Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Erweitern des Vergleichs verschiedener N-Körper-Algorithmen auf diversen Hardwareplattformen unter Verwendung von SYCL

Moritz Mahling

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dirk Pflüger
Betreuer/in:	Marcel Breyer, M.Sc.
Beginn am:	5. April 2023
Beendet am:	5. Oktober 2023

Kurzfassung

N-Körper Beschleunigungsberechnungen werden in diversen Bereichen der Wissenschaft durchgeführt, zum Beispiel bei der Simulation von Sonnensystemen in der Astronomie oder auch beim Simulieren von Teilchensystemen im Molekularbereich. Da diese Simulationen eine große Anzahl an Körpern enthalten können, gibt es bereits Algorithmen, die die notwendige Anzahl an Berechnungen während der Simulation mit möglichst geringer Auswirkung auf die Genauigkeit verringern. Ein Beispiel hierfür ist der baumbasierte Barnes-Hut Algorithmus. Im Vorfeld dieser Arbeit wurde bereits gezeigt, dass der Barnes-Hut Algorithmus auf GPUs mithilfe von SYCL erfolgreich implementiert werden kann. Da Barnes-Hut jedoch per Definition mit Bäumen arbeitet, ist er nicht optimal für die Ausführung auf GPUs geeignet. In dieser Arbeit soll der normale Barnes-Hut Algorithmus, bei dem der Baum im Top-Down Verfahren aufgebaut wird, so abgeändert werden, dass der Baum von den Blattknoten aufwärts erstellt wird, und dabei möglichst viele Operationen parallel ausgeführt werden können.

Die Erstellung der Baumstruktur kann durch die Verwendung des Bottom-Up Ansatzes beschleunigt werden. Außerdem können GPUs nun mehr Knoten des Baumes in der gleichen Zeit abarbeiten als beim Top-Down Verfahren. Durch die Umstruktierung des Baumes ist die Vereinfachung der Beschleunigungsberechnung mit dem Barnes-Hut Algorithmus jedoch nicht mehr so effektiv. Die Gesamtlaufzeit ist nun deutlich höher, da bei gleichem Theta mehr Körper in der Beschleunigungsberechnung einzeln bearbeitet werden. Die Laufzeit kann verbessert werden, indem für größere Datensätze die Anzahl an Kindknoten pro innerem Knoten im Baum erhöht wird, jedoch skaliert der Top-Down Barnes-Hut Algorithmus trotz dieser Optimierung besser. Der Bottom-Up Ansatz hat jedoch Potential für weitere Optimierungen, die ihn in Zukunft auf GPUs konkurrenzfähig zum Top-Down Verfahren machen könnten.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

N-Körper Simulationen werden in verschiedenen Bereichen der Wissenschaft durchgeführt. So beschäftigt man sich in der Astronomie beispielsweise mit der Simulation der Bewegungen von Himmelskörpern, und mit der Simulation von Wechselwirkungen zwischen Teilchen in der Molekularphysik. Ohne weitere Optimierungen beträgt die Komplexität dieser Simulationen $O(N^2)$. Grund dafür ist das zugrundeliegende N-Körper Problem, sprich die paarweise Berechnung der Gravitation zwischen allen an der Simulation beteiligten Körpern. Für N viele Körper müssen zur Lösung dieses Problems mit einem naiven Algorithmus N^2 viele Berechnungen durchgeführt werden, was zur entsprechenden Komplexität führt. Es gibt jedoch eine Vielzahl verschiedener Algorithmen, die die Komplexität des N-Körper Problems durch die Reduzierung der notwendigen Berechnung verbessern können. Ein Beispiel hierfür ist der Barnes-Hut Algorithmus, der die Körper nach ihrer räumlichen Lage in einen Baum einsortiert. [BH86] Dieser Baum soll dabei helfen zu entscheiden, welche Berechnungen zur Vereinfachung der Simulation zusammengefasst werden können, ohne die Genauigkeit der Simulation zu stark zu beeinträchtigen.

