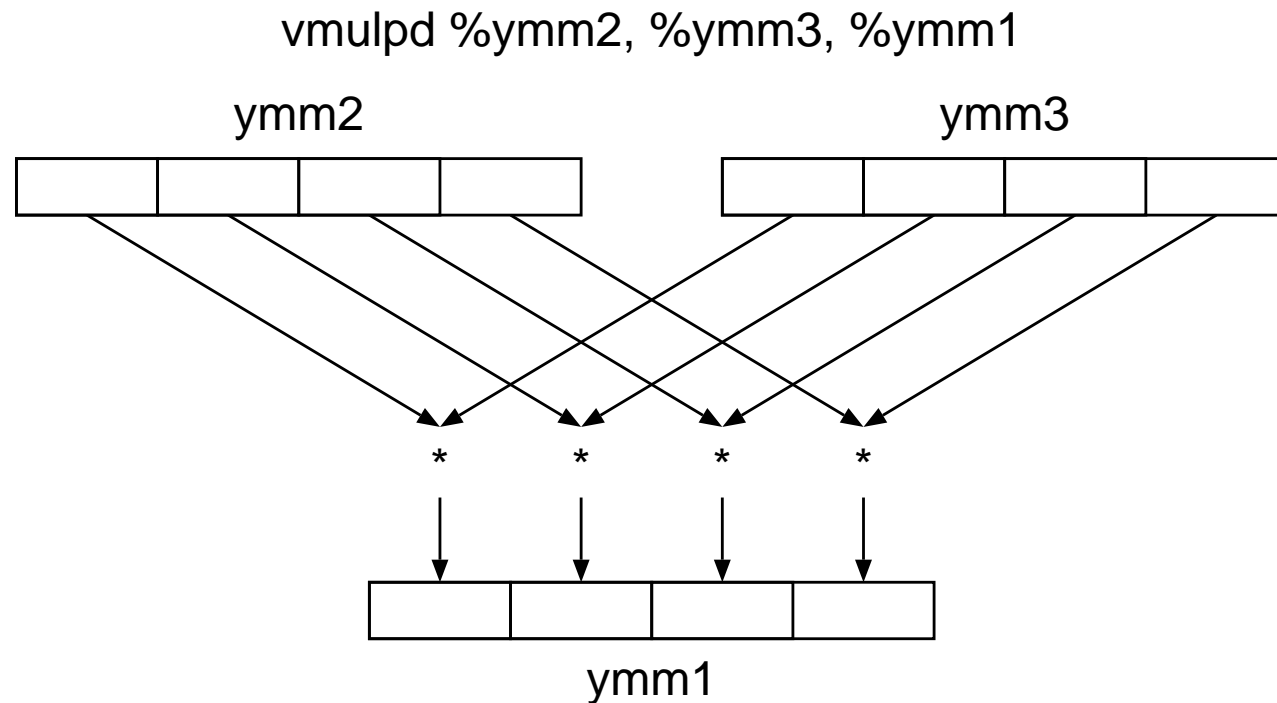


Software Vector Chaining

M. Anton Ertl
TU Wien

Data Parallelism and SIMD instructions

- Data parallelism in programming problems
- Hardware provides SIMD instructions
Cray-1 vector instructions, Intel/AMD SSE/AVX, ARM Neon/SVE



