CONSEQUENCES OF BUFFER OVERFLOWS
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption
CONSEQUENCES OF BUFFER OVERFLOWS
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption
CONSEQUENCES OF BUFFER OVERFLOWS
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption
CONSEQUENCES OF BUFFER OVERFLOWS
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption

Segmentation Faults
CONSEQUENCES OF BUFFER OVERFLOWS
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption  |  Segmentation Faults

Segmentation Faults

Data Corruption
CONSEQUENCES OF BUFFER OVERFLOWS
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption

Segmentation Faults
CONSEQUENCES OF BUFFER OVERFLOWS
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption

Segmentation Faults

Altered Control Flow (Security Subversion)
CONSEQUENCES OF BUFFER OVERFLOWS
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption
Segmentation Faults
Altered Control Flow
(Security Subversion)
CONSEQUENCES OF BUFFER OVERFLOWS
DEGRADING USER EXPERIENCE, AND SECURITY RISKS

Data Corruption

Segmentation Faults

Altered Control Flow (Security Subversion)

Elegant 0-day unicorn underscores “serious concerns” about Linux security

Scriptless exploit bypasses state-of-the-art protections baked into the OS.

DAN GOODIN - 11/22/2016, 3:48 PM
BACKGROUND: NORMAL BUFFER FILL

\[
\begin{align*}
&\text{buf}[n+1] \\
&\text{memcpy}(\text{buf}, \text{src}, n+1)
\end{align*}
\]
BACKGROUND: NORMAL BUFFER FILL

\texttt{memcpy(buf, src, n+1)}

\begin{center}
\begin{tabular}{cccc}
buf & buf+1 & buf+2 & \ldots & buf+n & \\
\end{tabular}
\end{center}
BACKGROUND: BUFFER OVERFLOW

```c
buf[n+1]
memcpy(buf, src, n+5)
```

```
buf   buf+1   buf+2   ...   buf+n
```

```
return addr
```
BACKGROUND: BUFFER OVERFLOW

- buf[n+1]
- memcpy(buf, src, n+5)
BACKGROUND: BUFFER OVERFLOW

- buf[n+1]
- memcpy(buf, src, n+5)
Overflows on GPU can cause remote GPU code execution

- B. Di, J. Sun, and H. Chen. *A Study of Overflow Vulnerabilities on GPUs.*
Overflows on GPU can cause remote GPU code execution

- B. Di, J. Sun, and H. Chen. *A Study of Overflow Vulnerabilities on GPUs.*

![Diagram showing GPU buffers also overflow](image)
Overflows on GPU can cause remote GPU code execution

- B. Di, J. Sun, and H. Chen. *A Study of Overflow Vulnerabilities on GPUs.*
Overflows on GPU can cause remote GPU code execution

– A. Miele. *Buffer Overflow Vulnerabilities in CUDA: A Preliminary Analysis*.
Overflows on GPU can cause remote GPU code execution
- B. Di, J. Sun, and H. Chen. *A Study of Overflow Vulnerabilities on GPUs.*
GPU BUFFERS ALSO OVERFLOW

SHARED MEMORY CORRUPTION
GPU BUFFERS ALSO OVERFLOW

GPU can overflow buffers in system memory
– Over Interconnects like PCIe®
GPU can overflow buffers in system memory
– Over Interconnects like PCIe®
GPU can overflow buffers in system memory
– Over Interconnects like PCIe®

$\mathbf{x = y + z}$
GPU can overflow buffers in system memory
– Over Interconnects like PCIe®
GPU can overflow buffers in system memory
– Over Interconnects like PCIe®
GPU can overflow buffers in system memory
– Over Interconnects like PCIe®
GPU BUFFERS ALSO OVERFLOW

SHARED MEMORY CORRUPTION

GPU can overflow buffers in system memory
– Over Interconnects like PCIe®
GPU can overflow buffers in system memory

- Over Interconnects like PCIe®
GPU can overflow buffers in system memory
– Over Interconnects like PCIe®

```
assert(x == y + z)
```

**SYSTEM MEMORY**

- CPU
- GPU Buffer
- CPU Data

**GPU MEMORY**

- GPU
- PCIe
GPU buffers also overflow

- Shared Memory Corruption

GPU can overflow buffers in system memory
- Over Interconnects like PCIe®
GPU BUFFERS ALSO OVERFLOW

SHARED MEMORY CORRUPTION

CPU and GPU as part of the same package

Diagram showing CPU and GPU accessing shared memory, with the possibility of buffer overflow and shared memory corruption.
CPU and GPU as part of the same package

```plaintext
assert(x == y + z)
```

```
MEMORY

GPU Buffer

0
4
...
...n-3

CPU Data

x
y
z
```
CPU and GPU as part of the same package
- Every GPU buffer overflow may affect CPU data
GOALS
BUILDING CLARMOR

Software tool to detect buffer overflows caused by GPU

Runnable with most OpenCL™ applications

Low runtime overhead
GOALS
BUILDING CLARMOR

- Software tool to detect buffer overflows caused by GPU
  - clARMOR found 13 GPU buffer overflows in 7 programs

- Runnable with most OpenCL™ applications

- Low runtime overhead
GOALS
BUILDING CLARMOR

Software tool to detect buffer overflows caused by GPU
- clARMOR found 13 GPU buffer overflows in 7 programs

Runnable with most OpenCL™ applications
- Tested for GPU and CPU device types from multiple vendors

Low runtime overhead
GOALS
BUILDING CLARMOR

- Software tool to detect buffer overflows caused by GPU
  - clARMOR found 13 GPU buffer overflows in 7 programs

- Runnable with most OpenCL™ applications
  - Tested for GPU and CPU device types from multiple vendors