Die Lösung des N-Körper Problems scheint eine Aufgabe zu sein, die gut auf Grafikkarten zugeschnitten ist, da eine Vielzahl von Berechnungen durchgeführt werden muss, die jedoch unabhängig voneinander sind und parallel ausgeführt werden können. Das gleiche gilt jedoch nicht allgemein für die optimierten Algorithmen wie den Barnes-Hut Algorithmus, da die Erstellung des Barnes-Hut Baumes nicht vollständig parallel durchgeführt werden kann und die Reihenfolge, in der die Körper bei der Gravitationsberechnung abgearbeitet werden wechseln kann, je nachdem, wie sie zum aktuellen Zeitpunkt der Simulation im Barnes-Hut Baum liegen, wodurch die Rechenkapazität der Grafikkarte nicht vollständig ausgeschöpft werden kann.

Ausgangslage und Forschungsfrage

Die Herausforderung liegt nun darin, einen Weg zu finden, eine N-Körper Simulation trotz der Verwendung des Barnes-Hut Algorithmus auf eine Art und Weise durchzuführen, die die Stärken von Grafikkarten bestmöglich ausnutzt. Dabei lässt sich auf der bereits vorhandenen Implementierung eines Top-Down Barnes-Hut Algorithmus von Thüring [Thü23] aufbauen, die sich dank SYCL sowohl auf CPUs als auch auf GPUs ausführen lässt. Diese Implementierung soll nun um einen Barnes-Hut Algorithmus erweitert werden, bei dem der Barnes-Hut Baum bei den Battknoten beginnend Bottom-Up aufgebaut wird, um zu untersuchen, ob sich dies im Vergleich zu Top-Down positiv auf die Laufzeit auswirkt und somit eine sinnvolle Herangehensweise für die Durchführung von N-Körper Simulationen auf Grafikkarten ist. Ebenfalls sollen die Laufzeiten dieses neuen Algorithmus auf verschiedenen Systemen miteinander verglichen werden.

In Kapitel 2 wird zunächst auf wissenschaftliche Arbeiten eingegangen, die sich mit demselben Themenbereich beschäftigen, und diese Arbeit von ihnen abgegrenzt. In Kapitel 3 werden einige Grundlagenkenntnisse zu den behandelten Themen vermittelt, die für das Verständnis von Vorteil

sein könnten. In Kapitel 4 wird zunächst auf die theoretischen Überlegungen für die Erstellung des neuen Algorithmus eingegangen, bevor die konkrete Implementierung des Programms genauer betrachtet wird. In Kapitel 5 werden schließlich einige Messwerte und Laufzeiten des Bottom-Up Algorithmus vorgestellt und mit den Werten des Top-Down Algorithmus verglichen. Ebenfalls werden mögliche Erklärungen und Ursachen für das Verhalten des Bottom-Up Algorithmus geschildert. In Kapitel 6 wird schließlich nochmal ein Überblick über die im Rahmen dieser Arbeit gesammelten Erkenntnisse gegeben, bevor in **??** einige Vorschläge gemacht werden, wie der Bottom-Up Barnes-Hut Algorithmus in Zukunft noch verbessert und genauer untersucht werden könnte.

2 Related Work

In diesem Kapitel werden nun einige Arbeiten vorgestellt, die sich mit ähnlichen Themen wie in dieser Arbeit befasst haben. Die Optimierung von N-Körper Simulationen ist ein Thema, in dem bereits seit Jahrzenten geforscht wird. Es gibt jedoch auch schon Arbeiten, die sich mit der Implementierung solch optimierter N-Körper Algorithmen auf Grafikkarten beschäftigt haben. Nylons et al. haben sich damit beschäftigt, einen naiven N-Körper Algorithmus unter Verwendung von CUDA für die Ausführung auf Grafikkarten zu implementiern, und haben sich bei der Optimierung auf die Effizienz der Speicherzugriffe und die optimale Auslastung der von Ihnen getesteten Grafikkarte konzentriert. Sie stellen außerdem die Vermutung an, dass kaum ein N-Körper Algorithmus besser für die Ausführung auf Grafikkarten geeignet sein kann als der naive Algorithmus, was die Auslastung der vorhandenen Ressourcen der Grafikkarte angeht [NHP07].

