ERROR ANALYSIS USING AUTOMATIC DIFFERENTIATION IN HPC APPLICATIONS

Harshitha Menon
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

August 11, 2020
Earth simulation
Brain simulation
Drug research
Weapons reliability

Sierra
Computers are Expected to Produce Correct Results

Factors that can affect the correctness

Faults in Hardware

Numerical Errors

Precision Errors

...
Study the impact of errors on HPC applications.
DISCVAR: Discovering Critical Variables that are impacted by Soft Errors

ADAPPT: Floating-Point Precision Analysis

Future Work
DISCVAR for SOFT ERRORS

FUNDING: DOE ASCR - Validating Extreme Scale Resilience with Veracity

Collaborators:
LLNL: Kathryn Mohror
UT Austin: Chun-Kai Chang, Mattan Erez,
Univ Utah: Zhimin Li, Yarden Livnat, Valerio Pascucci


SOFT ERRORS

Impact of soft errors

- Crash
- Masked or Benign
- Silent Data Corruption (SDC)

Image credit: CML at ASU
EFFECTS FROM THE HALLOWEEN 2003 SOLAR STORM

Aurorae were observed as low as CO, CA
Damages to satellites, flights diverted
At Los Alamos, the “Q” cluster had 26 errors a week
At Virginia Tech, System X architect said “Felt like they had not only built the world’s third fastest supercomputer, but also one of the world’s best cosmic ray detector”
SDC ERRORS - AMAZON S3

Error rates in all Amazon S3 datacenters went up significantly

Large number of servers were sending information about system state (gossiping) and failing

**Cause:** Message corruption where a single bit was corrupted leading to incorrect system state information.
PROTECTION AGAINST ERRORS

Hardware fault detection mechanism

- Redundancy in hardware - Expensive

Software-level detectors

- Full duplication - Significant overhead (2x)
- Protect vulnerable regions of the code - Lower cost
STUDY THE IMPACT OF SOFT ERRORS

One Program Input

Artificially introduce a fault

Program Execution

Observe Failure

Repeat for thousands of samples for the same input

SDC
Crash
Benign

Image credit: UBC
Study the impact of soft errors on an application without large number of fault injection runs.
APPROACH

Predict the outcome of errors in variables

Use automatic differentiation (AD) to predict the impact of faults

An analytical method to understand the soft errors
HOW DOES THE OUTPUT CHANGE WITH RESPECT TO ITS INPUTS?

For a given $y = f(x)$

First order Taylor series approximation at $x=a$

$$y = f(x)$$

$$= f(a) + f'(a)(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \ldots$$

$$\approx f(a) + f'(a)(x - a).$$

$\Delta y = f'(a) \Delta x$
USING AUTOMATIC DIFFERENTIATION

We assume that the application can be represented as a differential function.

Compute the derivative of the output of the function with respect to all the inputs and intermediate variables.

Use the reverse mode of AD.

Use CoDiPack (Code Differentiation Package).
SINGLE BIT FLIP FAULT MODEL

Determine $\Delta x$ due to a single bit flip:

Unsigned integer: flip $i^{th}$ bit an error of $2^i$

Floating point double: IEEE 754 (0-51 mantissa, 52-62 exponent, 63 sign)

Error in mantissa bit $i$: $2^{i-52} \times 2^{e-1023}$
Artificially introduce a fault

\[ x = 0100000010000\ldots \]

\[ x = 0100000010001\ldots \]

\[ \Delta x = 0.25 \]

\[ \Delta y = \frac{\partial y}{\partial x} \cdot \Delta x \]

\[ \Delta y > \text{threshold} \quad \text{SDC} \]
METRICS FOR EVALUATION

SDC rate: Fraction of faults that result in output error
EVALUATION

Fault model: single bit flips in variables

AD tools: Tapenade

Benchmarks:

- FFT - Splash-2 benchmark,
- Conjugate Gradient benchmark (CG)

Comparison to exhaustive fault injection campaign

- 2 - 5 million fault runs for the exhaustive campaign
- 2 runs with our approach
SDC RATE FOR FFT

![Graph showing SDC rate for FFT with Source Code Line Number on the x-axis and SDC Rate on the y-axis. The graph compares Fault Injection and DisCVar.]
CONJUGATE GRADIENT BENCHMARK

![Graph showing SDC Rate vs Ap Variable Instance for DisCVar]
CONJUGATE GRADIENT BENCHMARK

```c
void solve_cg(double A, double b, double x, double Ax, double p, double r) {
    double normr;
    double rtrans = 0, oldrtrans = 0;
    double p_ap_dot = 0;
    double alpha = 0;
    double double tolerance = 0.06;
    double double alpha = 0;
    double double tolerance = 2.22045e-16;
    int k;

    for (k = 1; k <= MAX_ITER && normr > tolerance; ++k) {
        if (k == 1) {
            wapby(1, x, 0, x, p, p);
            matvec(A, p, Ap);
            wapby(1, b, -1, Ap, r);
            rtrans = dot_r2(r);
            normr = sqrt(rtrans);
            } else {
                oldrtrans = rtrans;
                rtrans = dot_r2(r);
                double beta = rtrans/oldrtrans;
                daxby(1, r, beta, p);
                }
            normr = sqrt(rtrans);
            matvec(A, p, Ap);
            p_ap_dot = dot(Ap, p);
            alpha = rtrans/p_ap_dot;
            daxby(alpha, p, 1, x);
            danby(-alpha, Ap, 1, r);
        }

        double golden=1.41756-62;
        printf("normr: %4.6f\n",normr);;
        printf("iterations: %d\n",k);
        logTestResult("normrdiff", fabs(golden-normr));
    } //printVec(x);

    int main(int argc, char* argv[])
        double A;
        double b;
```
SUMMARY

Developed an error estimate model for soft errors without requiring millions of fault injection runs.

