ADAPT Floating-Point Precision Tuning

Ignacio Laguna, Harshitha Menon, Tristan Vanderbruggen
Lawrence Livermore National Laboratory

Michael Bentley, Ian Briggs, Ganesh Gopalakrishnan
University of Utah

Cindy Rubio González
University of California at Davis

ADAPT : Algorithmic Differentiation Applied to Floating-Point Precision Tuning

- 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
- Manually optimizing for mixed precision is challenging

GOAL

Develop an automated analysis technique for using the lowest precision sufficient to achieve a desired output accuracy to improve running time and reduce power and memory pressure.

ADAPT

- Estimates the output error due to lowering the precision
- Identifies variables that can be in lower precision
- Automatic floating-point sensitivity analysis
  - Identifies critical code regions that need to be in higher precision

ADAPT APPROACH

Used first order Taylor series approximation to estimate the rounding errors in variables.

$$\Delta y = f'(a)\ \Delta x \text{ for } y=f(x) \text{ at } x=a$$

Generalizing it

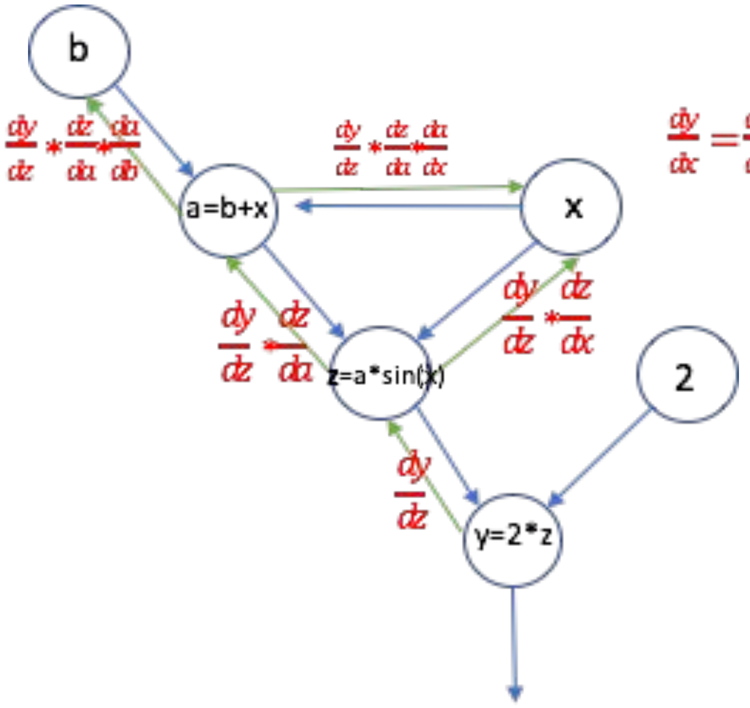
$$\Delta y = f_{x1}'(a_1)\ \Delta x_1 + \ldots + f_{xn}'(a_n)\ \Delta x_n \text{ for } y=f(x_{1,} x_2,\ldots,x_n) \text{ at } x_i=a_i$$

Obtained f'(a) at x=a using algorithmic differentiation (AD)

Reverse mode of AD - all the variables with respect to the output in a single execution.