Yokota und Barba sind noch einen Schritt weiter gegangen und haben mit dem Barnes-Hut Algorithmus und der Fast Multipole Methode fortgeschrittene N-Körper Algorithmen unter Verwendung von CUDA für die Ausführung auf Grafikkarten implementiert. Je nach Anzahl von Körpern, die simuliert wurden, haben sie mit diesen Algorithmen in der Spitze eine Beschleunigung von mehreren Größenordnungen erreichen können [YB11]. Singh et al. haben sich mit der Optimierung der Barnes-Hut und Fast Multipole Method Algorithmen für die Ausführung auf parallelen Systemen mit vielen Rechenkernen beschäftigt. Durch eine angepasste Partitionierung und Scheduling ist es ihnen gelungen, eine optimale Auslastung des Systems zu erreichen, die auch skaliert, wobei Speicherzugriffe so sehr wie möglich auf lokale Caches beschränkt werden. Sie merken jedoch an, dass die Optimierung der Algorithmen für die parallele Ausführung nicht trivial ist und deshalb nicht von einem Compiler oder automatischen Scheduler übernommen werden kann [SHT+95].

Es gibt deutlich weniger Veröffentlichungen, die sich bei der Implementierung der N-Körper Algorithmen auf SYCL stützen. Faqir-Rhazoui und Garcia untersuchen die Performance verschiedener Plattformen für portable Programmierung, unter anderem SYCL. Sie tun dies mithilfe einer Test-Suite, die als ein Testprogramm einen N-Körper Benchmark enthält, welcher sich jedoch auf den naiven N-Körper Algorithmus beschränkt. Nichtsdestotrotz kommen sie zu dem Schluss, dass SYCL die beste Performance für parallele Programme wie eine N-Körper Simulation bietet [FG23]. Es gibt anscheinend noch keine Arbeit, in der eine Optimierung des Barnes-Hut Algorithmus für parallele Ausführung vorgenommen wird, indem der Barnes-Hut Baum Bottom-Up anstatt Top-Down erzeugt wird, geschweige denn in SYCL. Daher handelt es sich beim Inhalt dieser Arbeit um einen neuen Ansatz, der nicht bereits Teil einer vorhandenen Arbeit ist.

3 Grundlagen

In diesem Kapitel werden einige theoretische Grundlagen erläutert, die für diese Arbeit relevant sind. Zu diesen Grundlagen gehört eine Einführung in N-Körper Simulationen allgemein sowie eine detailliertere Beschreibung des Barnes-Hut Algorithmus, wie er ursprünglich entworfen wurde. Anschließend wird die Idee zum Bottom-Up Barnes-Hut Algorithmus erläutert, zusammen mit einigen Schwierigkeiten, die sich aus der Umstruktierung des Algorithmus ergeben. Dazu wird noch die Hilbert-Kurve eingeführt, da sie für die Implementierung des Bottom-Up Barnes-Hut Algorithmus relevant ist. Zuletzt wird ein kleiner Überblick über die Fähigkeiten und Vorteile von SYCL gegeben, damit ersichtlich wird, warum dieser Standard für die Implementierung des Bottom-Up Algorithmus gewählt wurde.

3.1 N-Körper Problem

Das N-Körper Problem beschäftigt sich damit, die Bewegung einer Menge von Körpern durch die Berechnung der Kräfte, die zwischen ihnen wirken, vorherzusagen. Da diese Gravitationskräfte immer paarweise zwischen zwei Körpern berechnet werden müssen, und das für alle möglichen Paare der N Körper, hat der grundlegende All Pairs Algorithmus, bei dem die Anziehungskräfte einfach für alle Paare explizit berechnet werden, eine Komplexität von $O(N^2)$. Doch schon seit Jahrzenten beschäftigen sich Wissenschaftler mit Algorithmen, die Approximationen des N-Körper Problems in weniger als $O(N^2)$ lösen können. Seit jedoch Grafikkarten als leistungsstarke, hochparallele Prozessoren immer häufiger geworden sind, kam auch die Frage auf, ob das N-Körper Problem auch effizient und parallel gelöst werden könnte, um das Simulieren von einer großen Menge an Körpern in Echtzeit möglich zu machen. Mit einem naiven Ansatz ist das N-Körper Problem bestens für die Ausführung auf Grafikkarten geeignet, jedoch sind die optimierten N-Körper Algorithmen oft hierarchisch und daher nicht so gut für Grafikkarten geeignet. In dieser Arbeit geht es ausschließlich um die Variante des N-Körper Problems, die sich mit den Anziehungskräften zwischen Planeten und sonstigen Himmelskörpern beschäftigt, um ihre Bewegungen zu simulieren und damit auch vorhersagen zu können.

