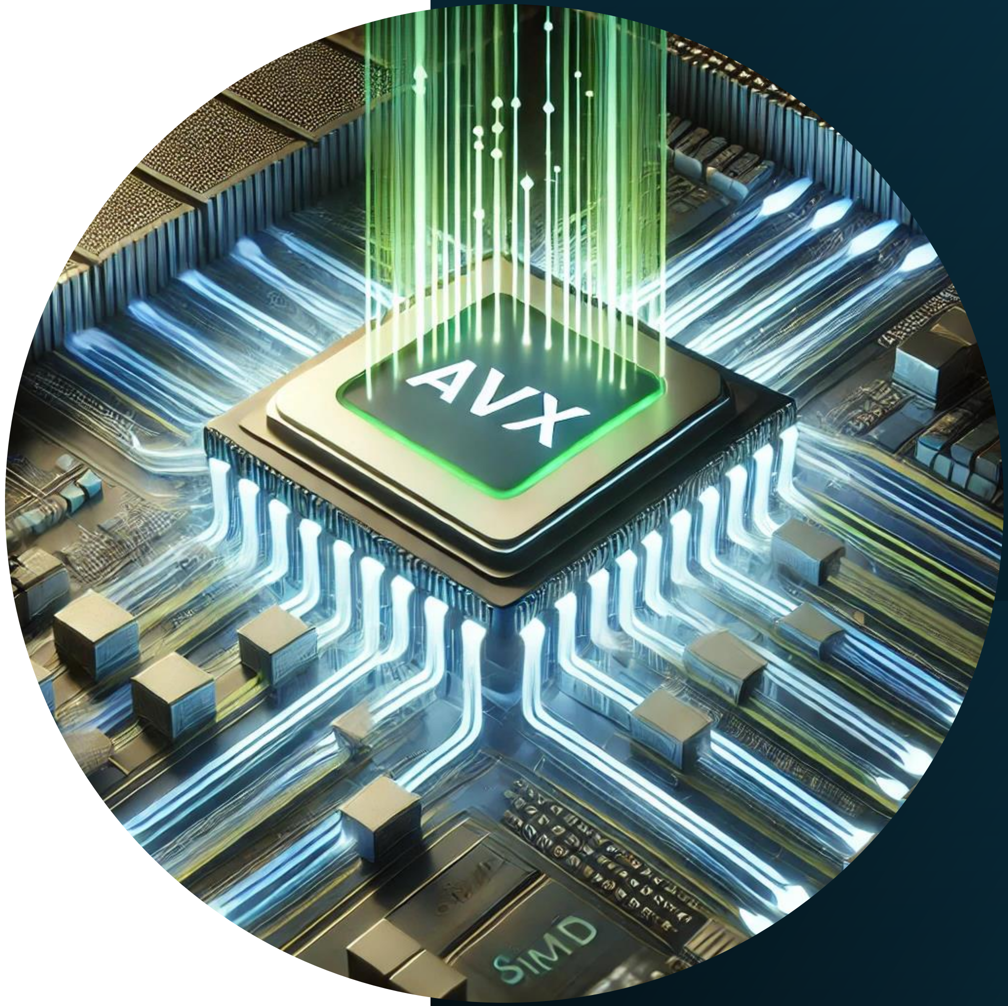
