ADAPT / FloatSmith
Floating-Point Precision Tuning

Ignacio Laguna, Harshitha Menon
Lawrence Livermore National Laboratory

Michael Bentley, Ian Briggs, Pavel Panchekha, Ganesh Gopalakrishnan
University of Utah

Hui Guo, Cindy Rubio González
University of California at Davis

Michael O. Lam
James Madison University

http://fpanalysistools.org/

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344, via LDRD project 17-SI-004 (LLNL-PRES-796478).

CONTEXT

- 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 APPROACH

Uses 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.

ALGORITHMIC DIFFERENTIATION (AD)

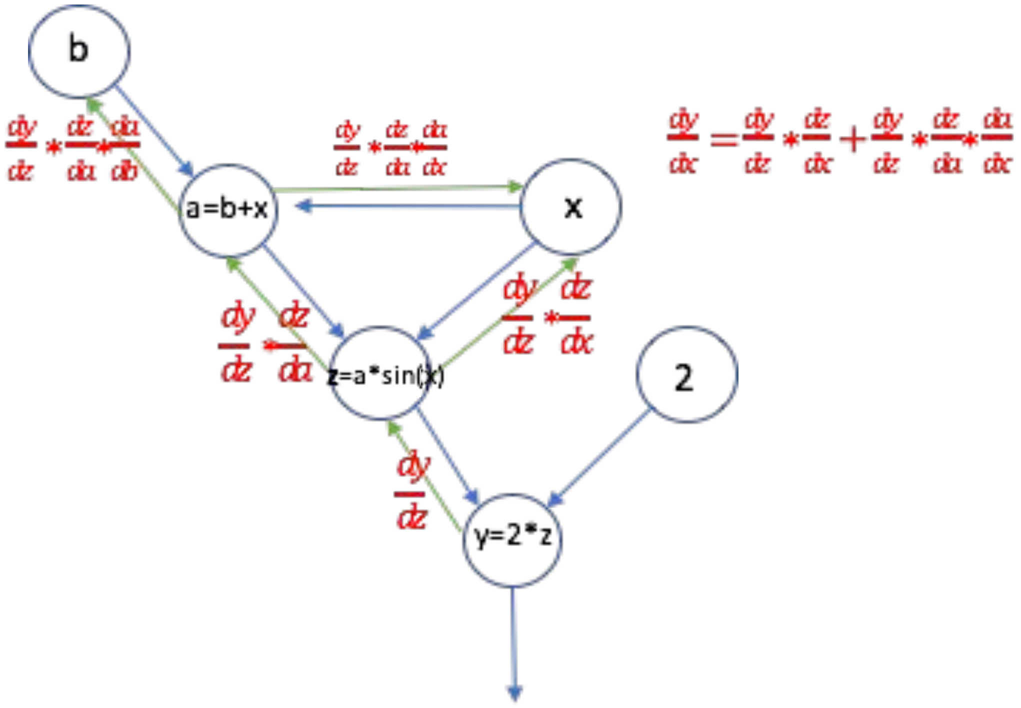
Compute the derivative of the output of a function with respect to its inputs

- A program is a sequence of operations
- Apply the chain rule of differentiation
- AD has been used in sensitivity analysis in various domains
- AD tools: CoDiPack, Tapenade

