breakdowns for GTX Platform (Tuned Parboil)

- spmv
- stencil
- lbm
- mri-gridding
- sgemm
- histo
- sad

Benchmarks

Execution Time (milliseconds)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>35</th>
<th>315</th>
<th>8104</th>
<th>699</th>
<th>14</th>
<th>109190</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>23</td>
<td>236</td>
<td>7763</td>
<td>386</td>
<td>10</td>
<td>38645</td>
<td>5</td>
</tr>
</tbody>
</table>

Kernel Time (GPU) Communication Time CPU + Sync Time
Special Forces (6) / Tuned Parboil Ratio Comp

**Tuned Parboil**

- **Small Input**
  - Spmv: 0.61
  - Stencil: 0.76
  - Lbm: 1.36
  - Mr-gridding: 1.22
  - Sgemm: 10.99
  - Histo: 9.97
  - Sad: 10.38

- **Large Input**

**Standard Parboil**

- **Small Input**
  - Lbm: 1.05
  - Spmv: 5.53
  - Stencil: 5.64
  - Mr-gridding: 5.2

- **Large Input**

---

Christos Margiolas CGO 2014