Exam Assignments Algorithm Engineering

Florian Zeidler 194888

1. März 2022

Inhaltsverzeichnis

1 Course Intro and OpenMP

1.1 Describe how parallelism differs from concurrency.

Hat man z.B. eine CPU und mehrere Aufgaben, so kann man es sich bei Concurrency so vorstellen, dass die CPU zwischen den Aufgaben hin und her schaltet (wenn z.B. auf Speicherzugriffe für eine Aufgabe gewartet werden muss). Bei Parallelism hat man mehrere Ärbeiter"(mehrere Kerne oder CPU's), welche gleichzeitig (simultan möglich) mehrere Aufgaben bearbeiten. Dieser Zusammenhang wird in Abbildung 1 verdeutlicht.

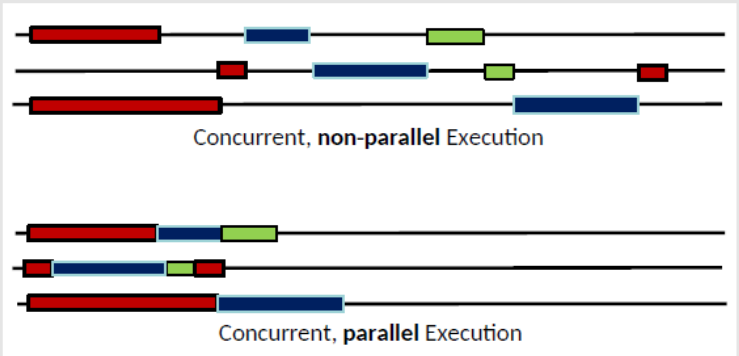


Abbildung 1: Concurrency vs. Parallelism [1]

1.2 What is fork-join parallelism?

Dieses Modell beschreibt die Arbeitsweise innerhalb von OpenMP. Es liegt ein Masterthread vor, welcher die gesamte Ausführungszeit existiert. Dieser Thread erstellt dann weitere Threads (Fork), sobald eine parallele Region betreten wird. Verlässt man den parallelen Bereich, so werden die Threads wieder eingesammelt (Join) und es läuft nur der Masterthread weiter (entspricht sequentiellen Anteilen). Dies kann wiederholt in einem Programm auftreten. Dieser Zusammenhang wird in Abbildung 2 verdeutlicht.

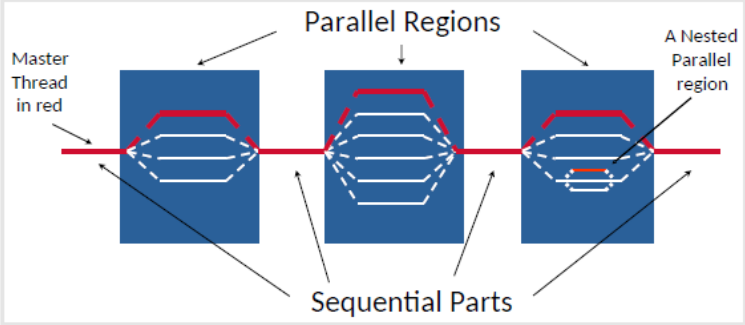


Abbildung 2: Fork Join Parallelism [1]

1.3 Discussion: A Programmer's Perspective.

Thema: Single-Instruction, Multiple-Data (SIMD) Parallelism Zu dieser Thematik habe ich etwas recherchiert und herausgefunden, dass es verschiedene Arten von Unterteilungen nach Flynn gibt, in welcher SIMD vorkommt.
S steht für Single, M steht für Multiple, I steht für Instruction und D steht für Data.

SISD
Klassischer Rechner von früher. Hier wird eine Instruktion mit einer Zahl nacheinander bearbeitet.

MISD
Hierbei hätte man einen starken Performanceverlust, weshalb diese Kombination nicht verwendet wird.

SIMD (Vektor)
Wird z.B. bei Vektoraddition verwendet (Eine Addition welche sich nicht auf eine Zahl sondern auf Vektoren bezieht) Man spart sich hierbei die Befehle für die einzelnen Additionen.

MIMD (Parallele Arbeit)
Mehrere Befehle, welche sich auf mehrere Daten beziehen. Mehrere unabhängige Recheneinheiten (CPU's) für Bearbeitung von Threads werden hier genutzt (Multithreading).

1.4 Performance gains after Moore's law ends.

Moore's Gesetz beschreibt den Miniaturization Trend, welcher besagt, dass sich die Anzahl der Transistoren innerhalb eines Computers alle 2 Jahre verdoppeln, und dies indirekt dann auch für die Performance eines Computers gilt. Dieser Trend konnte einige Jahre beobachtet werden, doch mittlerweile scheint dieses Wachstum ausgeschöpft worden zu sein. Somit ist es sinnvoll weitere Optionen zu betrachten, mit welchen man die Performance eines Computers steigern kann.

Software
Die Entwicklungszeitminimierung hat zu Performanceeinbußen bei der Entwicklung von Software geführt, welche nun optimiert werden könnten. (z.B. parallele Prozessoren + Vektoreinheiten in den Fokus setzen)

Algorithmen
Durch das Anpassen des Algorithmuses an die jeweilige Hardware, kann ein großer Speed Up erreicht werden. (Typisches Beispiel ist die Verbesserung des Maximum Flow Problems)

Hardware
Hier könnte man bestimmte Hardware entwerfen, welche in einer bestimmten Domain optimal arbeitet. Zudem kann es sinnvoll sein, die Anzahl an parallel arbeitenden Prozessoren zu erhöhen.

Diese neuen Grundsäulen werden in Abbildung 3 noch einmal verdeutlicht.

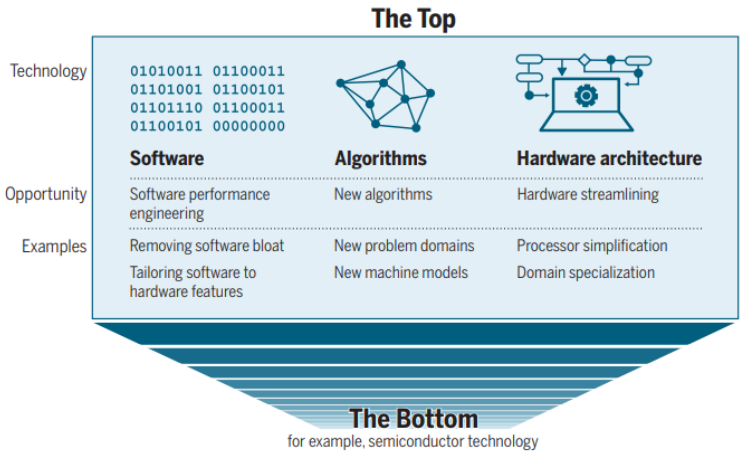


Abbildung 3: Performance gains after Moore's law end [1]

2 FS RC and Schedules

2.1 What causes false sharing?

Man hat hierbei beispielsweise eine Situation, in der zwei CPU's gleichzeitig arbeiten (Siehe Abb. 4). Nun greifen die beiden CPU's mittels Threads auf die gleiche Cacheline zu (aber auf eine andere Stelle innerhalb der Cache Line). Sobald nun eine der CPU's eine Veränderung an der Cacheline vornimmt, wird ein Update notwendig sein, da es sonst zu Inkohärenzen kommen könnte, da die andere CPU mit der veralteten Cacheline arbeiten würde. Dies ist im Allgemeinen ungünstig, da man die veränderte Variable vielleicht auch gar nicht benötigt in dem anderen Thread und man somit einen unnötigen Performanceverlust hätte, da die Cacheline erneut ausgelesen werden muss.

