Unsafe Floating-point to Unsigned Integer Casting
Check for GPU Programs

Wei-Fan Chiang   Ganesh Gopalakrishnan   Zvonimir Rakamarić

School of Computing, University of Utah

April 13, 2015

Supported in part by NSF CCF 7298529, NSF CCF 1346756, and DOE Award Number 10034350
Photo courtesies to Lesica Lab Coding in Neuronal Populations, ROAL living energy, geek.com, and groundreport.com
Manually analyzing GPU programs is extremely difficult.

- large code base, hierarchical memory and thread groups, etc.

Examples of bug categories:
- data race, out-of-bound memory access, unsafe type-casting, etc.
Detecting unsafe **floating-point-to-unsigned-int** casting (FP2UI)

- Undefined behavior in C standard and many GPU programming language e.g. CUDA
- *Example:* unsigned int \( u = -1.23 \);

Unsafe FP2UI found in real-world medical software [Yablonski 2011]

---

Photo courtesies to clker.com and artlure.blogspot.com
Main Contribution

Unsafe FP2UI detection for GPU programs

Static detection for unsafe FP2UI

Integrate to a general GPU program checking framework

- **GKLEE** [Li, PPoPP’12] [Li, SC’13, SC’14]

Close related work: dynamic unsafe type-casting prevention

- **IOC** [Dietz, ICSE’12]
Integration of Static Unsafe FP2UI Detection to GKLEE

CUDA → C → LLVM Bit-code

LLVM Bit-code

Static Analyses

Unsafe FP2UI Detection

Concolic Analysis based Test-case Generation

SMT

Random Testing

transformed LLVM Bit-code

Test Case 1

Test Case 2

Test Case N

(School of Computing, University of Utah)
Static Analysis for Unsafe FP2UI Detection

Key ideas

Tracking non-negative values

- Check if any FP2UI’s operand is possibly negative
- Track both floating-point and integer values

Reduce false alarms:

- Manual hints
- Pre-defined function contracts

Results:

- Discovered 1 unsafe type cast in a real-world GPU library
- Only 1 false alarm raised among 7 benchmarks
Adding **non-negative float** to the original type system

**Creations of non-negative float type:**
- constant assignment *e.g.* `float f = 1.0;`
- language built-in variables *e.g.* `threadIdx.x` in CUDA

**Conservative sign inference over binary operations, phi node, and type-casting**
Values inferred as non-negative are marked in blue.

Non-negative Value Inference

float $f_1 = 1.23$, $f_2 = 2.34$, $f_3 = -1.23$;
float $w = f_1 + f_2$;

Conservative Inference

float $x = f_1 + f_3$; // (sum13 = 0)
float $y = f_2 - f_1$; // (diff21 = 1.11)
float $z = f_1 - f_3$; // (diff13 = 2.46)
Handling phi node is similar to handling branch or select instruction.

- In this example, rand() returns a random boolean.

### Non-negative Value Inference

```c
float f1 = 1.23, f2 = 2.34, f3 = -1.23;
float x = ( rand() ? f1 : f2 );
```

### Conservative Inference

```c
float y = ( true ? f1 : f3 );
```
Sign Inference Over Type-casting

Cast From Non-negative Value
float \( f_1 = 1.23, f_2 = 2.34, f_3 = -1.23 \);
int \( v = (\text{int}) f_1 \);
float \( w = (\text{float}) v \);
unsigned int \( x = (\text{unsigned int}) w \);

Unsafe FP2UI
unsigned int \( y = (\text{unsigned int}) f_3 \);

False Alarm
unsigned int \( z = (\text{unsigned int}) f_1 + f_3; \quad // f_1 + f_3 = 0.0 \)
Example of Reducing False Alarms

FDTD3d

1: procedure FDTDGPU(int dimx, int dimy)
2: 
3:   dim3 dimGrid;
4:   int userBSIZE = (rand() ? min(max(rand(), 128), 512) : 512));
5:   unsigned int dbx = 32;
6:   unsigned int dby = (((userBSIZE / 32) < 16) ? (userBSIZE / 32) : 16);
7:   dimGrid.x = (unsigned int) ceil((float)dimx / dbx);
8:   dimGrid.y = (unsigned int) ceil((float)dimy / dby);
9: end procedure