- Low runtime overhead
  - 14% overhead across 175 applications in 16 GPU benchmark suites
GOALS
BUILDING CLARMOR

Software tool to detect buffer overflows caused by GPU
– clARMOR found 13 GPU buffer overflows in 7 programs

Runnable with most OpenCL™ applications
– Tested for GPU and CPU device types from multiple vendors

Low runtime overhead
– 14% overhead across 175 applications in 16 GPU benchmark suites
GOALS
BUILDING CLARMOR

Software tool to detect buffer overflows caused by GPU
– clARMOR found 13 GPU buffer overflows in 7 programs

Runnable with most OpenCL™ applications
– Tested for GPU and CPU device types from multiple vendors

Low runtime overhead
– 14% overhead across 175 applications in 16 GPU benchmark suites
BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION
BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION
BUFFER OVERFLOW DETECTION METHODOLOGY

CANARY-BASED DETECTION

- Inserting known values around a protected region.
Inserting known values around a protected region.

- \texttt{buf[n+1]}
- \texttt{memcpy(buf, src, n+1)}
BUFFER OVERFLOW DETECTION METHODOLOGY
CANARY-BASED DETECTION

Inserting known values around a protected region.

buf[n+1]
memcpy(buf, src, n+1)
Inserting known values around a protected region.

- `buf[n+1]`
- `memcpy(buf, src, n+1)`

---

**BUFFER OVERFLOW DETECTION METHODOLOGY**

**CANARY-BASED DETECTION**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>buf+1</td>
<td>buf+2</td>
<td>...</td>
<td>buf+n</td>
<td>verify</td>
<td></td>
</tr>
</tbody>
</table>
BUFFER OVERFLOW DETECTION METHODOLOGY
CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+1)

![Diagram showing buffer and canary values]

```c
buf[n+1]
memcpy(buf, src, n+1)
```
BUFFER OVERFLOW DETECTION METHODOLOGY
CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+5)

```
buf
buf+1
buf+2
...  buf+n

... canary ... return addr
```
buffers[n+1]
memcpy(buf, src, n+5)

Inserting known values around a protected region.
BUFFER OVERFLOW DETECTION METHODOLOGY
CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+5)

```c
buf[n+1]
memcpy(buf, src, n+5)
```

overflow src[n+1] – [n+4]
BUFFER OVERFLOW DETECTION METHODOLOGY
CANARY-BASED DETECTION

- Inserting known values around a protected region.

- `buf[n+1]`
- `memcpy(buf, src, n+5)`

![Diagram showing buffer overflow detection methodology with canary values around a protected region.](image)
BUFFER OVERFLOW DETECTION METHODOLOGY
CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+5)

![Diagram showing buffer overflow detection methodology with canary and return address verification.](image)
Inserting known values around a protected region.

buf[n+1]

memcpy(buf, src, n+5)
BUFFER OVERFLOW DETECTION METHODOLOGY
CANARY-BASED DETECTION

- Inserting known values around a protected region.

- buf[n+1]
- memcpy(buf, src, n+5)

Absence of known canary values alerts to invalid writes.
GOALS
BUILDING clARMOR

Software tool to detect buffer overflows caused by GPU
– clARMOR found 13 GPU buffer overflows in 7 programs

Runnable with most OpenCL™ applications
– Tested for GPU and CPU device types from multiple vendors

Low runtime overhead
– 14% overhead across 175 applications in 16 GPU benchmark suites
LAUNCHING AN OPENCL™ KERNEL
LAUNCHING AN OPENCL™ KERNEL

Buffer Create

Buffer
LAUNCHING AN OPENCL™ KERNEL

Set Arguments

Buffer
Kernel
LAUNCHING AN OPENCL™ KERNEL

Set Arguments
LAUNCHING AN OPENCL™ KERNEL
LAUNCHING AN OPENCL™ KERNEL

Launch Kernel
LAUNCHING AN OPENCL™ KERNEL
LAUNCHING AN OPENCL™ WITH CLARMOR

Buffer Create

Buffer
LAUNCHING AN OPENCL™ WITH CLARMOR

Buffer Create

Buffer
Canary
LAUNCHING AN OPENCL™ WITH CLARMOR

Buffer Create

Buffer Metadata

Buffer Canary
LAUNCHING AN OPENCL™ WITH CLARMOR

Set Arguments
- Buffer Metadata

Kernel Information
- Buffer
- Canary

Kernel
Set Arguments

Kernel Information

Buffer Metadata

Buffer
Canary

Kernel
LAUNCHING AN OPENCL™ WITH CLARMOR

Set Arguments

Kernel Information

Buffer Metadata

Kernel

Buffer

Canary
LAUNCHING AN OPENCL™ WITH CLARMOR

Kernel Information

Buffer Metadata

Launch Kernel

Kernel

Buffer

Canary
LAUNCHING AN OPENCL™ WITH CLARMOR

Kernel Information
Buffer Metadata
Launch Kernel
LAUNCHING AN OPENCL™ WITH CLARMOR
LAUNCHING AN OPENCL™ WITH CLARMOR

Kernel Information

Buffer Metadata

Canary Verification

Buffer  Canary
LAUNCHING AN OPENCL™ WITH CLARMOR

Kernel Information

Buffer Metadata

Canary Verification

Buffer  Canary
LAUNCHING AN OPENCL™ WITH CLARMOR

Kernel Information
Buffer Metadata
Canary Verification
LAUNCHING AN OPENCL™ WITH CLARMOR

Kernel Information

Buffer Metadata

Canary Verification

Canary
LAUNCHING AN OPENCL™ WITH CLARMOR

Kernel Information

Buffer Metadata

Canary Verification
cLARMOR is a Linux® library that uses LD_PRELOAD to wrap OpenCL™ library calls