Alternatives to AD : Symbolic differentiation, Finite difference

REVERSE MODE OF ALGORITHMIC DIFFERENTIATION

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

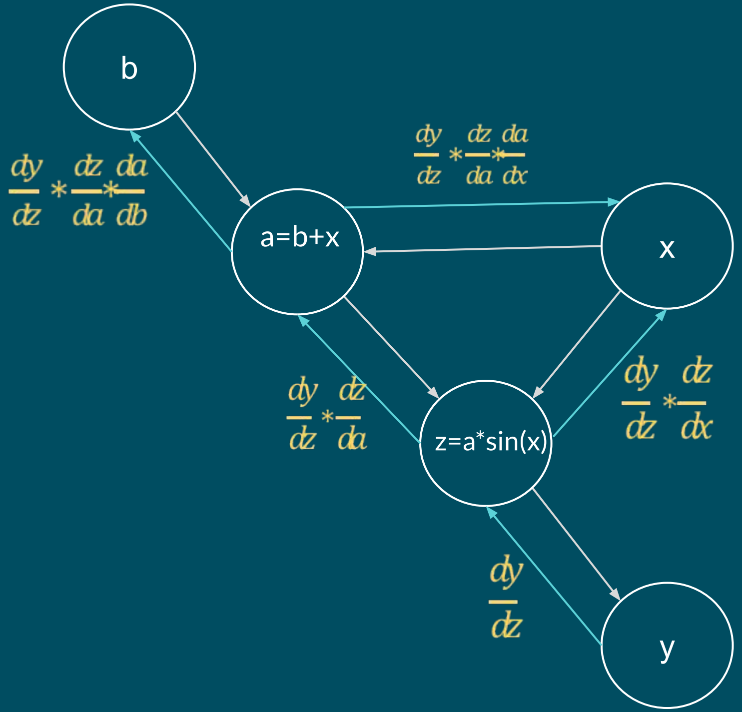


ADAPT

- Estimate the output error due to lowering the precision
- Identify variables that can be in lower precision
- Use mixed-precision to achieve a desired output accuracy while improving performance
- Automatic floating-point sensitivity analysis
    - Identifies critical code regions that need to be in higher precision

REVERSE MODE OF ALGORITHMIC DIFFERENTIATION

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



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

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

LIMITATIONS OF ADAPT

Analysis limited to the input used

  Use representative datasets

Control-flow divergence

  Consider control-flow variables as one of the dependent variables

Memory requirements

  Periodic checkpointing

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

Questions?

Author contacts: lam2mo@jmu.edu, harshitha@llnl.gov

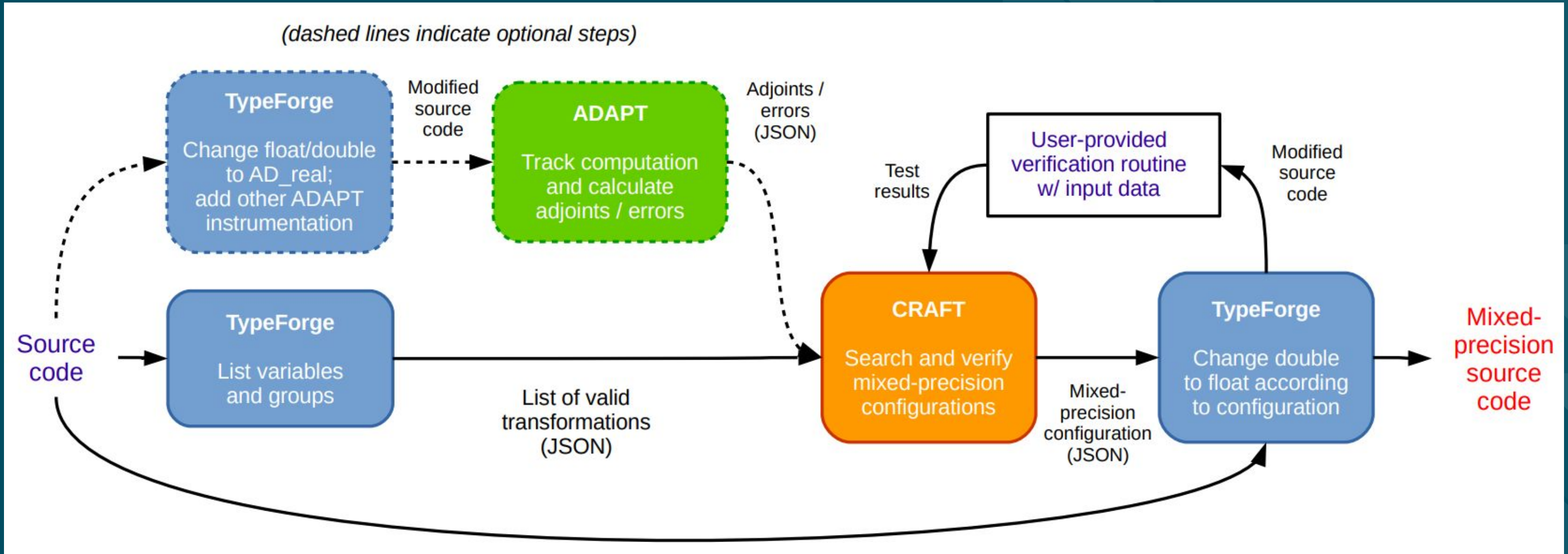
Harshitha Menon, Michael O. Lam, Daniel Osei-Kuffuor, Markus Schordan, Scott Lloyd, Kathryn Mohror, Jeffrey Hittinger. ADAPT: Algorithmic Differentiation Applied to Floating-Point Precision Tuning. In Proceedings of SC'18.