3.2 Barnes-Hut Algorithmus

3.2.1 Top-Down Barnes-Hut

Da N-Körper Simulationen mit einer quadratischen Komplexität viel Rechenzeit in Anspruch nehmen, ist es interessant, sich mit Möglichkeiten zu beschäftigen, die Anzahl der Berechnungen zu reduzieren. Eine solche Möglichkeit bietet der Barnes-Hut Algorithmus, indem er die Körper in der Simulation in einem Baum organisiert und anschließend für bestimmte Berechnungen

zusammenfasst. [BH86]. Ein Barnes-Hut Baum ist nach folgenden Prinzipien aufgebaut: jeder Knoten im Barnes-Hut Baum stellt entweder einen Körper dar, falls es ein Blattknoten ist, oder einen Bereich im Raum, falls es ein innerer Knoten ist. In diesem Fall werden die Körper, die sich im Bereich dieses inneren Knotens befinden, zu seinen Kindknoten. Den größten Raum umfasst der Wurzelknoten, nämlich die Axis-Aligned Bounding Box (AABB). In ihr sind alle Körper der Simulation enthalten. Je nachdem, in wie vielen Dimensionen sich die Simulation abspielt, hat jeder innere Knoten im Barnes-Hut Baum eine maximale Anzahl an Kindknoten. Im zweidimensionalen Raum kann ein Knoten bis zu vier Kindknoten haben, im dreidimensionalen Raum bis zu acht. Jeder Kindknoten stellt dabei einen Quadranten beziehungsweise einen Oktanten des Bereichs dar, den sein Elternknoten umfasst. Wird nun ein Körper in einen der Kindknoten hinzugefügt, muss er in den entsprechenden Kindknoten einsortiert werden, je nachdem in welchem Oktant er liegt. Gibt es in diesem Oktanten bereits einen Körper, wird der Kindknoten aufgesplittet. Das bedeutet, dass er selbst zu einem inneren Knoten wird und nun einen Bereich darstellt, nämlich den jeweiligen Oktanten, den er relativ zu seinem Elternknoten belegt hat. Dieser Bereich wird nun wiederum in acht kleinere Oktanten aufgeteilt, in die wieder Körper eingefügt werden können. Durch dieses Verfahren ergibt sich für jeden Körper im Raum eine eindeutige Position im Barnes-Hut Baum. Außerdem geht der Baum nur dort in die Tiefe, wo es notwendig ist, weil viele Körper dicht zusammen liegen. In weniger dicht besetzten Bereichen bleibt der Baum hingegen simpel und hat keine unnötigen Knoten.