- Call Wrapping
  - Buffer and Image creates
  - Argument setters
  - Kernel launches
  - Information functions
cIARMOR is a Linux® library that uses LD_PRELOAD to wrap OpenCL™ library calls

Call Wrapping
- Buffer and Image creates
- Argument setters
- Kernel launches
- Information functions
WRAPPING THE OPENCL™ API
BUFFER AND IMAGE CREATION

- Attach canaries to memory objects

- Buffer Creation
  - Calls to `clCreateBuffer`, `clCreateSubBuffer` or `clSVMAlloc`
  - Increase space requested, fill end with canary
Attach canaries to memory objects

Buffer Creation
- Calls to `clCreateBuffer`, `clCreateSubBuffer` or `clSVMAlloc`
- Increase space requested, fill end with canary
WRAPPING THE OPENCL™ API
BUFFER AND IMAGE CREATION

Attach canaries to memory objects

- **Buffer Creation**
  - Calls to `clCreateBuffer`, `clCreateSubBuffer` or `clSVMAlloc`
  - Increase space requested, fill end with canary

- **Image Creation**
  - Calls to `clCreateImage`, `clCreateImage2D`, or `clCreateImage3D`
  - Potential for multi dimensional overflow
  - Add canary regions to each dimension
WRAPPING THE OPENCL™ API

BUFFER AND IMAGE CREATION

Attach canaries to memory objects

Buffer Creation
- Calls to `clCreateBuffer`, `clCreateSubBuffer` or `clSVMAlloc`
  - Increase space requested, fill end with canary

Image Creation
- Calls to `clCreateImage`, `clCreateImage2D`, or `clCreateImage3D`
  - Potential for multi dimensional overflow
  - Add canary regions to each dimension
## WRAPPING THE OPENCL™ API

### BUFFER AND IMAGE CREATION

- **Attach canaries to memory objects**

- **Buffer Creation**
  - Calls to `clCreateBuffer`, `clCreateSubBuffer` or `clSVMAlloc`
  - Increase space requested, fill end with canary

- **Image Creation**
  - Calls to `clCreateImage`, `clCreateImage2D`, or `clCreateImage3D`
  - Potential for multi dimensional overflow
  - Add canary regions to each dimension

---

<table>
<thead>
<tr>
<th>Buffer</th>
<th>Canary</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Image Row0</th>
<th>Canary</th>
<th></th>
<th>Canary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canary</td>
<td></td>
<td>0</td>
<td>Canary</td>
</tr>
<tr>
<td>Canary</td>
<td></td>
<td>1</td>
<td>Canary</td>
</tr>
<tr>
<td>Canary</td>
<td></td>
<td>2</td>
<td>Canary</td>
</tr>
<tr>
<td>Canary Row</td>
<td></td>
<td></td>
<td>Canary</td>
</tr>
</tbody>
</table>

---

---

---
Attach canaries to memory objects

Buffer Creation
- Calls to `clCreateBuffer`, `clCreateSubBuffer` or `clSVMAlloc`
- Increase space requested, fill end with canary

Image Creation
- Calls to `clCreateImage`, `clCreateImage2D`, or `clCreateImage3D`
- Potential for multi dimensional overflow
- Add canary regions to each dimension
OpenCL allows buffer creation using an existing memory allocation.
OpenCL allows buffer creation using an existing memory allocation.

Make this a buffer.

Data Array
OpenCL allows buffer creation using an existing memory allocation

Make this a buffer.

Adjacent Memory  Data Array  Adjacent Memory
OpenCL allows buffer creation using an existing memory allocation
– Cannot extend buffer
OpenCL allows buffer creation using an existing memory allocation
– Cannot extend buffer

![Diagram](image-url)
OpenCL allows buffer creation using an existing memory allocation

- Cannot extend buffer
- Cannot move buffer
OpenCL allows buffer creation using an existing memory allocation

- Cannot extend buffer
- Cannot move buffer
OpenCL allows buffer creation using an existing memory allocation

- Cannot extend buffer
- Cannot move buffer
OpenCL allows buffer creation using an existing memory allocation
– Cannot extend buffer
– Cannot move buffer
– Work around by using a temporary copy at run time
OpenCL allows buffer creation using an existing memory allocation
– Cannot extend buffer
– Cannot move buffer
– Work around by using a temporary copy at run time
WRAPPING THE OPENCL™ API

SET ARGUMENTS

- clARMOR needs to know which buffers/images to check for overflows
- Kernel information object
  - map kernel argument number to buffer information
- Update on call to `clSetKernelArg` or `clSetKernelArgSVMPointer`
WRAPPING THE OPENCL™ API

SET ARGUMENTS

- clARMOR needs to know which buffers/images to check for overflows
- Kernel information object
  - map kernel argument number to buffer information
- Update on call to `clSetKernelArg` or `clSetKernelArgSVMPointer`
WRAPPING THE OPENCL™ API

SET ARGUMENTS

- clARMOR needs to know which buffers/images to check for overflows
- Kernel information object
  - map kernel argument number to buffer information
- Update on call to `clSetKernelArg` or `clSetKernelArgSVMPointer`
WRAPPING THE OPENCL™ API

KERNEL LAUNCH

- Do the work of detecting buffer overflows
- On call to `clEnqueueNDRangeKernel`
  - Enqueue the kernel
  - Retrieve affected buffers
  - Run the canary check
  - Report errors
WRAPPING THE OPENCL™ API

KERNEL LAUNCH

- Do the work of detecting buffer overflows
- On call to `clEnqueueNDRangeKernel`
  - Enqueue the kernel
  - Retrieve affected buffers
  - Run the canary check
  - Report errors
WRAPPING THE OPENCL™ API