TOOL INTEGRATION FOR MIXED PRECISION

- Goal: automate source-level mixed-precision search and prototyping
- Method: integrate three existing software tools
  - TypeForge (detects possible changes; performs source translation)
  - CRAFT (searches for speedup)
  - ADAPT (optional; used to narrow search space for CRAFT)
- Result: automated pipeline requiring minimal user input (FloatSmith)
  - E.g., for a simple Make-based projects where the output should remain unchanged:

```
floatsmith -B --run "./your_program"
```

FLOATSMITH


(dashed lines indicate optional steps)
TypeForge
Change float/double to AD_real; add other ADAPT instrumentation
Modified source code
ADAPT
Track computation and calculate adjoints / errors
Adjoints / errors (JSON)
User-provided verification routine w/ input data
Test results
Modified source code
Source code
TypeForge
List variables and groups
List of valid transformations (JSON)
CRAFT
Search and verify mixed-precision configurations
Mixed-precision configuration (JSON)
TypeForge
Change double to float according to configuration
Mixed-precision source code

MIXED-PRECISION SEARCHING

- Reduce search space w/ recommendations from ADAPT
  - Only consider recommended replacements
- Reduce search space w/ static analysis info from TypeForge
  - Identify type dependencies
  - Only consider feasible change sets
- Vary search strategy in CRAFT
  - Combinational, compositional, delta-debugging, and hierarchical+compositional

SEARCH STRATEGIES

- Combinational
  - All combinations--not feasible for most programs
- Compositional
  - Try each variable individually then compose passing changes
- Delta debugging
  - Binary search (algorithm from Precimonious)
- Hierarchical + Compositional
  - Breadth-first search on program structure, then compositional

Source code available:
https://github.com/crafthpc/floatsmith

Docker container available:
https://hub.docker.com/r/lam2mo/floatsmith

Questions?

Author contact: lam2mo@jmu.edu

Michael O. Lam, Tristan Vanderbruggen, Harshitha Menon, Markus Schordan. Tool Integration for Source-Level Mixed Precision. To appear, Correctness'19 workshop at SC'19.

Workshop presentation TOMORROW at 12:00pm (noon) in room 505

Exercises

Exercises with ADAPT and FloatSmith

1. ADAPT
   a. Annotate the code with ADAPT annotations
   b. Specify the tolerated output error
   c. Compile and run the code
2. FloatSmith
   a. Specify how to run the code

```
/Module-ADAPT_Floatsmith
     |---/exercise-1
     |---/exercise-2
     |---/exercise-3
     |---/exercise-4
     |---/exercise-5
     |---/exercise-6
```

Exercise 1

Exercise 1: Compiling with ADAPT

- Open Makefile file
- Note ADAPTFLAGS options (must include ADAPT and CoDiPack)
- Open simpsons-adapt.cpp
- Take a look at the annotations
  - AD_begin()
  - AD_INDEPENDENT()
  - AD_INTERMEDIATE()
  - AD_DEPENDENT()
  - AD_report()
- Execute:
  - $ make clean
  - $ make

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: Run with FloatSmith

- Open run-exercise3.sh
  - Note environment variables
  - Most dependencies are just git clones
  - TypeForge requires Rose compiler framework
- Command: `floatsmith -B --run "./simpsons" --adapt`
  - -B "batch" mode; no interactive questions
  - --run how to invoke program (built by default with "make")
  - --adapt use ADAPT to narrow search

Exercise 3: Run with FloatSmith

- Run run-exercise3.sh
  - Note similar ADAPT results (now via automated instrumentation)
  - Search to find speedup (none found)
- Examine `.floatsmith/search/final/simpsons.cpp`
  - Same (non-speedup) replacement as in Exercise 2
  - Can build with "make" and run with "./simpsons"

```
=== BEGIN ADAPT REPORT ===
6000010 total independent/intermediate variables
1 dependent variables
Mixed-precision recommendation:
  Replace variable ::main(int,char **,)::b
  Replace variable ::main(int,char **,)::a
  Replace variable ::main(int,char **,)::h
  Replace variable pi
  Replace variable ::fun(double,)::result
  DO NOT replace   ::main(int,char **,)::x
  DO NOT replace   ::main(int,char **,)::s1
=== END ADAPT REPORT ===
```

```
Total candidates:                    5
Total configs tested:               31
  Total executed:                   31
  Total passed:                     31
  Total failed:                      0
  Total aborted:                     0
Done.  [Total elapsed walltime: 0:01:12]
```

```
...
  float a;
  float b;
  float h;
  double s1;
  double x;
...
```

