

# SIMD Intrinsic on Managed Language Runtimes



Alen Stojanov  
ETH Zurich



Ivaylo Toskov  
ETH Zurich



Tiark Rompf  
Purdue



Markus Püschel  
ETH Zurich

*Published at Proc. Symposium on Code Generation and Optimization (CGO), 2018, pp. 2-15*

ETH zürich

PURDUE  
UNIVERSITY

## Managed Languages



### Pros

- General Purpose
- Robust & Portable
- High-level features

### Cons

- Lack low-level control
- Access to machine specific instructions

# Low-level Languages



## Pros

- Architecture and micro-architecture specific
- Fast code

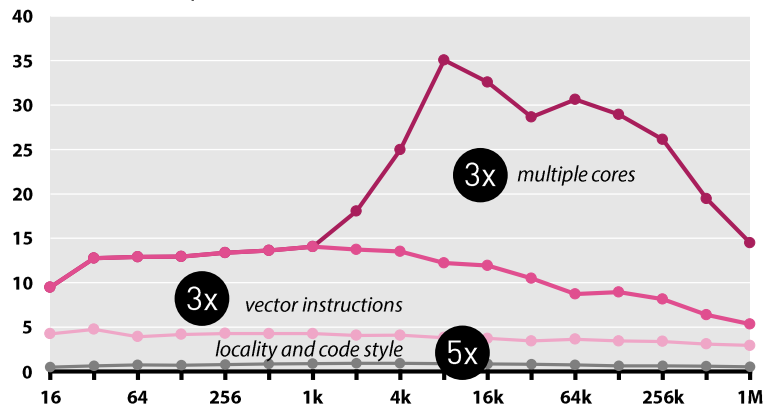
## Cons

- Lack high-level features

3

## Discrete Fourier transform on Intel Core i7 (4 cores)

Performance [Gflop/s]



4

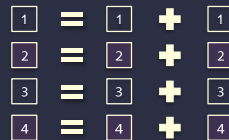
Can we combine benefits of both?

Yes, by supporting SIMD  
in the managed language.

What is SIMD?



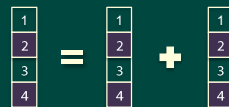
SISD



Single  
Instruction  
Multiple Data



SIMD

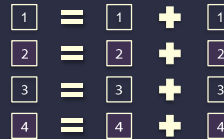


## Scalar

```
#define T double
void add(T* x, T* y, T* z, int N) {
    for(int i = 0; i < N; ++i) {
        T x1, y1, z1;
        x1 = x[i];
        y1 = y[i];
        z1 = x1 + y1;
        z[i] = z1;
    }
}
```



## SISD

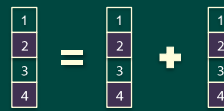


## AVX x4

```
#define T double
void add(T* x, T* y, T* z, int N) {
    for(int i = 0; i < N; i += 4) {
        __m256d x1, y1, z1;
        x1 = _mm256_loadu_pd(x + i);
        y1 = _mm256_loadu_pd(y + i);
        z1 = _mm256_add_pd(x1, y1);
        _mm256_storeu_pd(z + i, z1);
    }
}
```



## SIMD



7

## Scalar

```
#define T double
void add(T* x, T* y, T* z, int N) {
    for(int i = 0; i < N; ++i) {
        T x1, y1, z1;
        x1 = x[i];
        y1 = y[i];
        z1 = x1 + y1;
        z[i] = z1;
    }
}
```



## SISD

```
LBB0_3:
movsd (%rdi,%rax,8), %xmm0
addsd (%rsi,%rax,8), %xmm0
movsd %xmm0, (%rdx,%rax,8)
incq %rax
cpl %eax, %r9d
jne LBB0_3
```

## AVX x4

```
#define T double
void add(T* x, T* y, T* z, int N) {
    for(int i = 0; i < N; i += 4) {
        __m256d x1, y1, z1;
        x1 = _mm256_loadu_pd(x + i);
        y1 = _mm256_loadu_pd(y + i);
        z1 = _mm256_add_pd(x1, y1);
        _mm256_storeu_pd(z + i, z1);
    }
}
```



