ANECDOTE
DISCOVERING A BUFFER OVERFLOW
DISCOVERING A BUFFER OVERFLOW
ANECDOTE
DISCOVERING A BUFFER OVERFLOW

[Diagram showing data transfer between CPU and GPU]

MEMORY

Data | Data | Data

MEMORY

Data | Data | Data
ANEC DOTE
DISCOVERING A BUFFER OVERFLOW
DISCOVERING A BUFFER OVERFLOW
BACKGROUND: NORMAL BUFFER FILL

```c
buf[n+1]
memcpy(buf, src, n+1)
```
BACKGROUND: NORMAL BUFFER FILL

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

<table>
<thead>
<tr>
<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></td>
</tr>
</tbody>
</table>
BACKGROUND: BUFFER OVERFLOW

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

\[
\begin{align*}
\text{buf}[n+1] \\
\text{memcpy(buf, src, n+5)}
\end{align*}
\]

![Diagram showing buffer and source data with memcpy operation]
BACKGROUND: BUFFER OVERFLOW

buf[n+1]

memcpy(buf, src, n+5)
GPU can overflow buffers in system memory
– Over Interconnects like PCIe®
GPU INDUCED 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®
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 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®
CPU and GPU as part of the same package
CPU and GPU as part of the same package
- Every GPU buffer overflow may affect CPU data
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 core with buffer overflow](image)
Overflows on GPU can cause remote GPU code execution


![Diagram showing GPU induced 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

- 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 induced overflow](image)
GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

Software tool to detect buffer overflows caused by GPU

Runnable with most OpenCL™ applications

Low runtime overhead
GOALS
clARMOR: AMD Research Memory Overflow Reporter for OpenCL

Software tool to detect buffer overflows caused by GPU
- Memory buffers, Sub buffers, SVM, Images
- Overflow and Underflow detection

Runnable with most OpenCL™ applications

Low runtime overhead
GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

Software tool to detect buffer overflows caused by GPU
- Memory buffers, Sub buffers, SVM, Images
- Overflow and Underflow detection

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

Low runtime overhead
Software tool to detect buffer overflows caused by GPU
- Memory buffers, Sub buffers, SVM, Images
- Overflow and Underflow detection

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

Low runtime overhead
- 9.7% average overhead
clARMOR: AMD Research Memory Overflow Reporter for OpenCL

Software tool to detect buffer overflows caused by GPU
- Memory buffers, Sub buffers, SVM, Images
- Overflow and Underflow detection

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

Low runtime overhead
- 9.7% average overhead
GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

Software tool to detect buffer overflows caused by GPU
- Memory buffers, Sub buffers, SVM, Images
- Overflow and Underflow detection

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

Low runtime overhead
- 9.7% average overhead
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.

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

```
<table>
<thead>
<tr>
<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></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)

```
<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></td>
<td></td>
</tr>
</tbody>
</table>
```
Inserting known values around a protected region.

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

```
<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>
```
Inserting known values around a protected region.

- 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  adjacent data
```
Inserting known values around a protected region.

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

<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></td>
<td></td>
</tr>
</tbody>
</table>
Inserting known values around a protected region.

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

BUFFER OVERFLOW DETECTION METHODOLOGY
CANARY-BASED DETECTION

adjacent data

overflow src[n+1] – [n+4]

canary

src[0] src[1] src[2] ... src[n] buf buf+1 buf+2 ... buf+n
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]


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

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)

Absence of known canary values indicates an invalid write.
Inserting known values around a protected region.

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

Absence of known canary values indicates an invalid write.
Can find underflow as well!
GOALS

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

- Software tool to detect buffer overflows caused by GPU
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection

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

- Low runtime overhead
  - 9.7% average overhead
LAUNCHING AN OPENCL™ KERNEL
LAUNCHING AN OPENCL™ KERNEL

Buffer Create

Buffer
LAUNCHING AN OPENCL™ KERNEL

Set Arguments
Set Arguments
LAUNCHING AN OPENCL™ KERNEL

Launch Kernel

Kernel

Buffer
LAUNCHING AN OPENCL™ KERNEL

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

Buffer Create
LAUNCHING AN OPENCL™ KERNEL WITH cIARMOR

Buffer Create

Buffer  Canary
LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Buffer Create