- Little programming language support

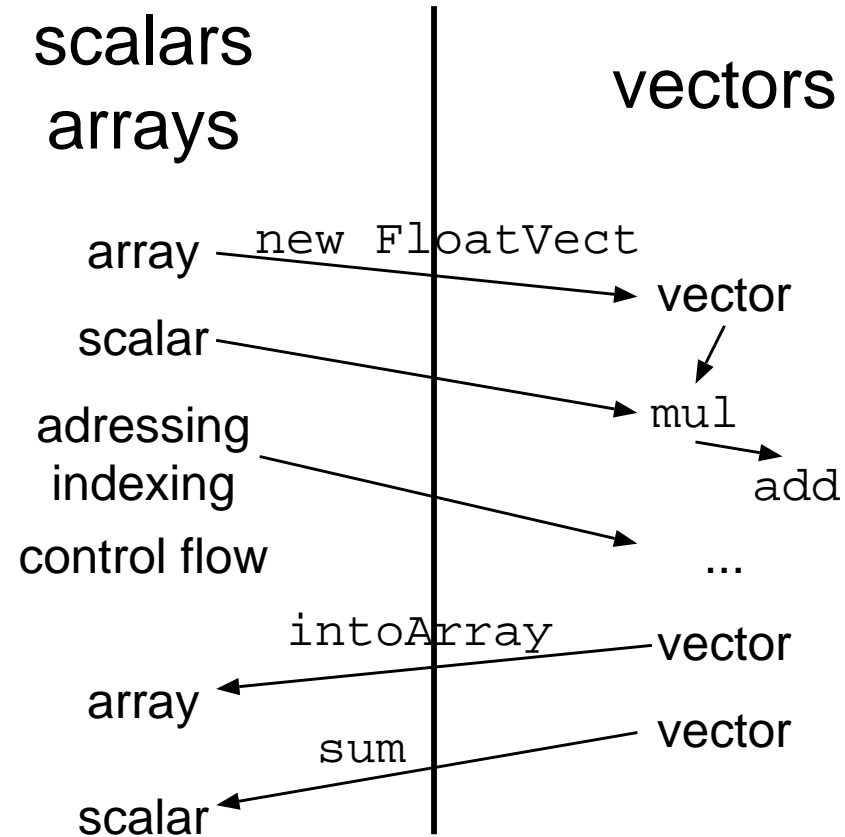
Programming language support: How?

- Manual Vectorization
- Application vector length
- Opaque, immutable vectors with value semantics
- Vector stack

```
: vcomp ( va vb -- vc )  
  vdup sf*v vswap vdup sf*v sf+v sfnegatev ;
```

Properties, benefits and drawbacks

- Vectors are immutable (value semantics)
 - Explicit conversion from/to memory
 - + gives control to programmer, who can make conversions infrequent
 - + Padding to SIMD granularity
 - + Aligning to SIMD granularity
 - + No aliasing problems
 - + Results do not overlap input operands
 - + only explicit dependences
 - + vectors are a separate world
 - + Compiler can arrange computations



Implementation

simple sf+v

simple:

```
vmovaps (%rdi,%r10,1),%ymm0
vaddps  (%rsi,%r10,1),%ymm0,%ymm0
vmovaps %ymm0, (%rdx,%r10,1)
add     $0x20,%r10
cmp     %r10,%rcx
ja     simple
```

fused vcomp

fused:

```
vmovaps (%rdi,%r10,1),%ymm0
vmulps  %ymm0,%ymm0,%ymm1
vmovaps (%rsi,%r10,1),%ymm2
vmulps  %ymm2,%ymm2,%ymm3
vaddps  %ymm1,%ymm3,%ymm1
vxorps  %ymm1,%ymm4,%ymm1
vmovaps %ymm0, (%rdx,%r10,1)
add     $0x20,%r10
cmp     %r10,%r9
ja     fused
```

... but how?

Who performs vector loop fusion?