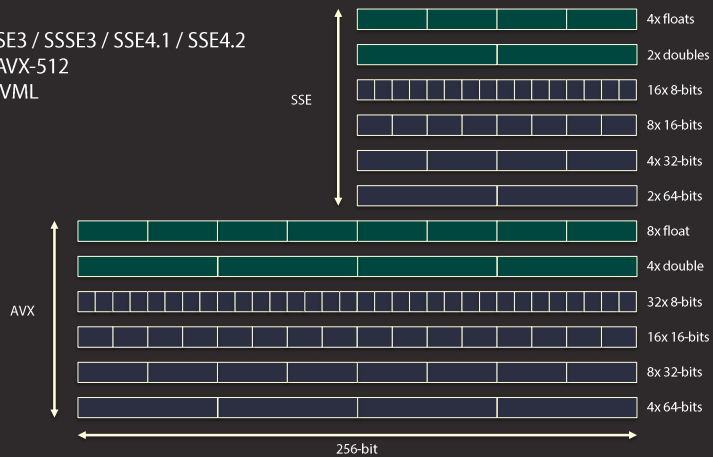
## SIMD

```
LBB0_3:
vmovupd (%rdi,%r10,8), %ymm0
vaddpd (%rsi,%r10,8), %ymm0, %ymm0
vmovupd %ymm0, (%rax)
addq $4, %r10
addq $32, %rax
addq $1, %rcx
jne LBB0_3
```

8

- MMX
- SSE / SSE2 / SSE3 / SSSE3 / SSE4.1 / SSE4.2
- AVX / AVX2 / AVX-512
- FMA / KNC / SVML

operands  
for each



## That's not all

### Shuffles:

- `_mm256_permutevar_pd`
- `_mm256_shufflehi_epi16`
- ...

### Strings:

- `_mm_cmpestrm`
- `_mm_cmpistrm`
- ..

### Bitwise operators:

- `_mm256_bslli_epi128`
- `_mm512_rol_epi32`
- ...

### Statistics:

- `_mm_avg_epu8`
- `_mm256_cdfnorm_pd`
- ...

### Logical:

- `_mm256_or_pd`
- `_mm256_andnot_pd`
- ...

### Crypto:

- `_mm_aesdec_si128`
- `_mm_sha1msg1_epu32`
- ...

### Loads:

- `_mm_i32gather_epi32`
- `_mm256_broadcast_ps`
- ...

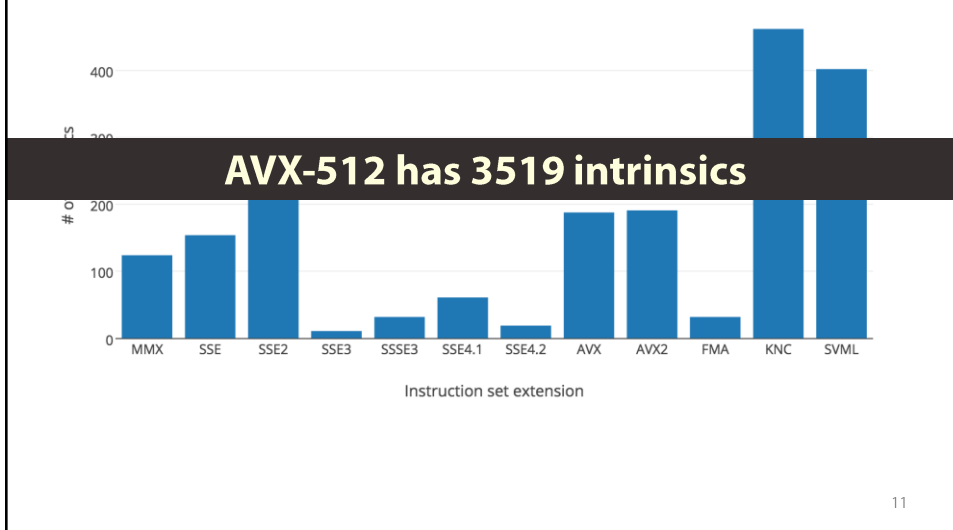
### Stores:

- `_mm512_storenrngo_pd`
- `_mm_store_pd1`
- ...