Buffer

Metadata

Canary
LAUNCHING AN OPENCL™ KERNEL WITH cIARMOR

Set Arguments

Kernel Information

Buffer Metadata

Buffer
Canary

Kernel
LAUNCHING AN OPENCL™ KERNEL WITH cIARMOR

Set Arguments

Kernel Information

Buffer Metadata

Buffer

Canary

Kernel
LAUNCHING AN OPENCL™ KERNEL WITH c|ARMOR

Set Arguments

Kernel Information

Buffer Metadata

Kernel

Buffer

Canary
LAUNCHING AN OPENCL™ KERNEL WITH cIARMOR
LAUNCHING AN OPENCL™ KERNEL WITH cARMOR

Kernel Information

Buffer Metadata

Launch Kernel

Buffer Canary
LAUNCHING AN OPENCL™ KERNEL WITH cIARMOR
LAUNCHING AN OPENCL™ KERNEL WITH cIARMOR

Kernel Information
Buffer Metadata
Canary Verification

Buffer Canary
LAUNCHING AN OPENCL™ KERNEL WITH cIARMOR

Kernel Information

Buffer Metadata

Canary Verification

Buffer

Canary
LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Kernel Information

Buffer Metadata

Canary Verification

Buffer Canary
LAUNCHING AN OPENCL™ KERNEL WITH cIARMOR

Kernel Information
Buffer Metadata
Canary Verification
LAUNCHING AN OPENCL™ KERNEL WITH clARMOR

Kernel Information
Buffer Metadata
Canary Verification

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

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

- Call Wrapping
  - Buffer, SVM, and Image creates
  - Argument setters
  - Kernel launches
  - Information functions
**GOALS**

clARMOR: AMD Research Memory Overflow Reporter for OpenCL

- Software tool to detect buffer overflows caused by GPU
  - Memory buffers, Sub buffers, SVM, Images
  - Overflow and Underflow detection

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

- Low runtime overhead
  - 9.7% average overhead
TEST SETUP

HARDWARE SPECIFICATIONS AND BENCHMARKS SUITES

- AMD Ryzen™ 7 1800X CPU
- AMD Radeon™ Vega Frontier Edition discrete GPU
- ROCm 1.7
- 143 benchmarks in 14 benchmark suites

<table>
<thead>
<tr>
<th>Suite</th>
<th>Num Benchmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMDAPP</td>
<td>33</td>
</tr>
<tr>
<td>FINANCEBENCH</td>
<td>1</td>
</tr>
<tr>
<td>GPUSTREAM</td>
<td>1</td>
</tr>
<tr>
<td>HETEROMARK</td>
<td>9</td>
</tr>
<tr>
<td>MANTEVO</td>
<td>2</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>8</td>
</tr>
<tr>
<td>POLYBENCH</td>
<td>21</td>
</tr>
<tr>
<td>PROXYAPPS</td>
<td>6</td>
</tr>
<tr>
<td>RODINIA</td>
<td>19</td>
</tr>
<tr>
<td>SHOC</td>
<td>14</td>
</tr>
<tr>
<td>VIENNACL</td>
<td>8</td>
</tr>
</tbody>
</table>
PERFORMANCE EVALUATION
APPLICATION RUNTIME: WITH / WITHOUT TOOL

Lower is better

<table>
<thead>
<tr>
<th>Application</th>
<th>Normalized Runtime with cARMOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>RODINIA</td>
<td>1.1</td>
</tr>
<tr>
<td>SHOC</td>
<td>1.2</td>
</tr>
<tr>
<td>OPENDWARFS</td>
<td>1.3</td>
</tr>
<tr>
<td>PROXKAPS</td>
<td>1.4</td>
</tr>
<tr>
<td>AMDAPP</td>
<td>1.5</td>
</tr>
<tr>
<td>PARBOIL</td>
<td>1.6</td>
</tr>
<tr>
<td>PANNOTA</td>
<td>1.7</td>
</tr>
<tr>
<td>POLYBENCH</td>
<td>1.8</td>
</tr>
<tr>
<td>FINANCEBENCH</td>
<td>1.9</td>
</tr>
<tr>
<td>GPUDSTREAM</td>
<td>2.0</td>
</tr>
<tr>
<td>MANTEVO</td>
<td>3.39</td>
</tr>
<tr>
<td>HETEROMARK</td>
<td>1.1</td>
</tr>
<tr>
<td>VIENNACL</td>
<td>1.1</td>
</tr>
<tr>
<td>NPB_OCL</td>
<td>1.1</td>
</tr>
<tr>
<td>ALL</td>
<td>0.96</td>
</tr>
</tbody>
</table>

