Tuning Autovectorization in Graal

Yunjie Pan
April 2, 2022

Yunjie Pan

She/Her
Intern at #TwitterVMTeam in 2021 summer

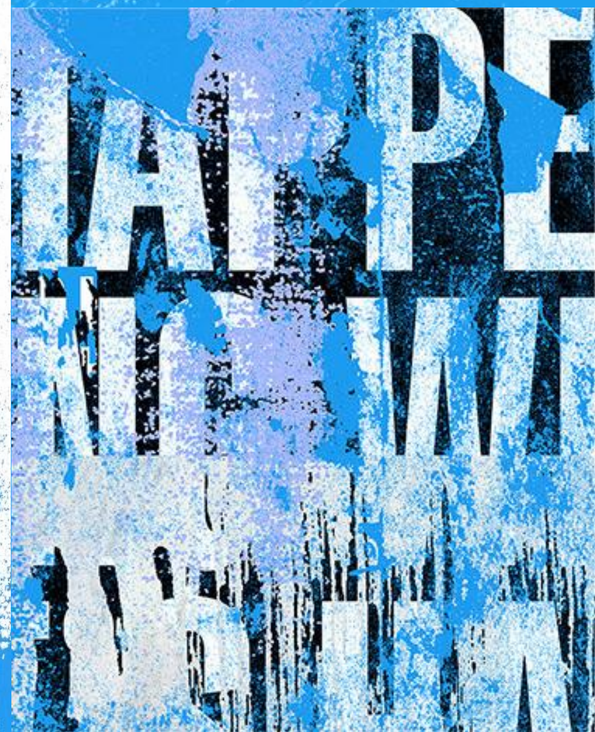
UNIVERSITY OF MICHIGAN

3rd-year Ph.D. student
Computer architecture

GO BLUE!

Outline

1. Recap of Graal & Autovectorization
2. Challenges in Autovectorization
3. Results
4. Next Steps



Recap of Graal & Autovectoriztion

Autovectorization Example: Original Code

```
public float[] add_2_arrays_elements() {
    for (int i = 0; i < size; i++) {
        C[i] = A[i] + B[i];
    }
    return C;
}
```

Autovectorization Example: Unrolled Loop

```
public float[] add_2_arrays_elements() {
    for (int i = 0; i < size; i += 4) {
        C[i:i+4] = A[i:i+4] + B[i:i+4];
    }
    return C;
}
```

```
add [rax], 1
add [rax + 4], 2
add [rax + 8], 3
add [rax + 12], 4
```

Scalar

Speedup!

```
vmovps  mem[A],%xmm0
vaddps  mem[B],%xmm0,%xmm0
vmovps  %xmm0,mem[C]
```

Vector

Autovectorization in compilers

- Java Compiler
  - C2: Superword-level Parallelism (SLP)
    - Loop optimizations unroll partially or fully
    - seek isomorphisms packs
  - Graal EE: Autovectorize implemented
- C++ Compiler
  - LLVM: SLP Vectorizer and Loop Vectorizer

Autovectorization status so far

Graal EE

- Charge money
- Not open-source

Graal CE

- Open-source

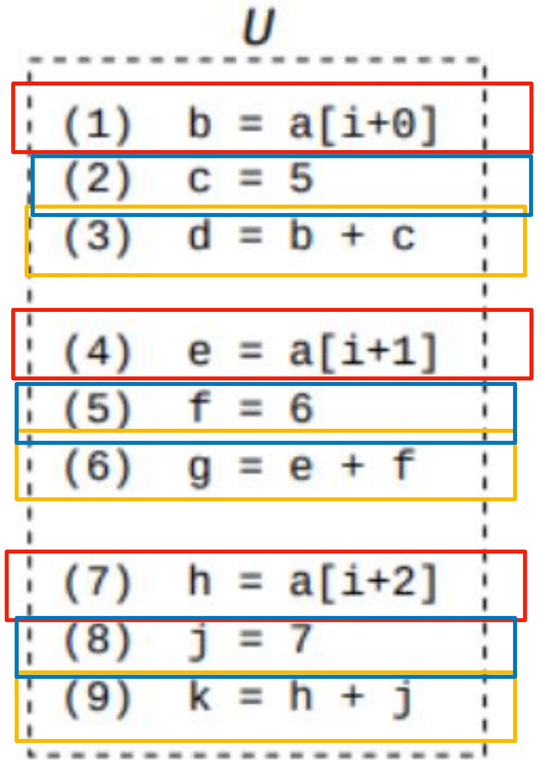
#TwitterVMTeam Part1

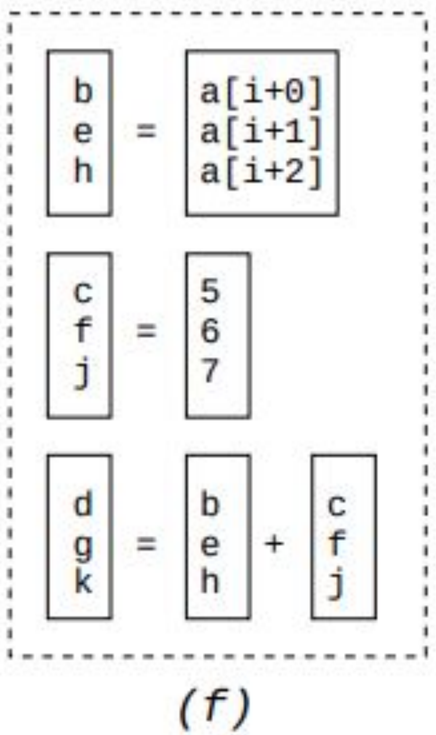
- **Superword-Level Parallelism**
- **Intel SSE/AVX Instructions**

Bravo Nik and David!

SLP Algorithm

1. Loop Unrolling if needed
2. Identifying Adjacent Memory References
3. Extend the Packlist
4. Pack Combination
5. **Filter Packs (If they are profitable)**
6. Generate Vector Instructions





[1] Larsen, Samuel, and Saman Amarasinghe. "Exploiting superword level parallelism with multimedia instruction sets." Acm Sigplan Notices 35.5 (2000): 145-156.

Challenges in Autovectorization

Challenges in Autovectorization

- Discoverability of vectorizable loops
  - **Pre-vectorization phases and optimizations**
    - **Bounds check for memory access**
    - **Commute inputs**
  - Memory dependency analysis
- Determination of the ROI of vectorization
  - Characteristics of the code being vectorized
  - **Cost Model for vectorization**
- Hardware support

[1] Larsen, Samuel, and Saman Amarasinghe. "Exploiting superword level parallelism with multimedia instruction sets." Acm Sigplan Notices 35.5 (2000): 145-156.