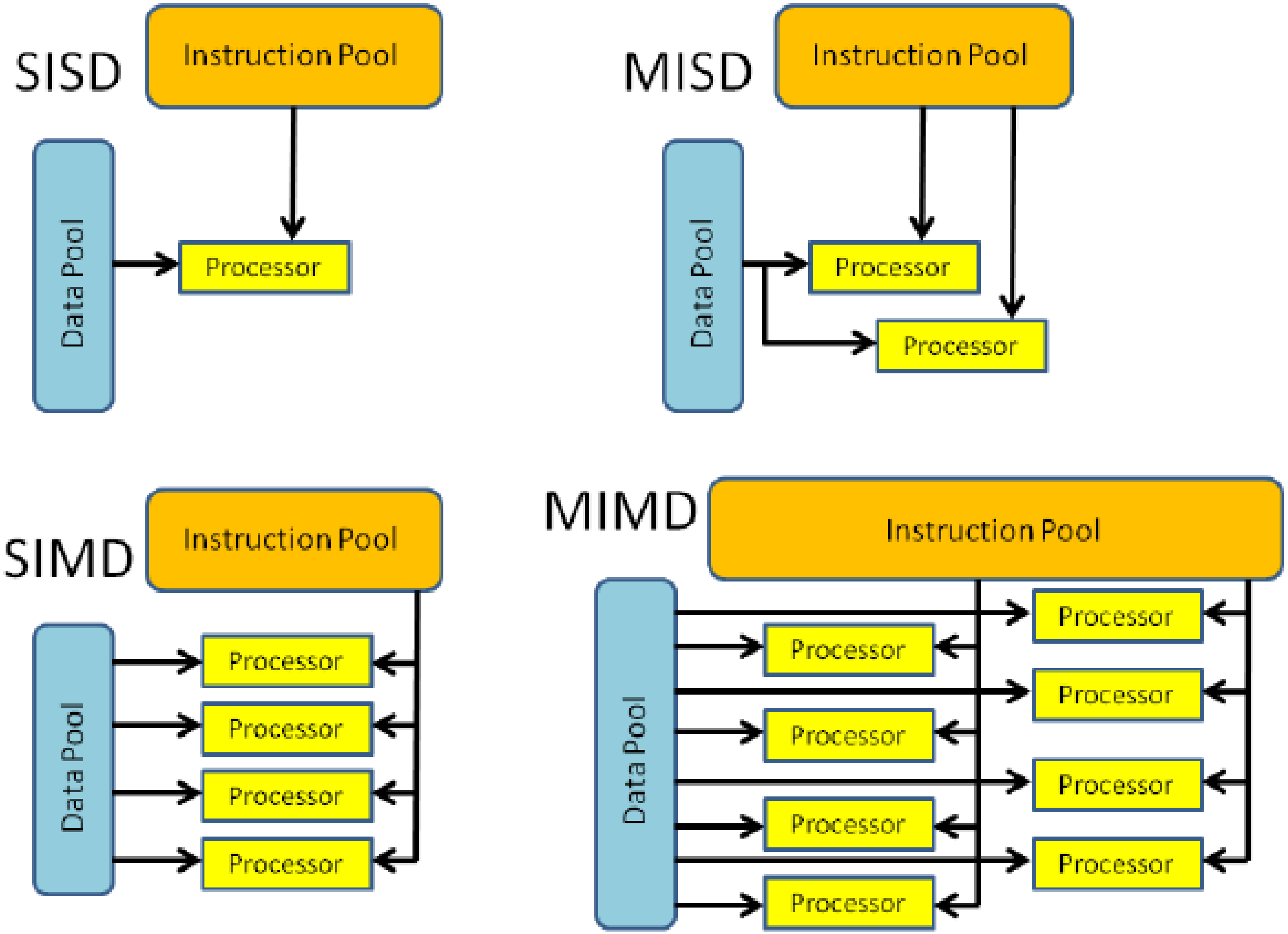
Code Profiling and Optimization

