

# CS 610: Vectorization

**Swarnendu Biswas**

Department of Computer Science and Engineering,  
Indian Institute of Technology Kanpur

Sem 2024-25-I



# Different Levels of Parallelization in Hardware

## Instruction-level Parallelism

Microarchitectural techniques like pipelining, OOO execution, and superscalar instruction issue

## Data-level Parallelism

Use Single Instruction Multiple Data (SIMD) vector processing instructions and units

## Thread-level Parallelism

Hyperthreading

# Vectorization

- Vectorization is the process of transforming a scalar operation on single data elements at a time (SISD) to an operation on multiple data elements at once (SIMD)
- Loop vectorization transforms a program so that the same operation is performed at the same time on several vector elements

*Don't use a single Vector lane/thread!*

*Un-vectorized and un-threaded software will under perform*



*Permission to Design for All Lanes*

*Threading and Vectorization needed to fully utilize modern hardware*

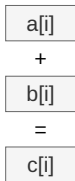


# Vectorization

```
double *a, *b, *c;  
for (int i = 0; i < N; i++)  
    c[i] = a[i] + b[i];
```

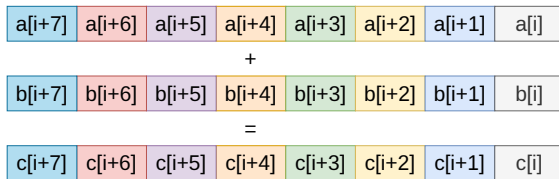
## Scalar mode

One instruction (e.g., vaddsd/vaddss) produces one result



## Vector mode

One instruction (e.g., vaddpd/vaddps) can produce multiple results



# Vectorization

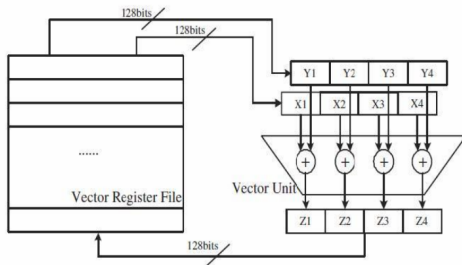
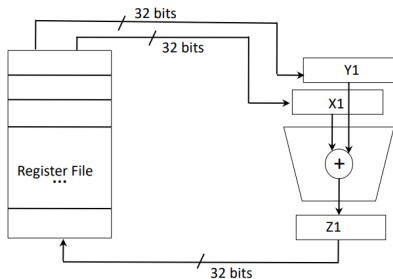
$n$  times {

```
ld r1, addr1
ld r2, addr2
add r3, r1, r2
st r3, addr3
```

```
for (i=0; i<n; i++) {
    c[i] = a[i] + b[i];
}
```

$\frac{n}{4}$  times {

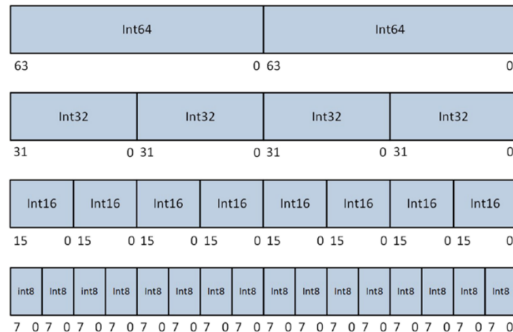
```
ldv vr1, addr1
ldv vr2, addr2
addv vr3, vr1, vr2
stv vr3, addr3
```



# SIMD Vectorization

- Use of SIMD units can speed up the program
- Intel SSE has 128-bit vector registers and functional units
- Assuming a single ALU, these SIMD units can execute 4 single precision floating point number or 2 double precision operations in the time it takes to do only one of these operations by a scalar unit

128-bit wide operands using integer types

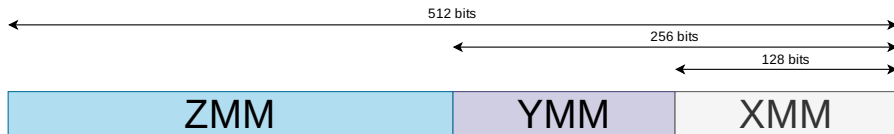


# Intel-Supported SIMD Extensions

SIMD Extensions	Width (bits)	DP calculations	SP calculations	Introduced
SSE2/SSE3/SSE4	128	2	4	~2001–2007
AVX/AVX2	256	4	8	~2011–2015
AVX-512	512	8	16	~2017

Other platforms that support SIMD have different extensions (e.g., ARM Neon and Power AltiVec)

# Intel-Supported SIMD Extensions



## 64-bit architecture

SSE	XMM0–XMM15
AVX	YMM0–YMM15
AVX-512	ZMM0–ZMM31

Low-order 128 bits of each YMM register is aliased to a corresponding XMM register

Low-order 256 and 128 bits are aliased to registers YMM0–YMM31 and XMM0–XMM31 respectively



## Example instructions

- Move: (V)MOV[A/U]P[D/S]
- Comparing: (V)CMP[P/S][D/S]
- Arithmetic operations: (V)[ADD/SUB/MUL/DIV][P/S][D/S]