KERNEL LAUNCH

- Do the work of detecting buffer overflows
- On call to `clEnqueueNDRangeKernel`
  - Enqueue the kernel
  - Retrieve affected buffers
  - Run the canary check
  - Report errors
WRAPPING THE OPENCL™ API

Do the work of detecting buffer overflows
On call to `clEnqueueNDRangeKernel`

- Enqueue the kernel
- Retrieve affected buffers
- Run the canary check
- Report errors
GOALS

BUILDING clARMOR

Software tool to detect buffer overflows caused by GPU
  –clARMOR found 13 GPU buffer overflows in 7 programs

Runnable with most OpenCL™ applications
  –Tested for GPU and CPU device types from multiple vendors

Low runtime overhead
  –14% overhead across 175 applications in 16 GPU benchmark suites
ACCELERATION
SELECTING A DEVICE FOR PERFORMING CANARY VERIFICATION

CPU is faster
– small / few canary regions (latency advantage)

GPU is faster
– large / many canary regions (throughput advantage with embarrassingly parallel workload)
– reduced transfers over PCIe® by keeping on GPU
ACCELERATION
SELECTING A DEVICE FOR PERFORMING CANARY VERIFICATION

CPU is faster
- small / few canary regions (latency advantage)

GPU is faster
- large / many canary regions (throughput advantage with embarrassingly parallel workload)
- reduced transfers over PCIe® by keeping on GPU
Maximizing asynchrony

- Event-based programming wherever possible
- GPU check kernels enqueue behind work kernels and wait on completion
- Evaluation of check kernel results is done with call-backs
Maximizing asynchrony

- Event-based programming wherever possible
- GPU check kernels enqueue behind work kernels and wait on completion
- Evaluation of check kernel results is done with call-backs
Maximizing asynchrony

- Event-based programming wherever possible
- GPU check kernels enqueue behind work kernels and wait on completion
- Evaluation of check kernel results is done with call-backs
Maximizing asynchrony

- Event-based programming wherever possible
- GPU check kernels enqueue behind work kernels and wait on completion
- Evaluation of check kernel results is done with call-backs
TEST SETUP
HARDWARE SPECIFICATIONS AND BENCHMARKS SUITES

▲ 3.7 GHz AMD A10-7850K CPU
  – 32 GB of DDR3-1866
▲ AMD FirePro™ W9100 discrete GPU
  – 930 MHz core frequency
  – 320 GB/s of memory bandwidth
  – 16 GB of GDDR5 memory
▲ 3rd Generation PCIe® x8 CPU–GPU connection
▲ 175 benchmarks in 16 benchmark suites

<table>
<thead>
<tr>
<th>Suite</th>
<th>Num Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMDAPP</td>
<td>46</td>
</tr>
<tr>
<td>FINANCEBENCH</td>
<td>2</td>
</tr>
<tr>
<td>GPUESTREAM</td>
<td>1</td>
</tr>
<tr>
<td>HETEROMARK</td>
<td>14</td>
</tr>
<tr>
<td>MANTEVO</td>
<td>4</td>
</tr>
<tr>
<td>NPB_OCL</td>
<td>8</td>
</tr>
<tr>
<td>OPENDWARFS</td>
<td>7</td>
</tr>
<tr>
<td>PANNOTIA</td>
<td>6</td>
</tr>
<tr>
<td>PARBOIL</td>
<td>9</td>
</tr>
<tr>
<td>PHORONIX</td>
<td>4</td>
</tr>
<tr>
<td>POLYBENCH</td>
<td>21</td>
</tr>
<tr>
<td>PROXYAPPS</td>
<td>6</td>
</tr>
<tr>
<td>RODINIA</td>
<td>21</td>
</tr>
<tr>
<td>SHOC</td>
<td>14</td>
</tr>
<tr>
<td>STREAMMR</td>
<td>4</td>
</tr>
<tr>
<td>VIENNACL</td>
<td>8</td>
</tr>
</tbody>
</table>
TEST SETUP

HARDWARE SPECIFICATIONS AND BENCHMARKS SUITES

- 3.7 GHz AMD A10-7850K CPU
  - 32 GB of DDR3-1866
- AMD FirePro™ W9100 discrete GPU
  - 930 MHz core frequency
  - 320 GB/s of memory bandwidth
  - 16 GB of GDDR5 memory
- 3rd Generation PCIe® x8 CPU–GPU connection

- 175 benchmarks in 16 benchmark suites

<table>
<thead>
<tr>
<th>Suite</th>
<th>Num Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMDAPP</td>
<td>46</td>
</tr>
<tr>
<td>FINANCEBENCH</td>
<td>2</td>
</tr>
<tr>
<td>GPUSTREAM</td>
<td>1</td>
</tr>
<tr>
<td>HETEROMARK</td>
<td>14</td>
</tr>
<tr>
<td>MANTEVO</td>
<td>4</td>
</tr>
<tr>
<td>NPB_OCL</td>
<td>8</td>
</tr>
<tr>
<td>OPENDWARFS</td>
<td>7</td>
</tr>
<tr>
<td>PANNOTIA</td>
<td>6</td>
</tr>
<tr>
<td>PARBOIL</td>
<td>9</td>
</tr>
<tr>
<td>PHORONIX</td>
<td>4</td>
</tr>
<tr>
<td>POLYBENCH</td>
<td>21</td>
</tr>
<tr>
<td>PROXYAPPS</td>
<td>6</td>
</tr>
<tr>
<td>RODINIA</td>
<td>21</td>
</tr>
<tr>
<td>SHOC</td>
<td>14</td>
</tr>
<tr>
<td>STREAMMR</td>
<td>4</td>
</tr>
<tr>
<td>VIENNACL</td>
<td>8</td>
</tr>
</tbody>
</table>