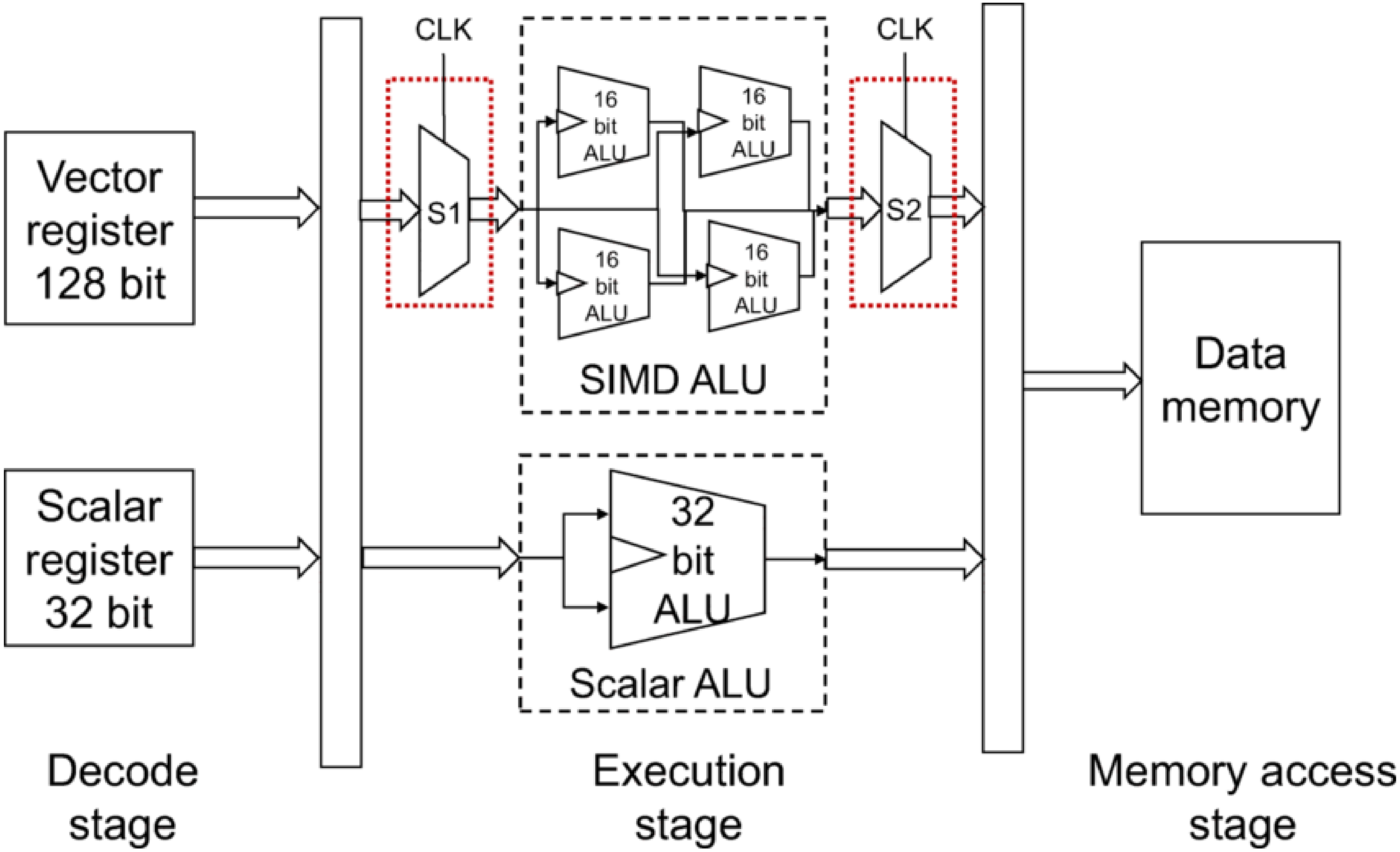


Vectorization
Subhrajit & Pratyush

Flynn’s Taxonomy



SISD v/s SIMD


Vector register 128 bit
Scalar register 32 bit
CLK
S1
16 bit ALU
16 bit ALU
16 bit ALU
16 bit ALU
SIMD ALU
CLK
S2
32 bit ALU
Scalar ALU
Data memory
Decode stage
Execution stage
Memory access stage

