Stochastic Optimization of Floating-point Programs with Tunable Precision

eric schkufza, rahul sharma, alex aiken
stanford university
Overview

- **Floating-point kernels**: Nearly ubiquitous in high-performance computing

- **Getting the best performance**: Requires full exploitation of the dark corners of an instruction set and how it interacts with machine resources

- **But we can do even better**: Give up on full precision for applications that don’t need it
Overview

- **Stochastic Optimization**: Automated method for generating high-performance kernels that trade a reduction in precision for improvements in code size and runtime
Goals and Related Work

- **Mixed fixed / floating-point kernels:** Build on previous work [schkufza 13, sharma 13]

- **Improved approximation:** Give user finer-grained control [lam 13, rubio-gonzalez 13]

- **Assembly level:** Direct optimization, no further trusted optimization kernel [sidiroglou-douskos 11]

- **Optimize kernels:** Improve end-to-end performance automatically [chilimbi 10, zou 12]
Example

```
vmovddup %xmm0, %xmm0
vmulpd (%rdi), %xmm0, %xmm2
vroundpd $0, %xmm2, %xmm2
vmulpd 0x10(%rdi), %xmm2, %xmm1
vcvtppd2dq %xmm2, %xmm3
vmulpd 0x20(%rdi), %xmm2, %xmm2
vaddpd %xmm1, %xmm0, %xmm1
vmovapd 0x30(%rdi), %xmm0
vpaddd 0x40(%rdi), %xmm3, %xmm3
vpslld $20, %xmm3, %xmm3
vpshufd $114, %xmm3, %xmm3
vaddpd %xmm2, %xmm1, %xmm1
vmulpd 0x50(%rdi), %xmm1, %xmm2
vaddpd 0x60(%rdi), %xmm2, %xmm2
vaddpd 0x70(%rdi), %xmm2, %xmm2
vmulpd 0x80(%rdi), %xmm2, %xmm2
vaddpd 0x90(%rdi), %xmm2, %xmm2
vaddpd 0xa0(%rdi), %xmm2, %xmm2
vaddpd 0xb0(%rdi), %xmm2, %xmm2
vaddpd 0xc0(%rdi), %xmm2, %xmm2
vaddpd 0xd0(%rdi), %xmm2, %xmm2
vaddpd 0xe0(%rdi), %xmm2, %xmm2
vaddpd 0xf0(%rdi), %xmm2, %xmm2
vaddpd %xmm1, %xmm2, %xmm2
vaddpd %xmm0, %xmm1, %xmm0
vmulpd %xmm3, %xmm0, %xmm0
retq
```

STOKE

```
vmulpd (%rdi), %xmm0, %xmm2
vroundpd $0xfffffffffffffff, %xmm2, %xmm2
vcvtppd2dq %xmm2, %xmm3
vmulpd 0x10(%rdi), %xmm2, %xmm1
vldq %r90(%rdi), %xmm2
vaddpd %xmm1, %xmm0, %xmm1
vmulpd %xmm1, %xmm2, %xmm2
vpaddw 0x40(%rdi), %xmm3, %xmm3
vmovapd 0x30(%rdi), %xmm0
vpsslq $0x84, %xmm3, %xmm3
vaddpd %r1x0(%rdi), %xmm2, %xmm2
vmulpd %xmm1, %xmm2, %xmm2
vaddpd %r1x0(%rdi), %xmm2, %xmm2
vmulsd %xmm1, %xmm2, %xmm2
vaddpd %r1x0(%rdi), %xmm2, %xmm2
vaddpd %r1x0(%rdi), %xmm2, %xmm2
vaddpd %r1x0(%rdi), %xmm2, %xmm2
vaddpd %r1x0(%rdi), %xmm2, %xmm2
vaddpd %r1x0(%rdi), %xmm2, %xmm2
vpshufd $0x3, %xmm3, %xmm3
vmulpd %xmm1, %xmm2, %xmm2
vaddpd %xmm0, %xmm1, %xmm0
vmulpd %xmm3, %xmm0, %xmm0
retq
```
Optimization Notes

- **Smaller kernel**: 38 LOC reduced to 28 LOC

- **Performance improvement**: 57% kernel speedup, produces a 27% overall task speedup

- **Highly specialized**: Obeys application-specific error bound requirements for all inputs between -3.0 and 0
Intuition

Abstract space of programs (*rewrites*): Blue regions contain points that are bit-wise equivalent to the *target*
Intuition

Few opportunities: Floating-point instruction set semantics complicate optimization
Relax correctness: Most applications don’t require full precision results for all possible inputs
New opportunities: Gain access to high performance optimizations that were previous inaccessible.
Intuition

Non-trivial task: Semantics preserving transformations ill-suited to this task; prefer stochastic search [schkufza 13]
What’s Required

1. **Search Procedure**: Markov Chain Monte Carlo Sampling

2. **Cost Function**: Formal encoding of rewrite quality to guide search; should balance competing constraints of precision and performance
MCMC Sampling

• **Widely used:** For many domains, the only known tractable solution method for high dimensional irregular search spaces [andrieu 03][chenney 00]

• **Guarantees:** Draws samples in proportion to their value; higher value points are sampled more frequently

• **No claim of convergence rate:** Works well in practice for the benchmarks that we consider
MCMC Sampling

Algorithm:

