Code vectorization in the JVM: Auto-vectorization, intrinsics, Vector API

Kishor Kharbas

Software Engineer
Intel Corp.

Vladimir Ivanov

HotSpot JVM Compilers
Java Platform Group
Oracle Corp.

September 17, 2019

Safe Harbor

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at http://www.oracle.com/investor. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.

Notices & Disclaimers

Intel processors of the same SKU may vary in frequency or power as a result of natural variability in the production process.

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

The cost reduction scenarios described are intended to enable you to get a better understanding of how the purchase of a given Intel based product, combined with a number of situation-specific variables, might affect future costs and savings. Circumstances will vary and there may be unaccounted-for costs related to the use and deployment of a given product. Nothing in this document should be interpreted as either a promise of or contract for a given level of costs or cost reduction.

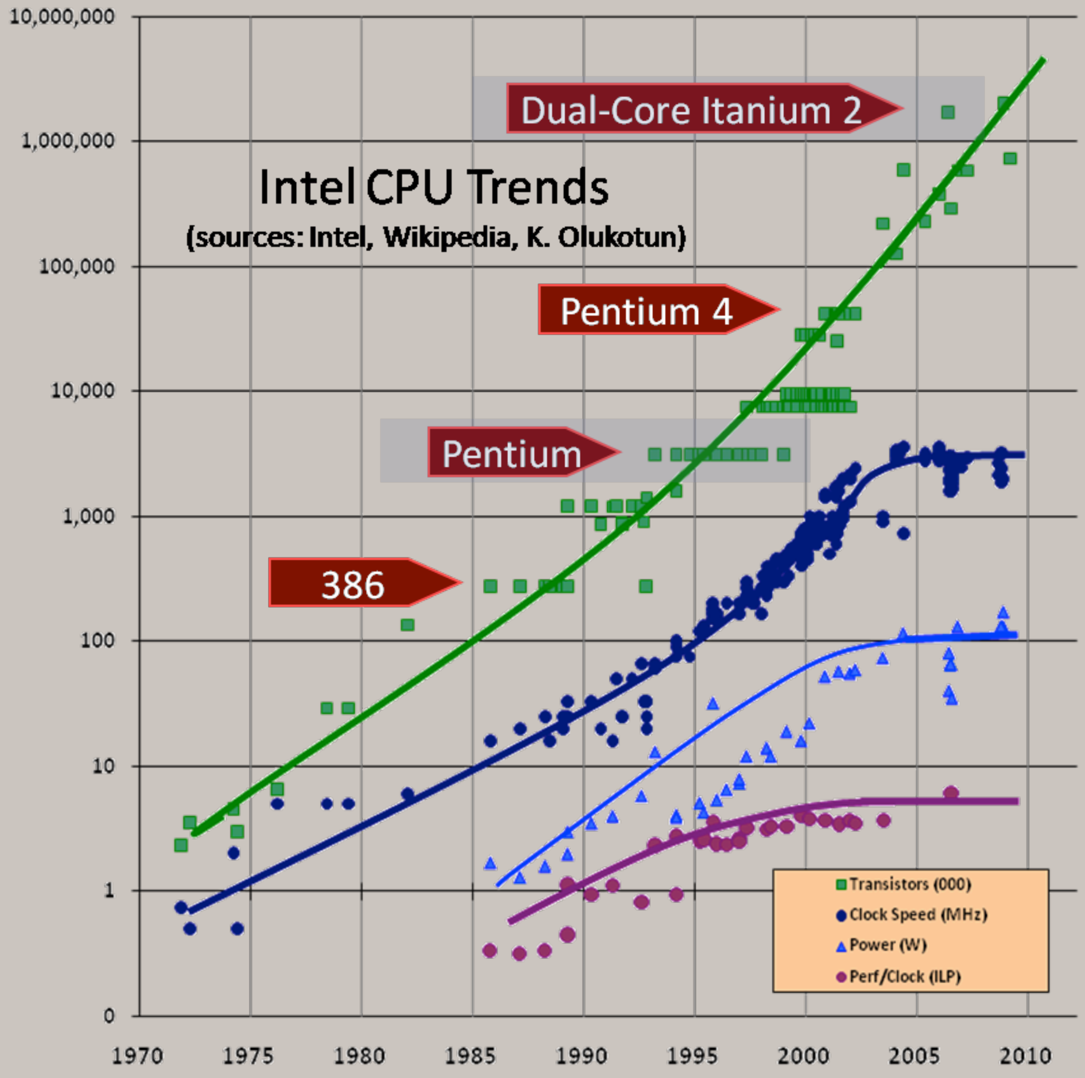
Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Performance results are based on testing by Intel as of September 16, 2019 and may not reflect all publicly available security updates. See configuration disclosure for details. No product or component can be absolutely secure.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to www.intel.com/benchmarks.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804.