False alarms at line 7 and 8!
Reducing False Alarms by Manual Hints

Introducing manual hint at line 2

FDTD3d

1: procedure FDTDGPU(int dimx, int dimy)
2:  assume(0 ≤ dimx, dimy);
3:  dim3 dimGrid;
4:  int userBSIZE = (rand() ? min(max(rand(), 128), 512) : 512));
5:  unsigned int dbx = 32;
6:  unsigned int dby = (((userBSIZE / 32) < 16) ? (userBSIZE / 32) : 16);
7:  dimGrid.x = (unsigned int) ceil((float)dimx / dbx);
8:  dimGrid.y = (unsigned int) ceil((float)dimy / dby);
9:  end procedure
Reducing False Alarms by Pre-defined Function Contracts

Maximum (max) and Minimum (min)

float \textbf{max} = \text{max}(1.23, -1.23);
float \textbf{min} = \text{min}(1.23, 2.34);

FDTD3d

1: \textbf{procedure} \texttt{FDTDGPU}(\texttt{int} \texttt{dimx}, \texttt{int} \texttt{dimy})
2: \textbf{assume}(0 \leq \texttt{dimx, dimy});
3: \texttt{dim3} \texttt{dimGrid};
4: \texttt{int userBSIZE} = (\text{rand()} ? \text{min}(\text{max}(\text{rand()}, 128), 512) : 512));
5: \texttt{unsigned int dbx} = 32;
6: \texttt{unsigned int dby} = (((\text{userBSIZE} / 32) < 16) ? \text{userBSIZE} / 32 : 16);
7: \texttt{dimGrid.x} = (\texttt{unsigned int}) \text{ceil}((\text{float})\texttt{dimx} / \texttt{dbx});
8: \texttt{dimGrid.y} = (\texttt{unsigned int}) \text{ceil}((\text{float})\texttt{dimy} / \texttt{dby});
9: \textbf{end procedure}
Reducing False Alarms by Pre-defined Function Contracts

### Ceiling (ceil)

float `ceil1 = ceil(1.23)`;
float `ceil2 = ceil(-1.23)`;

### FDTD3d

1: `procedure FDTDGPU(int dimx, int dimy)`
2: `assume(0 ≤ dimx, dimy);`
3: `dim3 dimGrid;`
4: `int userBSIZE = (rand() ? min(max(rand(), 128), 512) : 512);`
5: `unsigned int dbx = 32;`
6: `unsigned int dby = (((userBSIZE / 32) < 16) ? (userBSIZE / 32) : 16);`
7: `dimGrid.x = (unsigned int) ceil((float)dimx / dbx);`
8: `dimGrid.y = (unsigned int) ceil((float)dimy / dby);`
9: `end procedure`
Experimental Results

Currently handle C + CUDA programs

Artificial examples (2 examples)

CUDA SDK samples (4 examples extracted from version 6.0)
- Thread group size computation and information compaction
- 1 false alarm: can be eliminated by revised implementation

CUDPP: a GPU computation library (1 example)
- Found an unsafe FP2UI in the sanitation check code
Unsafe FP2UI (rand_gold.cpp)

1: `procedure void FF(int i, float p)`
2: `float t = sin((float) i) * p;`
3: `unsigned int trigFunc = (unsigned int) t;`
4: `end procedure`

Safe FP2UI (rand_cta.cuh)

1: `procedure __DEVICE__ void FF(int i, float p)`
2: `float t = sin(__int_as_float(i)) * p;`
3: `unsigned int trigFunc = __float2uint_rd(t);`
4: `end procedure`
Suggested Safe FP2UI (rand_gold.cpp)

1: procedure void FF(int i, float p)
2:   float t = sin((float) i) * p;
3:   unsigned int trigFunc;
4:   if t < 0.0 then
5:     trigFunc = 0;
6:   else
7:     if MAX_UNSIGNED_INT ≥ (long)t then
8:       trigFunc = (unsigned int) t;
9:     else
10:    trigFunc = MAX_UNSIGNED_INT;
11:   end if
12: end if
13: end procedure
Future Integration to GPU Program Analyzers

Current integration with GKLEE

- A static analysis layer before test case generation

Future integration approach

- A light-weight reduction of heavy but precise analysis
Conclusions

Static analysis is effective in detecting unsafe FP2UI.

Discovered unsafe FP2UI in real-world

Integration into general analyzers could be useful in practice.

github.com/wfchiang/llvm-utcc