**Charakteristika der Befehlssätze SSE, AVX(2) und AVX-512.**

SSE (Streaming SIMD Extensions) wurde im Jahr 1999 eingeführt und arbeitet mit 128-Bit-Registern. Es ermöglicht grundlegende SIMD-Berechnungen wie Addition und Multiplikation, ist aber in der Parallelisierung begrenzt, da es keine Maskierungsfunktionen unterstützt.

AVX und AVX2 wurden ab 2011 eingeführt und erweitern die Registergröße auf 256 Bit, was die Verarbeitung von doppelt so vielen Daten ermöglicht. AVX2 brachte zudem Unterstützung für Ganzzahloperationen und Gather-Befehle für nicht-lineare Speicherzugriffe. Jedoch können diese Erweiterungen zu höherem Energieverbrauch führen, was manchmal eine Drosselung der Prozessorleistung erfordert.

AVX-512, seit 2017 verfügbar, nutzt 512-Bit-Register und bietet fortgeschrittene Features wie Maskenregister für selektive Operationen innerhalb eines Vektors. Das erlaubt eine noch höhere Parallelisierung und Flexibilität, ist aber aufgrund des erhöhten Energieverbrauchs und der Verfügbarkeit meist auf High-End-Prozessoren beschränkt, die in Bereichen wie High-Performance-Computing oder Machine Learning eingesetzt werden.

**Wie kann Speicher-Aliasing die Performance beeinflussen?**

Speicher-Aliasing tritt auf, wenn mehrere Zeiger auf dieselbe Speicheradresse zeigen, sodass der Compiler nicht sicher sein kann, ob eine Veränderung an einem Speicherort nicht auch andere Bereiche beeinflusst. Dies hindert den Compiler daran, bestimmte Optimierungen wie Loop-Unrolling oder Vektorisierung durchzuführen, weil er vorsichtig sein muss. Die Folge sind weniger effiziente Codeausführungen mit unnötigen Speicherzugriffen und reduzierter Parallelisierung, was die Performance beeinträchtigt. Um das zu vermeiden, kann man in C/C++ das \_\_restrict\_\_-Schlüsselwort verwenden, um dem Compiler mitzuteilen, dass Zeiger nicht überlappen, oder man nutzt temporäre Variablen, um direkte Speicherreferenzen zu minimieren.

**Was sind die Vorteile von Unit-Strides (Schrittweite 1) gegenüber größeren Strides (z. B. Schrittweite 8)?**

Unit-Strides, also eine Schrittweite von 1, bieten den Vorteil effizienter Speicherzugriffe. Durch den kontinuierlichen Zugriff auf aufeinanderfolgende Speicheradressen werden Latenzen reduziert und die Speicherbandbreite optimal genutzt. Außerdem funktionieren Speichervorabrufe (Prefetching) besser, und der Compiler kann leichter Vektorisierung anwenden, da die Speicherzugriffsmuster regelmäßig sind. Bei größeren Strides wie Schrittweite 8 sind die Speicherzugriffe nicht kontinuierlich, was zu zusätzlichen Speicheroperationen und erhöhten Cache-Misses führt. Das erschwert die Optimierung und kann die Performance deutlich beeinträchtigen.

**Wann würde man es vorziehen, Datensätze im Speicher als Structure of Arrays zu arrangieren?**

Man würde eine Structure of Arrays (SoA) bevorzugen, wenn man vektorisierbare Berechnungen durchführen möchte oder wenn man häufig auf einzelne Felder aller Datensätze zugreifen muss. In SoA werden die einzelnen Felder in separaten Arrays gespeichert, was sequenzielle Speicherzugriffe ermöglicht und somit ideal für SIMD-Operationen ist. Das ist besonders vorteilhaft in Anwendungen wie physikalischen Simulationen oder bei der Verarbeitung großer Datenmengen. Allerdings kann SoA die Datenlokalität reduzieren, da die zusammengehörigen Felder eines Datensatzes verstreut im Speicher liegen. Wenn man oft auf alle Felder eines einzelnen Datensatzes zugreifen muss, könnte ein Array of Structures (AoS) effizienter sein, da hier alle Felder eines Datensatzes zusammenhängend gespeichert sind.