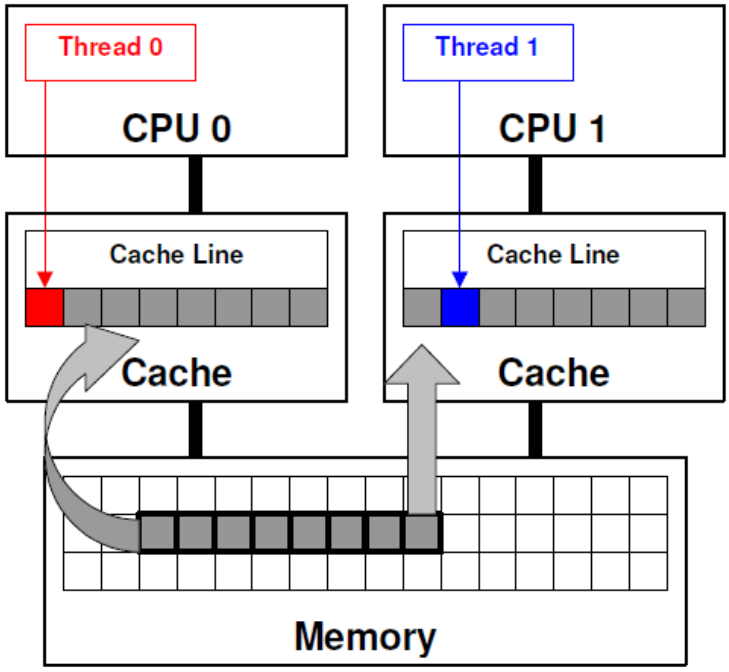


Abbildung 4: False Sharing [2]

2.2 How do mutual exclusion constructs prevent race conditions?

Race Conditions treten auf, wenn zwei oder mehrere Threads die selbe geteilte Variable verändern möchten. Hierbei ist ein exklusiver Zugang für die Threads sinnvoll, sodass Inkohärenzen vermieden werden. Ein derartige Zugriff könnte über einen Mutex erstellt werden, oder aber auch über OpenMP Direktiven wie z.B. pragma omp critical. Im Allgemeinen sollten solche Situationen aber minimiert werden, da hierdurch die Performance verschlechtert werden könnte.

2.3 Explain the differences between static and dynamic schedules in OpenMP.

Static: Jeder Thread bekommt einen festen Arbeitsteil zugewiesen.
Dynamic: Während der Laufzeit des Programms wird dynamisch entschieden, welcher Thread das nächste Arbeitspäckchen bearbeitet.

Schedule(dynamic) ist hilfreich, wenn der Arbeitsaufwand der Päckchenünterschiedlich groß ist. Bei gleichem Arbeitsaufwand ist Schedule(static) für die Performance besser. Es ist auch möglich die Größe der Arbeitspäckchen (Chunks) zu variieren.

2.4 What can we do if we've found a solution while running a parallel for loop in OpenMP, but still have many iterations left?

Vorgehensweise:
Man erstellt eine shared Variable, auf welche nur atomar zugegriffen werden kann. Wird diese dann auf einen Wert gesetzt (z.B. False/True), sobald man eine Lösung gefunden hat, kann man eine Bedingung in der for-Schleife nutzen, welche dann besagt, dass in den weiteren Durchläufen keine Arbeit mehr verrichten werden soll (Mittels continue).

Wichtig: Der Loop wird hierdurch nicht gestoppt, aber es wird unnötige Arbeit verhindert, sodass die Performance davon profitiert

3 OC Reductions Barriers and more

3.1 How does the ordered clause in OpenMP work in conjunction with a parallel for loop?

Die Ordered Clause funktioniert folgendermaßen: Verschiedene Threads werden gleichzeitig ausgeführt, bis sie auf die Ordered Region stoßen, die dann in der gleichen Reihenfolge ausgeführt wird, wie sie in einer seriellen Schleife ausgeführt werden würde. Dies ermöglicht immer noch ein gewisses Maß an Concurrency, insbesondere wenn der Codeabschnitt außerhalb des geordneten Bereichs eine beträchtliche Laufzeit hat. Hierdurch werden Race Conditions vermieden Somit ist diese Klausel sinnvoll, wenn eine geordnete Reihenfolge für nicht rechenintensive Aufgaben benötigt wird.

3.2 What is the collapse clause in OpenMP good for?

Parallelisierung von miteinander verknüpften for-Schleifen. Ohne diese Klausel würde dann nur die erste for-Schleife parallelisiert werden. Collapse(2) besagt beispielsweise, dass die zwei darauffolgenden for-Schleifen zu einer for-Schleife kollabiert werden.

3.3 Explain how reductions work internally in OpenMP.

Implementierung: reduction(op:list)
op: Entspricht auszuführende Operation
list: Enthält die Variablen, welche per Komma getrennt werden

Von jeder Listenvariablen wird eine lokale Kopie erstellt und abhängig von der durchzuführenden Operation initialisiert. Aktualisierungen erfolgen dann auf der lokalen Kopie. Die lokalen Kopien werden dann auf einen einzigen Wert reduziert und mit dem ursprünglichen globalen Wert kombiniert.

3.4 What is the purpose of a barrier in parallel computing?

Jeder Thread muss diese Barriere erreichen, bis das Programm weiter ausgeführt werden kann. Dies kann sinnvoll für Synchronisationszwecke sein.

3.5 Explain the differences between the library routines.

omp_get_num_threads()
Anzahl der innerhalb einer parallelen Region aktiven Threads. Außerhalb einer parallelen Region wird man eins zurückbekommen.

omp_get_num_procs()
Liefert die Anzahl logischer Kerne zurück. (Nicht unbedingt physische Kerne – 4 logische Kerne bei zwei CPU's denkbar aufgrund von Hyperthreading). Sinnvoll um die Anzahl von Threads festzulegen.

omp_get_max_threads()
Gibt an, wie viele Threads innerhalb einer parallelen Region unterstützt werden. Sinnvoll um herauszufinden, ob man sich in einem parallelen Bereich befindet.

3.6 Clarify how the storage attributes private and firstprivate differ from each other.

Öffnet man eine parallele Region, so kann man festlegen, wie die Variablen innerhalb einer parallelen Region behandelt werden sollen.
shared (per default):
Die Variable ist geteilt für alle Threads und Race-Conditions sind hierbei denkbar.
private:
Hier hat man eine nicht initialisierte Kopie der Variable auf jedem Thread. Vorherig abgespeicherter Wert wird hierbei nicht für die Variable übernommen.
firstprivate:
Jeder Thread bekommt eine Kopie der Shared Variable und der Wert wird hierbei übernommen). Diese Variable ist für jeden Thread lokal. Am globalen Variablenwert wird hierdurch nichts verändert.

4 Tasks and Merge Sort

4.1 Explain how divide and conquer algorithms can be parallelized with tasks in OpenMP.

Die Hauptidee ist hierbei, dass bei jedem Divide Schritt zwei neue Tasks erstellt werden. Dies wird dann rekursiv durchgeführt, sodass man zahlreiche voneinander unabhängige Arbeitspäckchen erhält. Diese können dann parallel bearbeitet werden. Es ist sinnvoll Taskwait zu verwenden, sodass sichergestellt wird, dass die Aufteilung der Tasks durchgeführt wurde und es somit zu keinen Problemen mit der darauffolgenden Bearbeitungsphase kommt. Zudem sollten die Arbeitspäckchen nicht zu klein werden, da deren Erstellung auch gewisse Ressourcenïn Anspruch nimmt.

4.2 Describe some ways to speed up merge sort.

1) Heap-Stack beachten
Es sollte nicht für jedes Element Speicher auf dem Heap angelegt werden. Unterschreitet der benötigte Speicher eine gewisse Größe, so könnte man per if-Abfrage sicherstellen, dass der Speicher auf dem Stack genutzt wird, welcher günstiger ist.