### Casts:

- `_mm256_castps_pd`
- `_mm256_cvtps_epi32`
- ...

There are *many* SIMD instructions



**How can you include all intrinsics?**  
**Idea #1: Get a Master student to do it**



Ivaylo Toskov  
ETH Zurich

**Idea #2: Generate them automatically**

**Intel Intrinsic Guide**

The Intel Intrinsic Guide is an interactive reference tool for Intel intrinsic instructions, which are C style functions that provide access to many Intel instructions - including Intel® SSE, AVX, AVX-512, and more - without the need to write assembly code.

**Technologies**

- MMX
- SSE
- SSE2
- SSE3
- SSSE3
- SSE4.1
- SSE4.2

<code>__m128i __mm_abs_epi16 (__m128i a)</code>	vpabsw
<code>__m128i __mm_mask_abs_epi16 (__m128i src, __mmask8 k, __m128i a)</code>	vpabsw
<code>__m128i __mm_maskz_abs_epi16 (__mmask8 k, __m128i a)</code>	vpabsw
<code>__m256i __mm256_abs_epi16 (__m256i a)</code>	vpabsw

## data-3.3.16.xml

<input type="checkbox"/> KNC	<code>__m512i __mm512_maskz_abs_epi16 (__mmask32 k, __m512i a)</code>	vpabsw
<input type="checkbox"/> SVMML	<code>__m128i __mm_abs_epi132 (__m128i a)</code>	vpabsw
<input type="checkbox"/> Other	<code>__m128i __mm_mask_abs_epi132 (__m128i src, __mmask8 k, __m128i a)</code>	vpabsw
	<code>__m128i __mm_maskz_abs_epi132 (__mmask8 k, __m128i a)</code>	vpabsw
<b>Categories</b>	<code>__m256i __mm256_abs_epi132 (__m256i a)</code>	vpabsw
<input type="checkbox"/> Application-Targeted	<code>__m256i __mm256_mask_abs_epi132 (__m256i src, __mmask8 k, __m256i a)</code>	vpabsw
<input type="checkbox"/> Arithmetic	<code>__m256i __mm256_maskz_abs_epi132 (__mmask8 k, __m256i a)</code>	vpabsw
<input type="checkbox"/> Bit Manipulation	<code>__m512i __mm512_abs_epi132 (__m512i a)</code>	vpabsw
<input type="checkbox"/> Cast	<code>__m512i __mm512_mask_abs_epi132 (__m512i src, __mmask16 k, __m512i a)</code>	vpabsw
<input type="checkbox"/> Compare	<code>__m512i __mm512_maskz_abs_epi132 (__mmask16 k, __m512i a)</code>	vpabsw
<input type="checkbox"/> Convert		
<input type="checkbox"/> Cryptographic		

13

**But how do we  
"include"  
the intrinsics?**



Use JNI and invoke intrinsics functions directly in Java ?



Modify the JVM?

- Jitrino JIT with JVI
- Panama JVM
- GraalVM



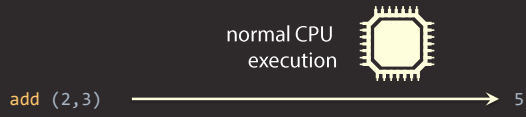
Staging: Embedded DSL +  
Lightweight Modular Staging (LMS)

**what is staging ?**



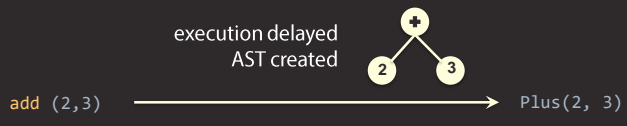
# Non-Staged

```
def add (  
  a: Float,  
  b: Float  
) : Float = {  
  a + b  
}
```



# Staged

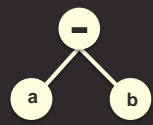
```
def add (  
  a: Rep[Float],  
  b: Rep[Float]  
) : Rep[Float] = {  
  a + b  
}
```