“The Free Lunch Is Over”, Herb Sutter, 2005



Going Parallel

Machines	< 10^3-10^6	(servers)
Hadoop (Map/Reduce), Apache Spark		
Cores/hardware threads	< 10s-100s	(threads)
Java Stream API Fork/Join framework		
CPU SIMD extensions	< 10s	(elements)
x86: SSE ..., AVX, …, AVX-512		

Going Parallel: CPUs vs Co-processors

CPUs

- SIMD ISA extensions (Single Instruction-Multiple Data)
- threads (Multiple Instructions-Multiple Data)

Co-processors

- GPUs, FPGAs, ASICs

SIMD vs MIMD

Machines up to 12 cards / server	< 10^3-10^6	12x	(servers)
Intel® Xeon® Platinum 9282 2 threads x 56 cores	< 10s-100s	112	(threads)
AVX-512 2 units / core	< 10s	16 SP	(elements)

SIMD vs MIMD

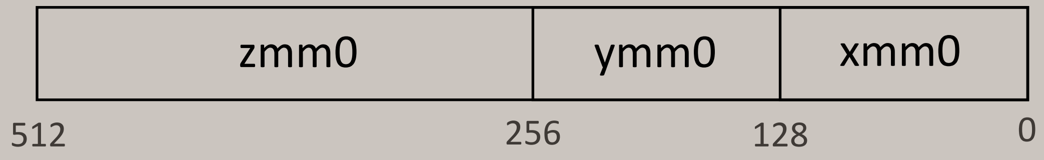
Machines up to 12 cards / server	< 10^3-10^6	12x	(servers)
Intel® Xeon® Platinum 9282 2 threads x 56 cores	< 10s-100s	112	(threads)
AVX-512 2 units / core	< 10s	16 SP	(elements)

1792-way

x86 SIMD Extensions

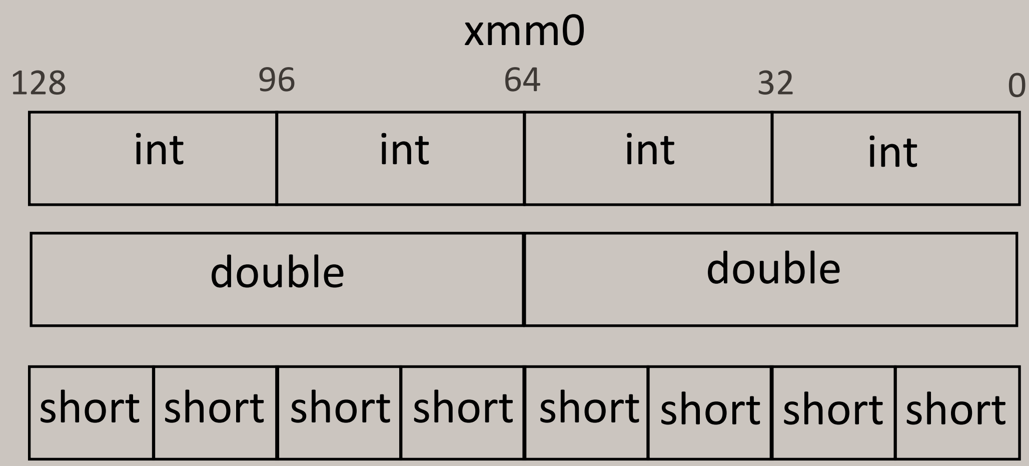
Wide (multi-word) registers

- 128-bit (xmm)
- 256-bit (ymm)
- 512-bit (zmm)