2) Insertion Sort
Diese Art der Sortierung ist für kleinere Arrays effizienter als Merge Sort.

3) Parallelisierung
Durch das Erstellen von unabhängigen Arbeitspäckchen (Tasks), können mehrere Prozessoren gleichzeitig an dem Problem arbeiten.

4) Multithreaded Merging
In der Kombinierungsphase verringert sich die Anzahl der verwendeten Threads bei steigender Arraygröße, was einer Performanceeinbuße entspricht. Dies kann mit Multithreaded Merging optimiert werden.

5) Speicher
Durch das Allokieren eines Speichers der Größe n (n=Elemente in Array) können die Merge Operationen stets auf einem Speicherbereich angewandt werden.

4.3 What is the idea behind multithreaded merging?

In der Kombinierungsphase werden bei steigender Arraygröße weniger Threads eingesetzt. Dem soll mit Multithreaded Merging entgegengewirkt werden.
Situation: Man hat zwei sortierte Arrays vorliegen und man möchte die beiden Arrays miteinander mergen.
Vorgehensweise: Im größeren Array betrachtet man den Median (Mitte des Arrays). Mit binärer Suche findet man die Elemente welche kleiner/größer als der Median sind. Diese Subarrays könnten dann gemerged werden.Dies ist separat in verschiedenen Threads möglich.

Hierbei hat man dann nicht die „teure“ Kombinierungsphase, welche in diesem Fall nur von einem Thread ausgeführt werden würde. Rekursives Vorgehen ist hierbei dann möglich. Zusammengefasst ergibt sich eine schnelle Divide Phase und keine Combine Phase. Das beschriebene Vorgehen wird in Abb. 5 dargestellt.

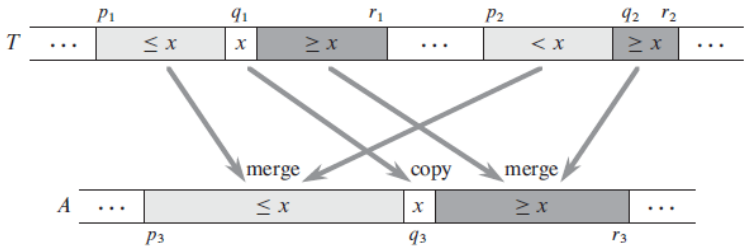


Abbildung 5: Multithreading Merging [3]

4.4 Discussion: What every systems programmer should know about concurrency.

Pipelines: Hierbei wird eine komplexe Operation in mehrere Stufen aufgeteilt. Diese Stages können dann unabhängig voneinander rechnen. Diese Form der Parallelität liegt auch in tiefen Computerarchitekturen vor (Instruction Level Parallelism). Es ist hierbei sinnvoll, dass die Pipeline ständig "gefüttert"wird, damit ein hoher Durchsatz vorliegt. In Abb. 6 wird eine Pipeline für den Instruction Level Parallelism dargestellt.

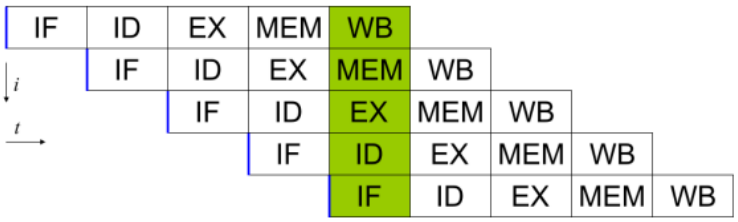


Abbildung 6: Pipeline

Atomare Operationen: Hierbei handelt es sich um ununterbrechbare Operationen. Im Allgemeinen kann ein Thread unterbrochen werden, da das Programm meist in kleinere Aufgaben zerlegt wird. Wird z.B. eine Variable geupdated (s=s+1), so besteht dies aus mehreren Operationen. Lade s, addiere 1 und speichere s. Diese aufeinanderfolgenden Befehle könnten unterbrochen werden. Atomare Operationen ermöglichen somit einen Hardwaremechanismus, sodass die gewünsche Operation am Stück ausgeführt wird. Dies sollte natürlich nicht für zu große Aufgaben verwendet werden, da es sonst zu Performanceeinbußen kommt. Es kann z.B. beim Laden/Schreiben sehr lange dauern, wenn der Speicher weit weg liegt. In OpenMP ist dies vergleichbar mit kritischen Regionen.

5 CMake and Optimization Process

5.1 What is CMake?

CMake ist eine Kommandosprache. Mittels CMake können Build Files erstellt werden, welche auf verschiedenen Betriebssystemem ausgeführt werden können. Compiler nutzt dann diese Build Files, um die dort genannten „Schritte“ auszuführen. Beispielsweise wird hier die Source Datei eines Projektes und die hierbei eingebundenen Bibliotheken angegeben. Die Erstellung von Build Files ist für verschiedene IDEs denkbar.

5.2 What role do targets play in CMake?

Ein CMake-basiertes Buildsystem ist als eine Reihe von logischen Targets auf hoher Ebene organisiert. Jedes Target entspricht einer ausführbaren Datei oder Bibliothek oder ist ein benutzerdefiniertes Ziel, das eigene Befehle enthält. Die Abhängigkeiten zwischen den Zielen werden im Buildsystem ausgedrückt, um die Build-Reihenfolge und die Regeln für die Wiederherstellung bei Änderungen festzulegen.

5.3 How would you proceed to optimize code?

Zunächst sei gesagt, dass die Korrektheit eines Programms stehts an erster Stelle steht und somit trotz Optimierungen die Ergebnisse korrekt bleiben müssen. Eine gute Vorgehensweise zur Optimierung ist es, das Problem zunächst ohne Optimierungne zu lösen. Hierbei wird dann sichergestellt, dass das Programm korrekt läuft. Ist die Performance gut genug, so ist man fertig. Ansonsten startet man mit der Optimierung der Implementierung, indem man die Bottlenecks herausfindet und hierfür dann Lösungen findet. Diese Optimierung könnte dann durch eine Verbesserung des Algorithmuses erreicht werden, indem beispielsweise Parallelisierung und Vektorisierung im Vordergrund steht. Zum Finden von Bottlenecks können Profiler verwendet werden. Nach den Optimierungen sollte wieder sichergestellt werden, dass das Programm korrekt läuft.

6 Auto Vectorization

6.1 Name some characteristics of the instructions sets: SSE, AVX(2) and AVX-512.

Je nach verwendeter CPU hat man verschiedene Instruktionssätze vorliegen. Hierbei haben die verwendeten Vektoren, je nach Instruktionssatz, eine unterschiedliche Länge. Die genauen Zahlen können hierzu in Abb. 7 entnommen werden. Es sei gesagt, dass AVX2 spezifischer Code auch mit AVX 512 ausgeführt werden könnte, wobei man hierbei nicht von der größeren Vektorlänge profitieren würde.

	Vector Length	Years of Launch	Registers
SSE – SSE4.2	128-bit	1999 – 2009	xmm0 – xmm15
AVX, **AVX2**	256-bit	2011, 2013	ymm0 – ymm15
AVX-512	512-bit	2017	zmm0 – zmm31

Abbildung 7: Vector Instruction Sets [4]

6.2 How can memory aliasing affect performance?

Werden an eine Funktion mehrere Pointer übergeben, so kann es für den Compiler unklar sein, ob diese Pointer auf den selben Speicherbereich zeigen, bzw. eine Überlappung der Speicherbereiche vorliegt. Ist man sich darüber bewusst, dass diese Speicherbereiche nicht überlappen, so kann der Compiler davon profitieren, wenn man ihm das vorab mitteilt (restricted pointer). Hierdurch wird der ausgeführte Code performanter und ist einfacher vektorisierbar für den Compiler.