# Staging eDSLs

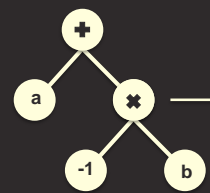
```
float add (  
  float a,  
  float b  
) = {  
  return a - b;  
}
```

Unparsing  
to C code



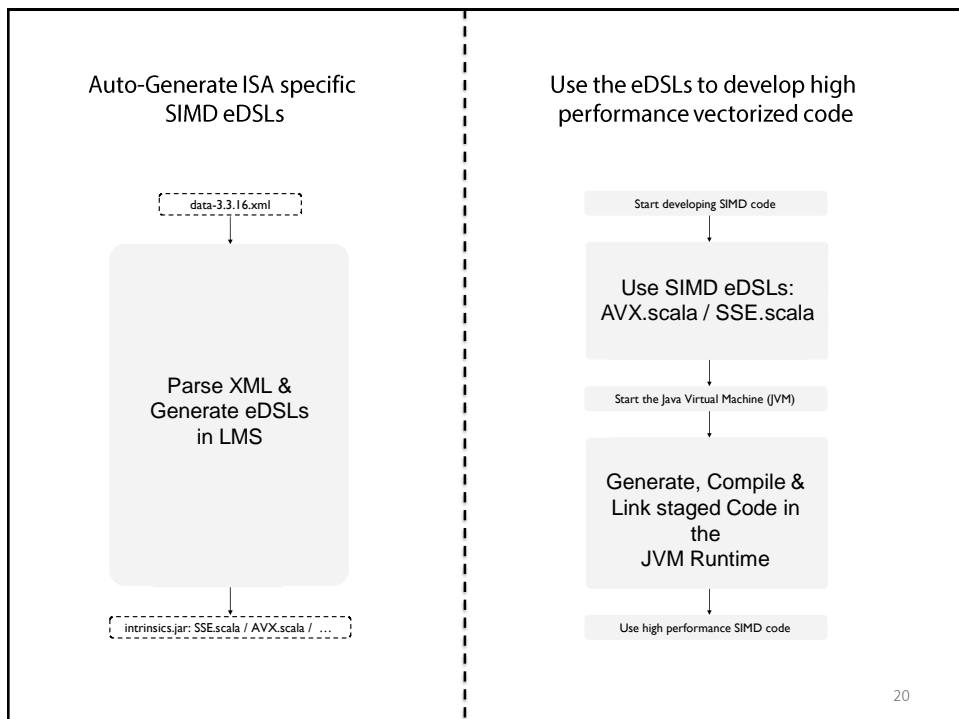
```
def add (  
  a: Rep[Float],  
  b: Rep[Float]  
) : Rep[Float] = {  
  a + b * (-1)  
}
```

Staging



- Write eDSL statements
- Generate AST
- Optimize, rewrite

# How do we make use of staging?



## Auto-Generate ISA specific SIMD eDSLs

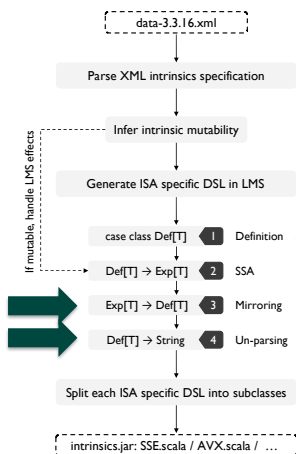


```

<intrinsic tech='AVX' rettype='__m256d'
  name='__mm256_add_pd'>
  <type>Floating_Point</type>
  <CPUID>AVX</CPUID>
  <category>Arithmetic<category>
  <parameter varname='a' type='__m256d' />
  <parameter varname='b' type='__m256d' />
  <instruction name='vaddpd' form='ymm, ymm, ymm' />
  <header>immintrin.h</header>
</intrinsic>
  
```

21

## Auto-Generate ISA specific SIMD eDSLs



```

case class MM256_ADD_PS (a: Exp[__m256], b: Exp[__m256])
  extends IntrinsicDef[__m256] {
  val category = List(IntrinsicsCategory.Arithmetic)
  val intrinsicType = List(IntrinsicsType.FloatingPoint)
  val performance = Map.empty[MicroArchType, Performance]
  val header = "immintrin.h"
}
  
