Evaluating Floating-Point Superoptimization with STOKE

Author: Jop Schaap – Supervisor: Dennis Sprokholt - Professor: Soham Chakraborty

1. BACKGROUND INFORMATION

- **Superoptimizers** generally search through all possible programs to find the fastest version of the program supplied at the input [1]
- **STOKE** performs in contrast to most superoptimizers a stochastic search
- A **stochastic search** uses randomness to search through a subset of the entire search space
- This allows **STOKE** to find an optimum faster and for larger programs, however this might not be the true optimum [1]
- STOKE by default, does not formally verify the results instead it relies on randomised tests
- **Floating-point** errors arise normally by the order/type of operation performed because of rounding errors in between operations [2]
    - This also makes that $0.1 + 0.2 - 0.3 \neq 0.3 - 0.1 - 0.2$
- STOKE contains an extension that optimizes **floating-point** programs and allows for defining the maximum precision error [3]

2. RESEARCH QUESTION

What classes of floating-point programs cause STOKE to give well optimized results?

3. METHOD

Three small C programs

The C programs are compiled/ optimized with **GCC version 4.9** and optimized with **STOKE**
- STOKE runs multiple times with different maximum allowed errors

The C programs are compiled/optimized with **GCC version 12.1**
- The flag **ffast-math** is set this allows GCC to violate some constraints of the standard for floating point arithmetic
- The binary is also compiled without this flag

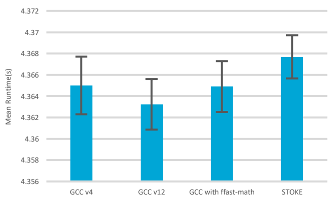
The function runtime is measured

4. STOKE'S PERFORMANCE

Signum

- Returns the **sign** of the input or 0 for 0
- The resulting execution speeds where **not statistically significant**
- The resulting function was incorrect for specific input values
    - For instance: signum(0.0) → 1

```
int signum(double x){
    if(x > 1.0) {
        return 1;
    } else if (x == 0.0) {
        return 0;
    } else {
        return -1;
    }
}
```



Average runtime for 10000000 calls to the signum function after being optimized

Range Sum

- Sums n equally spaced values between start and end
- Can be computed without a loop By realizing that the calculation is very similar to **GAUSS sum**[4]
- The results would then be computed using:

```
double range_sum(double start,
                 double end, int n){
    double total = 0.0;
    double stepSize = (end - start) / n;
    for(int i = 0; i < n; i++) {
        doube current = start
            + ((double) i * stepSize);
        total += current;
    }
    return total;
}
```

$$stepsize * \frac{n * (n+1)}{2} + \frac{start * (start+1)}{2}$$

- Stoke did not find this optimization
- Instead it found one that **malfunctioned** on specific input

```
addl $0x1, %eax
cvtsi2sdl %eax, %xmm2
mulsd %xmm1, %xmm2
```

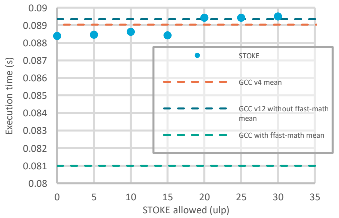
→

```
addsd  %xmm1, %xmm2
```

Incorrect optimization made by STOKE when optimizing the function

Find a Zero

- Takes as input the coefficients for a **third order polynomial** $f(x) = ax^3 + bx^2 + cx + d$, the function then tries to find a **root**(an x s.t. $f(x) \approx 0$) The algorithm uses **hill climbing** in order to find an x s.t. $f(x) \approx 0$
- STOKE obtained a runtime reduction of **1.08%**
    - For allowed error of **0 ulp**
- Increasing allowed error resulted in slower programs



Mean runtime of 100000 iterations the optimized Zero Finder function

5. DISCUSION

- For the tests the best runtime reduction was **1.08%** compared to GCC version 12.1
- During all tests STOKE was unable to generate results comparable to the original study
- Since the experiment only covers 3 algorithms are the results **Not generalizable to all** programs
    - They still can be used for answering the research question

6. CONCLUSION

- STOKE **struggled to find satisfactory optimizations** for all programs presented
- STOKE was never able to generate optimizations that outmatched **GCC with ffast-math** enabled
- The STOKE **test-case generator** fails to generate tests for floating-point number
    - It failed to prevent infinite loops

7. FUTURE WORK

- Future research could focus on experimenting with **different test-case generators**
- To better understand the total capabilities of STOKE, future studies should focus on **different program classes**

[1] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic Superoptimization," ACM SIGARCH Computer Architecture News, vol. 41, pp. 305–316, 3 2013.
[2]D. Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic," ACM Comput. Surv., vol. 23, no. 1, pp. 5–48, Mar. 1991, doi: 10.1145/103162.103163.
[3] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic Optimization of Floating-point Programs With Tunable Precision," ACM SIGPLAN Notices, vol. 49, pp. 53–64, 6 2014.
[4] J. DeMaio, "Counting Triangles to Sum Squares," The College Mathematics Journal,vol. 43, pp. 297–303, Sept. 2012.Number: 4 Publisher: Taylor & Francis _eprint:https://doi.org/10.4169/college.math.j.43.4.297.