Detect GPU Buffer Overflows
Compatible With Most OpenCL™
Low Runtime Overhead
TEST SETUP
HARDWARE SPECIFICATIONS AND BENCHMARKS SUITES

- 3.7 GHz AMD A10-7850K CPU
  - 32 GB of DDR3-1866
- AMD FirePro™ W9100 discrete GPU
  - 930 MHz core frequency
  - 320 GB/s of memory bandwidth
  - 16 GB of GDDR5 memory
- 3rd Generation PCIe® x8 CPU–GPU connection

- 175 benchmarks in 16 benchmark suites

<table>
<thead>
<tr>
<th>Suite</th>
<th>Num Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMDAPP</td>
<td>46</td>
</tr>
<tr>
<td>FINANCEBENCH</td>
<td>2</td>
</tr>
<tr>
<td>GPUSTREAM</td>
<td>1</td>
</tr>
<tr>
<td>HETEROMARK</td>
<td>14</td>
</tr>
<tr>
<td>MANTEVO</td>
<td>4</td>
</tr>
<tr>
<td>NPB_OCL</td>
<td>8</td>
</tr>
<tr>
<td>OPENDWARFS</td>
<td>7</td>
</tr>
<tr>
<td>PANNOTIA</td>
<td>6</td>
</tr>
<tr>
<td>PARBOIL</td>
<td>9</td>
</tr>
<tr>
<td>PHORONIX</td>
<td>4</td>
</tr>
<tr>
<td>POLYBENCH</td>
<td>21</td>
</tr>
<tr>
<td>PROXYAPPS</td>
<td>6</td>
</tr>
<tr>
<td>RODINIA</td>
<td>21</td>
</tr>
<tr>
<td>SHOC</td>
<td>14</td>
</tr>
<tr>
<td>STREAMMR</td>
<td>4</td>
</tr>
<tr>
<td>VIENNAACL</td>
<td>8</td>
</tr>
</tbody>
</table>

Detect GPU Buffer Overflows
Compatible With Most OpenCL™
Low Runtime Overhead
PERFORMANCE EVALUATION
APPLICATION RUNTIME: WITH / WITHOUT TOOL

Lower is better
PERFORMANCE EVALUATION
APPLICATION RUNTIME: WITH / WITHOUT TOOL

Lower is better
PERFORMANCE EVALUATION
APPLICATION RUNTIME: WITH / WITHOUT TOOL

Lower is better

Normalized Runtime

RODINIA  SHOC  PHORONIX  OPENDWARFS  PROXYAPPS  AMDAPP  PARBOIL  PANNOTIA  STREAMMR  POLYBENCH  FINANCEBENCH  GPUESTREAM  MANTEVO  HETEROMARK  VIENNACL  NPB_OCL  ALL

14%
PERFORMANCE EVALUATION
APPLICATION RUNTIME: WITH / WITHOUT TOOL

Lower is better

[Graph showing normalized runtime with various applications, indicating a 14% reduction with tool activated.]
PERFORMANCE EVALUATION
APPLICATION RUNTIME: WITH / WITHOUT TOOL

Lower is better

Normalized Runtime

RODINIA  SHOC  PHORONIX  OPENDWARFS  PROXYAPPS  AMDAPP  PARBOIL  PANNOTIA  STREAMMR  POLYBENCH  FINANCEBENCH  GPUSTREAM  MANTEVO  HETEROMARK  VIENNACL  NPB_OCL  ALL

Detect GPU Buffer Overflows
Compatible With Most OpenCL™
Low Runtime Overhead

14%
PERFORMANCE EVALUATION
APPLICATION RUNTIME: WITH / WITHOUT TOOL

Lower is better
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU

Application
Prelaunch

GPU
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU

Application Prelaunch  clARMOR Prelaunch

GPU

User Kernel

Launch Delay
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
- Application Prelaunch
- cIARMOR Prelaunch
- cIARMOR Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay

SNAP_MPI Synchronization
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
Application Prelaunch
clARMOR Prelaunch
clARMOR Postlaunch

GPU
User Kernel
Canary Check

Launch Delay

SNAP_MPI Synchronization
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU

Application Prelaunch | clARMOR Prelaunch | clARMOR Postlaunch

GPU

User Kernel | Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch

CPU
- Application Prelaunch

GPU
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch

GPU
- User Kernel

Launch Delay
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU

Application Prelaunch | clARMOR Prelaunch | clARMOR Postlaunch

GPU

User Kernel | Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch

CPU

Application Prelaunch | clARMOR Prelaunch | clARMOR Postlaunch | Application Postlaunch

GPU

User Kernel | Canary Check

Launch Delay
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch
- Application Postlaunch
- Application Prelaunch

GPU
- User Kernel
- Canary Check

Launch Delay
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch
- Application Postlaunch
- Application Prelaunch

GPU
- User Kernel
- Canary Check
- User Kernel

Launch Delay

Launch Delay
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch
- Application Postlaunch
- Application Prelaunch

GPU
- User Kernel
- Canary Check
- User Kernel

Launch Delay

Launch Delay
ANALYSIS OF TOOL OVERHEAD WITH SNAP_MPI

Example SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch
- Application Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch

CPU
- Application Prelaunch
- clARMOR Prelaunch
- clARMOR Postlaunch
- Application Postlaunch
- Application Prelaunch