REVERSE MODE OF ALGORITHMIC DIFFERENTIATION

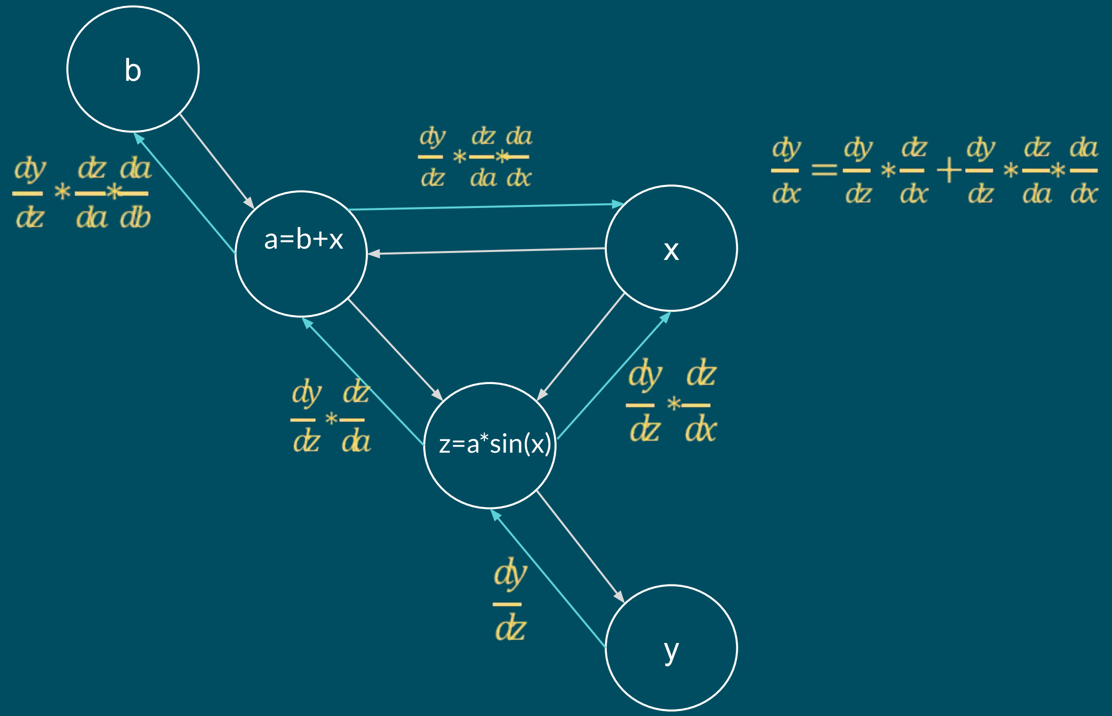
a = b+x
z = a*sin(x);
y = 2*z;



$$\frac{dy}{dx} = \frac{dy}{dz} * \frac{dz}{dx} + \frac{dy}{dz} * \frac{dz}{da} * \frac{da}{dx}$$

REVERSE MODE OF ALGORITHMIC DIFFERENTIATION

a = b + x;
z = a * sin(x);
y = 2 * z;



OUTPUT ERROR ESTIMATION

Obtain f_{xi}'(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

Estimate the error due to lowering the precision of every dynamic instance of a variable

Aggregate the error over all dynamic instance of the variable

Greedy approach

- Sort variables based on error contribution
- Variables switched to lower precision - estimated error contribution within threshold

Source code available:
https://github.com/LLNL/adapt-fp

Questions?

Author contact: harshitha@llnl.gov

Harshitha Menon, Michael O. Lam, Daniel Osei-Kuffuor, Markus Schordan, Scott Lloyd, Kathryn Mohror, Jeffrey Hittinger. ADAPT: Algorithmic Differentiation Applied to Floating-Point Precision Tuning. In Proceedings of SC'18.

Exercises

Exercises with ADAPT

1. Annotate the code with ADAPT annotations
2. Specify the tolerated output error
3. Compile and run the code
4. Output:
    a. Variables that can be converted to lower precision and the expected output error.
    b. Floating-point precision profile.

Directory Structure

```
/Module-ADAPT
     |---/exercise-1
     |---/exercise-2
     |---/exercise-3
     |---/exercise-4
     |---/exercise-5
```

Exercise 1

Exercise 1: Compiling with ADAPT

- Open Makefile file
- Take a look at this compilation options:
    - FLAGS = -I/opt/adapt-install/CoDiPack/include -I/opt/adapt-install/adapt-fp
- Open exercise1-adapt.cpp
- Take a look at the annotations
    - AD_Begin()
    - AD_INTERMEDIATE
    - AD_INDEPENDENT
    - AD_report()
- Execute:
    - $ make clean
    - $ make

Exercise 1: Output

```
$ make
g++-7 -O3 -Wall -o simpsons simpsons.cpp -lm
g++-7 -O3 -Wall --std=c++11 -I/opt/adapt-install/CoDiPack/include
-I/opt/adapt-install/adapt-fp -DCODI_ZeroAdjointReverse=0
-DCODI_DisableAssignOptimization=1 -o simpsons-adapt simpsons-adapt.cpp -lm
```

Exercise 1: Evaluate using ADAPT

- Run the code:
  - ./run-exercise1.sh
- Internally the scripts runs:
  - ./simpsons
  - ./simpsons-adapt