9.6%
PERFORMANCE EVALUATION
APPLICATION RUNTIME: WITH / WITHOUT TOOL
EXAMPLE USAGE
BAD_CL_MEM TEST

```
bin/clarmor tests/bad_cl_mem/bad_cl_mem.exe
```

```
cLARMOR: Final command line to run: LD_PRELOAD="/tools/clARMOR/bin/../lib/libclbuff erwrapper.so.1.0" PATH="/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin" CLARMOR_LOG_PREFIX="clARMOR: " CLARMOR_ERROR_EXITCODE=-1 tests/bad_cl_mem/b ad_cl_mem.exe

clARMOR: Loaded CL_WRAPPER
Searching for platforms...
  Using platform: AMD Accelerated Parallel Processing
Searching for devices...
  Using device: gfx803

Running Bad cl_mem Test...
  Using buffer size: 1048566
Launching 262144 work items to write up to 262144 entries.
This will write 1048576 out of 1048566 bytes in the buffer.

clARMOR: 
clARMOR: *********** Buffer overflow detected ***********
clARMOR: Kernel: test, Buffer: cl_mem_buffer
clARMOR: Write Overflow 1 byte(s) past end.
clARMOR: Done Running Bad cl_mem Test.
clARMOR: Done
```
EXAMPLE USAGE

GOOD_CL_MEM TEST

```
cLARMOR: Final command line to run: LD_PRELOAD=/tools/clARMOR/bin/../lib/libclbuf
ferwrapper.so.1.0' PATH= '/tools/clARMOR/bin:/usr/local/sbin:/usr/local/bin:
usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local
/games:/snap/bin' CLARMOR_LOG_PREFIX="clARMOR: " CLARMOR_ERROR_EXITCODE=-1 tests/good_cl_mem/good_cl_mem.exe

cLARMOR: Loaded CL.WRAPPER
Searching for platforms...
   Using platform: AMD Accelerated Parallel Processing
Searching for devices...
   Using device: gfx803

Running Good cl_mem Test...
   Using buffer size: 1048576
Launching 262144 work items to write up to 262144 entries.
This will write 1048576 out of 1048576 bytes in the buffer.
Done Running Good cl_mem Test.
cLARMOR: Done!
```
What do the wrapped OpenCL™ library calls have to do?
- Buffer and Image creates
- Argument setters
- Kernel launches
- Information functions

What are we doing to make the check faster?
WRAPPING THE OPENCL™ API
BUFFER AND IMAGE CREATION

Buffer Creation

- Calls to `clCreateBuffer` or `clSVMAlloc`
  - Allocate buffer
  - Create sub buffer for user
  - Surround with canary
WRAPPING THE OPENCL™ API
BUFFER AND IMAGE CREATION

Buffer Creation
- Calls to `clCreateBuffer` or `clSVMAlloc`
  - Allocate buffer
  - Create sub buffer for user
  - Surround with canary
Buffer Creation

- Calls to `clCreateBuffer` or `clSVMAlloc`
  - Allocate buffer
  - Create sub buffer for user
  - Surround with canary
WRAPPING THE OPENCL™ API
BUFFER AND IMAGE CREATION

Buffer Creation
- Calls to `clCreateBuffer` or `clSVMAlloc`
  - Allocate buffer
  - Create sub buffer for user
  - Surround with canary
WRAPPING THE OPENCL™ API
BUFFER AND IMAGE CREATION

Buffer Creation
- Calls to **clCreateBuffer** or **clSVMAlloc**
  - Allocate buffer
  - Create sub buffer for user
  - Surround 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

- **Buffer Creation**
  - Calls to *clCreateBuffer* or *clSVMAlloc*
    - Allocate buffer
    - Create sub buffer for user
    - Surround 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

Buffer Creation
- Calls to `clCreateBuffer` or `clSVMAlloc`
  - Allocate buffer
  - Create sub buffer for user
  - Surround with canary

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

Annotations for location of canaries, etc.
OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)

- Cannot extend buffer
OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
– Cannot extend buffer
OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
- Cannot extend buffer
- Cannot move buffer
OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
- Cannot extend buffer
- Cannot move buffer
OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)

- Cannot extend buffer
- Cannot move buffer
WRAPPING THE OPENCL™ API
BUFFER CREATION FROM EXISTING ALLOCATIONS

- OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)
  - Cannot extend buffer
  - Cannot move buffer
  - Solution using a temporary copy at run time

![Diagram showing buffer creation from existing allocations]
OpenCL allows buffer creation using an existing memory allocation (host pointers and sub buffers)

- Cannot extend buffer
- Cannot move buffer
- Solution 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
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

Diagram:
- Kernel Information
  - Buffer Metadata
  - Kernel
- Buffer
- Canary
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

![Diagram of kernel launch process]
WRAPPING THE OPENCL™ API
GETTERS AND SETTERS

- GetMemObjectInfo, GetImageInfo
  - Reserve space for canaries

- Enqueue Functions
  - Read / Write / Fill / Copy
  - Buffer / BufferRect / Image
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
ACCELERATION

USING OPENCL™ EVENTS TO INCREASE THROUGHPUT

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

```
synchronous

```

```
asynchronous

```

CPU

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
CONCLUSION

Canary-based detection scheme finds GPU write overflows
- Memory buffers, Sub buffers, SVM, Images
- Overflow and Underflow detection

Works for most OpenCL™ applications
- Running on GPU or CPU, not vendor specific

Near real-time detection
- 9.7% average overhead

Open Sourced
- https://github.com/ROCM-Developer-Tools/clARMOR - MIT

Technical Details
- Dynamic buffer overflow detection for GPGPUs, CGO 2017
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.