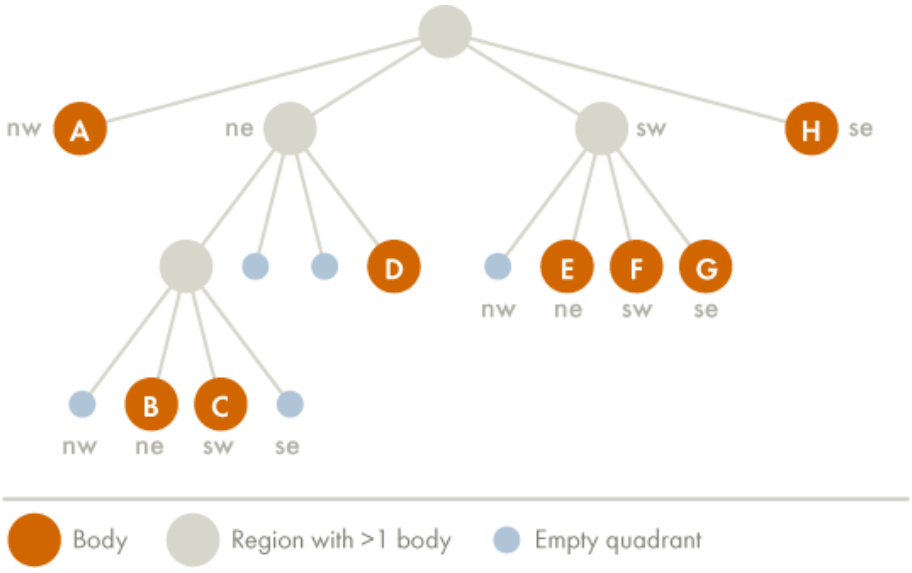

Abbildung 3.1: Ein beispielhafter zweidimensionaler Barnes-Hut Baum. Gut zu sehen ist die Bedeutung der einzelnen Kindknoten und die verschiedenen Knotentypen. Bei den beigen Knoten handelt es sich um die inneren Knoten, die selbst keine Körper enthalten und lediglich einen Bereich umfassen, in dem sich Körper befinden können. Die orangenen Knoten sind die Blattknoten, in denen Körper enthalten sind. Die blauen Punkte sind keine Knoten, sondern stellen in diesem Fall lediglich freie Quadranten des inneren Knoten dar, in denen sich bis jetzt noch kein Körper befindet.

In Abbildung 3.1[1] ist ein beispielhafter Barnes-Hut Baum zu sehen, an dem die eben genannten Prinzipien gut in Aktion zu sehen sind. Bei den beigen Knoten handelt es sich um die inneren Knoten, die selbst keine Körper enthalten, sondern diejenigen Körper zusammenfassen, die sich im Bereich des inneren Knoten befinden und somit in einem seiner Kindknoten enthalten sind. Die orangenen Knoten sind Blattknoten, die jeweils einen Körper darstellen. Die blauen Punkte sind nur Platzhalter an Stellen, wo sich aktuell noch kein Knoten befindet. Sie werden hier dargestellt, da in einem Barnes-Hut Baum jeder Kindknoten eine klare Zuordnung zu einem Quadranten hat, auch wenn noch Plätze für weitere Kindknoten frei sind. Dieser Barnes-Hut Baum wurde für einen zweidimensionalen Raum erstellt, weshalb jeder innere Knoten bis zu 4 Kindknoten haben kann, einen für jeden Quadranten. Wird ein Blattknoten in den Baum eingefügt, muss er an genau die Stelle eingefügt werden, an die er durch seine Position gehört. Er wandert also beginnend bei der Wurzel Ebene für Ebene durch den Baum, und bei jedem Schritt wird überprüft, in welchem Quadranten er liegt. Bei der Wurzel betrifft dies also die Viertel der gesamten AABB, im nächsten Schritt dann die Quadranten des Viertels, dem er auf der Wurzelebene zugewiesen wurde. Dies wird so oft wiederholt, bis der Körper entweder einem freien Kindknoten zugewiesen wird, oder an eine Stelle einsortiert wird, wo sich bereits ein Blattknoten befindet. Dann findet das bereits erwähnte Splitten dieses Blattknotens statt.

Die Beschleunigung der N-Körper Simulation ergibt sich nun daraus, dass nicht mehr alle Berechnungen für jedes Paar von Körpern ausgeführt werden. Stattdessen wird, abhängig von einem Parameter Theta, bei manchen Berechnungen der jeweilige Elternknoten des Körpers stellvertretend verwendet. Die Berechnung einmal für den Pseudoknoten durchgeführt, und die Berechnungen für alle seine Kindknoten entfallen. Dabei beträgt die Masse des Pseudoknotens die Summe der Massen der Kindknoten, und seine Koordinaten sind der gemeinsame Schwerpunkt der Kindknoten. Die Entscheidung, ob eine bestimmter Pseudoknoten für die Berechnung verwendet wird oder nicht, hängt vom Verhältnis seiner Distanz zum aktuell betrachteten Körper zur Kantenlänge dieses Pseudoknotens ab. Die Kantenlänge eines Pseudoknoten ist jeweils die Hälfte der Kantenlänge seines Elternknoten, und entspricht der Kantenlänge des Bereichs, in dem sich alle seine Kindknoten befinden. Ist das Verhältnis der Kantenlänge zur Distanz kleiner als Theta, werden die Kindknoten zusammengefasst und es müssen nicht alle Berechnungen einzeln ausgeführt werden. Wurde als Theta beispielsweise 0,5 gewählt, dann bedeutet das, dass die Pseudoknoten für die Berechnung verwendet werden, deren Kantenlänge weniger als die Hälfte ihrer Entfernung zum aktuell betrachteten Knoten beeträgt. Die Komplexität von N-Körper Simulationen kann durch die Verwendung dieses Algorithmus von $O(N^2)$ auf O(N log N) reduziert werden.