GPU
- User Kernel
- Canary Check
- User Kernel

Launch Delay

Launch Delay
CLARMOR DETECTION RESULTS
LIST OF BENCHMARKS WITH BUFFER OVERFLOWS

- Parboil
  - mri-gridding

- StreamMR
  - kmeans
  - wordcount

- Hetero-Mark
  - OpenCL™ 1.2 kmeans
  - OpenCL 2.0 kmeans
  - OpenCL 1.2 sw, 4 errors
  - OpenCL 2.0 sw, 4 errors
CLARMOR DETECTION RESULTS
LIST OF BENCHMARKS WITH BUFFER OVERFLOWS

- Parboil
  - mri-gridding

- StreamMR
  - kmeans
  - wordcount

- Hetero-Mark
  - OpenCL™ 1.2 kmeans
  - OpenCL 2.0 kmeans
  - OpenCL 1.2 sw, 4 errors
  - OpenCL 2.0 sw, 4 errors

- Detect GPU Buffer Overflows
- Compatible With Most OpenCL™
- Low Runtime Overhead
CLARMOR DETECTION RESULTS
LIST OF BENCHMARKS WITH BUFFER OVERFLOWS

- Parboil
  - mri-gridding

- StreamMR
  - kmeans
  - wordcount

- Hetero-Mark
  - OpenCL™ 1.2 kmeans
  - OpenCL 2.0 kmeans
  - OpenCL 1.2 sw, 4 errors
  - OpenCL 2.0 sw, 4 errors

Detect GPU Buffer Overflows
Compatible With Most OpenCL™
Low Runtime Overhead
CLARMOR DETECTION RESULTS

LIST OF BENCHMARKS WITH BUFFER OVERFLOWS

- Parboil
  - mri-gridding

- StreamMR
  - kmeans
  - wordcount

- Hetero-Mark
  - OpenCL™ 1.2 kmeans
  - OpenCL 2.0 kmeans
  - OpenCL 1.2 sw, 4 errors
  - OpenCL 2.0 sw, 4 errors

Detect GPU Buffer Overflows
Compatible With Most OpenCL™
Low Runtime Overhead
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

**Kernel**

```c
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
}
```

**Host**

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_; ... 

cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
...

const size_t globalSize[2] = {m_len_, n_len_};
...

err |= clSetKernelArg(kernel_sw_compute0_, 6, sizeof(cl_mem),
    reinterpret_cast<void *>(&cu_));
...

err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);
```
Kernel

__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
) {
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}

Host

size_t sizeInBytes = sizeof(double) * m_len_ * n_len_
...
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
...
const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6, sizeof(cl_mem),
                     reinterpret_cast<void *>(&cu_));
...
earr = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```c
__kernel void sw_compute0(...)  
   const unsigned M_LEN,  
   __global double *cu,  
   ... ) {  
   int x = get_global_id(0);  
   int y = get_global_id(1);  
   cu[(y + 1) * M_LEN + x] = <input_equation>  
   ... }
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;  
...  
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,  
   sizeInBytes, NULL, &err);  
...  
const size_t globalSize[2] = {m_len_, n_len_};  
...  
err |= clSetKernelArg(kernel_sw_compute0_, 6,  
   sizeof(cl_mem),  
   reinterpret_cast<void *>(&cu_));  
...  
err = clEnqueueNDRangeKernel(cmdQueue_,  
   kernel_sw_compute0_, 2, NULL, globalSize,  
   localSize, 0, NULL, NULL);
```
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

__kernel void sw_compute0(…
    const unsigned M_LEN,
    …
    __global double *cu,
    …)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    …
}

Host

size_t sizeInBytes = sizeof(double) * m_len_ * n_len;
…
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
…
const size_t globalSize[2] = {m_len_, n_len_};
…
err |= clSetKernelArg(kernel_sw_compute0_, 6,
    sizeof(cl_mem),
    reinterpret_cast<void *>(&cu_));
…
err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize, localSize, 0, NULL, NULL);
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```c
__kernel void sw_compute0(...
  const unsigned M_LEN,
  ...
  __global double *cu,
  ...
) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
  ...
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len;
...
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
  sizeInBytes, NULL, &err);
...
const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6,
  sizeof(cl_mem),
  reinterpret_cast<void *>(&cu_));
...
err = clEnqueueNDRangeKernel(cmdQueue_,
  kernel_sw_compute0_, 2, NULL, globalSize,
  localSize, 0, NULL, NULL);
```
Kernel

```c
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
...
const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6, sizeof(cl_mem),
                       reinterpret_cast<void *>(&cu_));
...
err = clEnqueueNDRangeKernel(cmdQueue_,
                             kernel_sw_compute0_, 2, NULL, globalSize,
                             localSize, 0, NULL, NULL);
```
Kernel

__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}

Host

size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;  
...

cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,  
    sizeInBytes, NULL, &err);
...

const size_t globalSize[2] = {m_len_, n_len_};
...

err |= clSetKernelArg(kernel_sw_compute0_, 6,  
    sizeof(cl_mem),  
    reinterpret_cast<void *>(&cu_));
...

t = clEnqueueNDRangeKernel(cmdQueue_,  
    kernel_sw_compute0_, 2, NULL, globalSize,  
    localSize, 0, NULL, NULL);
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```c
__kernel void sw_compute0(
    ... const unsigned M_LEN,
    ... __global double *cu,
    ... ) {
int x = get_global_id(0);
int y = get_global_id(1);
cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
...
const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6,
    sizeof(cl_mem),
    reinterpret_cast<void *>(&cu_));
...
err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);
```
Kernel

```c
__kernel void sw_compute0(
    const unsigned M_LEN,
    ...__global double *cu,
    ... ) {
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;  
...  
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,  
    sizeInBytes, NULL, &err);  
...  
const size_t globalSize[2] = {m_len_, n_len_};  
...  
err |= clSetKernelArg(kernel_sw_compute0_, 6,  
    sizeof(cl_mem),  
    reinterpret_cast<void *>(cu_));  
...  
err = clEnqueueNDRangeKernel(cmdQueue_,  
    kernel_sw_compute0_, 2, NULL, globalSize,  
    localSize, 0, NULL, NULL);  
```
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```c
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
) {  
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;  
...

cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,  
    sizeInBytes, NULL, &err);
...

const size_t globalSize[2] = {m_len_, n_len_};
...

err |= clSetKernelArg(kernel_sw_compute0_, 6,  
    sizeof(cl_mem),  
    reinterpret_cast<void *>(&cu_));
...

err = clEnqueueNDRangeKernel(cmdQueue_,  
    kernel_sw_compute0_, 2, NULL, globalSize,  
    localSize, 0, NULL, NULL);
```
Kernel

__kernel void sw_compute0(...
  const unsigned M_LEN,
  ...
  __global double *cu,
  ...
) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
  ...
}

Host

size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...

   cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
...

   const size_t globalSize[2] = {m_len_, n_len_};
...

   err |= clSetKernelArg(kernel_sw_compute0_, 6, sizeof(cl_mem),
                         reinterpret_cast<void *>(&cu_));
...

   err = clEnqueueNDRangeKernel(cmdQueue_,
                                kernel_sw_compute0_, 2, NULL, globalSize,
                                localSize, 0, NULL, NULL);
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

**Kernel**

```c
__kernel void sw_compute0(...)  
    const unsigned M_LEN,  
    ...  
    __global double *cu,  
    ... ) {
    int x = get_global_id(0);  
    int y = get_global_id(1);  
    cu[(y + 1) * M_LEN + x] = <input_equation>  
    ...  
}
```

**Host**

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;  
...  
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,  
    sizeInBytes, NULL, &err);  
...  
const size_t globalSize[2] = {m_len_, n_len_};  
...  
err |= clSetKernelArg(kernel_sw_compute0_, 6,  
    sizeof(cl_mem),  
    reinterpret_cast<void *>(&cu_));  
...  
err = clEnqueueNDRangeKernel(cmdQueue_,  
    kernel_sw_compute0_, 2, NULL, globalSize,  
    localSize, 0, NULL, NULL);
```
Kernel

```c
__kernel void sw_compute0(...) {
    const unsigned M_LEN,
    ...__global double *cu,
    ...)
    {
        int x = get_global_id(0);
        int y = get_global_id(1);
        cu[(y + 1) * M_LEN + x] = <input_equation>
    }
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;... 
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, 
        sizeInBytes, NULL, &err);
        ... 
        const size_t globalSize[2] = {m_len_, n_len_};
        ... 
        err |= clSetKernelArg(kernel_sw_compute0_, 6, 
            sizeof(cl_mem),
            reinterpret_cast<void *>(&cu_));
        ...
        err = clEnqueueNDRangeKernel(cmdQueue_,
            kernel_sw_compute0_, 2, NULL, globalSize,
            localSize, 0, NULL, NULL);
```
Kernel

```c
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
) {
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...

cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
    sizeInBytes, NULL, &err);
...

const size_t globalSize[2] = {m_len_, n_len_};
...

err |= clSetKernelArg(kernel_sw_compute0_, 6,
    sizeof(cl_mem),
    reinterpret_cast<void *>(&cu_));
...

err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);
```
**Kernel**

```c
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
) {
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}
```

**Host**

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
...
const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6,
    sizeof(cl_mem),
    reinterpret_cast<void *>(&cu_));
...
err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);
```
Kernel

```c
__kernel void sw_compute0(
    ...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
) {

    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...

cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
...

const size_t globalSize[2] = {m_len_, n_len_};
...

err |= clSetKernelArg(kernel_sw_compute0_, 6, sizeof(cl_mem),
                        reinterpret_cast<void *>(&cu_));
...

er = clEnqueueNDRangeKernel(cmdQueue_,
                             kernel_sw_compute0_,
                             2, NULL,
                             globalSize,
                             localSize, 0, NULL, NULL);
```
**Kernel**

```c
__kernel void sw_compute0(...
  const unsigned M_LEN,  
  ...  
__global double *cu,
  ... ) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
  ...  
}
```

- `(y + 1) * M_LEN + x`
- `x = m - 1`
- `y = n - 1`
- `m == M_LEN`

**Host**

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;  
...

cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,  
  sizeInBytes, NULL, &err);
...

const size_t globalSize[2] = {m_len_, n_len_};
...

err |= clSetKernelArg(kernel_sw_compute0_, 6,  
  sizeof(cl_mem),  
  reinterpret_cast<void *>(&cu_));
...

err = clEnqueueNDRangeKernel(cmdQueue_,  
  kernel_sw_compute0_, 2, NULL, globalSize,  
  localSize, 0, NULL, NULL);
```
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```c
__kernel void sw_compute0(...) {
    const unsigned M_LEN,
        ... __global double *cu,
        ... ) { 
        int x = get_global_id(0);
        int y = get_global_id(1);
        cu[(y + 1) * M_LEN + x] = <input_equation>
        ... 
    } 

    (y + 1) * M_LEN + x
    (y + 1) * m + x
    x = m - 1
    y = n - 1
    m == M_LEN
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_; 
...

cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
...
const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6, sizeof(cl_mem), 
    reinterpret_cast<void *>(&cu_));
...
err = clEnqueueNDRangeKernel(cmdQueue_, 
    kernel_sw_compute0_, 2, NULL, globalSize, 
    localSize, 0, NULL, NULL);
```
Kernel

```c
__kernel void sw_compute0(...
    const unsigned M_LEN, 
    ...
    __global double *cu,
    ...
) {