© 2018 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, AMD Ryzen, Radeon, 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.
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

Launch Delay

GPU

User Kernel
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
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
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
- Application 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
- cIARMOR Prelaunch
- cIARMOR Postlaunch
- Application Postlaunch

GPU
- User Kernel
- Canary Check

Launch Delay

SNAP_MPI Synchronization

Possible improvement for SNAP_MPI kernel launch

CPU
- Application Prelaunch
- cIARMOR Prelaunch
- cIARMOR 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
- 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
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);
```
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 = \texttt{sizeof(double)} * \texttt{m\_len\_} * \texttt{n\_len\_};
...

\texttt{cu\_} = \texttt{clCreateBuffer} (context_, \texttt{CL\_MEM\_READ\_WRITE},
    sizeInBytes, \texttt{NULL}, &\texttt{err});
...

const size_t globalSize[2] = {\texttt{m\_len\_}, \texttt{n\_len\_}};
...

err |\texttt{=} \texttt{clSetKernelArg}(kernel_sw_compute0\_, 6,
    sizeof(cl\_mem),
    reinterpret\_cast<\texttt{void *}>(&\texttt{cu\_});
...

err = \texttt{clEnqueueNDRangeKernel}(cmdQueue_,
    kernel_sw_compute0\_, 2, \texttt{NULL}, globalSize,
    localSize, \texttt{0}, \texttt{NULL}, \texttt{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

__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] = \textless \text{input\_equation} \textgreater
    ...
}
```

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

```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_));
...
err = clEnqueueNDRangeKernel(cmdQueue_,
    kernel_sw_compute0_, 2, NULL, globalSize,
    localSize, 0, NULL, NULL);
```
ClARMOR: A Dynamic Buffer Overflow Detector for OpenCL Kernels

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);
```

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>

    ...
}

\[(y + 1) \times M\_LEN + x\]
\[(y + 1) \times m + x\]
\[(n) \times m + x\]

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>
    ...
}
```

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

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

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

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

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

Host

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

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

Kernel Host
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);
```
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);
```
EXAMPLE ERROR

clARMOR: Loaded CL_WRAPPER
clARMOR: ATTENTION: *************** Buffer overflow detected ***************
clARMOR: Kernel: sw_compute0, Buffer: cu
clARMOR: First observed writing 1 byte(s) past the end.
clARMOR: Exiting application because of buffer overflow.
The image contains a bar graph with the following details:

- **X-axis**: Number of SVM Buffers (2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
- **Y-axis**: Time to Check (μs) (0, 500, 1000, 1500, 2000, 2500, 3000, 3500)

The graph shows the time taken to check for buffer overflow with different numbers of SVM Buffers. The bars are color-coded as follows:

- **Blue**: CPU
- **Red**: G 1-Buff Host
- **Green**: Kern
- **Purple**: G AllBuff Host
- **Orange**: G Ptrs Host
- **Dark Blue**: Kern

The data indicates that the time to check increases with the number of SVM Buffers for all categories except CPU, which shows a relatively flat line.
A DYNAMIC BUFFER OVERFLOW DETECTOR FOR OPENCL KERNELS

Time to Check (ms)

Number of 256x256 Images

CPU
GPU One Buffer Host
Kernels
GPU All Buffers Host
Kernel

2 4 6 8 10 12 14 16 18 20

0 2.5 5 7.5 10 12.5 15 17.5 20
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

- SNU OpenCL
  - CG (data races resulting in negative indexing, underflow)

Note: These have been reported, and most fixed.
c|ARMOR 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
- SNU OpenCL
  - CG (data races resulting in negative indexing, underflow)
- Note: These have been reported, and most fixed.

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

- SNU OpenCL
  - CG (data races resulting in negative indexing, underflow)

- Note: These have been reported, and most fixed.
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

SNU OpenCL
  – CG (data races resulting in negative indexing, underflow)

Note: These have been reported, and most fixed.
CONSEQUENCES OF BUFFER OVERFLOWS

DEGRADING USER EXPERIENCE, AND SECURITY RISKS
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
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