Northeastern University

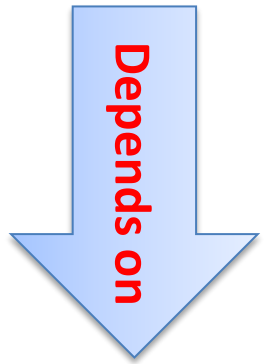
Identifying Volatile Numeric Expressions in OpenCL Applications

Miriam Leeser
mel@coe.neu.edu

Mahsa Bayati, Brian Crafton
Electrical and Computer Engineering
Yijia Gu and Thomas Wahl
College of Computer and Information Science
Northeastern University
Boston MA

Introduction

- Result of floating-point numerical computations



Execution platform, compiler

- Evaluation order is not standard:
  - similar floating point hardware, compilers have freedom when evaluating floating point expressions

Volatile expression

For the same input, expression value differs across platforms

With heterogeneous hardware, differences can become particularly large

Portability promised by OpenCL,

but NOT reproducibility

- Applications from Scalable Heterogeneous Computing (SHOC) Benchmark Suite

  MD: Molecular Dynamics performs an n-body pairwise Lennard-Jones potential computation

MD Largest absolute difference

	MD-InputSet1	MD-InputSet2
AMDCPU, AMDGPU	9.33E+17	1.53E+14
AMDCPU, Intel	0	0
AMDCPU, NVIDIA	0	2560
AMDGPU, Intel	9.33E+17	1.53E+14
AMDGPU,NVIDIA	9.33E+17	1.53E+14
Intel, NVIDIA	0	2560

- Same OpenCL code, same input on
  - Hardware platforms: AMD and Intel CPUs , NVIDIA Tesla GPUs, AMD Radeon APUs
- All *compliant with IEEE 754-2008*

More applications from SHOC

- Sparse Matrix-Vector Multiplication (SPMV)
- Stencil2D: 2D, 9-point single and double precision stencil computation$(100 \times 100\ input, 1000\ iterations)$

Stencil2D , SPMV Largest absolute difference

	Stencil2d-InputSet1	SPMV-Inputset1
AMDCPU, AMDGPU	5.091E+20	0
AMDCPU, Intel	0	0
AMDCPU, NVIDIA	68719476736	6.1E-5
AMDGPU, Intel	5.091E+20	0
AMDGPU,NVIDIA	9. 5.091E+20	6.1E-5
Intel, NVIDIA	68719476736	6.1E-5

What feedback can we give the programmer regarding these differences?

- Determine *tight bounds* for volatile expressions
  independent of the platform (hardware, compiler)

- Bounds can direct the programmer or compiler to focus on parts of the program where reproducibility is important

- Our approach addresses differences between platforms
  - others focus on differences between floating point and real numbers

Our approach

Takes a program p, a fixed input i, an expression x representing some intermediate result of the program

Our method determines an *upper bound I* on the range of values $x(i)$ that program p can produce for x, on input i, across different platforms.

I overapproximates the range, not all values contained in I correspond to values for x

Y. Gu, T. Wahl, M. Bayati, and M. Leeser, "Behavioral non-portability in scientific numeric computing," in Parallel and Distributed Computing (EURO-PAR), 2015

Tight bound method

- Compute Min and Max values for each volatile expression, under all possible evaluations
- Analysis uses dynamic programming
- Order of computation is polynomial in time with relation to size of expression x
- Min and Max values form the left and right boundaries of interval I

Unstable Inputs

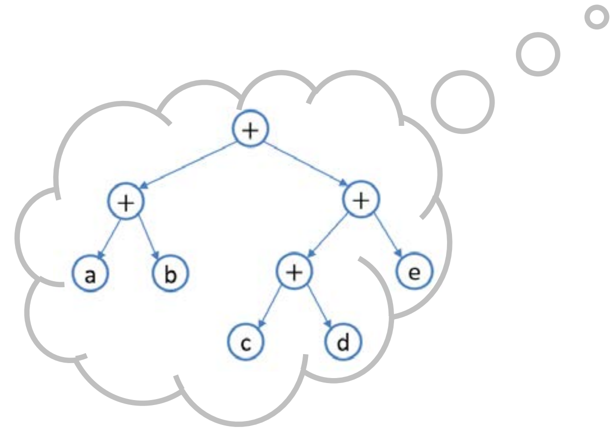
Definition:

Let q be a **Boolean**-valued FP expression. Input I is **unstable** if there exist evaluation models M_0 and M_1 such that

$$q(I, M_0) \neq q(I, M_1)$$

Minimizing $x_1 + x_2 + \ \dots \ + x_n$

… over all possible “parse trees”.