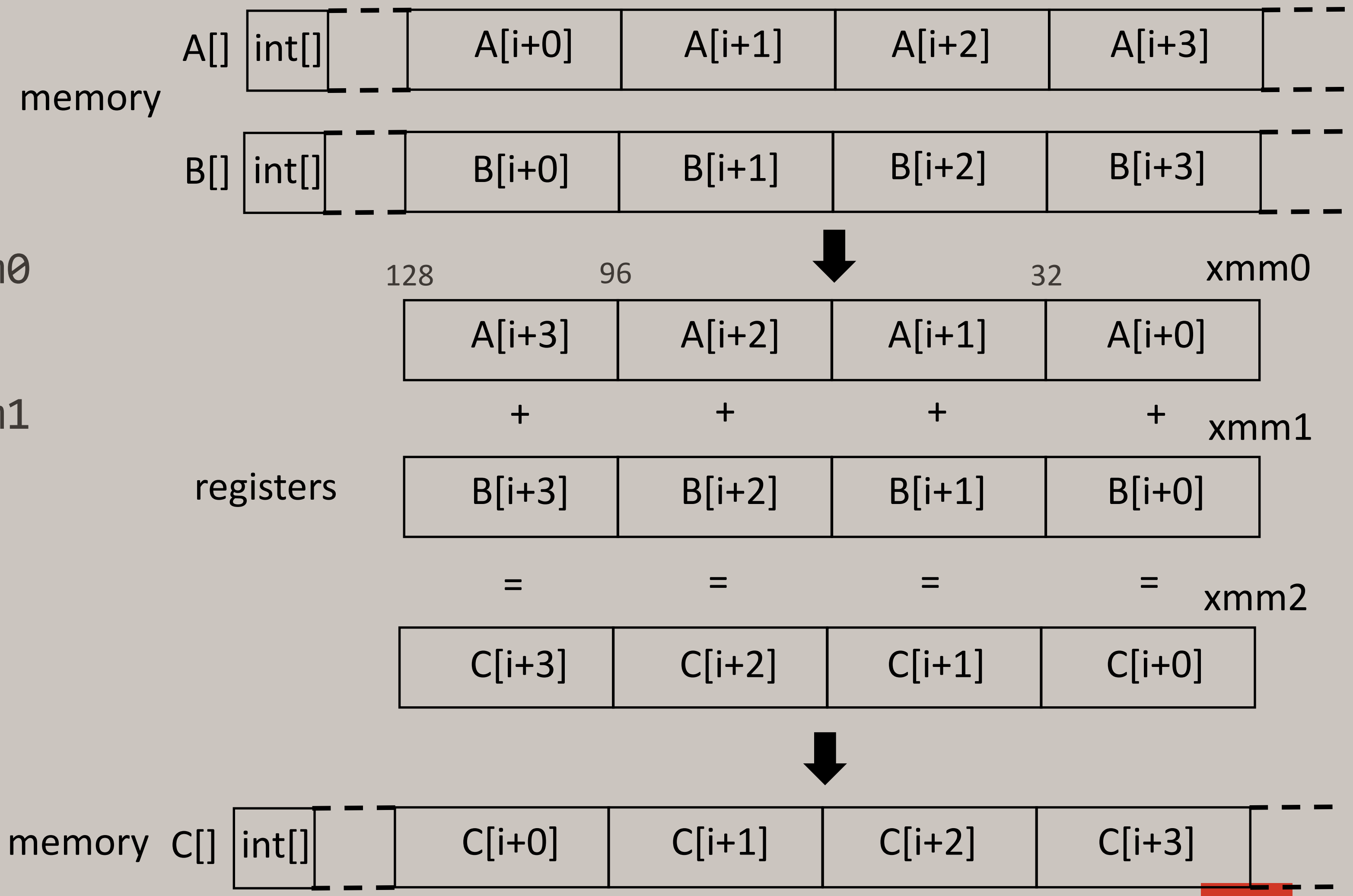
Instructions on packed vectors

- packed in a register or memory location
- short vectors of integer / FP numbers
  - 2 x double, 4 x int, 8 x short
- hard-coded vector size



x86 SIMD Extensions

```
// Load A[i:i+3]
vmovdqu 0x10(%rcx,%rdx,4),%xmm0
// Load B[i:i+3]
vmovdqu 0x10(%r10,%rdx,4),%xmm1
// A[i:i+3] + B[i:i+3]
vpaddd %xmm0,%xmm1,%xmm2
// Store into C[i:i+3]
vmovdqu %xmm2,0x10(%r8,%rdx,4)
```



SIMD today

x86: MMX, SSE, AVX, AVX2, AVX-512

- 8 64-bit registers (MMX) to 32 512-bit registers (AVX-512)