Pre-vectorization phases - BCE

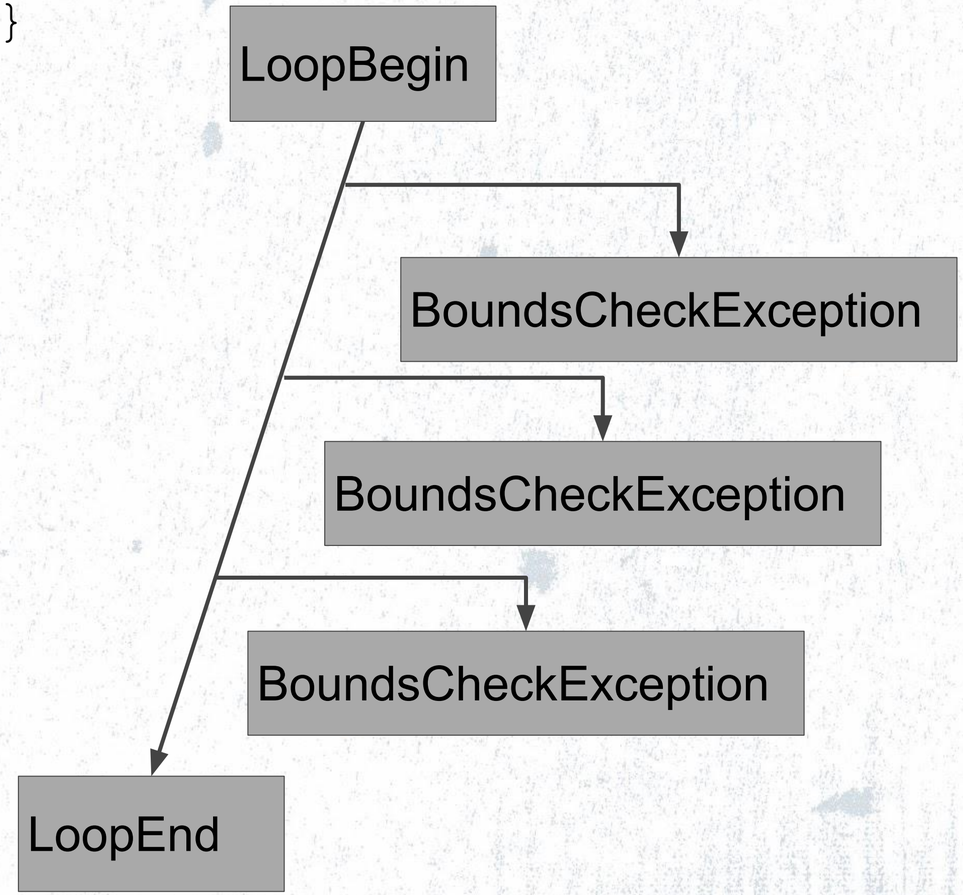
```
for (int i = 0; i < size; i++) {
    C[i] = A[i] + B[i];
}
```

Before:

- Graal features bounds checks for array accesses.
- Disables loop unrolling!
- Hacks around this:
  - Unroll earlier, before guards are lowered.
  - Remove bounds checks.

After:

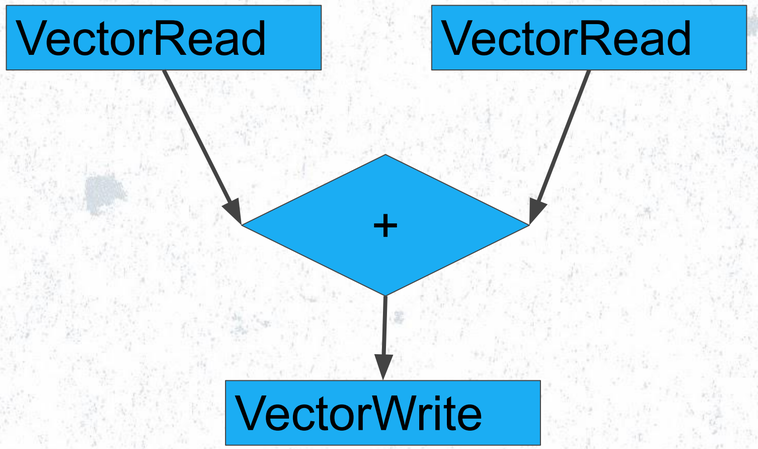
- Bounds check elimination (BCE) available in 20.3
- Problem solved



Ionut *add_2_arrays_elements*() Analysis - commute inputs

```
public int[] add_2_arrays_elements(int[] a, int[] b, int[] c) {
    for (int i = 0; i < ARRAY_SIZE; i++) {
        c[i] = a[i] + b[i];
    }
}
```

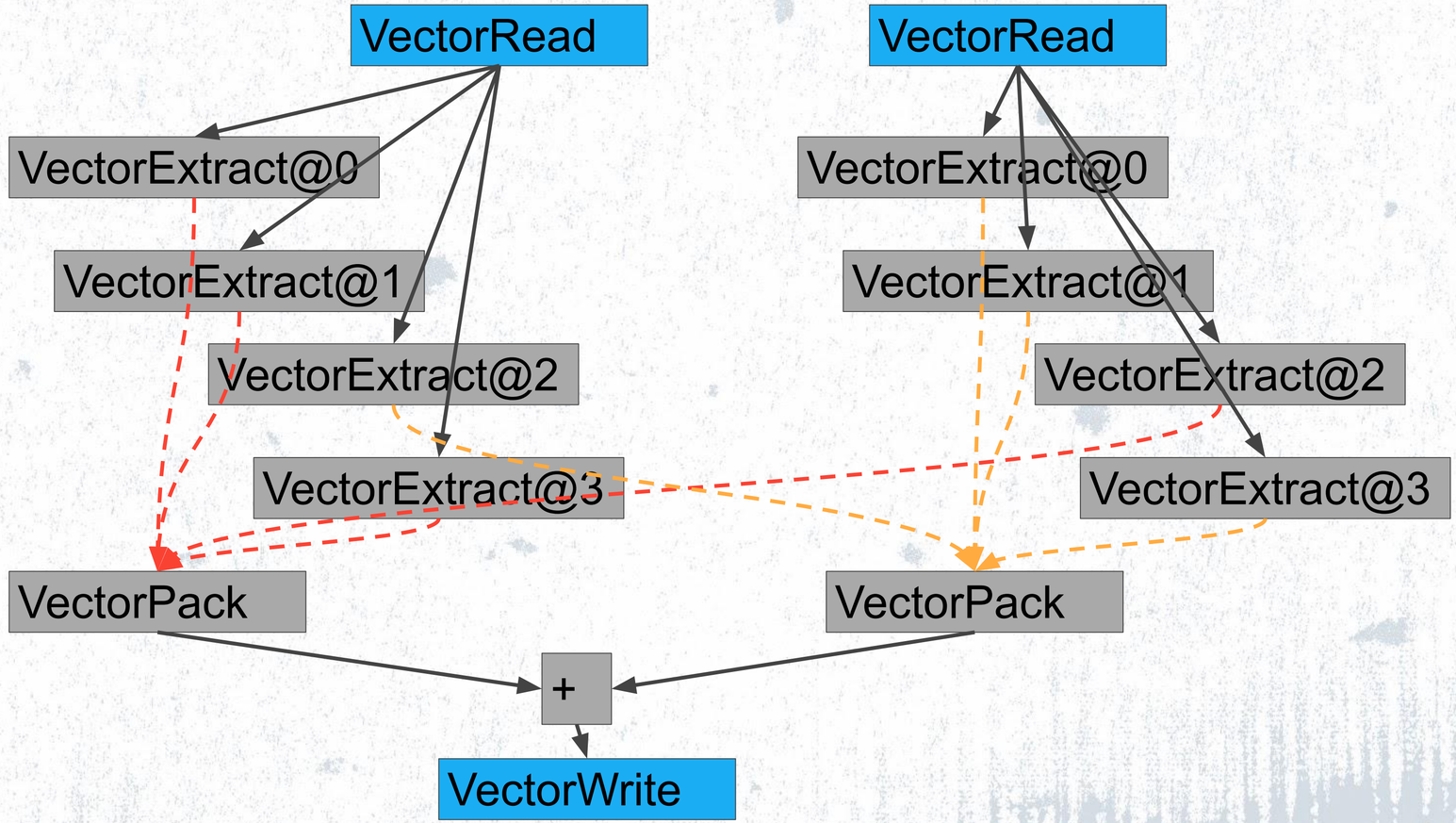
```
c[i]   = a[i]   + b[i];
c[i+1] = a[i+1] + b[i+1];
c[i+2] = a[i+2] + b[i+2];
c[i+3] = a[i+3] + b[i+3];
```