3.2.2 Bottom-Up „Barnes-Hut“

Um die Bottom-Up Erstellung des Baumes zu ermöglichen, müssen einige der Prinzipien des ursprünglichen Algorithmus übergangen werden, weshalb es sich streng genommen nicht mehr um einen Barnes-Hut Algorithmus handelt. Da alle Blattknoten eingefügt werden, bevor der erste innere Knoten erstellt wird, ist es nicht wie beim Top-Down Verfahren möglich, die Sortierung der Körper relativ zu den Körpern, die bereits im Baum vorhanden sind, vorzunehmen. Die Sortierung der Körper muss vor der Erstellung des Baumes stattfinden, und dabei trotzdem von den Positionen

[1] http://arborjs.org/docs/barnes-hut

der Körper relativ zueinander abhängen. Außerdem muss ein neuer Weg gefunden werden, die Kantenlängen der Bereiche der inneren Knoten zu berechnen, da sich diese nicht mehr nach den Kantenlängen ihrer Elternknoten richten können.

Der Anreiz, den Barnes-Hut Algorithmus derart umzubauen, ist, dass sich auf diese Art und Weise die Baumerzeugung besser parallelisieren lässt. Im Top-Down Verfahren kann man die Baumerzeugung nur parallelisieren, indem man den erzeugten Baum in seiner Höhe begrenzt und mehrere Bäume gleichzeitig erzeugen lässt. Diese Teilbäume werden anschließend zum Gesamtbaum zusammengesetzt. Aber auch hier müssen die Körper einzeln in die Teilbäume einsortiert und der Baum je nach Bedarf angepasst werden. Außerdem können sich die Blattknoten im Baum in unterschiedlicher Tiefe befinden, was schlecht beim parallelen Traversieren des Baumes ist. Beim Bottom-Up Verfahren können alle Knoten des Baumes, die auf einer Ebene liegen, parallel in den Baum eingefügt werden, da sich Knoten, die einmal hinzugefügt wurden, nachträglich nicht mehr verschieben. Der Baum ist außerdem ausbalanciert, sodass es immer gleich lang dauert, zu einem Blattknoten zu traversieren. Dies sollte bei der Verwendung des Baumes zur Beschleunigungsberechnung von Vorteil sein.

3.3 Hilbert-Kurve

Für das Vorsortieren der Körper für ihre Einsetzung in den Barnes-Hut Baum soll eine Hilbert-Kurve dienen. Bei der Hilbert-Kurve handelt es sich um eine raumfüllende Kurve, das heißt sie ist eine eindimensionale Kurve, die einen mehrdimensionalen Raum komplett durchlaufen kann, also jeden Punkt im Raum besucht. Sie tut dies in einem bestimmten Muster, bei dem sie zuerst alle Punkte in einem kleineren, quadratischen Bereich besucht, bevor sie sich zu einem anderen Bereich vorarbeitet. Legt man eine solche Kurve nun durch den Raum, in dem sich die N-Körper Simulation abspielt, kann man für jeden Körper bestimmen, an welcher Stelle auf der Hilbert-Kurve er sich befindet. Mithilfe dieser Information kann man näherungsweise bestimmen, wie die Körper zueinander liegen, da Körper, die auf der Hilbert-Kurve nahe zusammen liegen, tendenziell auch räumlich näher zusammen liegen werden.