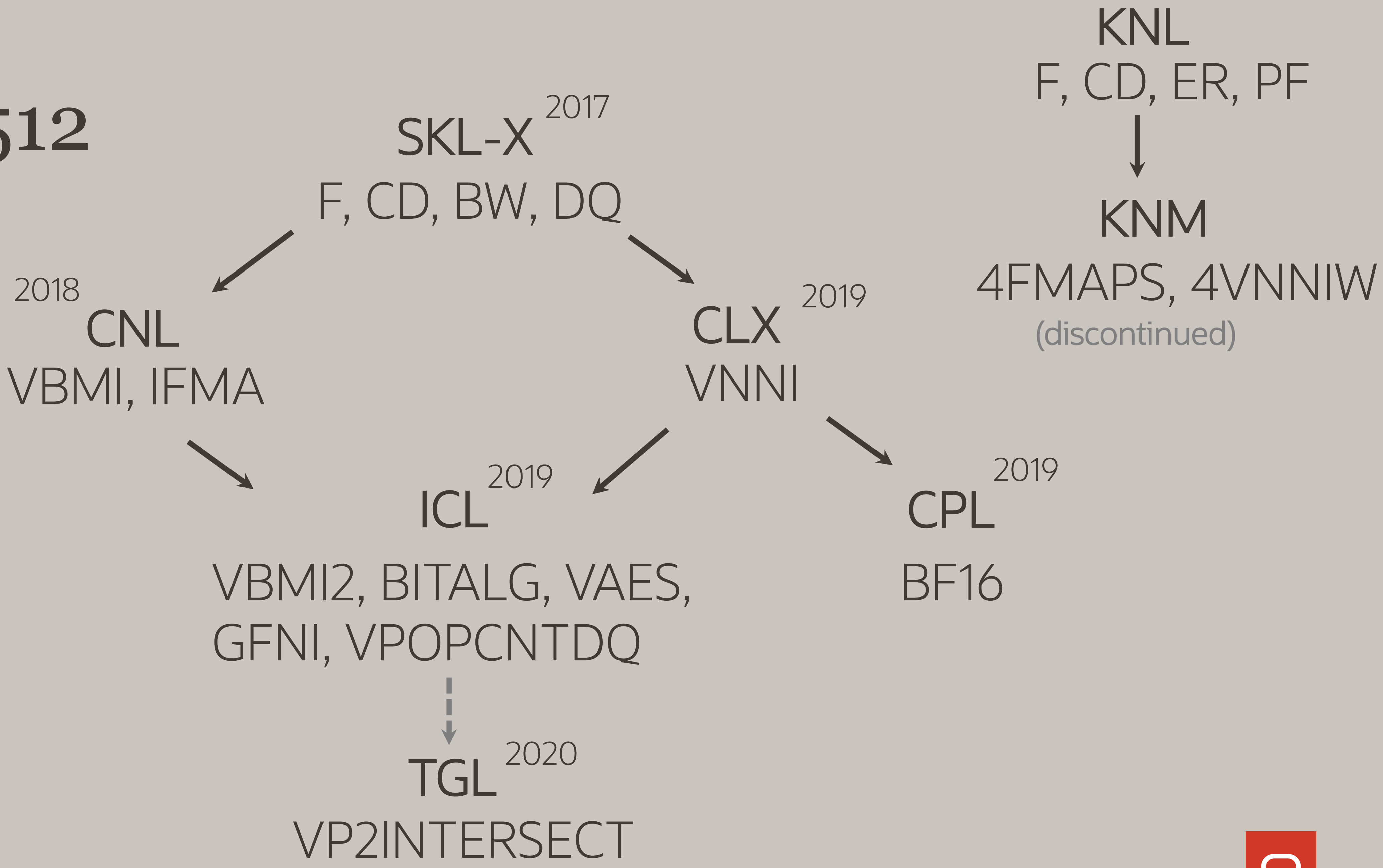
ARM: NEON, SVE, SVE2

- 32 128-bit registers (NEON) to 32 128-2048-bit in SVE

POWER: VMX/AltiVec

- 32 128-bit registers

AVX-512



SKL-X 2017
F, CD, BW, DQ

2018 CNL
VBMI, IFMA

CLX 2019
VNNI

ICL 2019
VBMI2, BITALG, VAES,
GFNI, VPOPCNTDQ

CPL 2019
BF16

TGL 2020
VP2INTERSECT

KNL
F, CD, ER, PF

KNM
4FMAPS, 4VNNIW
(discontinued)

How to utilize SIMD instructions?

Vectorization techniques

Automatic

- sequential languages and practices gets in the way

Semi-automatic

- Give your compiler/runtime hints and hope it vectorizes
  - OpenMP 4.0 #pragma omp simd

Code explicitly

- SIMD instruction intrinsics

Problem

If the code is compiled for a **particular instruction set** then it will be **compatible** with all CPUs that **support** this instruction set or any higher instruction set, but **possibly not** with **earlier** CPUs.

SSE 4.2 << AVX-512

JVM and SIMD today

JVM is in a good position:

1. Java bytecode is platform-agnostic
2. CPU probing at runtime (at startup)
   knows everything about the hardware it executes at the moment
3. Dynamic code generation
   only use instructions which are available on the host

JVM and SIMD today

Hotspot supports some of x86 SIMD instructions

Automatic vectorization of Java code

- Superword optimizations in HotSpot C2 compiler to derive SIMD code from sequential code

JVM intrinsics

- e.g., Array copying, filling, and comparison

JVM Intrinsics

JVM Intrinsics

“A method is intrinsified if the HotSpot **VM replaces the annotated method with hand-written assembly and/or hand-written compiler IR** -- a compiler intrinsic -- to improve performance.”

@HotSpotIntrinsicCandidate JavaDoc

```
public final class java.lang.Class<T> implements … {
    @HotSpotIntrinsicCandidate
    public native boolean isInstance(Object obj);
```

Vectorized JVM Intrinsics

Array copy

- System.arraycopy(), Arrays.copyOf(), Arrays.equals()

Array mismatch (@since 9)

- Arrays.mismatch(), Arrays.compare()
- based on ArraysSupport.vectorizedMismatch()

Auto-vectorization

by JVM JIT-compiler

Vectorization: Prerequisites

SuperWord optimization is:

1. implemented only in C2 JIT-compiler in HotSpot

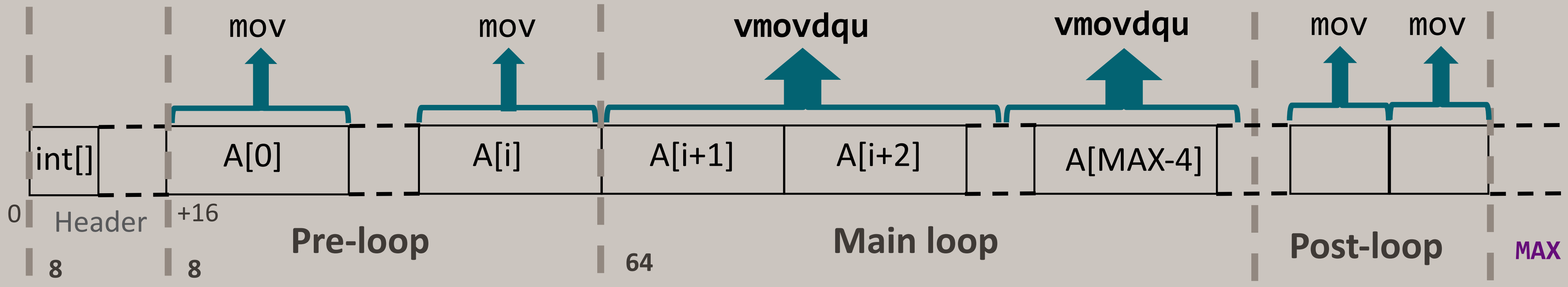
```
hotspot/src/share/vm/opto/c2_globals.hpp:
  product(bool, UseSuperWord, true,
          "Transform scalar operations into superword operations”)
```

2. applied only to unrolled loops

   unrolling is performed **only** for counted loops

```
int[] A, B, C
for (int i = 0; i < MAX; i++) {
    A[i] = B[i] + C[i];
}
```

```
…
// Main loop
vmovdqu 0x10(%rcx,%rdx,4),%xmm0
vpaddd 0x10(%r10,%rdx,4),%xmm0,%xmm0
vmovdqu %xmm0,0x10(%r8,%rdx,4)
add     $0x4,%edx
cmp     %r9d,%edx
jl      0x…
…
```



```
MAX = 1000
```