Ionut *add_2_arrays_elements*() Analysis - commute inputs

```
public int[] add_2_arrays_elements(int[] a, int[] b, int[] c) {
    for (int i = 0; i < ARRAY_SIZE; i++) {
        c[i] = a[i] + b[i];
    }
}
```

```
c[i]   = a[i]   + b[i];
c[i+1] = a[i+1] + b[i+1];
c[i+2] = b[i+2] + a[i+2];
c[i+3] = a[i+3] + b[i+3];
```



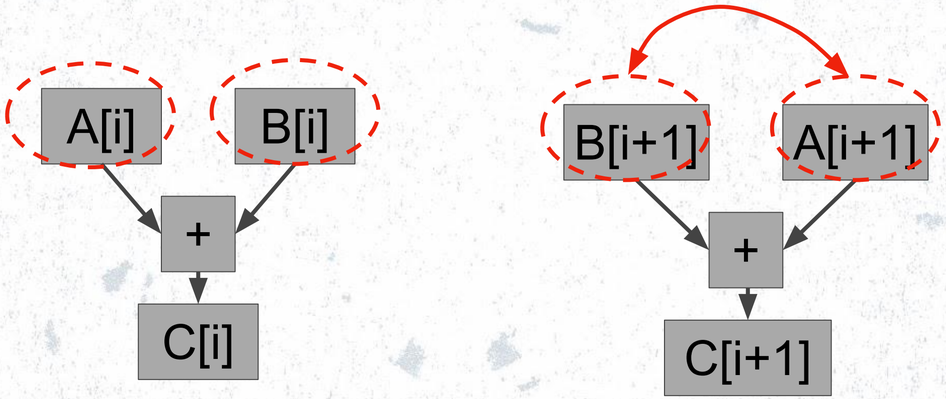
Issues: SLP slows if commute inputs

Quick Fix: disable commute inputs for memory access

- Pre-autovectorization phase
- But may have side-effect on other optimizations

Better choice (but may have higher overhead): Look-ahead SLP[1]

- Reorder inputs for commutative operations
- At Autovectorize phase



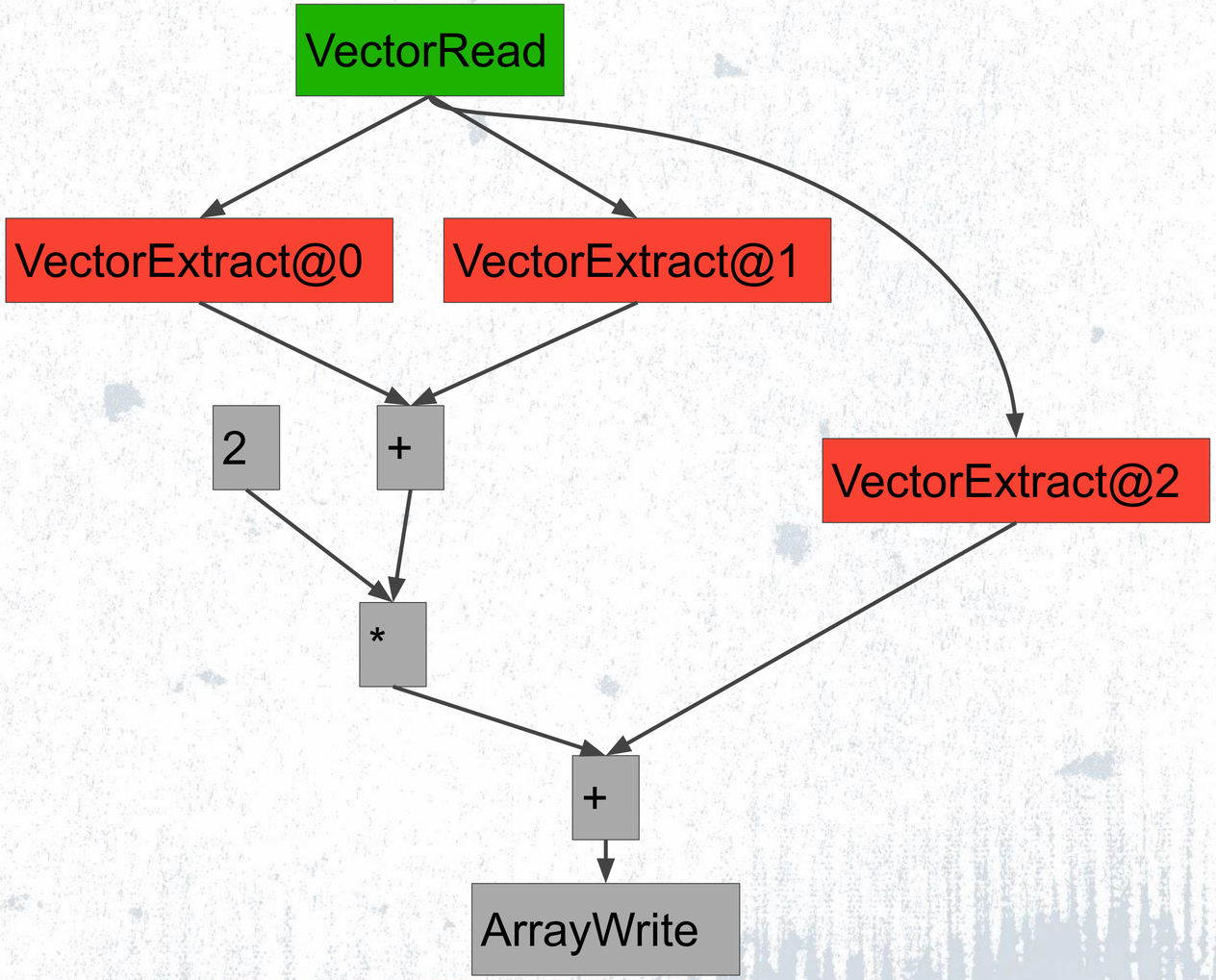
[1] Vasileios Porpodas, Rodrigo C. O. Rocha, and Luís F. W. Góes. 2018. Look-ahead SLP: auto-vectorization in the presence of commutative operations. In Proceedings of the 2018 International Symposium on Code Generation and Optimization (CGO 2018). Association for Computing Machinery, New York, NY, USA, 163–174. DOI:https://doi.org/10.1145/3168807

When is it not profitable to vectorize?

