Reproducibility BoF
Position, Solutions

Triaging Races, Floating Point, Other Concerns

Wei-Fan Chiang, Ganesh Gopalakrishnan,
Geof Sawaya, Simone Atzeni

School of Computing, University of Utah, Salt Lake City, UT

Acks to: Mark Baranowski, Ian Briggs, Zvonimir Rakamaric’, Alexey Solovyev,
Charles Jacobsen, LLNL Pruner Team

Supported by NSF ACI 1535032, CCF 1531140
Definition of Reproducibility

- No Rummy Bugs :-)
Definition of Reproducibility

- No Rummy Bugs :-)  
  - No Unknown Unknows
Definition of Reproducibility

● No Rummy Bugs :-)  
  ○ No Unknown Unknows
    ■ The ability to explain differing results
      ● upon subsequent execution(s)
      ● of the same (or functionally very similar) software,
      ● potentially under different compilers / platforms / thread schedules /
Our high-level message

- Need to plan for repro by
  - Considering various sources of non-reproducibility
  - Having a triaging scheme
- We consider both floating-point non-repro and data races in our work
- We plan to develop tools to stage their search, and amelioration/mitigation
- This presentation will emphasize
  - Floating-point non-reproducibility
  - Tools for its amelioration
FP is non-intuitive

Error Profile for Addition (computed using the Racket FP library developed by N. Toronto)

\[(\text{e}^x - 1) / \log(\text{e}^x)\]

(shown here: blue is actual error; red is formally analyzed error bound)

(can have less error in some intervals than \((\text{e}^x - 1) / x\) in some input value ranges!)

- More complex expressions can be analyzed
- Helps designers of arithmetic libraries
- Too hard to design for repro based on such knowledge
Examples: an MPI hang caused by floating-point...

- Xeon vs. Xeon-Phi hang (somewhat catastrophic)

**A Heterogeneity-induced bug**
(Berzins, Meng, Humphrey, XSEDE’12)

\[
P = 0.4218749999999944488848768742172978818416595458984375 \\
C = 0.0026041666666666665221063770019327421323396265506744384765625 \\
\text{Compute: floor}( P / C )
\]

Xeon \hspace{1cm} Xeon Phi

\[
P / C = 161.9999... \\
floor( P / C ) = 161 \\
\]

\[
P / C = 162 \\
floor( P / C ) = 162
\]

Authors’ fix : used double-precision for P/C
Other egregious examples

- Explosion of Compiler Flags ( + too many other confusing options!)
```
qfp=# select compiler, switches, sort, precision from tests
  where score='3f8e8010000000000000' and host='kingspeak1' and name
  = 'DoOrthoPerturbTest' and run = (select max(index) from runs)
  order by compiler, switches;

<table>
<thead>
<tr>
<th>compiler</th>
<th>switches</th>
<th>sort</th>
<th>precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;g++&quot;</td>
<td></td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td></td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-fassociative-math</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-ffast-math</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-ffinite-math-only</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-ffloat-store</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-fp-contract-on</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-fmerge-all-constants</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-fntrapping-math</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-frounding-math</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-frounding-nans</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-funsafe-math-optimizations</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-maxx</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-mfpmath=sse -mtune=native</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-00</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-01</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-02</td>
<td>gt</td>
<td>e</td>
</tr>
<tr>
<td>&quot;g++&quot;</td>
<td>-03</td>
<td>gt</td>
<td>e</td>
</tr>
</tbody>
</table>
```
Races have bizarre effects that often show up when one optimizes.

Initially: \( x[i] == y[i] == i \)

Warp-size = 32

```c
#include <stdio.h>

__global__ void kernel(int* x, int* y)
{
    int index = threadIdx.x;

    y[index] = x[index] + y[index];

    if (index != 63 && index != 31)
        y[index+1] = 1111;
}
```

The hardware schedules these instructions in “warps” (SIMD groups).

However, this “warp view” often appears to be lost.

E.g. When compiling with optimizations

**Expected Answer:** 0, 1111, 1111, ..., 1111, 64, 1111, ...

**New Answer:** 0, 2, 4, 6, 8, ...
Real issue for a practitioner: Hard to tell root-cause of bug!

- Race or FP or … ?
Case Studies, Tools we are building

- Xeon vs. Xeon-Phi
- Our FP tools applied to case studies
  - Round-off analysis: Symbolic Taylor Forms (FPTaylor, FM’15)
  - Quick triggering of divergence (LCPC’15)
  - FP precision sensitivity study (QFP tool under construction):
    - Takes compiler flags, reduction settings, machines
    - Generates Makefiles, makes and runs results
    - Collect results, supports SQL query based equivalencing
    - Computational divergence detection methods (being investigated)
  - 4 (sorts) * 3 (prec) * 2 (comp) * 25 (flags) * 6 (hosts) = 3,600 options!
    - 8 (tests) * 3600 (above) = 28,800 configured tests!!
  - Race detection tools
    - For GPUs (GKLEE)
    - For OpenMP (Archer)
What Solutions Would You Like to See

- **Tools for quick attribution**
  - What type of bug is it?
    - Quick check for races
    - Then check for FP maladies
    - Then explore other causes (depending on prog model)

- **Kernels of “FP-sensitive codes” - collect from the community**
  - Kernels extracted from real apps where
    - Precision matters
    - Precision does not matter

- **Case studies - collect buggy examples from the community, build tools**
  - With races alone
  - With FP issues alone
  - With both
Extra slides
Reduction Methods (used by QFP)