T	not vectorized, 8u	vectorized, 8u
byte	506 ±6	159 ±4
short	495 ±4	140 ±3
char	493 ±4	141 ±2
int	490 ±4	154 ±2
long	492 ±5	157 ±2
float	489 ±7	155 ±2
double	483 ±4	172 ±3

```
<any T> void add (T[] A, T[] B, T[] C) {
    for (int i = 0; i < MAX; i++) {
        A[i] = B[i] + C[i];
    }
}
```

Core i7, 1x2x2, Haswell (AVX2), ns/op

MAX = 1000

T	vectorized, 8u	vectorized, 11u
byte	159 ±4	69 ±3
short	140 ±3	69 ±4
char	141 ±2	68 ±2
int	154 ±2	74 ±1
long	157 ±2	141 ±1
float	155 ±2	80 ±3
double	172 ±3	167 ±2

```
<any T> void add (T[] A, T[] B, T[] C) {
    for (int i = 0; i < MAX; i++) {
        A[i] = B[i] + C[i];
    }
}
```

Core i7, 1x2x2, Haswell (AVX2), ns/op

```
int dotProduct(int[] A, int[] B) {
  int r = 0;
  for (int i = 0; i < MAX; i++) {
    r += A[i]*B[i];
  }
  return r;
}
```

```
// Vectorized post-loop

vmovdqu 0x10(%rdi,%r11,4),%ymm0

vmovdqu 0x10(%rbx,%r11,4),%ymm1

vpmulld %ymm0,%ymm1,%ymm0

vphaddd %ymm0,%ymm0,%ymm3

vphaddd %ymm1,%ymm3,%ymm3

vextracti128 $0x1,%ymm3,%xmm1

vpaddd %xmm1,%xmm3,%xmm3

vmovd  %eax,%xmm1

vpaddd %xmm3,%xmm1,%xmm1

vmovd  %xmm1,%eax

add    $0x8,%r11d

cmp    %r8d,%r11d

jl     0x117e23668
```

```java
public int sum(int[] A) {
    int sum = 0;
    for (int a : A) {
        sum += a;
    }
    return sum;
}
```

```
…
add  0x10(%r8,%rcx,4),%eax
add  0x14(%r8,%rcx,4),%eax
add  0x18(%r8,%rcx,4),%eax
add  0x1c(%r8,%rcx,4),%eax
add  0x20(%r8,%rcx,4),%eax
add  0x24(%r8,%rcx,4),%eax
add  0x28(%r8,%rcx,4),%eax
add  0x2c(%r8,%rcx,4),%eax

add  $0x8,%ecx
cmp  %r10d,%ecx
jl   …
```

JVM and SIMD today

Superword optimizations can be very brittle

- doesn’t (and can’t) cover all the use cases

Intrinsics are point fixes, not general

- powerful, lightweight, and flexible
- high development costs

JNI is hard to develop and maintain

- interoperability overhead between Java & native code
- CPU dispatching is required

Vector API

Embrace explicit vectorization

DEV-6764: “Vector API”

Vladimir Ivanov, Oracle
Kishor Kharbas, Intel Corp.

Monday, September 16,
04:00 PM - 04:45 PM
Moscone South - Room 303

https://youtu.be/tR0mXPMOUjw?t=12800

Vector API: Goals

Expressive and **portable** API

- “principle of least astonishment”
- uniform coverage operations and data types
- type-safe

Performant

- High quality of generated code
- Competitive with existing facilities for auto-vectorization

Graceful performance degradation

- fallback for "holes" in native architectures

```java
int[] A, B, C

for (int i = 0; i < MAX; i++) {
    A[i] = B[i] + C[i];
}
```

```java
var S = IntVector.SPECIES_PREFERRED;
for (int i = 0; i < MAX; i += S.length()) {
    var va = IntVector.fromArray(S, A, i);
    var vb = IntVector.fromArray(S, B, i);
    var vc = va.add(vb);
    vc.intoArray(C, i);
}
```