SOR.execute() - simplified

```
for(int i=0; i < size; i++){
  B[i]=(A[i-1]+A[i])*2+A[i+1];
}
```

When uses of vector operations are not vectors - Need extra VectorExtract



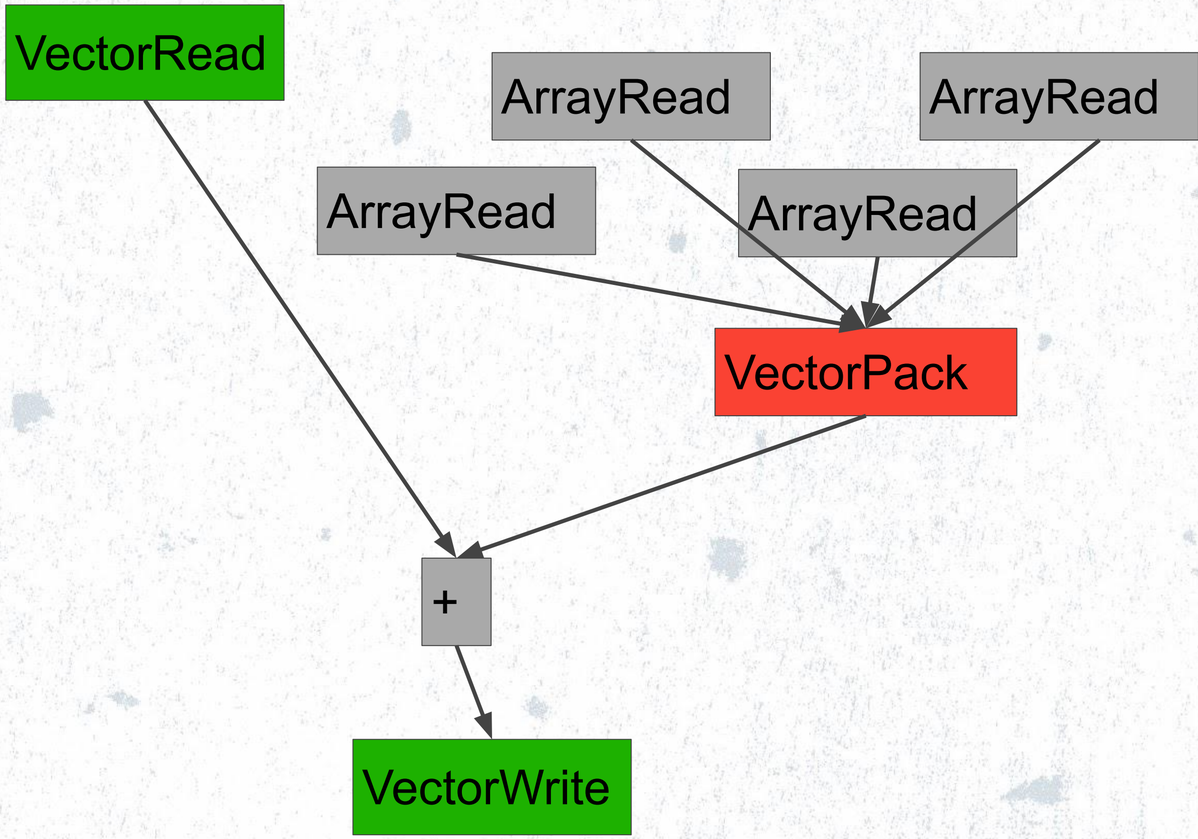
[1] scimark2 benchmark: https://math.nist.gov/scimark2/index.html

When is it not profitable to vectorize?

```
for(int i=0; i < size; i++){
  A[i]= A[i] + B[2*i] ;
}
```

When inputs of vector operations are not vectors - Need extra VectorPack

Except?



When is it not profitable to vectorize? Exceptions

```
for(int i=0; i < size; i++){
  A[i]= A[i] + 1 ;
}
```

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

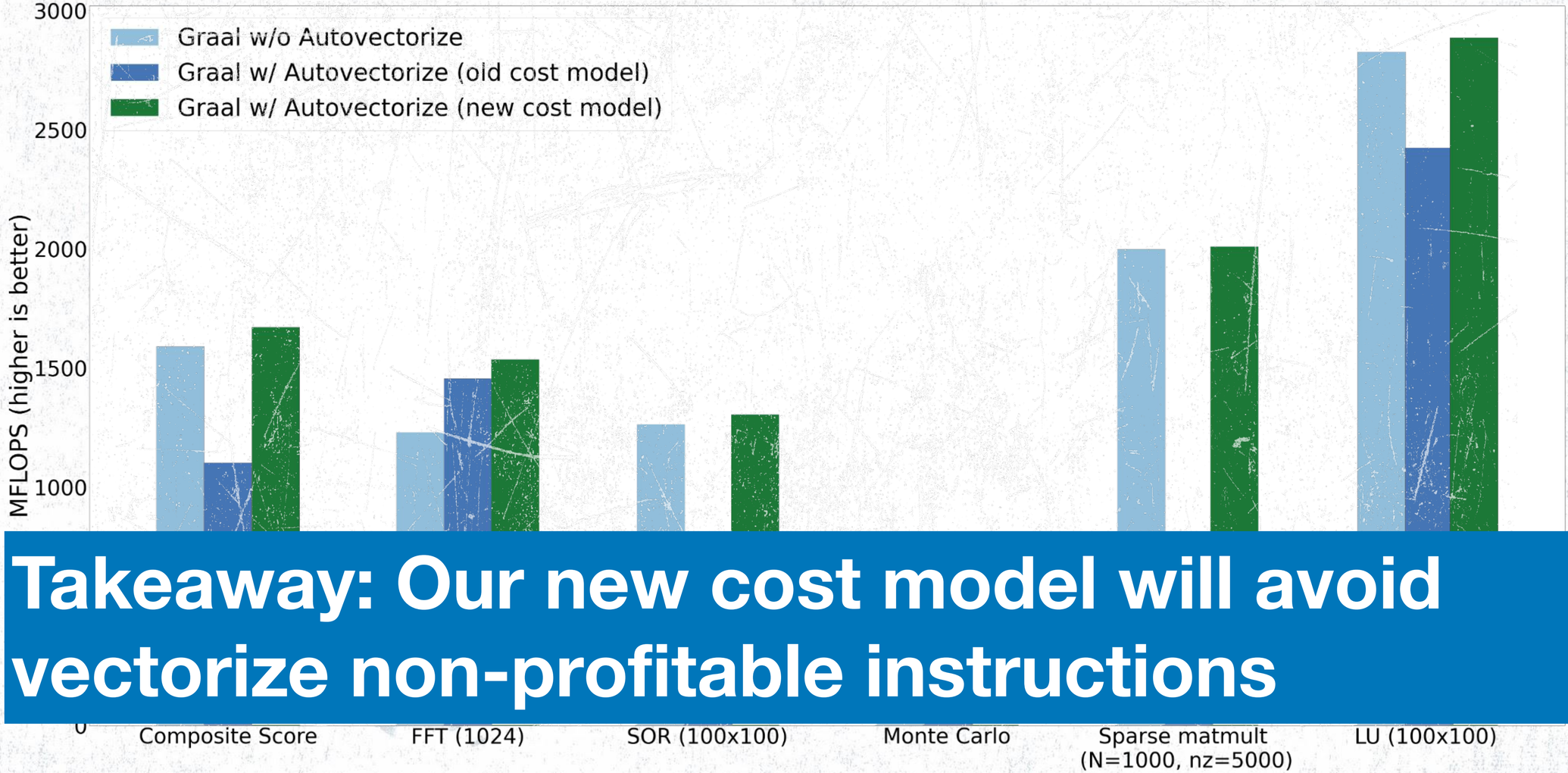
- When inputs of vector operations are constants or loop invariants
- Global Value Numbering (GVN) will hoist them outside loop