## Instruction decoding

V AVX

P,S packed, scalar

A,U aligned, unaligned

D,S double-, single-precision

B,W,D,Q byte, word, doubleword, quadword integers

# x86\_64 Vector Operations

**movss xmm1, xmm2** Move scalar single-precision floating-point value from xmm2 to xmm1

**vmovapd xmm1, xmm2** Move aligned packed double-precision floating-point values from xmm2 to xmm1

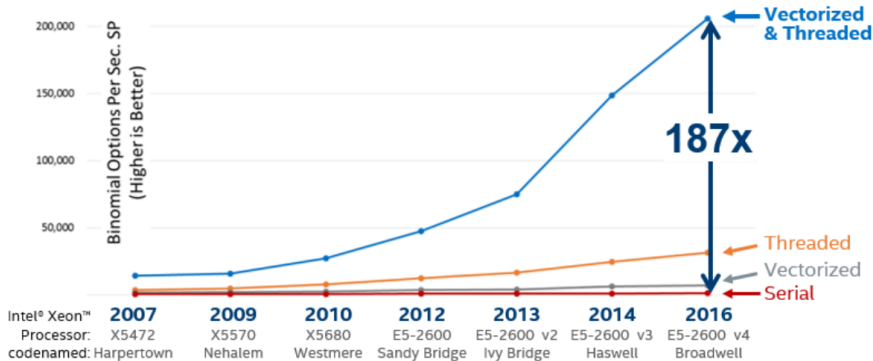
```
vaddss xmm0, xmm1, xmm2
```

```
xmm0[31:0] = xmm1[31:0] + xmm2[31:0]  
xmm0[127:32] = xmm1[127:32]  
ymm0[255:128] = 0
```

```
vaddsd xmm0, xmm1, xmm2
```

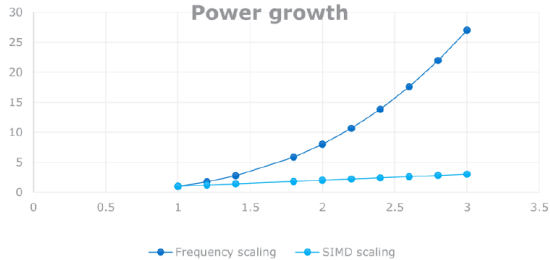
```
xmm0[63:0] = xmm1[63:0] + xmm2[63:0]  
xmm0[127:64] = xmm1[127:64]  
ymm0[255:128] = 0
```

# The combined effect of vectorization and threading



**The Difference Is Growing With Each New Generation of Hardware**

# Why SIMD vector parallelism?



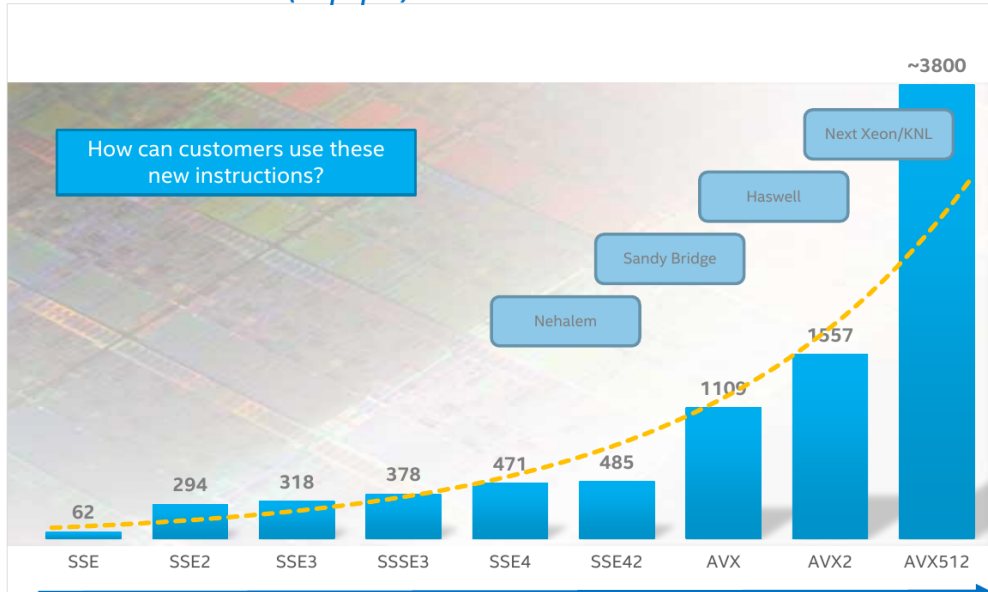
Wider SIMD -- Linear increase in area and power

Wider superscalar – Quadratic increase in area and power




Higher frequency – Cubic increase in power

With SIMD we can go faster with less power

# Cumulative (app.) # of Vector Instructions



# References

-  M. Voss. Topics in Loop Vectorization.
-  M. Garzarán et al. Program Optimization Through Loop Vectorization.
-  K. Rogozhin. Vectorization.