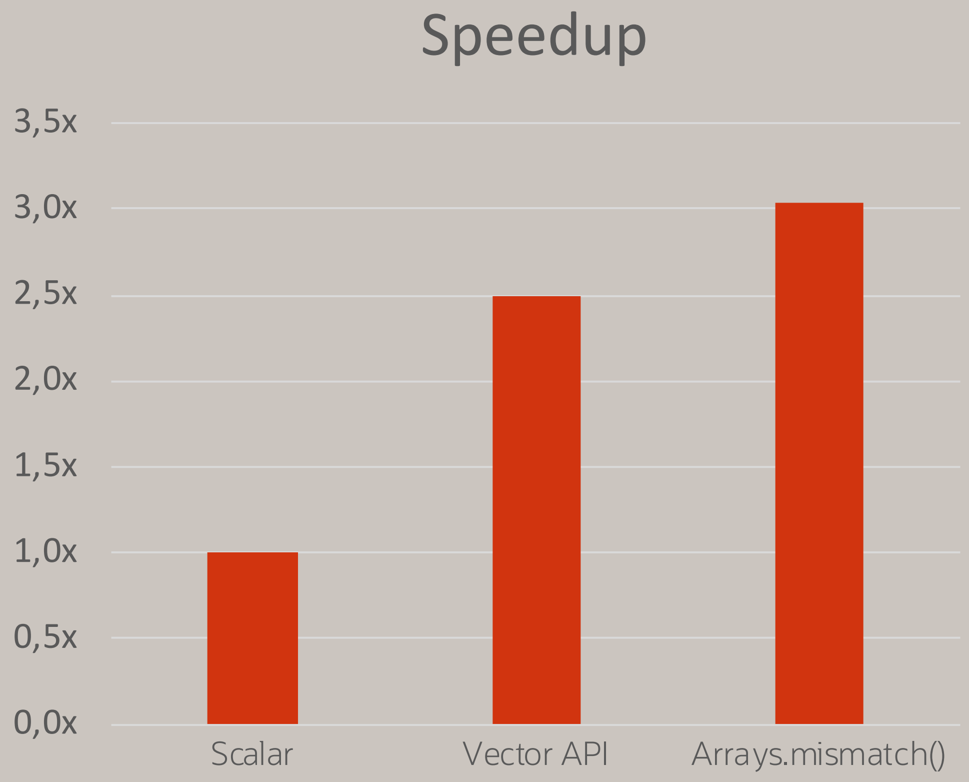
Arrays.mismatch()

```
…
var S = IntVector.SPECIES_PREFERRED;

for (int i = 0; i < MAX; i += S.length()) {
    var va = IntVector.fromArray(S, A, i);
    var vb = IntVector.fromArray(S, B, i);
    if (va.compare(NE, vb).anyTrue()) {
        break; // mismatch found
    }
}
…
```

VS

```
for (int i = 0; i < MAX; i++) {
    if (a[i] != b[i])
        return i;
}
```

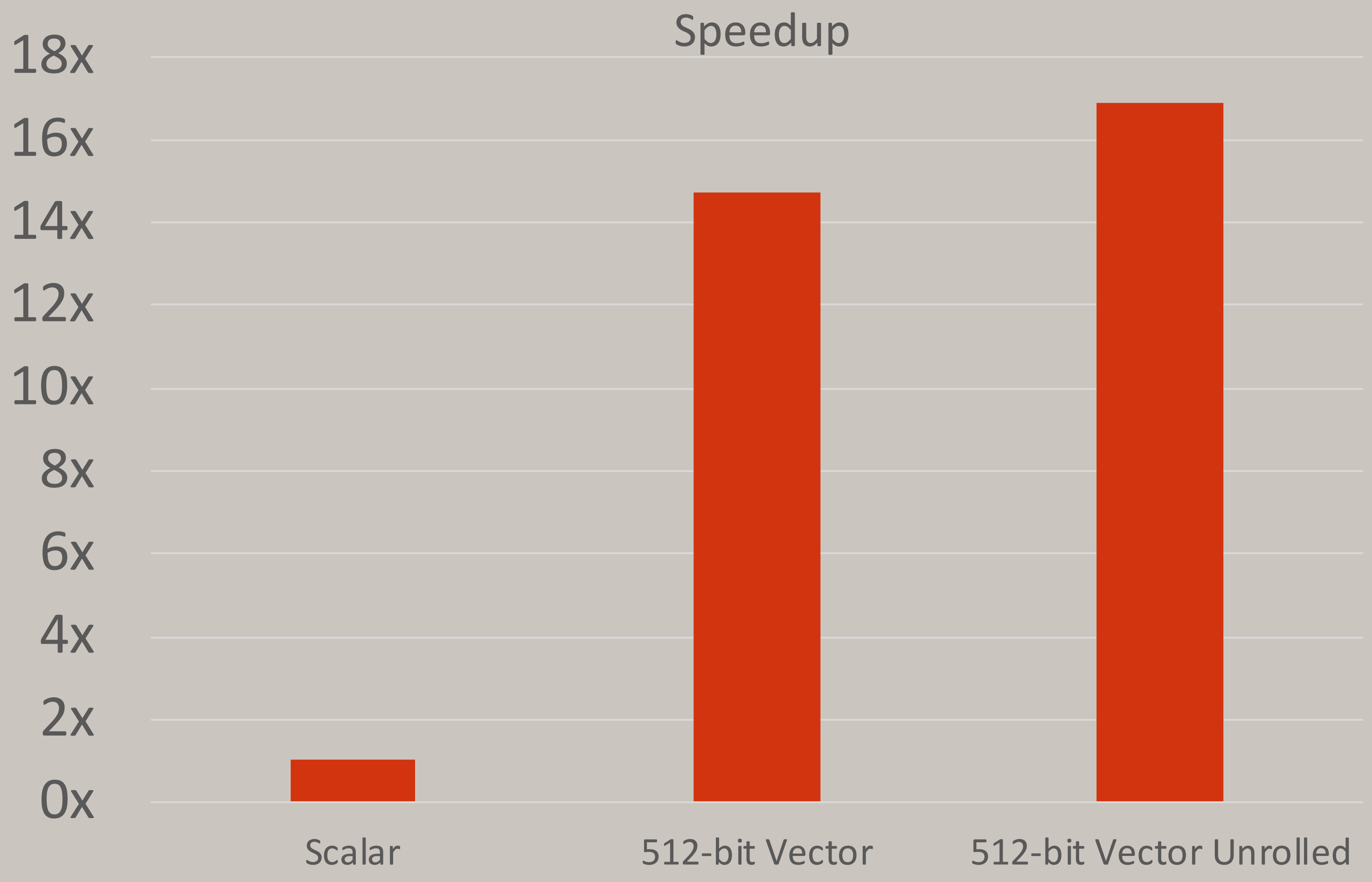


`MAX = 1000`

OpenJDK Panama project, parent: 56355:4ca845a25642, branch: vectorIntrinsics
Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz, 32 GB RAM, Windows 10, 64-bit

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
See slide #72 for configurations.