Results

JDK: TwitterJDK 11.0.9.0.7, Graal 20.3.1.2
OS: macOS Big Sur 11.4, x86_64

- **C2 w/ SuperWord**: -XX:+UseSuperWord
- **C2 w/o SuperWord**: -XX:-UseSuperWord
- **Graal w/ Autovectorize**: -XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI -XX:+UseJVMCICompiler -Dgraal.Autovectorize=true
- **Graal w/o Autovectorize**: -XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI -XX:+UseJVMCICompiler -Dgraal.Autovectorize=false

MFLOPS of scimark2 benchmark, old and new cost model

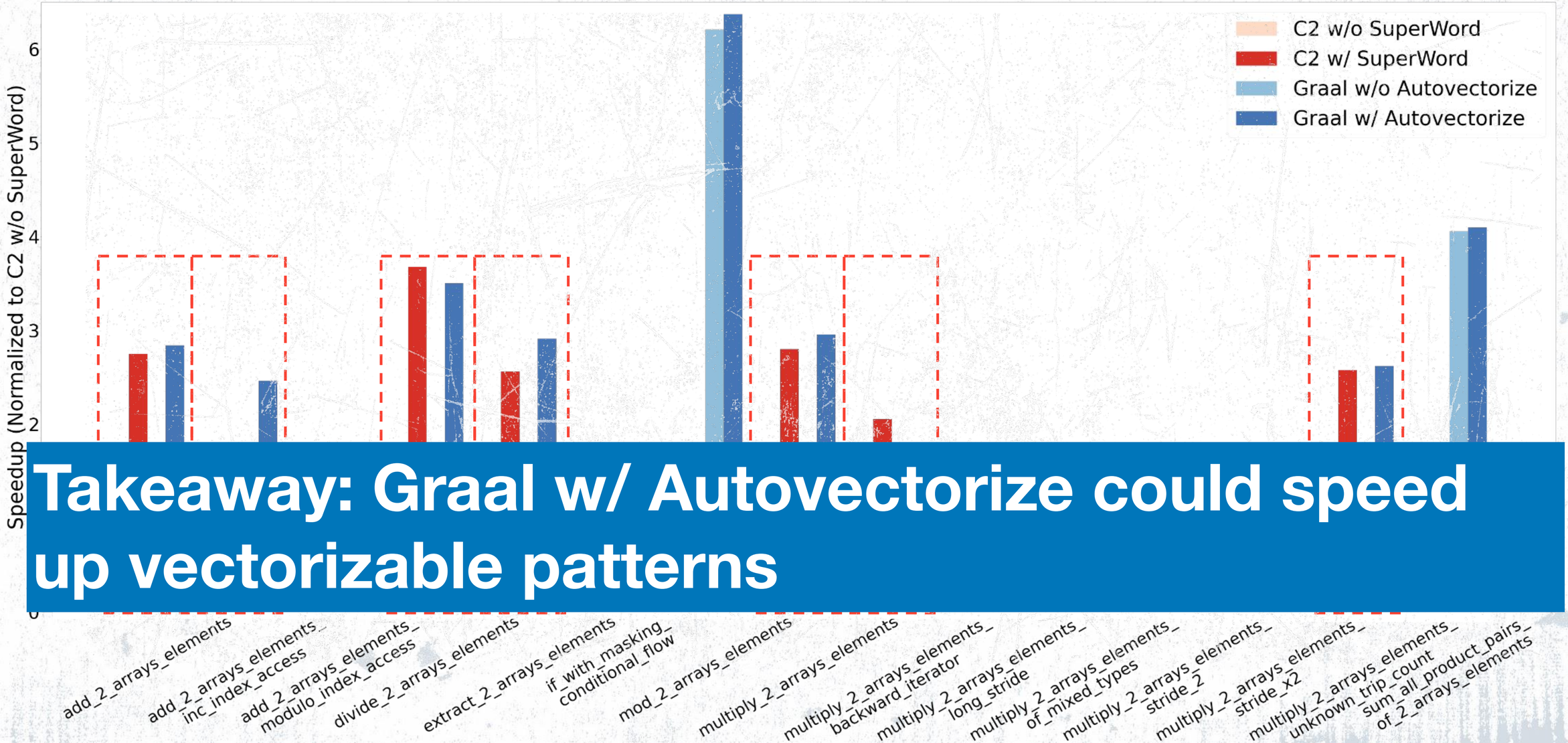


Takeaway: Our new cost model will avoid vectorize non-profitable instructions

[1] Java Scimark 2.0, https://math.nist.gov/scimark2/index.html

Date type: Float

Speedup of VectorizationPatternsMultipleFloatArraysBenchmark

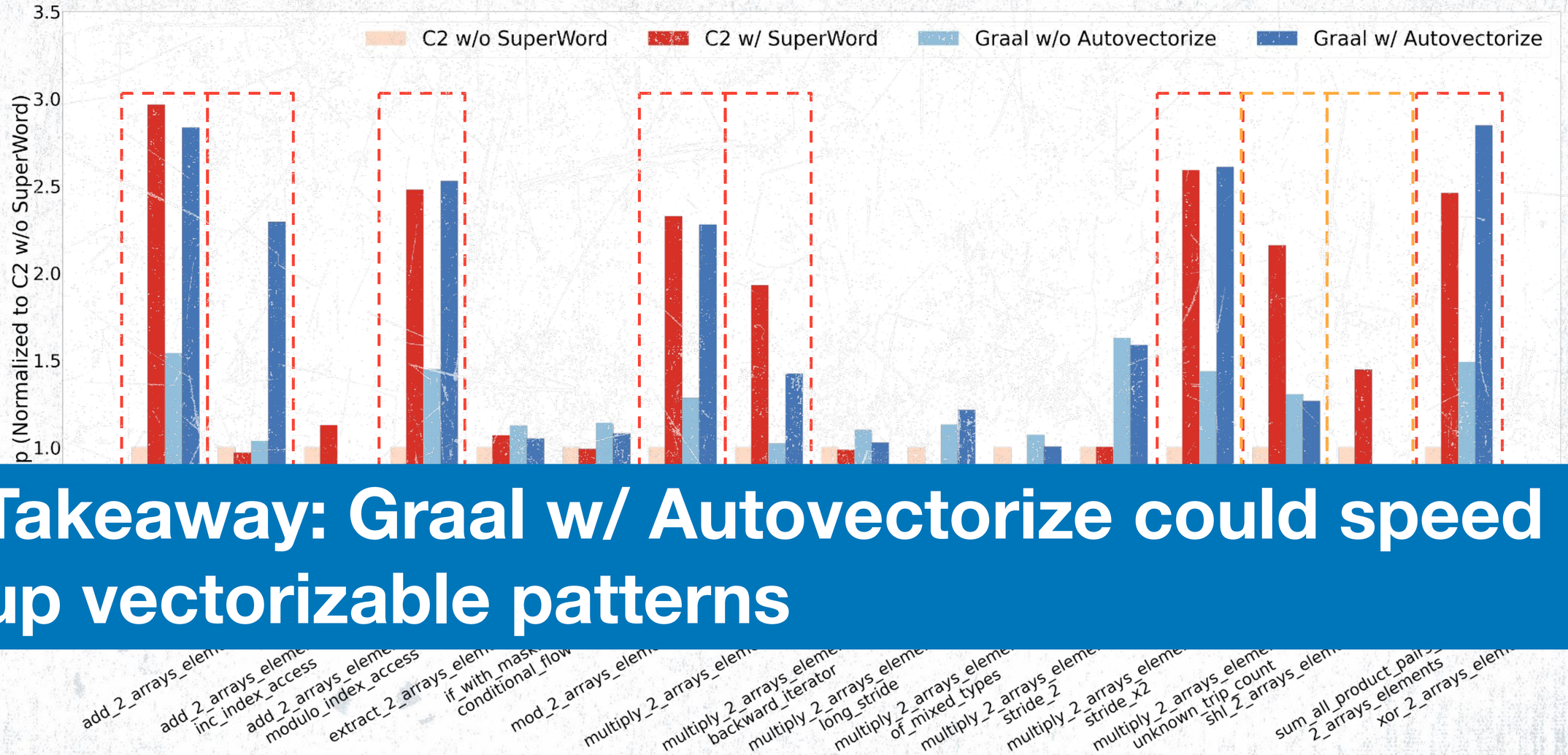


Takeaway: Graal w/ Autovectorize could speed up vectorizable patterns

[1] JVM JIT Compilers Benchmarks Report 19.11, https://ionutbalosin.com/2019/11/jvm-jit-compilers-benchmarks-report-19-11/

Date type: Int

Speedup of VectorizationPatternsMultipleIntArraysBenchmark