Compiler

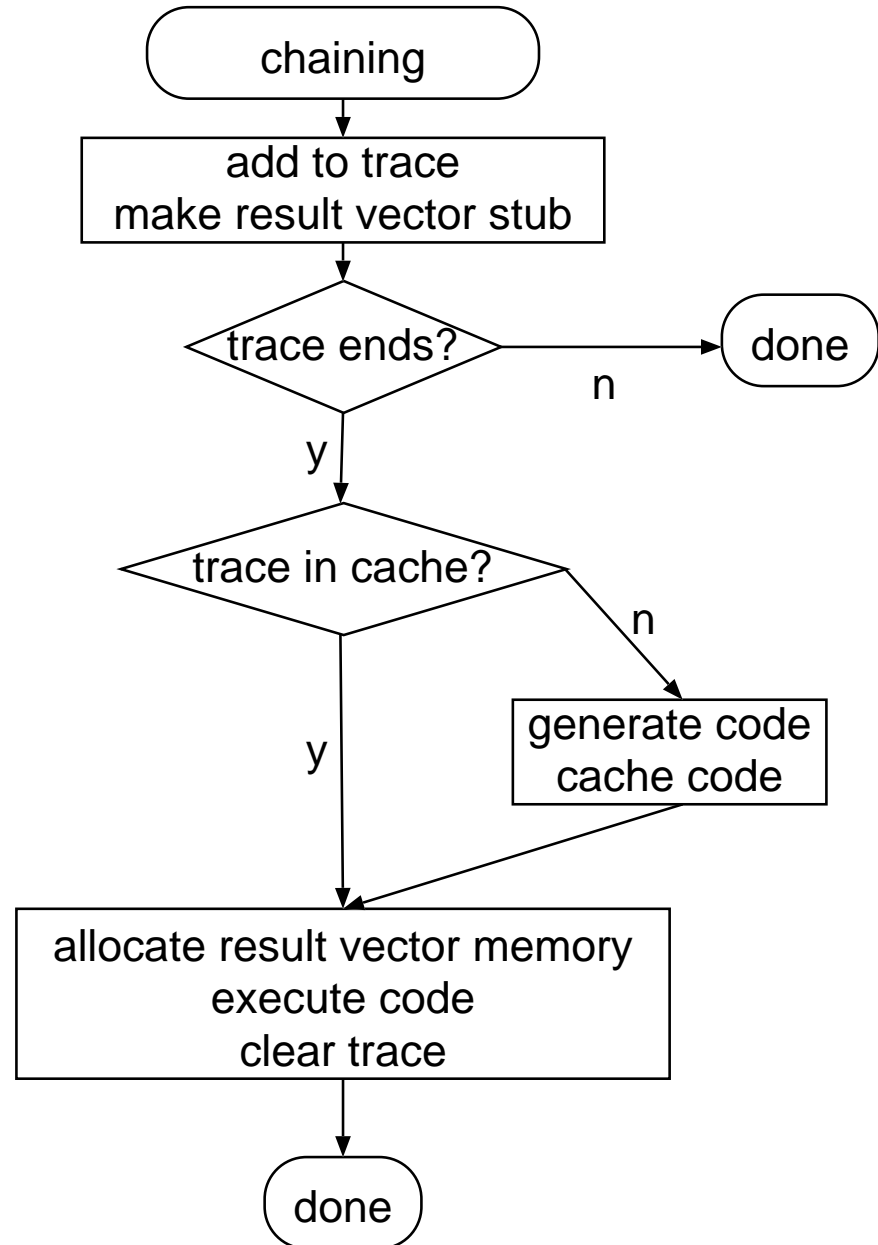
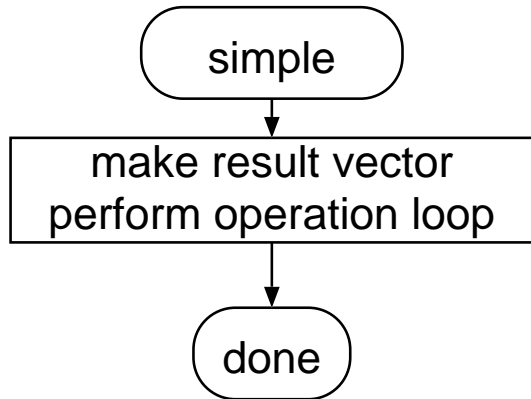
- + Low run-time overhead
- High implementation effort?
- Control-flow may limit fusion
- Aliasing plays a role

Run-time Library

- High run-time overhead
- + Low implementation effort
- + Fuses across control flow
- + Dependencies resolved