Dot Product

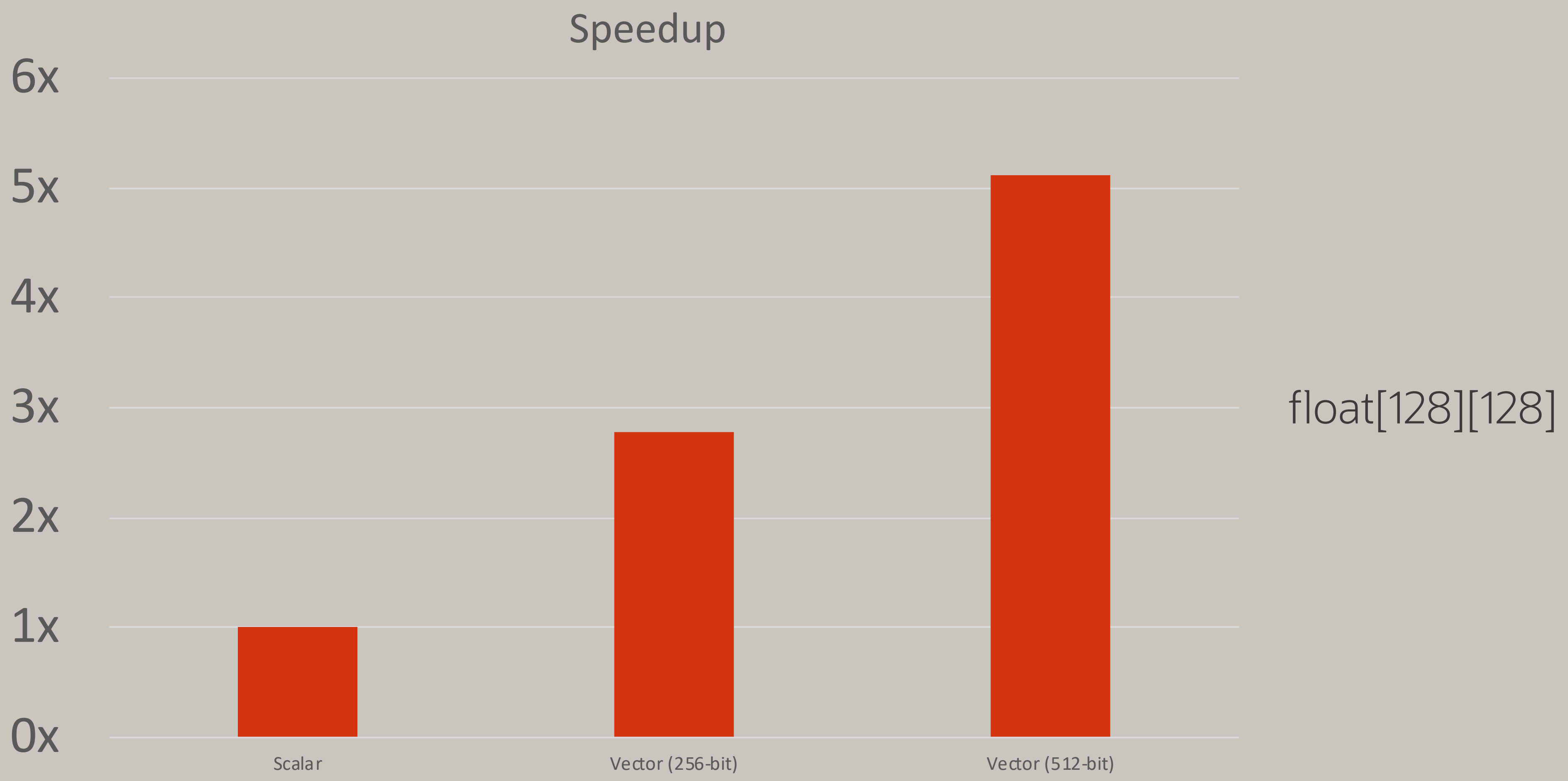


float[128]

OpenJDK Panama project, parent: 56355:4ca845a25642, branch: vectorIntrinsics
Red Hat Enterprise Linux Server release 7.6
Intel(R) Xeon(R) Platinum 8280L CPU @ 2.70GHz, 768 GB RAM

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
See slide #72 for configurations.

Matrix Multiplication



OpenJDK Panama project, parent: 56355:4ca845a25642, branch: vectorIntrinsics
Red Hat Enterprise Linux Server release 7.6
Intel(R) Xeon(R) Platinum 8280L CPU @ 2.70GHz, 768 GB RAM

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
See slide #72 for configurations.