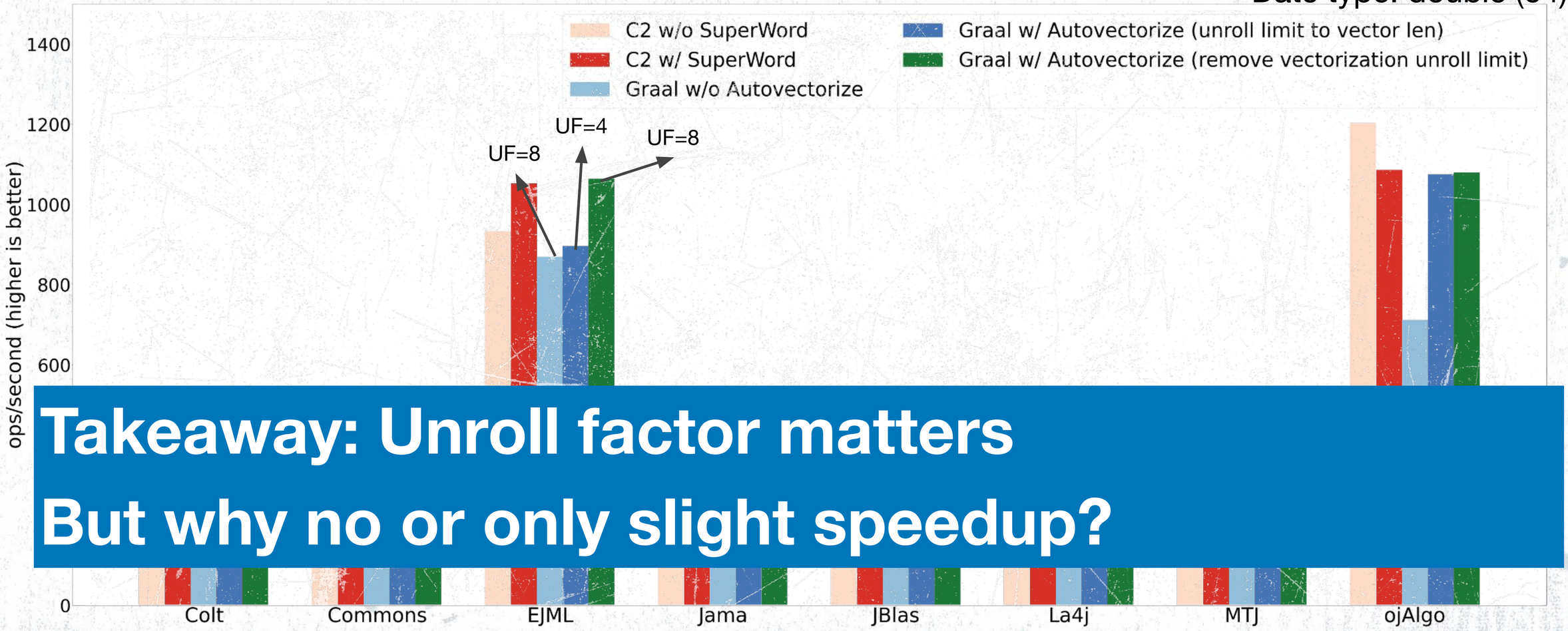


Takeaway: Graal w/ Autovectorize could speed up vectorizable patterns

[1] JVM JIT Compilers Benchmarks Report 19.11, https://ionutbalosin.com/2019/11/jvm-jit-compilers-benchmarks-report-19-11/

ops/second for Java-Matrix-Benchmark add()

Java linear algebra libraries
add() Operation: C = A + B
Matrix size: 1024
Date type: double (64)



Takeaway: Unroll factor matters

But why no or only slight speedup?

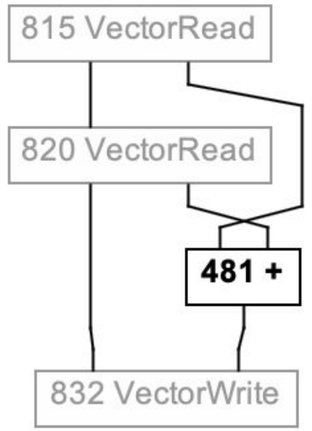
[1] Java Matrix Benchmark, https://lessthanoptimal.github.io/Java-Matrix-Benchmark/

Analysis: EJML Matrix add()

Java Code

```
for (int i = 0; i < length; i++) {
    output.set(i, a.get(i) + b.get(i));
}
```

Graal IR



[1] Manual vectorization using AVX vector intrinsics only runs about the same speed as 4 scalar FP adds on Ryzen? (2021, March 12). Stack Overflow. https://stackoverflow.com/questions/66602052/manual-vectorization-using-avx-vector-intrinsics-only-runs-about-the-same-speed
[2] https://www.uops.info/table.html

Analysis: EJML Matrix add() (continued)