Error estimate model uses Algorithmic Differentiation (AD).

This approach identifies critical variables for selective protection from soft errors.
ADAPT: FLOATING-POINT PRECISION ANALYSIS

FUNDING - LDRD SI: Variable Precision Computing PI: Jeffrey Hittinger

Collaborators: Michael O. Lam, Daniel Osei-Kuffuor; Markus Schordan, Scott Lloyd, Kathryn Mohror; Nathan Pinnow, Jeffrey Hittinger


MOTIVATION

HPC applications extensively use floating point arithmetic operations

Computer architectures support multiple levels of precision

Higher precision - improve accuracy

Lower precision - reduces running time, memory pressure, energy consumption
MIXED PRECISION ARITHMETIC

Using multiple levels of precision in a single program

Without affecting correctness

Improving performance

Manually optimizing for mixed precision is challenging
Develop an automated analysis technique for

Identifying variables that require higher precision to ensure correctness.

Using the lowest precision sufficient to achieve a desired output accuracy to improve running time and reduce power and memory pressure.
RELATED WORK

Automatically discovering unstable floating-point operations and applying transformations

Herbie [Panchekha’14 et al.], Darulova’18 et al.

Search based methods

CRAFT [Lam’13 et al.], Precimonious/HiFPTuner [Rubio’13 et al.]

Rigorous error analysis methods

FPTuner [Chiang’17 et al.], Rosa/Daisy [Darulova’14 et al.]

Using automatic differentiation for

Significance Analysis for approximate computing: Vassiliadis’16 et al.

Rounding error estimation: Langlois’02 et al., Iri’1991
Used first order Taylor series approximation to estimate the rounding errors in variables.

\[ \Delta y = f'(a) \Delta x \] for \( y = f(x) \) at \( x = a \)

Generalizing it for \( y = f(x_1, x_2, \ldots, x_n) \) at \( x_i = a_i \)

\[ \Delta y = f_{x_1}'(a) \Delta x_1 + \ldots + f_{x_n}'(a) \Delta x_n \]

Obtain \( f_{x_i}'(a) \) using algorithmic differentiation (AD)

Reverse mode of AD is used to compute the partial derivatives of all the variables with respect to the output in a single execution.
MIXED PRECISION ALLOCATION

Calculate the error contribution for each variable

Greedy approach

Variables sorted based on error contribution

Variables switched to lower precision - estimated error contribution within threshold
AUTOMATIC FLOATING-POINT PRECISION ANALYSIS

TypeForge
- List variables
- Change float/double to AD_real; add other ADAPT instrumentation

TypeForge
- List of valid transformations (JSON)

TypeForge
- Change double to float according to configuration

ADAPT
- Track computation and calculate output error estimate

CODIPACK
- Calculates adjoints

CRAFT
- Search and verify mixed-precision configurations

User-provided verification routine w/ input data

Mixed-precision configuration (JSON)

Test results

Modified source code

Modified source code

Source code

Modified source code

Mixed-precision source code
EVALUATION

Benchmarks and Mini-Apps:

6 benchmarks
HPCCG, Lulesh

Comparison with existing tools

Precimonious, CRAFT : search based
FPTuner : interval analysis
EVALUATION ON HPCCG

HPCCG from Mantevo benchmark suite
ADAPT is able to identify critical sections that need to be in higher precision
Mixed precision analysis version achieves **1.1x speedup.**
Used ADAPT on Lulesh to create mixed precision sensitivity profile

Used the profile as a guide to develop a mixed precision version for a CUDA implementation of Lulesh

Achieved speedup of 1.2x within error threshold of 1e-11 on GPU
COMPARISON WITH EXISTING TOOLS

CRAFT
Search based approach
Analyzes instructions

Precimonious
Search based approach
Explores hundreds of configurations for tiny benchmarks

FPTuner
Rigorous approach
Supports only real-valued expression language
## EVALUATION

<table>
<thead>
<tr>
<th>Program</th>
<th>Error Threshold</th>
<th>Output Error</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>arclength</td>
<td>1E-12</td>
<td>1.7E-13</td>
<td>1.11</td>
</tr>
<tr>
<td>simpsons</td>
<td>1E-12</td>
<td>4.5E-14</td>
<td>1.13</td>
</tr>
<tr>
<td>jetEngine</td>
<td>1E-13</td>
<td>2.5E-13</td>
<td>1.40</td>
</tr>
<tr>
<td>carbonGas</td>
<td>1E-10</td>
<td>1.0E-11</td>
<td>1.57</td>
</tr>
<tr>
<td>HPCCG</td>
<td>1E-10</td>
<td>2.8E-15</td>
<td>1.10</td>
</tr>
<tr>
<td>LULESH</td>
<td>1E-08</td>
<td>1.8E-11</td>
<td>1.20</td>
</tr>
</tbody>
</table>
Analysis time wrt to the application time. ADAPT has the lowest analysis time.
LIMITATIONS

Analysis limited to inputs used

Use representative datasets

Control-flow divergence:

Consider control-flow variables as one of the dependent variables

Memory requirements

Periodic checkpointing
FUTURE WORK
Extend the current framework to automatically identify potential data structures for lossy compression

Other number representation formats that might be less susceptible to silent data corruptions or precision errors

Root-cause for sensitivity to errors

Explore the application of AD in Approximate Computing
THANK YOU!

QUESTIONS?

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number(s) DE- SC0014098 and LDRD project 17-SI-004.