Exercise 4

Exercise 4: Speedup with FloatSmith

- Open axpy.cpp
  - Vectorizable arithmetic
- Open run-exercise4.sh
  - Command: `floatsmith -B --run "./axpy"`
  - No ADAPT here due to memory requirements
- Run run-exercise4.sh
  - "Speedup achieved!"

```
Candidate queue exhausted.  [Max queue length: ~3 item(s)]
Generating final configuration ... Done.
Testing final configuration ... Success!

          Top instrumented (passed):                Runtime (s)          Speedup (X)
            - a+x                                   0.92                 1.49
            - x                                     0.94                 1.46
            - a                                     1.37                 1.00

          Speedup achieved! (max: 1.49x, baseline: 1.37s)


Total candidates:                     3
Total configs tested:                 4
  Total executed:                     4
  Total passed:                       3
  Total failed:                       1
  Total aborted:                      0
Done.  [Total elapsed walltime: 0:01:33]
```

Exercise 4: Speedup with FloatSmith

- Examine `.floatsmith/search/final/axpy.cpp`
  - Can build with "`CXX=g++-7 make`" and run with "`./axpy`"
  - Run with "`/usr/bin/time -v ./axpy`"
    - Resident set size reduced by 25%
    - Page faults reduced by 25%

```
// can be float
float a = 10.0;
// can be float
float x[100000000];
// must be double
double y[100000000];
```

```
ORIGINAL:
    Command being timed: "./axpy"
    User time (seconds): 0.70
    System time (seconds): 0.66
    Percent of CPU this job got: 100%
    Elapsed (wall clock) time (h:mm:ss or m:ss): 0:01.37
    ...
    Maximum resident set size (kbytes): 1564080
    ...
    Minor (reclaiming a frame) page faults: 390685
```

```
MIXED-PRECISION:
    Command being timed: "./axpy"
    User time (seconds): 0.42
    System time (seconds): 0.49
    Percent of CPU this job got: 100%
    Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.92
    ...
    Maximum resident set size (kbytes): 1173480
    ...
    Minor (reclaiming a frame) page faults: 293029
```

Exercise 5

Exercise 5: Floating-Point analysis of HPCCG

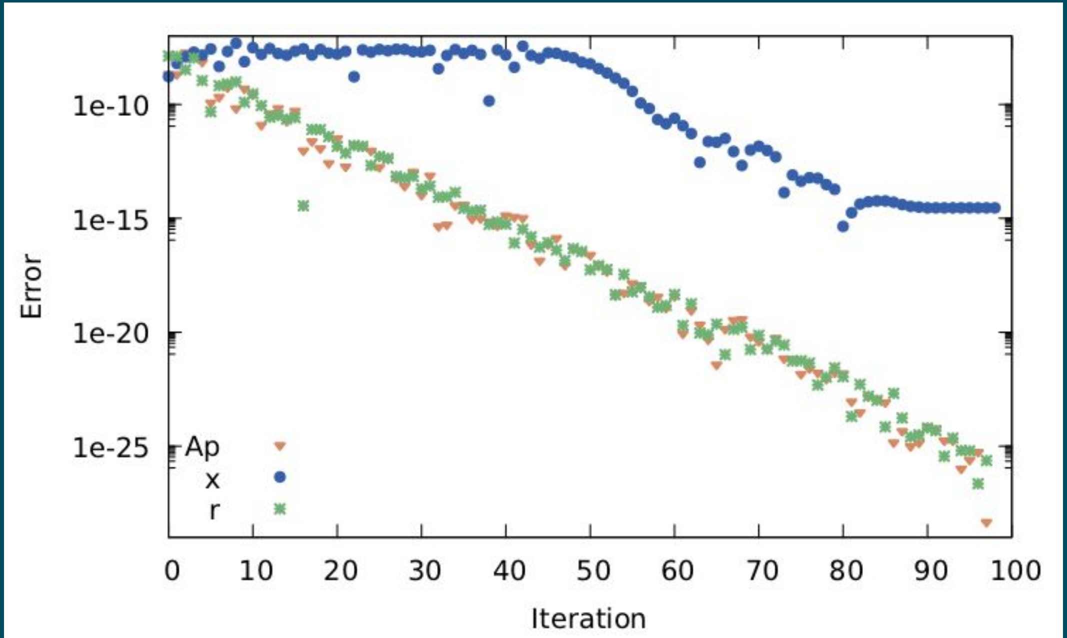
- HPCCG
  - Mini-application from the Mantevo benchmark suite
  - Conjugate gradient benchmark code
- HPCCG is an iterative application
  - We evaluate floating-point sensitivity of variables across different iterations

Exercise 5: HPCCG example with ADAPT

- Compile with ADAPT
  - make
- Run with ADAPT
  - sh run-exercise5.sh
- View resulting graph
  - evince variter.pdf

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

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



Exercise 6

Exercise 6: Mixed precision version of HPCCG

- Runs first 60 iterations in doubles and then in float
- Compile and run
  - make
  - sh run-exercise6.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:
  ...
```