6.3 What are the advantages of unit stride (stride-1) memory access compared to accessing memory with larger strides (for example, stride-8)?

stride-1: Dies bedeutet, dass man auf jedes aufeinanderfolgende Element eines Arrays zugreift.
stride-8: Zugriff auf jedes 8.te Element innerhalb eines Arrays.
Diese Stride Arten werden in Abb. 8 dargestellt.

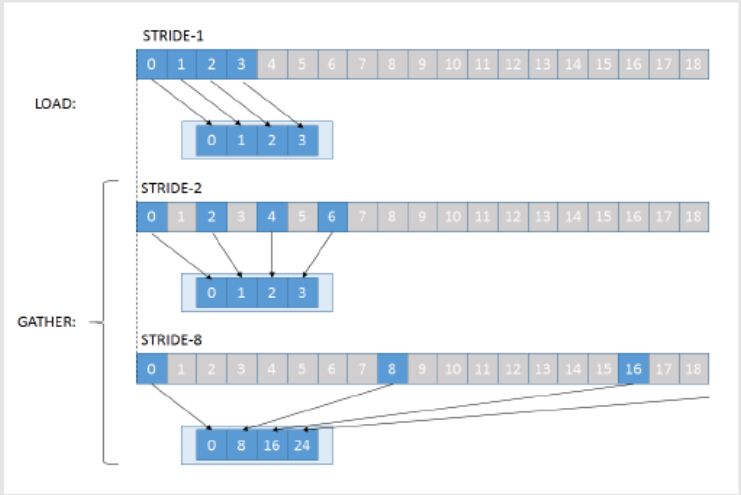


Abbildung 8: Strided Access [4]

Im Allgemeinen werden Daten aus dem Arbeitsspeicher mit Cachelines geladen (64 Byte groß). Eine CPU kann mit einer bestimmten Bandbreite Daten aus dem Arbeitsspeicher

laden und somit würde bei einer Optimierung hierbei das Zeil sein, die Bandbreite zu maximieren. Diese Optimierung liegt dann vor, wenn man unit stride (stride-1) verwendet, da hierbei die geladenen Daten komplett in der CPU verarbeitet werden. Zudem ist es für den Compiler deutlich einfacher vektorisiert zu arbeiten, wenn die zu bearbeitenden Elemente sequenziell im Speicher liegen

6.4 When would you prefer arranging records in memory as a Structure of Arrays?

Bei SoA (Structure of Arrays) sind die Objekte anhand ihrer Koordinaten angeordnet (Siehe Abb. 9. Diese Struktur ist dann hilfreich, wenn man Operationen für eine bestimmte Koordinate der Objekte durchführt. Ein mögliches Beispiel wäre hierbei die Bestimmung des kleinen x-Wertes, da diese Aufgabe aufgrund des gegebenen Formates gut vektorisierbar wäre. Ein Nachteil entsteht hierbei, wenn man bei dieser Struktur auf die Attribute (x,y,z - Koordinate) eines Objektes zugreifen möchte, da diese Informationen verstreut im Speicher liegen. Somit ist die Lokalität von einzelnen Objekten bei diesem Format nicht gegeben.

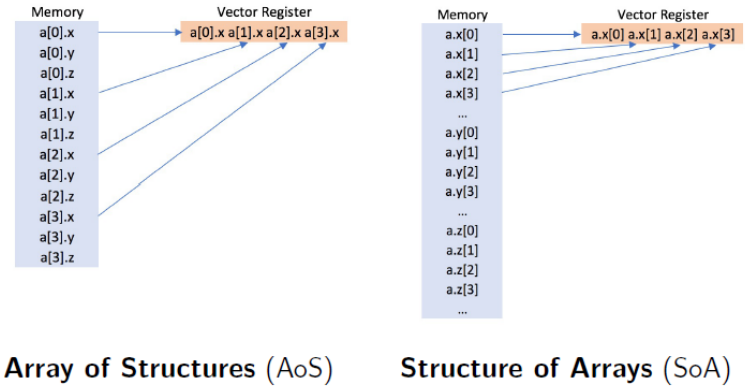


Abbildung 9: AoS and SoA [4]

7 Guided Vectorization and DT

7.1 Explain three vectorization clauses of your choice that can be used with pragma omp simd.

simdlen: Hierdurch wird die Breite der verwendeten Vektoren vorgeschlagen (Präferenz). Dies solle man aber dem Compiler überlassen, da inkorrekte Angeaben zu einem deutlichen Performance Verlust führen können. Die meisten Compiler ignorieren diese Klausel, wobei der Intel Compiler diese berücksichtigen würde.
safelen: Hierdurch wird vorgegeben, dass eine gewissen Vektorengröße nicht überschritten werden darf. Dies wäre in einer Situation denkbar, in der die Vektorgröße vorher begrent wurde und für die Korrektheit des Codes dies beibehalten werden muss.
aligned: Hiermit wird dem Compiler mitgeteilt, dass die angegebenen Vektoren innerhalb

der for-Schleife aligned sind zu einer bestimmten Byte Boundary. Diese Byte Boundary entspricht meist 64 Byte, da dies die Größe einer typischen Cache Line ist.

7.2 Give reasons that speak for and against vectorization with intrinsics compared to guided vectorization with OpenMP.

Man hat keine Garantie, dass der Compiler den Code in Vektorinstruktionen übersetzt, da es compilerabhängig ist, wie gut er mögliche Vektorisierungen erkennt. Es wird somit die vorgeschlagene Vektorisierung mit OpenMP nicht immer umgesetzt. Anders hierbei ist die Vektorisierung mit Intrinsics, bei welcher man sicherstellen kann, dass diese Vektorisierung stattfindet, da diese explizit im Code erstellt wird. Dies bedeutet natürlich einen gewissen Mehraufwand, welcher mit eingeplant werden sollte.

7.3 What are the advantages of vector intrinsics over assembly code?

Intrinsics Funktionen bieten den Vorteil, dass sie portabler zwischen verschiedenen Compilern sind, da es sich um validen C++ Code handelt. Zudem ist Vector Intrinsics deutlich einfacher zu Lernen, als Assembly Code und auch für Außenstehende deutlich einfacher zu verstehen. Vector Instrinsics führt im Endeffekt assembly Instruktionen aus. Mit den Optimierungsflags generiert der Compiler normalerweise ein one-to-one Mapping zwischen intrinsics Funktionen und den dazugehörigen assembly Instruktionen.

7.4 What are the corresponding vectors of the three intrinsic data types: __m256, __m256d and __m256i.

Es handelt sich hierbei um die Datentypen Float, Double und Integer. Es sei gesagt, dass die Integer Vektoren unterschiedlich interpretiert werden können (B, short, int, long long und doublequadword). Die folgenden Vektoren in Abb. 10 beziehen sich auf den Instruktionssatz AVX2.
__m256: Vektor mit 8 Floats (32 bit pro Float)
__m256d: Vektor mit 4 Doubles (64 bit pro Double)
__m256i: Vektor mit 256 Bit und i steht für Integer Vektor (kann signed oder unsigned sein)

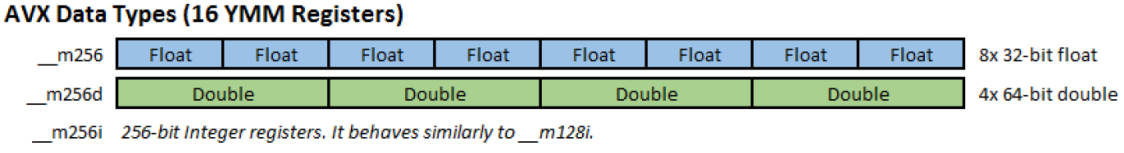


Abbildung 10: AVX Data Types [5]