Focusing on Intel SIMD

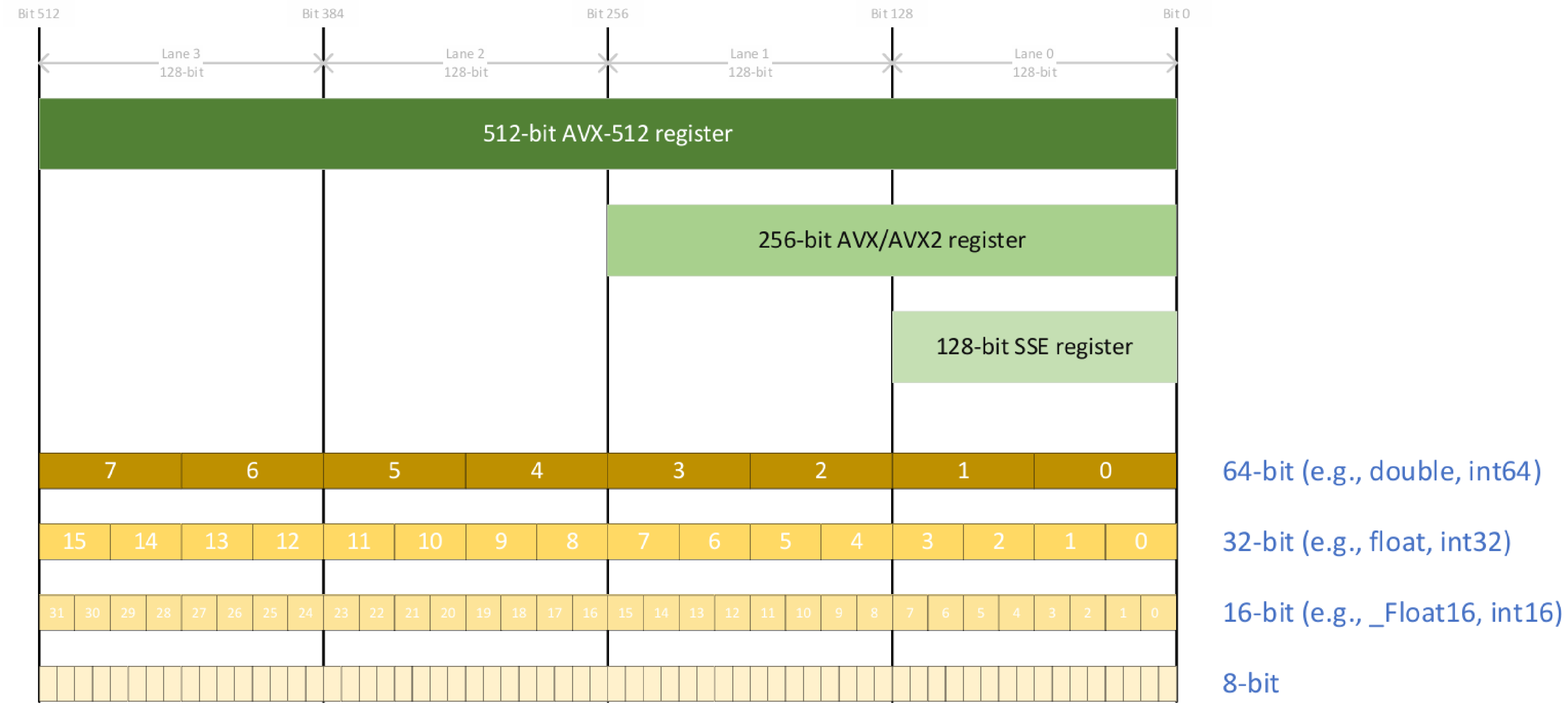


Figure 1. Layout of Various Sizes of SIMD Register and How Each Can Be Broken Down into Smaller Subgroups of Elements

Intel AVX

- The Intel® Advanced Vector Extensions (Intel® AVX) family of instruction sets on Intel processors provides a rich variety of capabilities for supporting many different single instruction, multiple data (SIMD) instructions and data types.

AVX Support

Does your CPU have AVX support?

Check the list of CPU flags using lscpu

Vectorization

Auto-Vectorization

Explicit-Vectorization

Auto-Vectorization

GCC and various other compilers have automatic vectorization support depending on the CPU

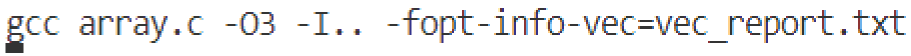
For gcc, use –O2 and –O3 flag.

Auto-Vectorization

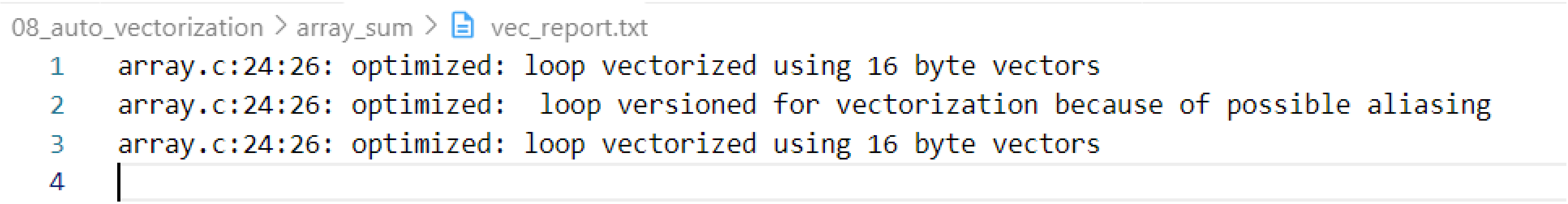
- GCC and various other compilers have automatic vectorization support depending on the CPU
- For gcc, use –O3 flag.
  - By default, it uses SSE (128-bit/16-Byte Vectors)
  - Vectorization also depends upon the alignment of the data. If the data is not aligned properly, the compiler may not be able to vectorize the code.
- https://www.intel.com/content/www/us/en/developer/articles/training/explicit-vector-programming-best-known-methods.html
  - 1. **Use Aligned Data Allocation** (`_mm_malloc(ptr,<alignment-size>)` and `_mm_free(ptr)` for memory allocation and deallocation)
    - 16-byte alignment for SSE
    - 32-byte alignment for AVX (and AVX2)
    - 64-byte alignment for AVX512
  - 2.**Hint the Compiler about Alignment** (use the `assume_aligned` attribute)