Unroll factor for autovec <= vector register len (256) / data type (64) = 4

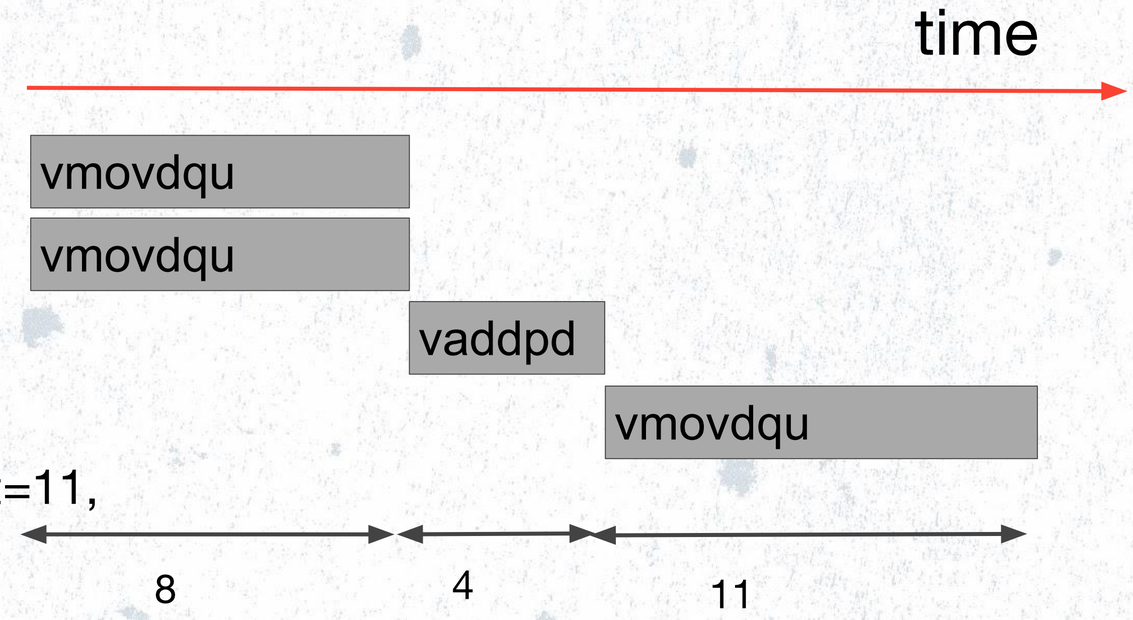
Assembly code w/ Autovectorize (unroll = 4)

```
vmovdqu 0x10(%r9,%rdi,8),%ymm0
vmovdqu 0x10(%r13,%rdi,8),%ymm1
vaddpd %ymm1,%ymm0,%ymm0
vmovdqu %ymm0,0x10(%rdx,%rdi,8)
```

vpadd
latency=4,
throughput=0.5

ymm wr m256, latency<=11,
throughput=0.5

m256 wr ymm, latency<=8,
throughput=0.5



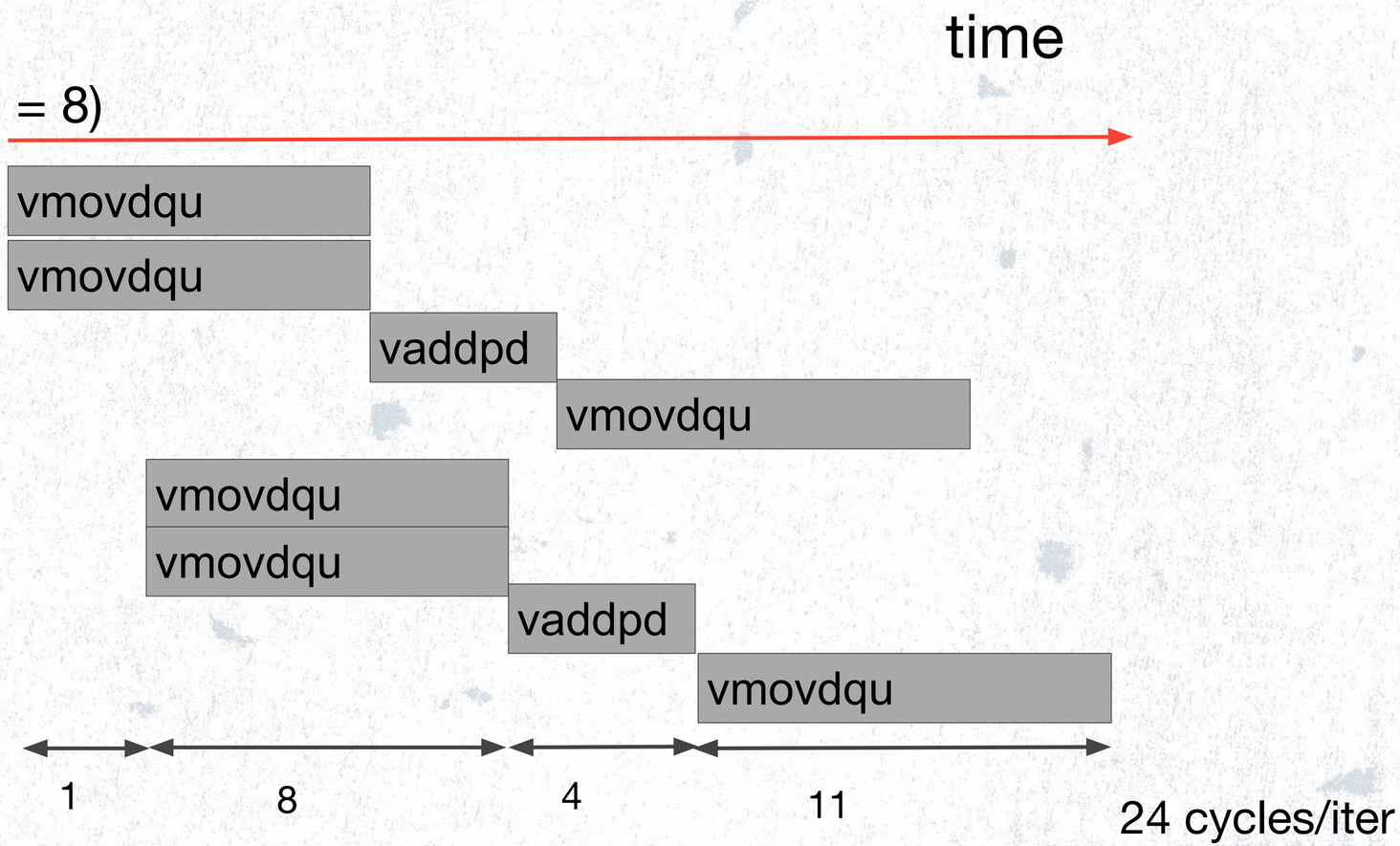
23 cycles/iter

[1] Manual vectorization using AVX vector intrinsics only runs about the same speed as 4 scalar FP adds on Ryzen? (2021, March 12). Stack Overflow. https://stackoverflow.com/questions/66602052/manual-vectorization-using-avx-vector-intrinsics-only-runs-about-the-same-speed
[2] https://www.uops.info/table.html

Analysis: EJML Matrix add() (continued)

Assembly code w/ Autovectorize (unroll = 8)

```
vmovdqu 0x10(%rdx,%r11,8),%ymm0
vmovdqu 0x10(%rdi,%r11,8),%ymm1
vaddpd %ymm1,%ymm0,%ymm0
vmovdqu %ymm0,0x10(%rcx,%r11,8)
movslq %r11d,%r8
vmovdqu 0x30(%rdi,%r8,8),%ymm0
vmovdqu 0x30(%rdx,%r8,8),%ymm1
vaddpd %ymm1,%ymm0,%ymm0
vmovdqu %ymm0,0x30(%rcx,%r8,8)
```