1. Select an initial program

2. Repeat (millions to billions of times)
   
   A. Propose a random change and evaluate cost
   
   B. If (decreased) { accept }
   
   C. If (increased) { maybe accept anyway }
Transformations

original

... 
movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...
Transformations

**insert**

```assembly
... movl ecx, ecx shrq 32, rsi andl ff, r9d movq rcx, rax movl edx, edx imulq r9, rax imulq rsi, rdx ...
```

**original**

```assembly
... movl ecx, ecx shrq 32, rsi andl ff, r9d movq rcx, rax movl edx, edx imulq r9, rax imulq rsi, rdx ...
```
Transformations

**insert**

```assemble
...  
movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx
...```

**delete**

```assemble
...  
movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...```
Transformations

**insert**

```assembly
... movl ecx, ecx shrq 32, rsi andl ff, r9d movq rcx, rax movl edx, edx imulq r9, rax imulq rsi, rdx ...```

**delete**

```assembly
... movl ecx, ecx shrq 32, rsi andl ff, r9d movq rcx, rax movl edx, edx imulq r9, rax ...```

**instruction**

```assembly
... movl ecx, ecx shrq 32, rsi andl ff, r9d movq rcx, rax movl edx, edx imulq r9, rax ...```
Transformations

**insert**

```
... 
movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx
... 
```

**delete**

```
... 
movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
... 
```

**opcode**

```
... 
movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
... 
```

**instruction**

```
... 
movl ecx, ecx
shrq 32, rsi
salq 16, rcx
movq rcx, rax
movl edx, edx
movl edx, edx
imulq r9, rax
imulq r9, rax
... 
```
Transformations

**insert**

```
... movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx ...
```

**opcode**

```
... movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax ...
```

**delete**

```
... movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax ...
```

**operand**

```
... movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
subl edx, edx
imulq r9, rax ...
```

**instruction**

```
... movl ecx, ecx
shrq 32, rsi
movq rcx, rax
movl edx, edx
imulq r9, rax ...
```

```
... movl ecx, ecx
shrq 32, rsi
movq rcx, rax
movl edx, edx
imulq r9, rax ...
```
Transformations

**insert**

```
... movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
imulq rsi, rdx
...```

**opcode**

```
... movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...```

**delete**

```
... movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...```

**instruction**

```
... movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
movl edx, edx
imulq r9, rax
...```

**swap**

```
... movl ecx, ecx
movl edx, edx
salq 16, rcx
movq rcx, rax
movl edx, edx
imulq r9, rax
...```

**operand**

```
... movl ecx, ecx
movl edx, edx
shrq 32, rsi
movq rcx, rax
movl edx, edx
imulq r9, rax
...```

```
... movl ecx, ecx
shrq 32, rsi
andl ff, r9d
movq rcx, rax
subl edx, edx
imulq r9, rax
...```
Comparison: Execute target and rewrite on identical state and compare live outputs
Precision Function

- **Uncertainty in Last Place:** Measures the distance between a real number and the closest representable floating-point value.

- **Widely Used:** Most scientific applications measure precision in terms of ULPs; 0.5 is the gold standard but very expensive to obtain, most settle for 1 to 2.
Verification

• **Guarantees:** How do we know that an optimization is “precise enough” for all possible inputs?

• **Decision Procedures:** Bit-blasting techniques don’t scale beyond a few lines of code; can’t handle mixed fixed- and floating-point codes [darulova 14]

• **Abstract Interpretation:** Can’t prove even bitwise equality in many common cases; can’t handle mixed fixed- and floating-point codes [haller 12]

• **Bottom Line:** No standard techniques can do this
Validation

• **Relaxed problem statement:** Claim with high confidence that there is no input that will cause an error in excess of maximum user bound

• **Error function:** Run original and optimized code on identical inputs and measure ULP Error

\[
\text{error}(x) = \text{ULP}(\text{eval}(\text{target}, x), \text{eval}(\text{rewrite}, x))
\]

• **Goal:** Show max error is below user-defined bound
Validation

Test case → test case’

Search: Use MCMC sampling to search for test case inputs that maximize the error function; just one transform
Termination

- **Mixing Tests:** Statistical tests [geweke 92] for producing a high-confidence guarantee that search has sampled uniformly across the domain of a function.

- **Therefore:** High confidence that search has discovered all local maxima implies high confidence that it has discovered the global maximum.
Evaluation

• **Numeric simulation:** S3D, 3-dimensional direct numerical solver for HCCI combustion

• **C library:** libimf, Intel’s hand-written implementation of the C numerics library math.h

• **Computer graphics:** A ray tracer
• **Bit-wise correct:** 30% speedup optimizing vector kernels

• **Depth of field blur:** Random perturbations made to viewing camera angle; 6% speedup by relaxing precision requirements

![Ray Tracer](image)

- bit-wise correct
- relaxed precision
- error pixels (white)
Overfitting

• **Over-relaxation**: If minimum tolerable error exceeds the variance of random perturbations, they are removed altogether.

• **Faster still**: But depth of field blur has been removed.
Summary

• **Micro-optimization**: For many interesting application domains, once data movement is orchestrated correctly, even a single instruction can make a difference.

• **New approach**: Use random search to experiment with imprecise intermediate optimizations.

• **New opportunities for optimization**: Identify applications that can tolerate a loss of precision and produce code that is specialized to that domain.

• **Download**: [www.github.com/eschkufz/stoke-release](http://www.github.com/eschkufz/stoke-release)