8 Vector Intrinsics and ILP

8.1 Explain the naming conventions for intrinsic functions.

vector size: Sagt aus, welcher Vektor hierbei zurückkommt.
SSE: mm für 128 Bit Vektor
AVX2: mm256 für 256 Bit Vektor
Es sind aber auch 128 Bit Vektoren bei „neuerem“ Instruktionssatz denkbar.

operation: Hier wird die Art der Berechnung vorgegeben, welche die intrinsische Funktion ausführen soll.

suffix: Hierbei wird der Datentyp des eingegeben Vektors vorgegeben.
pd steht für Double Vektoren.
ps steht für Float Vektoren.
ep steht für Integer Vektoren. (Verschiedene Größen möglich)

8.2 What do the metrics latency and throughput tell you about the performance of an intrinsic function?

Die Latenz sagt aus, wie viele Taktzyklen es dauert, bis das Ergebnis der Intrinics Funktion vorliegt.
Der Durchsatz gibt an, wie viele Taktzyklen man warten muss, bis man eine neue Intrinsics aufrufen kann.
Zur Verbesserung der Performance wäre es somit optimal die Latenz und den Durchsatz zu minimieren. Für die Minimierung des Durchsatzes gibt es die Möglichkeit des Softwarepipelinings. Hierbei würde man im optimalen Fall x Taktzyklen warten müssen, bis das erste Ergebnis herauskommt (x entspricht Latenz) und es würde dann in jedem weiteren Taktzyklus ein neues Ergebnis geliefert werden. Dies wäre dann möglich, wenn der Durchsatz 1 entspricht.

8.3 How do modern processors realize instruction-level parallelism?

Instruction Level Parallelism bedeutet, dass mehrere Instruktionen simultan auf einem CPU Kern pro Taktzyklus ausgeführt werden können. In Abb. 11 werden die verschiedenen Funktionseinheiten einer CPU dargestellt. Hierbei gibt es dann verschiedene Eigenschaften, welche den Instruction-Level Parallelism ermöglichen.
Superscalar: Mehrere Instruktionen können gleichzeitig bei jedem Taktzyklus gestartet werden.
Out of Order Execution: Programm wird in Maschinencode übersetzt, aber die exakte Ausführung auf der CPU (z.B. Vertauschen von Instruktionen) wird intern vorgenommen.
Branch Prediction: Hierbei wird erraten, welcher Branch in der Zukunft verwendet wird.
Speculative Execution: Ausführen der Operationen, obwohl noch nicht klar ist, ob die Branch Prediction korrekt war. Hierdurch wird der Instruction-Level Parallelism stark ausgenutzt,

da die CPU maximal ausgenutzt wird und im Worst Case müssten die Werte für den korrekten Branch neu berechnet werden.

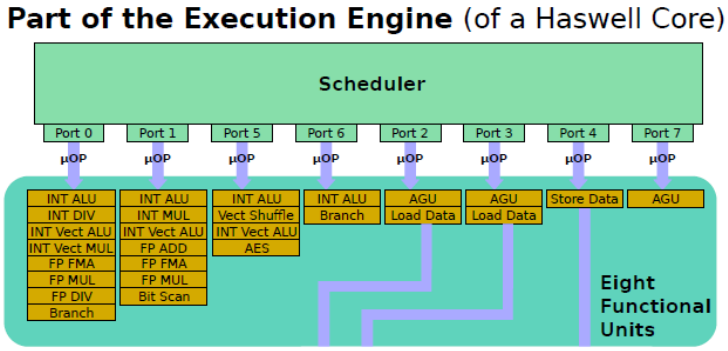


Abbildung 11: Execution Engine [6]

8.4 How may loop unrolling affect the execution time of compiled code?

Loop Unrolling bewirkt, dass innerhalb einer for-Schleife mehrere Instruktionen ausgeführt werden. Man erhofft sich hierdurch, dass der Instruction Level Parallelism erleichtert wird. Man muss hierbei aber beachten, dass hierbei die Laufzeit verlängert werden könnte, da z.B. zuviel zusätzlicher Speicher im Instruktionscache benötigt wird und eine zu große Nachfrage an Registern in der CPU entstehen könnte. Diese Abwägungen müssen dann je nach CPU Architektur getroffen werden.

8.5 What does a high IPC value (instructions per cycle) mean in terms of the performance of an algorithm?

IPC entspricht dem Verhältnis von Instruktionen zu Zyklen bei Betrachtung der Arbeitsweise eines Algorithmus. Ein hoher IPC Wert deutet auf eine effiziente CPU Verwendung hin, da hier scheinbar der Instruction Level Parallelism gut genutzt wird. Ein hoher IPC Wert heißt aber nicht dass der Algorithmus gut ist, sondern, dass die Implementierung effizient ist.

9 Cache and Main Memory

9.1 How do bandwidth-bound computations differ from compute-bound computations?

Compute-bound: Es wird hierbei kaum auf Arbeitsspeicher zugegriffen und CPU ist bei den Berechnungen "überlastet". In diesem Fall ist die CPU der Bottleneck

Bandwitdh-bound: Hierbei wird die CPU kaum ausgelastet, da der Speicherzugriff hierbei der Bottleneck ist. Dies wäre beispielsweise denkbar, wenn die Elemente einer sehr großen Matrix aufsummiert werden sollen. Für Programme, welche den Cache optimal nutzen, ist die bandwidth-bound von besonderem Interesse und das Ziel solle es dann sein die Daten möglichst schnell zur CPU zu bekommen.

9.2 Explain why temporal locality and spatial locality can improve program performance.

Moderne Computer verwenden SRAM basierte Caches um die Processor-Memory Gap (Siehe Abb. 12) zu überbrücken. Dies läuft besonders gut, wenn man die zeitliche und örtliche Nähe der Daten beachtet.

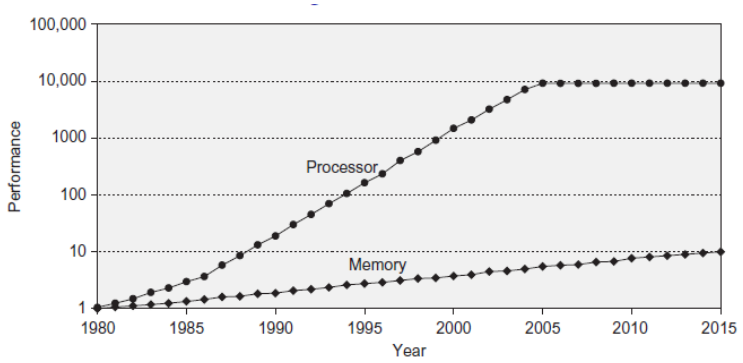


Abbildung 12: Processor Memory Gap

Im Allgemeinen versucht der Cache die Daten zu halten, welche häufig von der CPU verwendet werden. Für dieses Grundprinzipt kann die örtliche und zeitliche Nähe von Daten genutzt werden.
Örtliche Nähe: Greift man auf ein Element in einer Cacheline zu, so ist es sinnvoll auch auf die anderen Elemente in der Cacheline zuzugreifen, da hierdurch die Bandbreite optimal genutzt werden kann. Dies bedeutet, dass in einem gut geschriebenem Programm bei einer referenzierten Speicherstelle auch auf nahegelegene Elemente in naher Zukunft zugegriffen wird. Beispiel: Es ist bekannt, dass eine Matrix Zeilenorientiert gespeichert wurde und die Cachelines dementsprechend geladen werden.
Zeitliche Lokalität: Daten, welche im Cache liegen sollen auch vom Algorithmus so genutzt werden, dass es zu einem großen „Reuse“ dieser Cachelines kommt. Beispiel: Aufteilung eines Matrix Vektor Produktes in Blöpche, sodass einer großer Reuse von Cachelines ermöglicht wird.