- Ascending
- Descending
- Unsorted

Library Default
<table>
<thead>
<tr>
<th>Compiler flags explored by QFP</th>
<th>-mfpmath=sse</th>
<th>-ffast-math</th>
<th>-fcx-limited-range</th>
<th>-fp-model=precise</th>
</tr>
</thead>
<tbody>
<tr>
<td>-funsafe-math-optimizations</td>
<td>-fexcess-precision=fast</td>
<td>-ftz</td>
<td>-fp-model=source</td>
<td></td>
</tr>
<tr>
<td>-mavx</td>
<td>-fexcess-precision=standard</td>
<td>-no-ftz</td>
<td>-fp-model=double</td>
<td></td>
</tr>
<tr>
<td>-O0</td>
<td>-fassociative-math</td>
<td>-prec-div</td>
<td>-fp-model=extended</td>
<td></td>
</tr>
<tr>
<td>-O1</td>
<td>-freciprocal-math</td>
<td>-no-prec-div</td>
<td>-fpmodel=except</td>
<td></td>
</tr>
<tr>
<td>-O2</td>
<td>-ffinite-math-only</td>
<td>-fma</td>
<td>-fp-model=strict</td>
<td></td>
</tr>
<tr>
<td>-O3</td>
<td>-fno-trapping-math</td>
<td>-no-fma</td>
<td>-fp-model=fast=1</td>
<td></td>
</tr>
<tr>
<td>-ffp-contraction-on</td>
<td>-frounding-math</td>
<td>-fp-trap</td>
<td>-fp-model=fast=2</td>
<td></td>
</tr>
<tr>
<td>-fmerge-all-constants</td>
<td>-fsignaling-nans</td>
<td>-fp-port</td>
<td>-fp-model=precise</td>
<td></td>
</tr>
<tr>
<td>-ffloat-store</td>
<td>-fsingle-precision-constant</td>
<td>-mp1</td>
<td>[default]</td>
<td></td>
</tr>
</tbody>
</table>
QFP

tests := \{A\} \{H\} \times \{P\} \times \{S\} \times \{C\} \times \{F\}

Algorithm
Host
Precision
Sort
Compiler
Flags

Intel Xeon Sandybridge
Intel Xeon Phi
Intel Haswell
Arm64

GCC
Intel
Clang

Flags / Makefile

Compiler

Tests

Host/Platform

Compilation

Execution

SQL Database
Xeon (Sandybridge) vs Xeon Phi (Knights Corner)

- From a real world example, causing control flow divergence in a heterogeneous system

- **What**
  - 0.421874999999999944488848768742172978818416595458984375
divided by
  - 0.0026041666666666665221063770019327421323396265506744384765625

- **Xeon:** 162
- **Phi:** 161.999999999999971578290569595992565155029296875
- **MPFR:** 1.6199999999999987676524426660761717207372237845...
Or should we say SSE vs AVX512

- The Intel compiler generated a simple SSE divide instruction on the Sandy Bridge:
  - Compiler command line:
    - icpc -O2 test.cpp
  - Producing:
    - divsd 0x10(%rsp),%xmm0
- While for direct execution on Phi, it used a software divide algorithm (as there is no divide intrinsic in the Xeon Phi Coprocessor ISA)
  - icpc -O2 -mmic test.cpp
    - ->
AVX512 Software Division (some highlights of 58 lines of disassembly)

- `broadcastsd 0x20(%rsp),%zmm0`  #load dividend
- `mov $0x1,%edx`
- `vbroadcastsd 0x28(%rsp),%zmm1`  #load divisor
- `vgetexppd %zmm1,%zmm9{%k3}`  #extract (unbiased) exponent of divisor
- `vgetexppd %zmm0,%zmm8{%k3}`  #extract (unbiased) exponent of dividend
- `vgetmantpd $0x0,%zmm1,%zmm7{%k3}`  #extract mantissa of divisor
- `vmovapd %zmm26,%zmm0`
Sample QFP Query

qfp=# select count(distinct score0), name from tests where host =
    'kingspeak1' and run = (select max(index) from runs) group by
    name order by name;

<table>
<thead>
<tr>
<th>count</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>DoHariGSBasic</td>
</tr>
<tr>
<td>3</td>
<td>DoHariGSImproved</td>
</tr>
<tr>
<td>1</td>
<td>DoMatrixMultSanity</td>
</tr>
<tr>
<td>16</td>
<td>DoOrthoPerturbTest</td>
</tr>
<tr>
<td>3</td>
<td>DoSimpleRotate90</td>
</tr>
<tr>
<td>21</td>
<td>DoSkewSymCPRotationTest</td>
</tr>
<tr>
<td>8</td>
<td>RotateAndUnrotate</td>
</tr>
<tr>
<td>20</td>
<td>RotateFullCircle</td>
</tr>
</tbody>
</table>

(8 rows)
QFP Equivalence Classes

- Each execution of a test algorithm is assigned a score based on its distance from the known answer.

- For instance, the perturbed orthogonal vector test score is the ‘true’ dot product.

- The worse a test execution performs, the higher its score.

- Test results belong to the same equivalence class if they have the same score.
QFP Database Queries

- The QFP Test database is a SQL relational db (postgresql)

- This allows us to craft SQL queries that provide information about computational divergence across VERY similar executions.

- How do we get a list of tests from one host with a count of the equivalence classes for each test algorithm?