Automatic Generation of Mixed-Precision Programs

Logan Moody, Nathan Pinnow, Michael O. Lam, Harshitha Menon, Markus Schordan, G. Scott Lloyd, Tanzima Islam

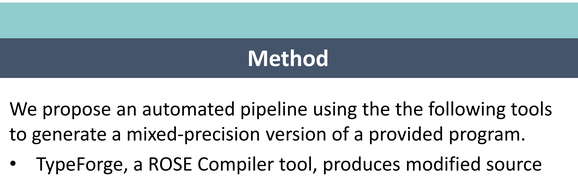
Lawrence Livermore National Laboratory

JAMES MADISON UNIVERSITY

WESTERN WASHINGTON UNIVERSITY

Motivation

- HPC applications use floating point operations extensively and computer architecture supports multiple levels of precision.
  - Higher precision improves accuracy.
  - Lower precision improves performance.



Solution

Goal: Automatically produce a mixed-precision version using 64 and 32 bit variable sizes depending on the level of accuracy needed.

Benefits:

- Minimal development time spent optimizing program
- Provide modified source code to the user
- Maintains specified level of accuracy
- Minimize runtime of the program

```
Original:
    double precise = 1.00000003;
    double loose   = 0.00000003;
    double output = precise + loose;
    printf("output: %.8g\n", output);
Output: 1.0000001

Uniform single:
    float precise = 1.00000003;
    float loose   = 0.00000003;
    float output = precise + loose;
    printf("output: %.8g\n", output);
Output: 1

Mixed-precision:
    double precise = 1.00000003;
    float loose   = 0.00000003;
    float output = precise + loose;
    printf("output: %.8g\n", output);
Output: 1.0000001
```

Method

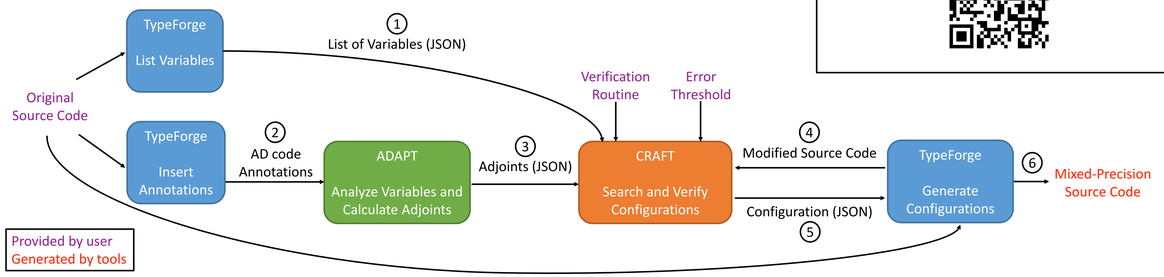
We propose an automated pipeline using the the following tools to generate a mixed-precision version of a provided program.

- TypeForge, a ROSE Compiler tool, produces modified source code and provides information about the program.
- ADAPT analyses variables to refine the search space.
- CRAFT uses the search space and modified source to test configurations and determines the optimal precision mixture.

Pipeline

- The user provides the original source code, verification routine, and an error threshold, then the pipeline will produce a mixed-precision version of the source.
- A JSON data interchange format was designed to facilitate communication between the tools and to allow for easy extensions in the future.

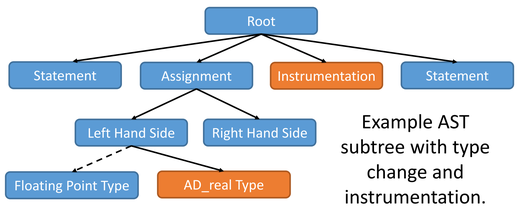
If you would like to learn more about the tool pipeline, scan the QR code below.



TypeForge

Initial Configuration:

1. Generate list of possible replacements with unique identifiers based on variable declaration.
2. Locate all assignments to a floating-point variables and inserts ADAPT instrumentation to instrument variable.
2. Replace floating-point types with AD_real types for ADAPT tracking.
2. Output modified source code with ADAPT instrumentation included.



Example AST subtree with type change and instrumentation.

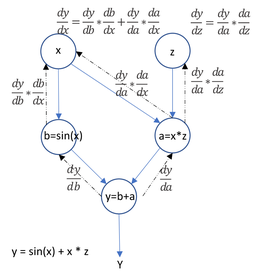
Generate Mixed-Precision:

4. Takes specifications from CRAFT to generate a test configuration.
6. Output final mixed-precision source code for the user.

ADAPT

- Calculates the numerical derivative of a function using reverse automatic differentiation
- A reverse pass calculates the impact on one output by all the inputs
- Stores all intermediate numerical values

3. With this information ADAPT is able to determine which variables in a program are more likely to need higher precision.



CRAFT

5. Generate configurations of variables for TypeForge, which will produce the mixed-precision program.
4. Test the new mixed-precision program to determine if the required error tolerance was maintained and if a speedup was achieved.

The CRAFT tool has also been expanded to include different search strategies which allows it to narrow down the search space, and reduce analysis time.

Results

Benchmarks		SUM2PI	FFT	EP
Floating-point Variables		7	25	63
Candidates		7	24	56
Original Runtime		0.01s	2.01s	2.86s
Configurations For Testing	Combinational	127	3.36E+07	7.21E+16
	Delta Debugging	22	11	222
	ADAPT + Delta Debugging	11	11	-
Pipeline Runtime (mm:ss)	Combinational	1:52	-	-
	Delta Debugging	1:41	2:43	34:54
	ADAPT + Delta Debugging	2:31	3:16	-
Largest Speedup Found	Combinational	1.0x	-	-
	Delta Debugging	1.0x	1.2x	1.0x
	ADAPT + Delta Debugging	1.0x	1.2x	-
Final Optimized Runtime		0.01s	1.69s	2.86s

- Using ADAPT to inform the delta debugging search can reduce the search space.
- The use of ADAPT on a small, trivial program will likely run slower due to the overhead of the AD analysis

Future Work

- We know that running mixed-precision on GPUs will lead to speedups on more codes than currently seen.
- ADAPT cannot currently instrument entire HPC codes at once, to address this, we could determine critical points or conduct multiple passes with varying sets of inputs.
- Continue exploring alternative search strategies for CRAFT in an attempt to reduce analysis time. Possibly implement a machine learning algorithm to narrow the search space.

Acknowledgements

This work has been supported by the LLNL-LDRD Program under Project No. 17-SI-004. The authors would also like to thank Jeff A.F. Hittinger for his support throughout this project.

References

1. Michael O. Lam and J.K. Hollingsworth. 2016. Fine-Grained Floating-Point Precision Analysis. International Journal of High Performance Computing Applications (jun 2016).
2. Harshitha Menon, Michael O. Lam, Daniel Osei-Kuffuor, Markus Schordan, Scott Lloyd, Kathryn Mohror, and Jeff Hittinger. (in press) 2018. ADAPT: Algorithmic Differentiation for Floating-Point Precision Tuning. In The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'18).
3. Dan Quinlan. 2000. ROSE: Compiler Support for Object-Oriented Frameworks. Parallel Processing Letters 10 (2000), 215–226.

LLNL-POST-756004

This work was performed under auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344.