Output error threshold set

ADAPT output

Estimated output error

```
$ sh run-exercise1.sh
============ All variables in double precision ============

ans: 2.000000000067576e+00

============ ADAPT Floating-Point Analysis ============

ans: 2.000000000067576e+00
Output error threshold : 1.000000e-07
=== BEGIN ADAPT REPORT ===
8000011 total independent/intermediate variables
1 dependent variables
Mixed-precision recommendation:
  Replace variable a        max error introduced: 0.000000e+00  count: 1          totalerr: 0.000000e+00
  Replace variable b        max error introduced: 0.000000e+00  count: 1          totalerr: 0.000000e+00
  Replace variable h        max error introduced: 4.152677e-15  count: 1          totalerr: 4.152677e-15
  Replace variable pi       max error introduced: 9.154282e-14  count: 1          totalerr: 9.569550e-14
  Replace variable xarg     max error introduced: 5.523091e-13  count: 2000002    totalerr: 6.480046e-13
  Replace variable result   max error introduced: 2.967209e-11  count: 2000002    totalerr: 3.032010e-11
  DO NOT replace   s1       max error introduced: 3.932171e-02  count: 2000002    totalerr: 3.932171e-02
  DO NOT replace   x        max error introduced: 4.219682e-02  count: 2000001    totalerr: 8.151854e-02
=== END ADAPT REPORT ===
```

Exercise 2

Exercise 2: Evaluate suggested mixed precision and all float

1. Open simpsons-mixed.cpp
2. Take a look at the variables converted to lower precision

```
float pi;

float fun(float xarg) {
  float result;
  result = sin(pi * xarg);
  return result;
}

int main( int argc, char **argv) {


  const int n = 1000000;
  float a; float b;
  float h; double s1; double x;
  ...
}
```

Exercise 2: Run mixed precision and all float

- Run make:
  - make
- Run the different versions:
  - ./run_exercise2.sh
- Internally the script runs:
  - ./simpsons
  - ./simpsons-float
  - ./simpsons-mixed

```
$ make
g++-7 -O3 -Wall -o simpsons simpsons.cpp -lm
g++-7 -O3 -Wall -o simpsons-float simpsons-float.cpp -lm
g++-7 -O3 -Wall -o simpsons-mixed simpsons-mixed.cpp -lm


$ sh run-exercise2.sh
============ All variables in double precision ============

ans: 2.000000000067576e+00

============ All variables in float ============

ans: 2.038122653961182e+00 output error: 3.81227e-02

============ Mixed precision version ============

ans: 2.000000000020178e+00 output error: 4.73981e-11
```

Mixed precision:
Output error: 4.73e-11
ADAPT predicted error: 3.03e-11

All float:
Output error: 3.81e-02
ADAPT predicted error: 8.15e-02

Exercise 3

Exercise 3: Floating-Point analysis of HPCCG

- HPCCG
  - Mini-application from the Mantevo benchmark suite
  - Conjugate gradient benchmark code
- We look at mixed precision suggestion given by ADAPT

Exercise 3: HPCCG example

- Compile HPCCG
  - make
- Run HPCCG
  - sh run-exercise3.sh
- Internally the script runs
  - ./test_HPCCG 20 30 160

```
Initial Residual = 1358.72
Iteration = 10   Residual = 66.0369
Iteration = 20   Residual = 0.87865
Iteration = 30   Residual = 0.0151087
Iteration = 40   Residual = 0.000381964
...
Iteration = 99   Residual = 7.8055e-15
Mini-Application Name: hpccg
Mini-Application Version: 1.0
Parallelism:
  MPI not enabled:
  OpenMP not enabled:
Dimensions:
  nx: 20
  ny: 30
  nz: 160
Number of iterations: : 99
Final residual: : 7.8055e-15
********** Performance Summary (times in sec) ***********:
Time Summary:
  ...
Difference between computed and exact (residual)  = 2.8866e-15
```

Exercise 3: HPCCG example with ADAPT

- Compile with ADAPT
  - cd adapt/
  - make
- Run with ADAPT
  - sh run-hpccg-adapt.sh

```
$ sh run-hpccg-adapt.sh

Initial Residual = 1358.72
Iteration = 10   Residual = 66.0369
Iteration = 20   Residual = 0.87865
...

=== BEGIN ADAPT REPORT ===
28704396 total independent/intermediate variables
1 dependent variables
Mixed-precision recommendation:
  Replace variable x:main.cpp:180           max error introduced: 0.000000e+00  count: 96000       totalerr:
0.000000e+00
  Replace variable b:main.cpp:181           max error introduced: 0.000000e+00  count: 96000       totalerr:
0.000000e+00
  Replace variable normr:HPCCG.cpp:105      max error introduced: 0.000000e+00  count: 1           totalerr:
0.000000e+00
  Replace variable normr:HPCCG.cpp:125      max error introduced: 0.000000e+00  count: 99          totalerr:
0.000000e+00
  DO NOT replace   beta:HPCCG.cpp:120       max error introduced: 6.350859e-21  count: 98          totalerr:
6.350859e-21
  DO NOT replace   alpha:HPCCG.cpp:138      max error introduced: 3.593344e-20  count: 99          totalerr:
4.228429e-20
  DO NOT replace   alpha:HPCCG.cpp:137      max error introduced: 5.615825e-20  count: 99          totalerr:
9.844254e-20
  DO NOT replace   r:HPCCG.cpp:142          max error introduced: 2.051513e-08  count: 9504000     totalerr:
2.051513e-08
  DO NOT replace   Ap:HPCCG.cpp:135         max error introduced: 4.205647e-08  count: 9504000     totalerr:
6.257160e-08
  DO NOT replace   x:HPCCG.cpp:140          max error introduced: 1.854875e-07  count: 9504000     totalerr:
2.480591e-07
=== END ADAPT REPORT ===
```

