Guarding Numerics Amidst Rising Heterogeneity

a presentation at correctness-workshop.github.io/2021
by members of the ComPort project
https://xstack-fp.github.io/ (evolving website)

Ganesh Gopalakrishnan, Ignacio Laguna, Ang Li, Pavel Panchekha, Cindy Rubio-Gonzalez, Zachary Tatlock
1,4: University of Utah, 2: LLNL, 3: PNNL, 4: University of California, Davis 5: University of Washington
ganesh@cs.utah.edu, ilaguna@llnl.gov, ang.li@pnnl.gov, pavpan@cs.utah.edu, crubio@ucdavis.edu,
ztatlock@cs.washington.edu

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, ComPort: Rigorous Testing Methods to Safeguard Software Porting, under Award Numbers DE-SC0022252 (1,4), SCW 1743 (2), SCW 78284 (3), UCD# (5), UW# (6)
Rising Heterogeneity, Mixed-Precision

Chinese team wins Gordon Bell Prize for Simulation of Quantum Circuit

RQC). Their simulation achieved a performance of 1.2 Eflops (one quintillion floating-point operations per second) single-precision, or 4.4 Eflops mixed-precision, using over 41.9 million Sunway cores (processors).
Rising Heterogeneity: Data Movement Reduction → Precision Reduction
Rising Heterogeneity

- CPUs along with GPUs and custom accelerators in support of:
  - HPC and ML workloads
- We focus on the consequences of GPU adoption
  - We first describe the broad spectrum of problems
    - Jointly compiled in our paper
  - Then the specifics of each problem
  - And how to solve them through community effort
GPUs: Moving/Evolving serving different needs...

<table>
<thead>
<tr>
<th>Format</th>
<th>Exponent</th>
<th>Mantissa</th>
<th>Sign</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double precision (FP64)</td>
<td>11 bits</td>
<td>52 bits</td>
<td>1 bit</td>
</tr>
<tr>
<td>Single precision (FP32)</td>
<td>8 bits</td>
<td>23 bits</td>
<td>1 bit</td>
</tr>
<tr>
<td>Half precision (FP16)</td>
<td>5 bits</td>
<td>10 bits</td>
<td>1 bit</td>
</tr>
<tr>
<td>Tensor floating-point 32 (TF32)</td>
<td>8 bits</td>
<td>10 bits</td>
<td>1 bit</td>
</tr>
<tr>
<td>Brain floating-point 16 (BF16)</td>
<td>8 bits</td>
<td>7 bits</td>
<td>1 bit</td>
</tr>
</tbody>
</table>
Challenges due to Increasing GPU/accelerator adoption

**Floating-Point Formats**
- Range of formats in GPUs
- Several rounding modes
- FMA complicates things

**Concurrency**
- Data races are hard to detect
- Checking that reductions are reproducible is challenging

**Mixed-Precision**
- Multiple precision levels supported
- Hard to tune and maintain

**Exceptions**
- Not supported in GPUs
- Detection required in codes
- Printf approach is suboptimal

**Compiler Effects**
- Aggressive optimizations
- Different compilers used in the GPU and the host CPU

**Testing**
- Error tolerance unknown
- Static testing is desirable
- More tools needed
Need better formal error models.

Build trust in them outside of ML.
Need GPU Race Checkers; none available now. Dire need to develop.

Need better formal error models. Build trust in them outside of ML.
Need better formal error models. Build trust in them outside of ML.

Nobody trusts or knows how brittle the code will get, and where. Need efficient analysis tools.
Need better formal error models. Build trust in them outside of ML.

Need GPU Race Checkers; none available now. Dire need to develop.

Nobody trusts or knows how brittle the code will get, and where. Need efficient analysis tools.

HW Exceptions not reported by many GPUs (currently printfs). Develop analysis tools.
Need better formal error models. Build trust in them outside of ML.

Need GPU Race Checkers; none available now. Dire need to develop.

Nobody trusts or knows how brittle the code will get, and where. Need efficient analysis tools.

HW Exceptions not reported by many GPUs (currently printfs). Develop analysis tools.

Need better formal error models. Build trust in them outside of ML.

Nobody trusts or knows how brittle the code will get, and where. Need efficient analysis tools.

Need GPU Race Checkers; none available now. Dire need to develop.

Testing Objectives, Oracles, Fuzzing, Scalable Tracing.
Open-Source Testing Tool
Components to be Shared.

Floating-Point Formats
- Range of formats in GPUs
- Several rounding modes
- FMA complicates things

Concurrency
- Data races are hard to detect
- Checking that reductions are reproducible is challenging

Mixed-Precision
- Multiple precision levels supported
- Hard to tune and maintain

Exceptions
- Not supported in GPUs
- Detection required in codes
- Printf approach is suboptimal

Compiler Effects
- Aggressive optimizations
- Different compilers used in the GPU and the host CPU

Testing
- Error tolerance unknown
- Static testing is desirable
- More tools needed

HW Exceptions not reported by many GPUs (currently printfs). Develop analysis tools.

Need better formal error models. Build trust in them outside of ML.

Nobody trusts or knows how brittle the code will get, and where. Need efficient analysis tools.

Need GPU Race Checkers; none available now. Dire need to develop.

Floating-Point Formats
- Range of formats in GPUs
- Several rounding modes
- FMA complicates things

Concurrency
- Data races are hard to detect
- Checking that reductions are reproducible is challenging

Mixed-Precision
- Multiple precision levels supported
- Hard to tune and maintain

Testing Objectives, Oracles, Fuzzing, Scalable Tracing. Open-Source Testing Tool Components to be Shared.