9.3 What are the differences between data-oriented design and object-oriented design?

Cache Effiziente Programme sollen hierdurch erstellt werden Hauptunterschied: Fokus auf Performance bei Data-Oriented Design

Data Oriented Design:
Hierbei liegt die Performance stark im Fokus. Man schaut sich an, was herauskommen soll und startet mit dem Design Prozess. Der Rechenprozess wird überlegt, welcher aus den Input Bytes die gewünschten Outputs Byte erstellt. Dies sollte so schnell und effizient wie möglich erfolgen. Funktionen hierbei haben generischen Nutzen. Es handelt sich dann um einfachere Funktionen, welche auf großen Datenmengen angewandt werden können. Vektorinstruktionen sollen dadurch dann gut nutzbar sein und die Parallelisierung sollte dadurch dann vereinfacht werden.

Object Oriented Design:
Man hat z.B. viele Objekte im Speicher, welche gemerged werden müssten für den gewünschten Output. Insgesamt dient dieses Design einer nutzerfreundlichen Bedienung.

9.4 What are streaming stores?

Zunächst muss man sich hierfür mit der allgemeinen Frage beschäftigen, wie man Elemente von der CPU in den RAM zurückspeicher kann.
Normal Stores: Man geht durch die komplette Cache Hierarchie bis in den RAM hinein.
Streaming Stores: Man speichert die Elemente direkt in den RAM hinein (Siehe Abb. 13) .
Streaming Stores können hilfreich sein, wenn man weiß, dass die gewünschten Daten nicht mehr gebraucht werden und somit direkt in den RAM zurückgespeichert werden können. Hierdurch spart man Cache für andere Daten und man könnte auch eine bessere Performance erhalten.

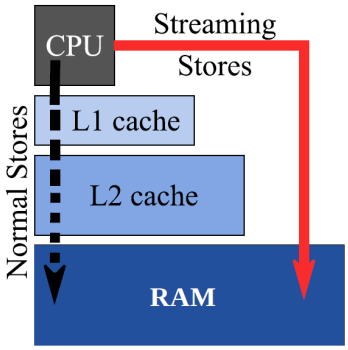
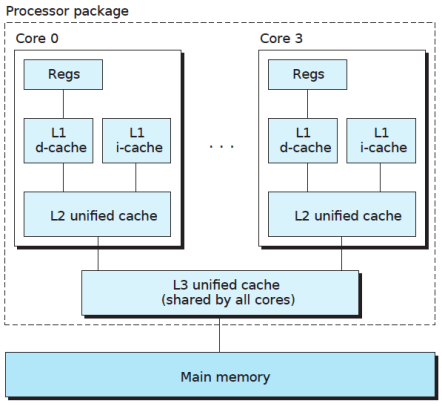


Abbildung 13: Streaming Stores [7]

9.5 Describe a typical cache hierarchy used in Intel CPUs.

Die typische Cache Hierarchie wird anhand eines Intel Core i7 beschrieben (Siehe Abb. 14).
Speicherhierarchie von Caches:
L1 Cache (Data Cache und Instruktions Cache wird hier unterschieden)
L2 Unified Cache (Für Instruktionen und für Daten)
L3 Cache unified Cache (Geteilt zwischen allen Kernen)

Eigenschaften der verschiedenen Caches:
Latenz: L1 Cache (4 Taktzyklen) / L2 Cache (10 Taktzyklen) / 40 – 75 Taktzyklen
Speicher: L1 Cache: 32 KB / L2 Cache: 256 KB / L3 Cache: 8 MB Cache Line: 64 Byte groß (Unterschiedliche Anzahl hiervon, dann je nach Cache Art)
In einem Bucket können dann bestimmte Blockanzahlen behalten werden. (Je nach Cache Art)



Cache type	Access time (cycles)	Cache size	Associativity	Block size	Sets
L1 i-cache	4	32 KB	8	64 B	64
L1 d-cache	4	32 KB	8	64 B	64
L2 unified cache	10	256 KB	8	64 B	512
L3 unified cache	40–75	8 MB	16	64 B	8192

Abbildung 14: Cache Hierarchy [7]

9.6 What are cache conflicts?

Ein Bucket kann nur eine bestimmte Zahl an Cachelines halten. N-way associative bedeutet, dass hier N Cache Lines in ein Bucket hineinpassen. Wird trotzdem eine Cacheline in ein volles Bucket geladen, so muss ein belegter Platz frei gemacht werden, was einem Cache Conflict entspricht. Ein ungünstiger Speicherzugriff könnte bewirken, dass auf einen vollen Bucket gespeichert werden soll, obwohl die anderen Buckets noch Platz frei haben. Dies kann sich dann negativ stark auf die Performance auswirken.

10 Debugging and Profiling

10.1 Name and explain some useful compiler flags during development.

-Wall: Hierdurch sind alle Warnings Nachrichten des Compilers angeschaltet

-g: Debugger Informationen werden ins Programm geschrieben.

-fsanitize=address: Hierdurch kann man Out-of-Bounds Access (z.B. außerhalb von Array), use-after-free und memory leaks Fehler erkennen.

-fsanitize=undefined: Zur Laufzeit kann festgestellt werden, wenn das Programm ein undefiniertes Verhalten hat.

10.2 How could Intel oneAPI help you write better programs?

Man ist daran gewöhnt mit den verschiedensten Geräten gleichzeitig zu arbeiten. Die Wiederverwendung von Code ist schwierig, da jeder dieser verwendeten Prozessoren seine eigenen Bibliotheken, Tools oder ein eigenes Programmiermodell hat. Die Vision von oneAPI ist es, eine architektur- und herstellerübergreifende Portabilität zu erreichen und gleichzeitig die erforderliche Leistung zu bieten.oneAPI überbrückt und abstrahiert all diese verschiedenen Geräte und bringt sie auf eine gemeinsame Basis.

10.3 What can we learn from the following quote? Premature optimization is the root of all evil (Donald Knuth).

Quote: „Premature optimization is the root of all evil“ Man sollte nicht blind einfach irgendwo im Code direkt mit Optimierungen starten. Man sollte so wenig wie möglich „invasiv“ in einem fremden Programm optimieren. Man kennt hier die ganzen Abhängigkeiten in einem Programm nicht auf Anhieb.
Keep it Short and Simple als sinnvolle Leitlinie. Es macht Sinn zunächst zu profilen und Bottlenecks zu finden, welche dann mittels Optimierungen gelöst werden können.

11 Cython Introduction

11.1 What is Cython?

Python hat das Problem recht langsam zu sein Cython soll bei Performance kritischen Berechnungen helfen, eine ähnlichen Speed wie bei C/C++ erreichen. Idee: Profiler verwenden und den Bottleneck mittels Cython bearbeiten. Cython ist eine kompilierte Sprache und Wird zur Beschleunigung von Python Code genutzt. Mit Hilfe von Cython können auch C / C++ Libraries gewrapped werden.

11.2 Describe an approach how Python programs can be accelerated with the help of Cython.

Beschleunigungsmöglichkeit 1:
Überhaupt erstmal die Grundfunktion über Cython laufen lassen. Hierbei ist auch schon Speed Up möglich.

Beschleunigungsmöglichkeit 2:
Verwendung von Cython Compiler Direktiven

Beschleunigungsmöglichkeit 3:
Variablentypen im Cython Code hinzufügen (cdef hierbei verwenden). Variablen müssen dann vor der Verwendung definiert werden

11.3 Describe two ways for compiling a .pyx Cython module.

Möglichkeit 1:
Funktion herausschreiben und kompilieren mit cythonize (z.B.: cythonize -i -a compute pi.pyx). Dies kann dann mittels einer Importfunktion verwendet werden und .py/.pyx wird dabei in eine C/C++ Datei umgewandelt. Dieses Extension Modul kann dann in Python importiert werden.