Software Vector Chaining

Implementing a vector operation



Generate code

```
vdup sf*v vswap vdup sf*v sf+v sfnegatev
```

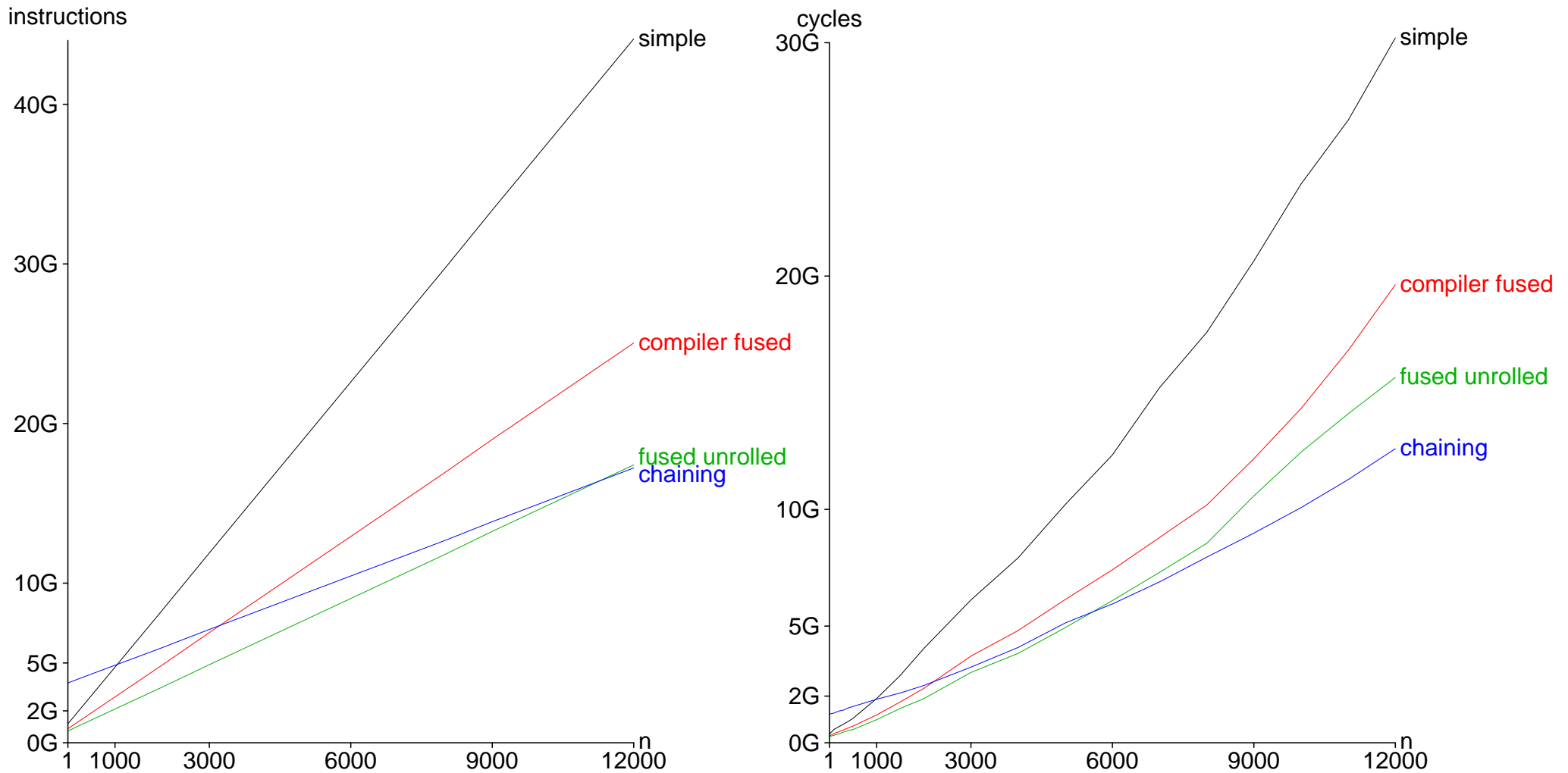
```
                $24147C0 refs= 0 bytes=16 $24147A0 :14  
                $2414B10 refs= 0 bytes=16 $2415150 :15  
sftimesv_      15 15 temporary                    :16  
sftimesv_      14 14 temporary                    :17  
sfplusv_       16 17 temporary                    :18  
sfnegatev_     18  0 $2415030 refs= 0 bytes=16 $2417300 :19
```

fused:

```
vmovaps (%rdi,%r10,1),%ymm0  
vmulps  %ymm0,%ymm0,%ymm1  
vmovaps (%rsi,%r10,1),%ymm2  
vmulps  %ymm2,%ymm2,%ymm3  
vaddps  %ymm1,%ymm3,%ymm1  
vxorps  %ymm1,%ymm4,%ymm1  
vmovaps %ymm0, (%rdx,%r10,1)  
add     $0x20,%r10  
cmp     %r10,%r9  
ja      fused
```


Evaluation

Multiply 50×50 with $50 \times n$ Double matrix for varying n , 500 times on Core i5 6600K (Skylake)



Conclusion

- How to use SIMD instructions for data parallelism?
- Manual vectorization, application vector size, opaque vectors gives freedom to the compiler/library writer
- Software vector chaining
 - Build trace at run-time
 - Compile if not cached
- + Can be implemented as library
 - 315 source lines of code
- + For long vectors $> 2\times$ as fast as *simple*
- High per-operation overhead
 - Useful only for long vectors
 - Select between *simple* and *chaining* per operation
- github.com/AntonErtl1/vectors
 - Paper at ManLang 2018
 - <https://www.complang.tuwien.ac.at/papers/ertl18.pdf>