    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>

}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_; 
...
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, 
    sizeInBytes, NULL, &err);
...
const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6, 
    sizeof(cl_mem), 
    reinterpret_cast<void *>(&cu_));
...
err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize, 
    localSize, 0, NULL, NULL);
```
Kernel

__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
) {

    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>

    ...
}

Host

size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;  
...

    cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,     
        sizeInBytes, NULL, &err);
...

    const size_t globalSize[2] = {m_len_, n_len_};
...

    err |= clSetKernelArg(kernel_sw_compute0_, 6,     
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
...

    err = clEnqueueNDRangeKernel(cmdQueue_,     
        kernel_sw_compute0_, 2, NULL, globalSize,     
        localSize, 0, NULL, NULL);
Kernel
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
    ) {
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}

Host
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE, sizeInBytes, NULL, &err);
...
const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6, sizeof(cl_mem),
    reinterpret_cast<void *>(&cu_));
...
err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

**Kernel**

```c
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
) {
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}
```

**Host**

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...

... cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
    sizeInBytes, NULL, &err);
...

const size_t globalSize[2] = {m_len_, n_len_};
...

err |= clSetKernelArg(kernel_sw_compute0_, 6,
    sizeof(cl_mem),
    reinterpret_cast<void *>(&cu_));
...

err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);
```
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```c
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len;
...
cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
                     sizeInBytes, NULL, &err);
...
const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6,
                      sizeof(cl_mem),
                      reinterpret_cast<void*>(&cu_));
...
err = clEnqueueNDRangeKernel(cmdQueue_,
                             kernel_sw_compute0_, 2, NULL, globalSize,
                             localSize, 0, NULL, NULL);
```

```c
(y + 1) * M_LEN + x
(y + 1) * m + x
(n) * m + x
n * m + m - 1
```

```
x = m - 1
y = n - 1
m == M_LEN
```
Kernel

```c
__kernel void sw_compute0(...
  const unsigned M_LEN,
  ...
  __global double *cu,
  ...
) {
  int x = get_global_id(0);
  int y = get_global_id(1);
  cu[(y + 1) * M_LEN + x] = <input_equation>
};
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...

const size_t globalSize[2] = {m_len_, n_len_};
...
err |= clSetKernelArg(kernel_sw_compute0_, 6,
  sizeof(cl_mem),
  reinterpret_cast<void *>(&cu_));
...
err = clEnqueueNDRangeKernel(cmdQueue_,
  kernel_sw_compute0_, 2, NULL, globalSize,
  localSize, 0, NULL, NULL);
```
Kernel

__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...)
{
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
}

(size_t sizeInBytes = sizeof(double) * m_len_ * n_len;
...

const size_t globalSize[2] = {m_len_, n_len_};
...

err |= clSetKernelArg(kernel_sw_compute0_, 6,
    sizeof(cl_mem),
    reinterpret_cast<void *>(&cu_));
...

err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);

(y + 1) * M_LEN + x
(y + 1) * m + x
(n) * m + x
n * m + m - 1
m * n - 1 + m > m * n - 1
Hetero-Mark OpenCL™ 1.2 SW Overflow Error

Kernel

```c
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...
) {
    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;  

...  

cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
    sizeInBytes, NULL, &err);

...  

const size_t globalSize[2] = {m_len_, n_len_};

...  

err |= clSetKernelArg(kernel_sw_compute0_, 6,
    sizeof(cl_mem),
    reinterpret_cast<void *>(&cu_));

...  

err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);
```

```
(y + 1) * M_LEN + x  
(y + 1) * m + x  
(n) * m + x  
n * m + m - 1  
m - n - 1 + m > m - n - 1  
m > 0
```

Kernel Host
Kernel

```c
__kernel void sw_compute0(...
    const unsigned M_LEN,
    ...
    __global double *cu,
    ...

    int x = get_global_id(0);
    int y = get_global_id(1);
    cu[(y + 1) * M_LEN + x] = <input_equation>
    ...
}
```

Host

```c
size_t sizeInBytes = sizeof(double) * m_len_ * n_len_;
...
    cu_ = clCreateBuffer(context_, CL_MEM_READ_WRITE,
        sizeInBytes, NULL, &err);
    ...
    const size_t globalSize[2] = {m_len_, n_len_};
    ...
    err |= clSetKernelArg(kernel_sw_compute0_, 6,
        sizeof(cl_mem),
        reinterpret_cast<void *>(&cu_));
    ...
    err = clEnqueueNDRangeKernel(cmdQueue_,
        kernel_sw_compute0_, 2, NULL,
        globalSize, 0, NULL, NULL);
```
CONCLUSION

CLARMOR IS READY FOR YOU TO USE

- Canary-based detection scheme finds GPU write overflows
  - 13 GPU buffer overflows in 7 programs
- Works for most OpenCL™ applications
  - Running on GPU or CPU, not vendor specific
- Near real time detection
  - 14% overhead across 175 applications in 16 GPU benchmark suites

- Open Sourced
  - https://github.com/GPUOpen-ProfessionalCompute-Tools/clARMOR
  - Branch available for reproducing paper measurements
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2017 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, AMD FirePro, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. OpenCL is a trademark of Apple Inc. used by permission by Khronos. PCIe is a registered trademark of PCI-SIG Corporation. Linux is a registered trademark of Linus Torvalds. Other names are for informational purposes only and may be trademarks of their respective owners.
MEMORY OVERHEAD
cLARMOR: Loaded CL.WRAPPER
**ATTENTION:** Buffer overflow detected ************
Kernel: sw_compute0, Buffer: cu
First observed writing 1 byte(s) past the end.
Exiting application because of buffer overflow.