```

```

def __mm256_add_ps(a: Exp[__m256], b: Exp[__m256])
  : Exp[__m256] = {
  MM256_ADD_PS(a, b)
}
  
```

```

override def mirror[A:Typ](e: Def[A], f: Transformer)
  (implicit pos: SourceContext): Exp[A] = (e match {
  case MM256_ADD_PS (a, b) => __mm256_add_ps(f(a), f(b))
  // Pattern match against all other nodes
  case _ => super.emitNode(sym, rhs)
})
  
```

```

override def emitNode(sym: Sym[Any], rhs: Def[Any]) =
  rhs match {
  case iDef@MM256_ADD_PS(a, b) =>
    headers += iDef.header
    emitValDef(sym, s"__mm256_add_ps(${quote(a)}, ${quote(b)})")
    // Pattern match against all other nodes
  case _ => super.emitNode(sym, rhs)
}
  
```

22



## Challenge #3

Type Mappings – unsigned?

- Use Scala Unsigned for unsigned operations.

## Challenge #4

Pointers?

- Disallow and use memory offsets instead

## Challenge #5

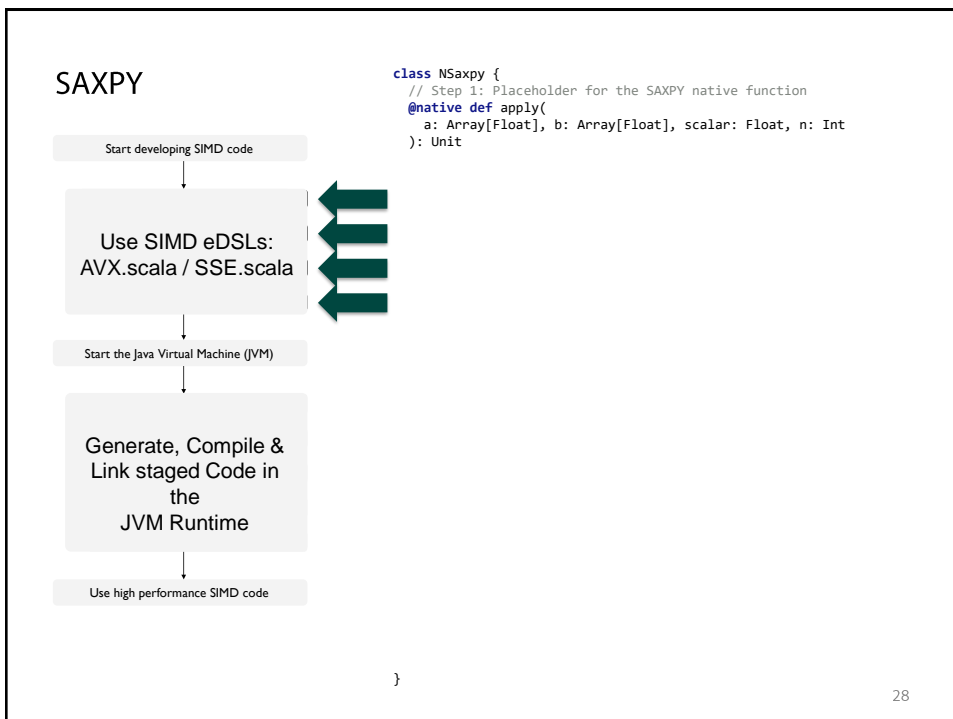
Implement Arrays only?

- Abstract containers for the need of the DSL

The screenshot shows the GitHub repository page for 'ivtoskov / lms-intrinsics'. At the top, there are navigation links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, and Insights. Below this, there is a message: 'No description, website, or topics provided.' The repository statistics show 31 commits, 1 branch, 0 releases, 2 contributors, and Apache-2.0 license. There are buttons for 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The commit history is listed below, showing the latest commit by ivtoskov: 'Fixed a typo in README.md' 5 hours ago. Other commits include 'Separate the implicit ArrayContainer from IntrinsicBase' (14 days ago), 'Added reset method for intrinsics headers.' (9 hours ago), 'Re-generated all intrinsics to accommodate newest changes.' (a day ago), 'Ignore IntelliJ Idea files.' (a month ago), 'Initial commit' (a month ago), 'Fixed a typo in README.md' (5 hours ago), 'Added sbt information for publishing to Maven central.' (23 days ago), 'Adjusted the developer's information to conform with different versio...' (13 days ago), and 'Added sbt information for publishing to Maven central.' (23 days ago).