Möglichkeit 2:
Mittels pyximport wird das Programm automatisch rekompiliert, wenn man ein .pyx module verwendet. Somit muss man folgende Code Zeilen in seinen Python Code einbauen:
import pyximport
pyximport.install()

Möglichkeit 3:
Verwendung von Jupyter Notebooks, da diese Cython sehr gut unterstützen. Hierbei verwendet man %load_ext cython, um Cython nutzen zu können. Hat man eine Zelle, welche Cython Code verwenden soll, so schreibt man zu Beginn der neuen Zelle cython.

11.4 Name and describe two compiler directives in Cython.

boundscheck (True / False)
Wird dieser Wert auf False gesetzt, kann Cython davon ausgehen, dass Indexierungsoperationen im Code keine Index Errors auslösen werden.

wraparound (True / False)
In Python können Arrays und Sequenzen relativ zum Ende indiziert werden. Zum Beispiel indiziert A[-1] den letzten Wert einer Liste. In C wird die negative Indizierung nicht unterstützt. Wenn der Wert auf False gesetzt ist, kann Cython weder auf negative Indizes prüfen noch diese korrekt behandeln, was zu Seg-Fehlern oder Datenbeschädigungen führen kann.

11.5 What is the difference between def, cdef and cpdef when declaring a Cython function?

def,cdef und cpdef unterscheiden sich darin, für welche Programmiersprache die jeweilige Funktion ausgelegt ist. Zudem kann bei cdef und cpdef der Rückgabetyp angegeben werden.
def: Ganz normale Pythonfunktion

cdef: Funktion kann nur in Cython verwendet werden.
cpdef: Diese Funktion kann in Python und in Cython aufgerufen werden.

11.6 What are typed memoryviews especially useful for in Cython?

Hierdurch kann die Übergabe von Python Arrays und Numpy Arrays an Cython deutlich vereinfacht werden. Es wird hierbei dann der Datentyp mit der zugehörigen Dimension angegeben, was einen großen Speed Up ermöglicht.

12 Extension Types and Interfacing

12.1 What are extension types in the context of Python?

Diese Extension Types werden wie normale Klassen in Python genutzt, aber es handelt sich hierbei eigentlich um kompilierten Code. Sie sind sehr hilfreich um C/C++ Code zu wrappen und ein Python ähnliches Interface darzustellen. Hierdurch kann dann ein großer Speed Up erreicht werden.

12.2 How do extension types data fields in Cython differ from data fields in Python classes?

Bei Cython Klassen müssen die Attribute angegeben, welche intern in der Klasse verwendet werden sollen. Diese sind innerhalb von Cython auch ohne Readonly / Public zugreifbar. In Python wäre dies aber notwendig, um die Attribute nutzen zu können. In einer Cythonklasse müssen dann alle Attribute angegeben werden. Die Objekte sind hierbei direkt im Speicher gesetzt und es wird, im Gegensatz zu Python, keine Dictionary Suche nach dem Attribut benötigt. Als Attribute können auch andere Cython Klassen verwendet werden. Hierdurch können verschiedenste Datenstrukturen definiert werden.

12.3 Give a simple description of how to wrap C / C++ code in Cython.

Zunächst einmal benötigt man eine Header Datei zum deklarieren der Funktion, welche in einer cpp Datei definiert wurde. Diese soll nun in Python genutzt werden, indem man eine Variable übergibt. Folgende Schritte sind hierfür notwendig:
pyx Datei hierfür erstellen (Datentyp importieren)
Compilerdirektiven zu Beginn niederschreiben
language = c++ (Hinweis auf C++ Code geben)
sources = pi.cpp (Source Datei mit angeben)
extra_compile_args = -fopenmp –fast-math
extra_links_args = -fopenmp
Man muss Cython erläutern, welche Funktionalität man auf dem pi Header verwenden möchte.

cdef extern from pi.h (Funktionsdeklaration angeben)
Schnittstelle wird hierdurch angegeben.
def Funktion dient für den Aufruf in Python. Hierbei wird dann auf die Funktion in C++ zugegriffen. Kompilieren von Cython Datei. Dann kann die Funktion (aus C++) innerhalb des Python Codes genutzt werden.

13 Designing SSD-Friendly Applications

13.1 Delimit from each other the following SSD parts: Cells, Pages and Blocks.

Cells: Hier werden einzelne Bits auf SSDs gespeichert. In den Zellen befinden sich eine gewisse Menge an Elektronen. Je nach Menge der Elektronen wird hierbei dann ein Wert von 1 oder 0 angenommen. Dafür gibt es dann einen sogenannen Schwellwert. Es gibt verschiedene Zellentypen (Siehe Abb. 15):
Single Level (SLC), Multiple Level (Pro Zelle werden zwei Bit gespeichert), Triple Level und Quad Level. Hierbei liegt dann eine feinere Unterteilung je nach Schwellenwert vor und es kann mehr Information abgespeichert werden. Billigste SSDs haben Quad Level. Hierbei hat man eine hohe Dichte, aber eine vergleichsweise geringe Performance. Je weniger „Schwellenwerte“ vorliegen, desto größer ist auch die Lebenszeit.SLCs können extremen Temperaturen standhalten (-40 bis +85 Grad Celsius).

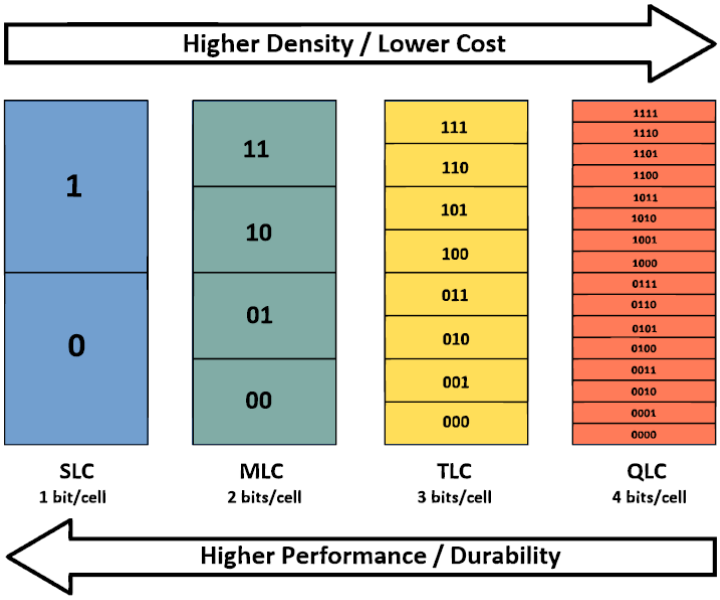


Abbildung 15: Cell Types [8]

Pages: Eine Gruppe von Zellen bildet eine Page, was der kleinsten Speichereinheit zum Lesen und Schreiben entspricht. Die typische Größe hierfür beträgt 4 KB. Eine Page kann nur einmal beschrieben werden, weshalb ein Überschreiben nicht möglich ist. Für ein Update

wird der Inhalt von der SSD kopiert, modifiziert und in eine neue Page geschrieben. Beim Kopieren wird die Page als abgestanden (invalid) markiert und verbleibt solange in diesem Zustand, bis sie gelöscht wird.
Blocks: Entspricht der kleinsten Einheit, welche gelöscht werden kann. Die typische Größe entspricht hierbei 512 KB oder 1 MB (128 oder 256 Pages). Eine wichtige Anmerkung hierbei ist, dass die Löschoperation eine deutlich größere Latenz, als die Lese/Schreibe Operationen besitzt. Die gültigen Pages in einem zu löschenden Block werden in einen neuen Block abgespeichert und dann kann der Block mit den ungültigen Pages gelöscht werden, was einer Entladung der Elektronen entsprich (Siehe Abb. 16).