Observation:

$$\min \ x_1 + \ \dots \ + x_n = \quad \min_k \ +_k$$

$$\boxed{\min x_1 + \cdots + x_k} \qquad \boxed{\min x_{k+1} + \cdots + x_n}$$

Minimizing $x_1 + x_2 + \ldots + x_n$

… over all possible parse trees:

Let $N[i,j] = \min\ x_i + \ldots + x_j$

Algorithm: fill array N "bottom up":

1. $N[1,1] = x_1$, … , $N[n,n] = x_n$
2. $N[1,2] = x_1 + x_2$, … , $N[n-1,n] = x_{n-1} + x_n$
3. $N[1,3] = \min\{\, N[1,1] + N[2,3] ,\ N[1,2] + N[3,3] \,\}$

   ⋮

Then $\min\ x_1 + \ldots + x_n = N[1,n]$.

General Volatile Expressions

Similarly (more or less):

- **max** $x_1 + \ldots + x_n$ (analogous)
- min $x_1 * \ldots * x_n$ (sign matters!)
- min $x_1 * y_1 + \ldots + x_n * y_n$ (may involve FMA)

From Expressions to Programs: Propagating Value Ranges

1. **Inputs:** $a \to [a, a]$, …

2. **Volatile expressions** y :
   $\mathrm{y} \to [\min_{M} y \,, \max_{M} y]$ as before

3. **Non-volatile:** use the domain's transfer operation:

$$[x + y] = [\downarrow x + \downarrow y, \uparrow x + \uparrow \mathrm{y}]$$

$$[x \times y] = [\min S \,, \max S]$$ for $S = \{\downarrow x, \uparrow x\} \times \{\downarrow y, \uparrow y\}$

```
x=a+b+c;
y=e*f+g*h;
⋮
```

Exposing Instabilities

- interval arithmetic **overapproximates:**
  $[\downarrow X, \uparrow X]$ overestimates set of possible values of X

- M_0, M_1 may not **materialize** on your (or any!) target platform (our approach is *platform-agnostic*)

- Goal is tight bounds

Finding Instabilities: Dynamic Analysis

... via code instrumentation + runtime analysis:

Before:

```
float A = a[0]*a[0] + a[1]*a[1] + a[2]*a[2];
```

After:

```
#include "unstable.h"
  ⋮
ufloat A = a[0]*a[0] + a[1]*a[1]+ a[2]*a[2];
```

Experiments and Results

Two applications :

- SOR: Jacobi Successive Over-Relaxation from SciMark benchmark
  - stencil computation: runs on a 100x100 grid
  - typical of finite difference applications
  - C code from SciMark, we rewrote it in OpenCL
- Stencil2D: 9-point single and double precision stencil computation
- Input matrices for both applications generated with random cell contents

Ran these programs on a diverse set of platforms:

	Type	Manufacture	Description	Year	FMA?
1	CPU	Intel	E52650	2012	*
2	CPU	AMD	A8-3850	2011	N
3	GPU	NVIDIA	GF108 Quadro 600	2010	N
4	GPU	NVIDIA	Tesla C2075	2011	Y
5	GPU	NVIDIA	Tesla K20	2013	Y
6	GPU	AMD	Radeon HD66550D	2011	N

SOR Code snippet

```
SOR ( global float* A,  int M, int N, float w)
 {
            for(int i=1; i<M-1; i++)  {
               for (int j=1; j<N-1; j++){

                   A[i*N + j] = (w/4) * (A[(i-1)*N + j]+ A[(i+1)*N + j]+
                   A[i*N + (j - 1)] + A[i*N + (j + 1)]) + (1.0-w) * A[i*N + j];

                 }
            }
   }
```

Running our tool on SOR

```
original value: 0.72078645229339600
volatile bound: 0.72078627347946167 0.72078663110733032
```

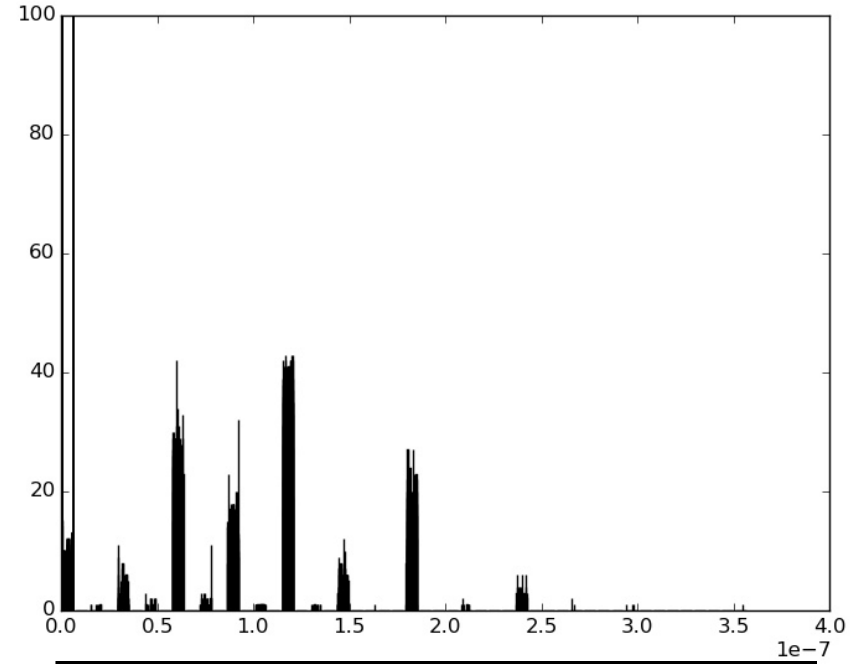
"original value" : Computed left to right, no FMA