Exercise 4

Exercise 4: Floating-Point analysis of HPCCG across iterations

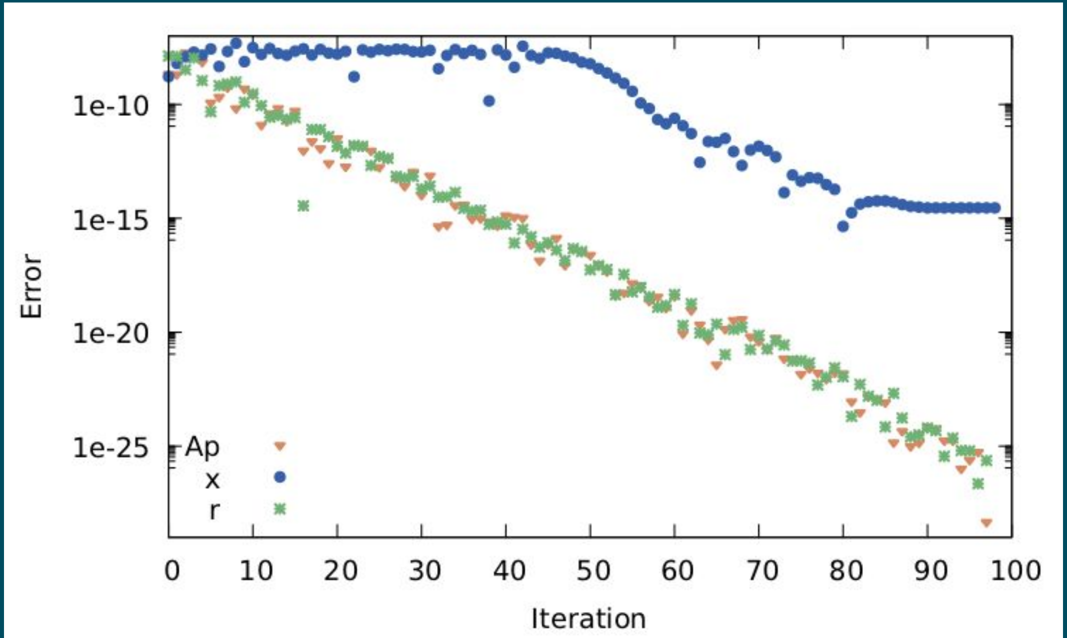
- HPCCG is an iterative application
- We evaluate floating-point sensitivity of variables across different iterations

Exercise 4: HPCCG example with ADAPT

- Compile with ADAPT
    - make
- Run with ADAPT
    - sh run-hpccg-adapt.sh

After 20 iterations error from *Ap* and *r* are below 1.0e-10

After 60 iterations error in *x* below 1.0e-10



Exercise 5

Exercise 5: Mixed precision iteration of HPCCG

- Runs first 60 iterations in doubles and then in float
- Compile and run
  - make
  - sh run-exercise5.sh
- Output error within threshold

```
Initial Residual = 1358.72
Iteration = 10   Residual = 66.0369
Iteration = 20   Residual = 0.87865
Iteration = 30   Residual = 0.0151087
Iteration = 40   Residual = 0.000381964
...
Iteration = 99   Residual = 7.81946e-15
Mini-Application Name: hpccg
Mini-Application Version: 1.0
Parallelism:
  MPI not enabled:
  OpenMP not enabled:
Dimensions:
  nx: 20
  ny: 30
  nz: 160
Number of iterations: : 99
Final residual: : 7.81946e-15
********** Performance Summary (times in sec) ***********:
Time Summary:
  ...
```