File	Description	Time
project	Separate the implicit ArrayContainer from IntrinsicBase	14 days ago
src	Added reset method for intrinsics headers.	9 hours ago
stats	Re-generated all intrinsics to accommodate newest changes.	a day ago
.gitignore	Ignore IntelliJ Idea files.	a month ago
LICENSE	Initial commit	a month ago
README.md	Fixed a typo in README.md	5 hours ago
build.sbt	Added sbt information for publishing to Maven central.	23 days ago
publish.sbt	Adjusted the developer's information to conform with different versio...	13 days ago
version.sbt	Added sbt information for publishing to Maven central.	23 days ago

# How do we develop code using the intrinsics ?



## Matrix-Matrix Multiplication In ~40 LOC

Using Scala features:

- Collections and iterators
- Pattern matching
- Closures

```
// Perform Matrix-Matrix-Multiplication
def staged_mmm_blocked (
  a  : Rep[Array[Float]],
  b  : Rep[Array[Float]],
  c_imm : Rep[Array[Float]],
  n  : Rep[Int] // assume n == 8k
): Rep[Unit] = {
  val c_sym = c_imm.asInstanceOf[Sym[Array[Float]]]
  val c = reflectMutableSym(c_sym)
  forloop(0, n, fresh[Int], 8, (kk: Exp[Int]) => {
    forloop(0, n, fresh[Int], 8, (jj: Exp[Int]) => {
      // Load the block of matrix B and transpose it
      val blockB = transpose((0 to 7).map { i =>
        _mm256_loadu_ps(b, (kk + i) * n + jj)
      })
      // Multiply all the vectors of a of the
      // corresponding block column with the running
      // block and store the result in matrix C
      forloop(0, n, fresh[Int], 1, (i: Exp[Int]) => {
        val rowA = _mm256_loadu_ps(a, i * n + kk)
        val mulAB = transpose(
          blockB.map(_mm256_mul_ps(rowA, _))
        )
        def f(l: Seq[Exp[_m256]]): Exp[_m256] = l.size match {
          case 1 => l.head
          case s =>
            val lhs = f(l.take(s/2))
            val rhs = f(l.drop(s/2))
            _mm256_add_ps(lhs, rhs)
        }
        val rowC = _mm256_loadu_ps(c, i * n + jj)
        val accC = _mm256_add_ps(f(mulAB), rowC)
        _mm256_storeu_ps(c, accC, i * n + jj)
      })
    })
  })
}
```

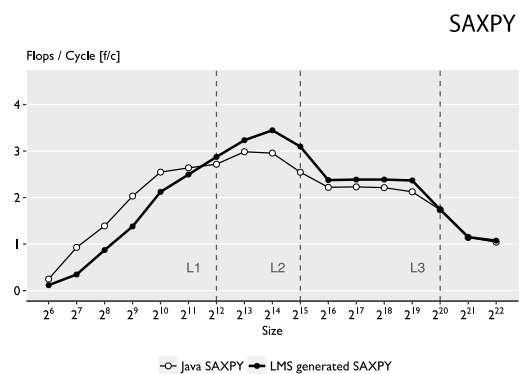
29

Intel(R) Xeon(R)  
E3-1285L v3 3.1 GHz  
AVX2, Debian 8 (jessie)