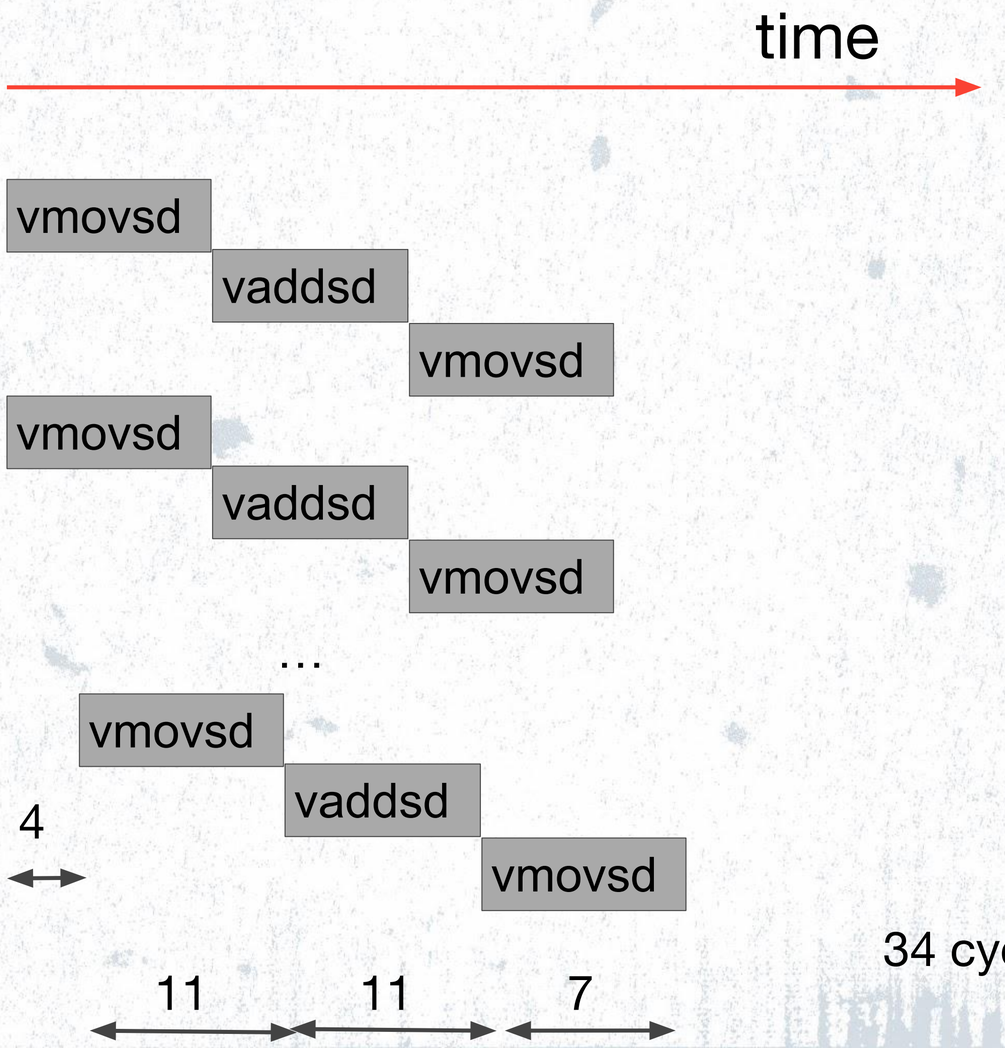
Analysis: EJML Matrix add() (continued)

Assembly code w/o Autovectorize (unroll = 8)

```
vmovsd 0x10(%r13,%rbx,8),%xmm0
vaddsd 0x10(%r14,%rbx,8),%xmm0,%xmm0
vmovsd %xmm0,0x10(%rax,%rbx,8)

vmovsd 0x18(%r13,%rbx,8),%xmm0
vaddsd 0x18(%r14,%rbx,8),%xmm0,%xmm0
vmovsd %xmm0,0x18(%rax,%rbx,8)
…

vmovsd 0x48(%r13,%rbx,8),%xmm0
vaddsd 0x48(%r14,%rbx,8),%xmm0,%xmm0
vmovsd %xmm0,0x48(%rax,%rbx,8)
```



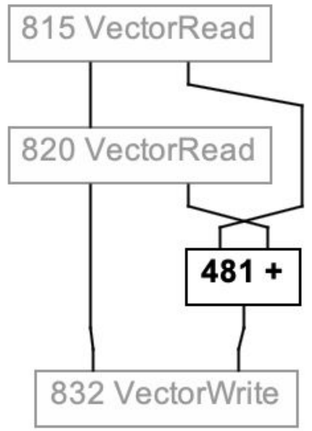
34 cycles/iter

Issues: low speedup with Autovectorize

EJML Matrix add()

```
for (int i = 0; i < length; i++) {
    output.set(i, a.get(i) + b.get(i));
}
```

Graal IR



Guess:

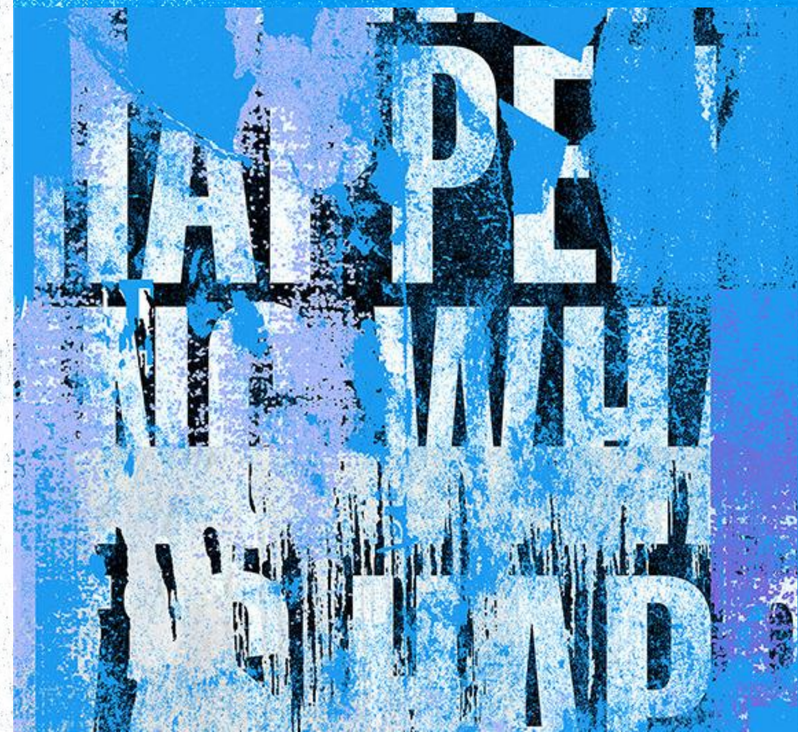
- Bottlenecked by memory access and loop-carried dependency

Possible Solution:

- Increase unroll factor
- take cycles/iteration estimation into consideration of cost model

[1] Manual vectorization using AVX vector intrinsics only runs about the same speed as 4 scalar FP adds on Ryzen? (2021, March 12). Stack Overflow. https://stackoverflow.com/questions/66602052/manual-vectorization-using-avx-vector-intrinsics-only-runs-about-the-same-speed

Next Steps



Existing cost models

- C2-like cost Model: *Veto* (implemented in Graal Autovectorization Part2)
- LLVM-like cost Model: cost = Vector cost - Scalar cost
- ML based cost Model[1]: NeuroVectorizer - Deep Reinforcement Learning to determine vectorization factor

[1] Ameer Haj-Ali, Nesreen K. Ahmed, Ted Willke, Yakun Sophia Shao, Krste Asanovic, and Ion Stoica. 2020. NeuroVectorizer: end-to-end vectorization with deep reinforcement learning. In Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization (CGO 2020). Association for Computing Machinery, New York, NY, USA, 242–255. DOI:https://doi.org/10.1145/3368826.3377928

Could we feed hardware support and latency to cost model?

Unroll factor, latency, throughput

Could we use ML model as cost model?

Linear Regression, NLP, Graph NN, etc.

Could we vectorize more using dynamic profiling?

```
for (int i = first; i < limit; i += step) {
        data[i] = left[i] + right[i];
    }
```

Thank You

Thanks Uma for the great mentorship and Graal expertise
Thanks Nik and David for their previous work on Graal Autovectorization and help