Auto-Vectorization

- Checking whether Vectorization Happened or not
- use the `-fopt-info-vec` flag to generate a report on vectorization.
- Ex:



```
gcc array.c -O3 -I.. -fopt-info-vec=vec_report.txt
```

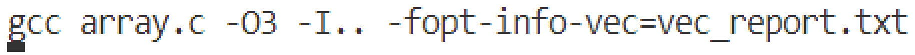


08_auto_vectorization > array_sum > vec_report.txt

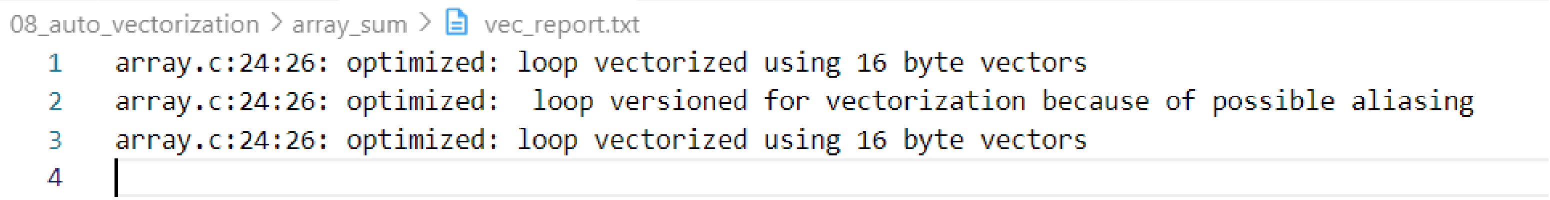
```
array.c:24:26: optimized: loop vectorized using 16 byte vectors
array.c:24:26: optimized:  loop versioned for vectorization because of possible aliasing
array.c:24:26: optimized: loop vectorized using 16 byte vectors

```

Auto-Vectorization

- But my CPU has AVX512 support, it used 16 Byte -> 128-bits, not 512-bit vectors
- Well, we have to provide one more flag for AVX512
- Use –mavx512f additionally

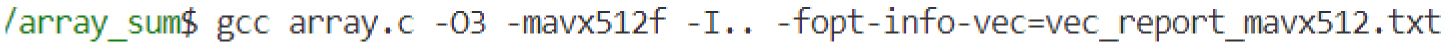


```
gcc array.c -O3 -I.. -fopt-info-vec=vec_report.txt
```

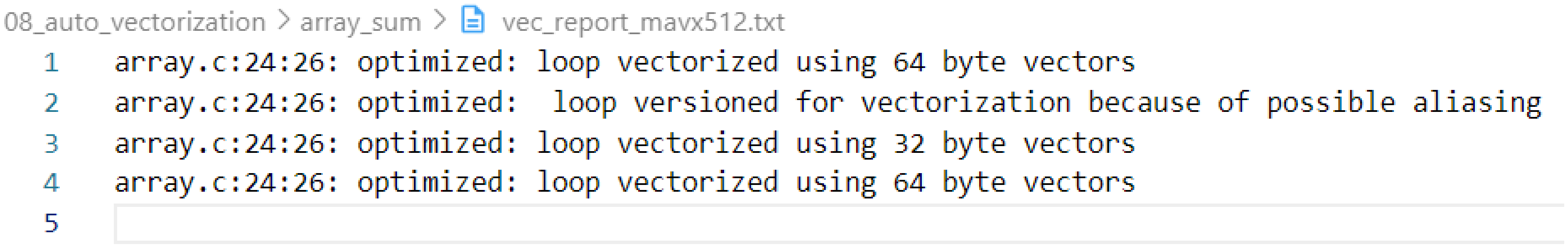


08_auto_vectorization > array_sum > vec_report.txt

```
array.c:24:26: optimized: loop vectorized using 16 byte vectors
array.c:24:26: optimized:  loop versioned for vectorization because of possible aliasing
array.c:24:26: optimized: loop vectorized using 16 byte vectors
```



```
/array_sum$ gcc array.c -O3 -mavx512f -I.. -fopt-info-vec=vec_report_mavx512.txt
```



08_auto_vectorization > array_sum > vec_report_mavx512.txt

```
array.c:24:26: optimized: loop vectorized using 64 byte vectors
array.c:24:26: optimized:  loop versioned for vectorization because of possible aliasing
array.c:24:26: optimized: loop vectorized using 32 byte vectors
array.c:24:26: optimized: loop vectorized using 64 byte vectors
```