Current Status (September, 2019)

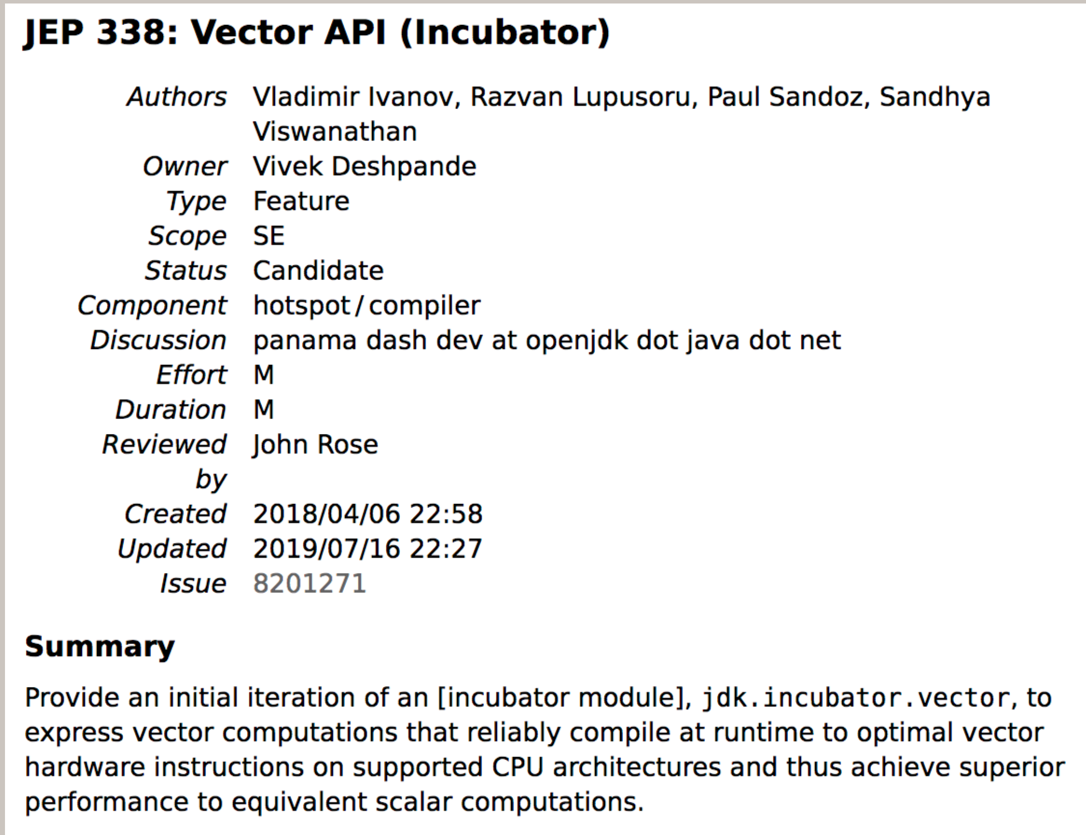
JEP 338: “Vector API (Incubator)”

in Candidate state

First version of API is in CSR

- https://bugs.openjdk.java.net/browse/JDK-8223348
- To be delivered in an upcoming OpenJDK release
- Will be an incubator project, pending integration with Valhalla
- Ongoing basic experimentation, including machine learning kernels
- Who uses it? What’s built on top of it? … is TBD. Ideas solicited.

Lots of work on productizing the implementation went in during last year



JEP 338: Vector API (Incubator)

Authors	Vladimir Ivanov, Razvan Lupusoru, Paul Sandoz, Sandhya Viswanathan
Owner	Vivek Deshpande
Type	Feature
Scope	SE
Status	Candidate
Component	hotspot / compiler
Discussion	panama dash dev at openjdk dot java dot net
Effort	M
Duration	M
Reviewed by	John Rose
Created	2018/04/06 22:58
Updated	2019/07/16 22:27
Issue	8201271

Summary

Provide an initial iteration of an [incubator module], `jdk.incubator.vector`, to express vector computations that reliably compile at runtime to optimal vector hardware instructions on supported CPU architectures and thus achieve superior performance to equivalent scalar computations.

Summary

- SIMD ISA extensions
  - very irregular on x86
  - hard to utilize in cross-platform manner

- JVM
  - auto-vectorization
    - brittle
    - can’t cover all the cases
  - intrinsics
    - pros: powerful, lightweight, and flexible
    - cons: point fixes, high development costs

Future

SIMD ISA extensions

- will continue to evolve

JVM

- better auto-vectorization
- more intrinsics

Vector API

- reliable way to write performant vectorized code
- next iterations of the API
  - easier to use
  - closer to hardware

Thank You!

Configuration

OpenJDK Panama project, parent: 56355:4ca845a25642, branch: vectorIntrinsics

Intel(R) Xeon(R) Platinum 8280L CPU:

2-socket Intel(R) Xeon(R) Platinum 8280L CPU @ 2.70GHz, 28 cores HT On Turbo ON Total Memory 768 GB (24 slots/ 32GB/ 2666 MHz), BIOS: SE5C620.86B.0X.02.0001.051420190324 (ucode:0x5000024), Red Hat Enterprise Linux Server 7.6 (Maipo)

All benchmarks are run in a single thread.

Intel(R) Core(TM) i7-6700 CPU:

Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz, 3401 Mhz, 4 cores, HT ON, Total Memory 768 GB, BIOS Version/Date, BIOS: American Megatrends Inc. F4, 10/21/2015, Microsoft Windows 10 Pro 10.0.18362 Build 18362