Exceptions
- Not supported in GPUs
- Detection required in codes
- Printf approach is suboptimal

Compiler Effects
- Aggressive optimizations
- Different compilers used in the GPU and the host CPU

Testing
- Error tolerance unknown
- Static testing is desirable
- More tools needed

HW Exceptions not reported by many GPUs (currently printfs). Develop analysis tools.

Challenges and Solutions (summary slide)

- **FP Formats, Formal Standards, Error Models**
  - FMA, SFU, Tensor Cores

- **Exceptions**
  - Develop techniques to detect at runtime or pre-analyze and prove absence

- **Schedule-dependency**
  - Races, Reduction Order Dependence

- **Compiler optimizations**
  - Performance-portability layers can provide a point to inject parametric solutions

- **Mixed precision**
  - Not just flashy results but robust engineering, runtime dynamic-range exhaustion detection

- **Testing and Fuzzing**
  - Need to specify interfaces, better goal-directed fuzzers
Proprietary Nature of GPUs is a reality

- **Nvidia is dominant**
  - Good and bad

- **AMD and Intel on the rise -- but very little experience**
  - Documented uses in HPC and ML hard to come by
<table>
<thead>
<tr>
<th>Tool Name</th>
<th>Instrumentation Method</th>
<th>GPU Availability</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPSpy</td>
<td>OS-level instrum.</td>
<td>No GPU</td>
<td>Available</td>
</tr>
<tr>
<td>FPChecker (Laguna, ASE’20)</td>
<td>LLVM instrumentation</td>
<td>GPU (initial)</td>
<td>Available</td>
</tr>
<tr>
<td>Verificarlo / Verrou</td>
<td>Montecarlo Arithmetic</td>
<td>No GPU</td>
<td>Available</td>
</tr>
<tr>
<td>FPDebug, NSan (CC’21), FPSanitizer</td>
<td>Shadow Value</td>
<td>No GPU</td>
<td>Some available</td>
</tr>
<tr>
<td>Herbgrind</td>
<td>Valgrind instrum.</td>
<td>No GPU</td>
<td>Install issues</td>
</tr>
<tr>
<td>Saman (Nestor),</td>
<td>Modeling error (library based)</td>
<td>No GPU</td>
<td>Available</td>
</tr>
<tr>
<td>Ariadne</td>
<td>Exception triggering</td>
<td>No GPU</td>
<td>?</td>
</tr>
<tr>
<td>FLiT</td>
<td>Optimization bisection</td>
<td>No GPU</td>
<td>Available</td>
</tr>
<tr>
<td>Blossom, S3FP, FPGen</td>
<td>Guided fuzzing</td>
<td>No GPU</td>
<td>Available</td>
</tr>
</tbody>
</table>
## Selected Numerical Issues, Solutions, Actionables

<table>
<thead>
<tr>
<th>Issue</th>
<th>Problems, Where Experienced</th>
<th>Status of Solutions</th>
<th>Most Promising Research Needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Number Systems, Exception</td>
<td>No common notions of error</td>
<td>Hype has overshot usage, tools</td>
<td>Fix IEEE issues first; Automate through translation; Invest in education</td>
</tr>
<tr>
<td>Precision tuning</td>
<td>Code can become very brittle at places</td>
<td>Tools to check for blown precision budgets are unavailable</td>
<td>Need precision pressure-relief valves; Avoid Precision Fragmentation; Invest more in data compression (&quot;bulk tuning&quot;)</td>
</tr>
<tr>
<td>Scalable Error Analysis</td>
<td>Many codes have have loops; No &quot;one-size fits all&quot;</td>
<td>Not all variables are alike (values, derivatives, FFT)</td>
<td>Domain-specific Error Definitions Appear Inevitable</td>
</tr>
<tr>
<td>Handling Compiler-Induced Variability</td>
<td>Made difficult by proprietary compilers</td>
<td>Very little progress; compilers don’t know what a variable models</td>
<td>Insist on clear compiler specs; Optimize specific to problem semantics</td>
</tr>
<tr>
<td>Combined HPC and ML</td>
<td>Increasing in Uptake</td>
<td>Hardly any Tools to support SW Testing</td>
<td>Urgent creation of verification benchmarks; Get traction by pitching around Trustworthy AI</td>
</tr>
</tbody>
</table>
Concluding Remarks

- **Community Action**
  - FPBench.org
  - X-Stack Project

- **Challenge Benchmark Creation**
  - See one benchmark suite proposal at [https://docs.google.com/presentation/d/1b6bAlmj_4xGKg8D7iW_fDMmdJODzlJA4/edit?usp=sharing&ouid=11149565245157297413&amp;rlz=1C15801_enUS863US863](https://docs.google.com/presentation/d/1b6bAlmj_4xGKg8D7iW_fDMmdJODzlJA4/edit?usp=sharing&ouid=11149565245157297413&amp;rlz=1C15801_enUS863US863)
  - Need to whittle down, provide a graded series of challenges
  - Tools to forecast what will happen when precision/platform/formats are changed

- **Need tool standardization, avoid duplication of effort**
  - Incentivize robust tool release, *value real impact of tools on HPC codes*
    - Change reward metrics!

- **Help from GPU vendors essential to stay abreast**
  - Force hands during procurement -- *not just for perf but also correctness tools!*
  - *Not just window-dressing but serious commitment*
    - Best Practices to Mix or Change Precision
    - C++11 memory model adoption (amidst CUDA Atomics, older idioms)
    - Weaning users away from coding practices such as the "C volatile holy-water sprinkles"