Intel C++ Composer 17.0.0  
kernel 3.16.43-2+deb8u3

Java: 1.8  
HotSpot: 25.144-b01  
Bench: ScalaMeter

Intel's Hyper-Threading: Off  
Intel Turbo Boost: Off



30

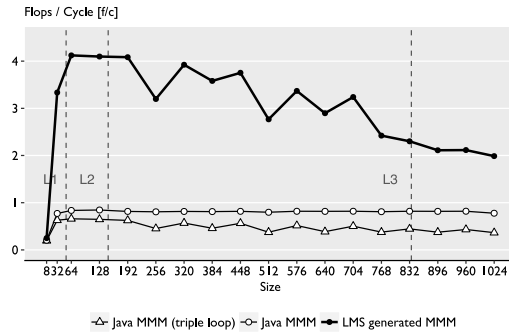
Intel(R) Xeon(R)  
E3-1285L v3 3.1 GHz  
AVX2, Debian 8 (jessie)

Intel C++ Composer 17.0.0  
kernel 3.16.43-2+deb8u3

Java: 1.8  
HotSpot: 25.144-b01  
Bench: ScalaMeter

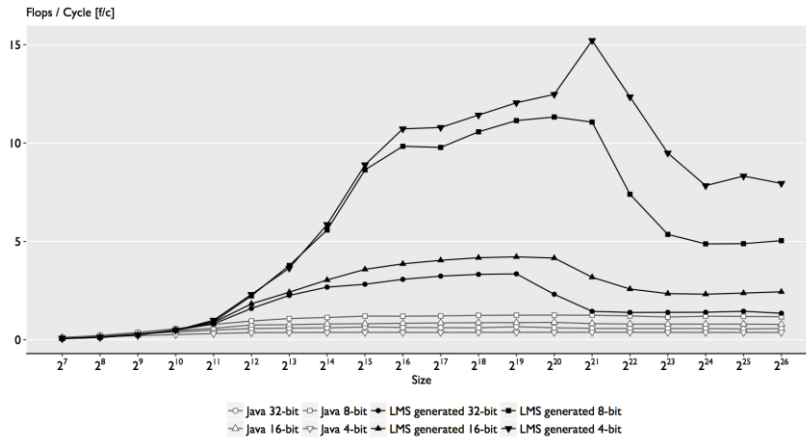
Intel's Hyper-Threading: Off  
Intel Turbo Boost: Off

### Matrix-Matrix-Multiplication



### Low Precision arithmetic

Dot product 32, 16, 8 and 4-bit precision (quantized arrays):  
up to **40x improvements**





astojanov / NGen

Unwatch 1 Star 0 Fork 0

Code Issues Pull requests Projects Wiki Insights Settings

No description, website, or topics provided. Edit

Add topics

2 commits 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

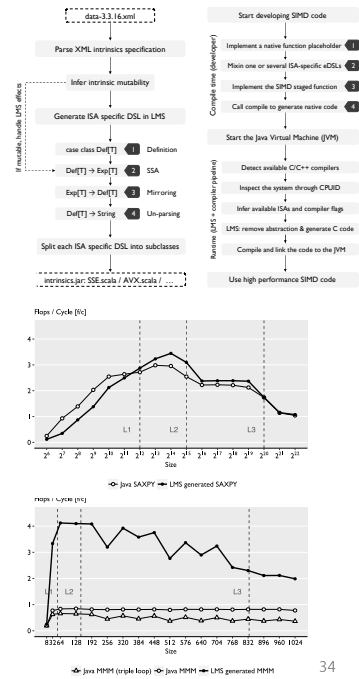
File	Description	Updated
bin/abt	CGO 2018 artifact evaluation.	a month ago
conf	CGO 2018 artifact evaluation.	a month ago
lib/model	CGO 2018 artifact evaluation.	a month ago
project	CGO 2018 artifact evaluation.	a month ago
resources	CGO 2018 artifact evaluation.	a month ago
src/ch/ethz/jael	Update to latest lms-intrinsics snapshot.	3 days ago
test-src/cgo	CGO 2018 artifact evaluation.	a month ago
.gitignore	CGO 2018 artifact evaluation.	a month ago
README.md	CGO 2018 artifact evaluation.	a month ago
build.abt	Update to latest lms-intrinsics snapshot.	3 days ago

Artifacts Available  
Results Reusable  
Artifacts Evaluated Reusable

33

# Summary

- Systematic generation of SIMD eDSL
- Metaprogramming & staging to give back control to the developer
- Support of low level instructions in managed language runtime
- Abstraction with no regret



34