SOR Results

$$Max_{Output\ diff} = 3.5E-07$$

- constraints between different loops limit the reordering of expressions
- The experimental results are within the tight theoretical bound

SOR output differences



	SOR-InputSet1
AMDCPU, AMDGPU	0
AMDCPU, Intel	0
AMDCPU, NVIDIA	2.38419E-07
AMDGPU, Intel	0
AMDGPU,NVIDIA	2.38419E-07
Intel, NVIDIA	2.38419E-07

Stencil2D Results

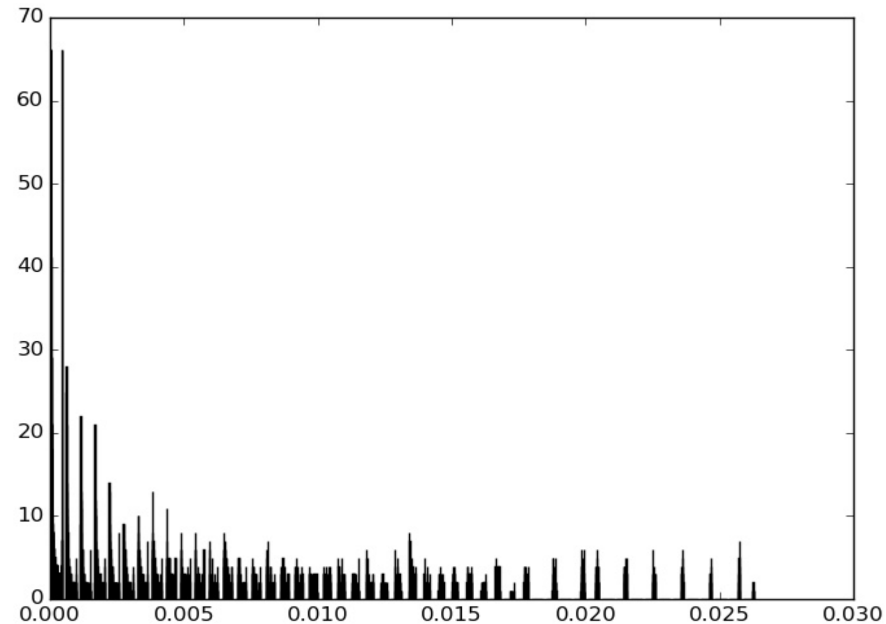
- $Max_{Output\ diff} = 0.03$

  input size 64x64

  30 iterations

- Experiments result shows the difference within the range

Stencil2D output differences



	Stencil2D-InputSet1
AMDCPU, AMDGPU	0
AMDCPU, Intel	0
AMDCPU, NVIDIA	0.00288
AMDGPU, Intel	0
AMDGPU,NVIDIA	0.0028
Intel, NVIDIA	0.0028

Conclusions

- We quantify differences that numeric programs produce, for the same input, across heterogeneous platforms
- Our experiments showed that differences are real and occur not only for specific critical inputs, but even for randomly chosen ones
- We demonstrated that the range of values predicted by our theoretical method are fairly tight
  … and accurately predict observed differences

Future work

- Automate annotating program for user
- Provide user with r***obustness tips*** for important/ volatile ***portions*** of the program
  - Inhibit use of Fused multiply add (FMA)
  - Force expression evaluation orders

- Should be applied to small regions of the program that contribute most to the computational differences, leaving the compiler free to rearrange other parts

Thank you!

- Miriam Leeser: mel@coe.neu.edu

- Floating point comparison for different platforms," http://www.coe.neu.edu/Research/rcl/projects/FloatingpointComparison/index.html

- Y. Gu, T. Wahl, M. Bayati, and M. Leeser, "Behavioral non-portability in scientific numeric computing," in Parallel and Distributed Computing (EURO-PAR), 2015.

National Science Foundation
WHERE DISCOVERIES BEGIN