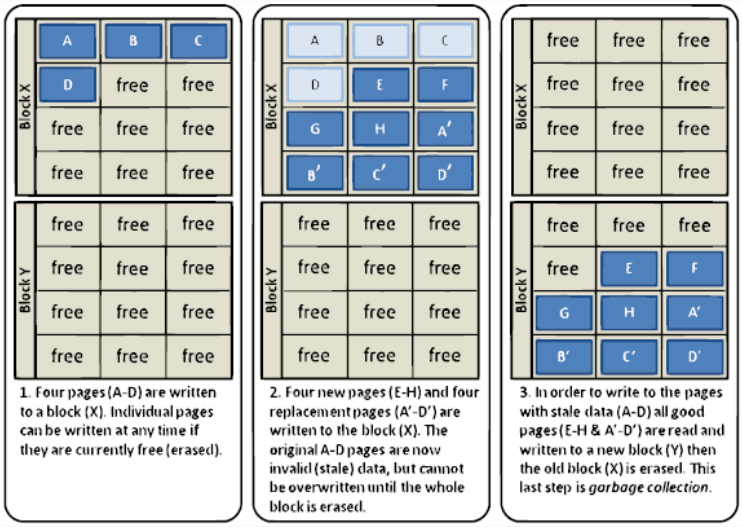


Abbildung 16: Blocks [8]

13.2 What is the purpose of garbage collection in SSDs?

Eine SSD hat einen Controller, welcher die Logik verwaltet. Dieser dient dazu, die Lebenszeit der SSD zu optimieren und performante Funktionen zu ermöglichen. Zudem ist dieser Controller für die Garbage Collection zuständig. Die Garbage Collection ist dafür da, Blöcke mit invaliden Pages zu bereinigen. Wird ein bestimmter Wert an invaliden Pages überschritten, so wird dem Controller signalisiert, dass dieser Blockfreigegeben werden soll. Die Write Amplification entspricht hierbei dem Verhältnis von wirklich geschriebenen Pages zu den tatsächlich geschriebenen Pages. Hierbei könnte es beispielsweise eine Situation geben, in der eine Page geschrieben werden soll, aber hierdurch zahlreiche Blöcke hin und hergeschoben werden müsste. Somit ist es das Ziel die Write Amplification zu reduzieren um eine höhere Performance zu ermöglichen und um die Lebenszeit der SSD zu schonen. Die Garbage Collection verschiebt zusammengehörige Datensegmente so, dass sie nebeneinander liegen. Um die Kosten für die Garbage Collection zu reduzieren, bieten einige SSDs eine Überbelegung an.

13.3 What is the purpose of wear leveling in SSDs?

Wear Leveling ist dazu da, die Lebenszeit einer SSD zu verlängern, indem es für eine ausgeglichene Abnutzung der Pages sorgt. Dieser Prozess kann aber die Performance einer SSD verschlechtern, da zusätzlicher IO entsteht.

13.4 Tell some interesting things about SSDs with an M.2 form factor.

M.2 ist eine Schnittstellenspezifikation, die mehrere Protokolle und Anwendungen wie PCI Express (PCIe) und SATA (Serial Advanced Technology Attachment) unterstützt. Hierdurch wird eine hohe Performance ermöglicht. Zudem sei gesagt, dass SSDs mit einem M.2 Formfaktor platzsparender sind und somit effizienter verbaut werden können. Diese Vorteile gehen natürlich einher mit höheren Kosten.

13.5 What influence do garbage collection and wear leveling have on write amplification of an SSD?

Die Garbage Collection hat einen tiefgreifenden Einfluss auf die Write Amplification in Flash-basierten SSDs, was wiederum deren Lebensdauer erheblich verkürzen kann. Die ungleiche Abnutzung von Datenblöcken trägt weiter zu dieser verkürzten Lebensdauer bei. Unoptimiert sorgt die Garbage Collection somit für eine Erhöhung der Write Amplification. Optimiert kann diese verringert werden, was eine bessere Performance und längere Lebenszeit der SSD zur Folge hat. Wear Leveling erhöht die Write Amplification. Hierdurch ensteht dann eine verringerte Performance, da zusätzlicher IO benötigt wird um Wear Leveling zu ermöglichen.

13.6 Discuss three different recommendations for writing code for SSDs.

SSD freundliche Datenstrukturen sollten möglichst kompakt sein. Dies gilt für das Lesen und für das Schreiben. Hierdurch erhält man dann eine bessere Performance und erhöht gleichzeitig die Lebensdauer der SSD.

Man sollte darauf achten, dass die SSD genügend Speicher frei hat, da es ansonsten zu extremen Problemen mit dem Write Amplification Faktor kommen kann und die Schreibe Performance hierdurch stark ins negative beeinflusst wird. Dies liegt daran, dass es bei Platzmangel zu vielen Blockbewegungen kommt und dabei viele Pages rumkopiert werden müssen.

Zur Ausnutzung des inneren Parallelismus der SSD ist es sinnvoll mit mehreren Threads bei kleinerem IO zu arbeiten. Hierbei wird dann das Overprovisioning ausgenutzt. Dies ergibt dann eine deutliche Performancesteigerung im Vergleich zur Implementierung mit nur einem Thread.

13.7 How could the CPU load for IO be reduced?

Folgende Möglichkeiten wären hierbei denkbar:
Asynchrone IO verwenden, sodass CPU bei Wartezeiten weiterarbeitet.
Streaming Stores verwenden.
OS Buffering deaktivieren. Hierbei cached das Betriebssystem nichts, sondern die Applikation selbst. Dieses Szenario ist aber für Datenbankmanagementsysteme gedacht.

13.8 How could you solve problems that do not fit in DRAM without major code adjustments?

Verwendung einer Linearen Algebra Bibliothek, welche optimiert mit Flash Speicher umgehen kann. Beispielsweise könnte hierfür BLAS-on-flash genutzt werden. SSD kann hierbei als RAM Ersatz genutzt werden und Pipelining wird hierbei verwendet. Dieses Pipelining bewirkt, dass Latenzen während der Berechnungen versteckt werden, indem weitere Ladeoperationen durchgeführt werden.

Literaturverzeichnis

[1] Mark Blacher Vorlesung Algorithm Engineering 01. https://drive.google.com/drive/folders/1WCROK88fgrnncZczSz7CdzU6yymyyqh1. [Online; Stand 22. Februar 2022].

[2] Mark Blacher Vorlesung Algorithm Engineering 02. https://drive.google.com/drive/folders/1WCROK88fgrnncZczSz7CdzU6yymyyqh1. [Online; Stand 22. Februar 2022].

[3] Mark Blacher Vorlesung Algorithm Engineering 04. https://drive.google.com/drive/folders/1WCROK88fgrnncZczSz7CdzU6yymyyqh1. [Online; Stand 22. Februar 2022].

[4] Mark Blacher Vorlesung Algorithm Engineering 06. https://drive.google.com/drive/folders/1WCROK88fgrnncZczSz7CdzU6yymyyqh1. [Online; Stand 22. Februar 2022].

[5] Mark Blacher Vorlesung Algorithm Engineering 07. https://drive.google.com/drive/folders/1WCROK88fgrnncZczSz7CdzU6yymyyqh1. [Online; Stand 22. Februar 2022].

[6] Mark Blacher Vorlesung Algorithm Engineering 08. https://drive.google.com/drive/folders/1WCROK88fgrnncZczSz7CdzU6yymyyqh1. [Online; Stand 22. Februar 2022].

[7] Mark Blacher Vorlesung Algorithm Engineering 09. https://drive.google.com/drive/folders/1WCROK88fgrnncZczSz7CdzU6yymyyqh1. [Online; Stand 22. Februar 2022].

[8] Mark Blacher Vorlesung Algorithm Engineering 13. https://drive.google.com/drive/folders/1WCROK88fgrnncZczSz7CdzU6yymyyqh1. [Online; Stand 22. Februar 2022].