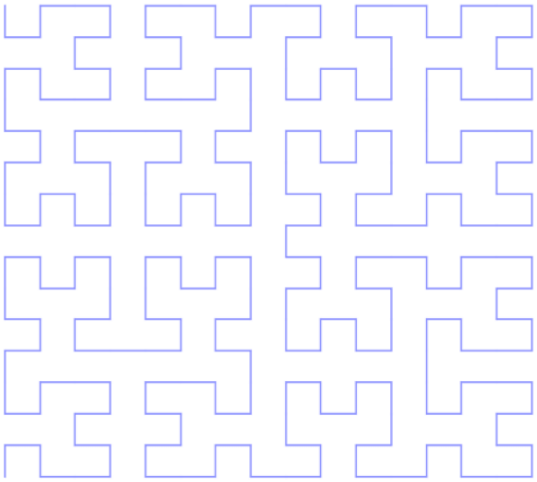

Abbildung 3.2: Beispiel einer zweidimensionalen Hilbert-Kurve. Gut zu erkennen ist das quadratische Muster, das sich rekursiv auf mehreren Ebenen durch die Kurve zieht. Sie traversiert die gesamte Fläche ohne größere Sprünge, jedoch nähert sie sich in manchen Fällen Bereichen wieder an, die sie bereits traversiert hat. Das kann dazu führen, dass manche Punkte, die auf der Hilbert-Kurve weiter voneinander entfernt sind, räumlich näher zusammen liegen als manche andere Punkte, die eigentlich auf der Hilbert-Kurve näher zusammen sind.

In Abbildung 3.2[2] ist eine zwedimensionale Hilbert-Kurve zu sehen, die die Struktur von Hilbert-Kurven allgemein gut zur Schau stellt und bei der Vorstellung einer Hilbert-Kurve im dreidimensionalen Raum helfen kann. Man erkennt hier klar das quadratische Muster, mit dem die Hilbert-Kurve die Fläche traversiert. Sie besucht alle Punkte in einem quadratischen Teilbereich der Fläche, bevor sie zu einem benachbarten quadratischen Teilbereich übergeht. Mehrere dieser Teilbereiche ergeben wiederum einen größeren quadratischen Teilbereich. Dieses Muster fährt so oft fort, bis der gesamte Bereich abgedeckt ist. Durch dieses Vorgehen kann man behaupten, dass Körper, die auf der Hilbert-Kurve nahe zusammenliegen (sprich, sie werden von der Hilbert-Kurve relativ direkt nacheinander besucht),tendenziell auch räumlich dicht zusammenliegen, da sie dann im selben quadratischen Teilbereich der Kurve liegen müssen. Für eine Sortierung anhand der Hilbert-Kurve kann man jedoch an diesem Schaubild auch ein Problem erkennen. Die Hilbert-Kurve arbeitet immer zuerst den Rand jedes quadratischen Teilbereichs ab, bevor sie zur Mitte zurückkehrt. Dadurch kann es sein, dass Körper, die mit einem größeren Abstand nacheinander von der Kurve besucht wurden, räumlich näher zusammen liegen als Körper, deren Abstand auf der Hilbert-Kurve kleiner ist. Das kann dazu führen, dass Körper, die vielleicht sogar direkt nebeneinander liegen, zum Beispiel in der Mitte der Fläche, einen großen Abstand auf der Hilbert-Kurve haben und so nicht zusammen einsortiert werden. Die Sortierung ist also keineswegs ideal.

[2] https://www.researchgate.net/publication/323003804_A_Spatial_Mapping_Algorithm_with_Applications_in_Deep_Learning-Based_Structure_Classification

3.4 Programmieren mit SYCL

Bei SYCL handelt es sich um einen relativ neuen C++-Standard, der sich zum Ziel setzt, ohne Spracherweiterungen das Erstellen von portablen Programmen, die sowohl auf GPUs als auch auf CPUs ausgeführt werden können, durch schreiben von einfachem C++-Code zu ermöglichen. Dabei werden im Hintergrund die jeweiligen spezifischen Backends des Geräts angesprochen, um den in SYCL geschriebenen Code in Code für die Maschine zu übersetzen. SYCL ist also im Prinzip eine Abstraktionsebene, die es ermöglicht, mit der gleichen Syntax Programme für diverse verschiedene Geräte zu schreiben.

Nichtsdestotrotz gibt es beim Programmieren mit SYCL einige Dinge zu beachten. Obwohl man in SYCL mit der normalen C++-Syntax schreiben kann, gibt es Einschränkungen für die sogenannten Kernels, also die Teile des Programms, die auf externenen Geräten ausgeführt werden. Manche Programmierparadigmen, wie zum Beispiel Rekursion, dürfen in den Kernels nicht verwendet werden, da sie nicht mit allen externen Geräten kompatibel sind und daher nicht übersetzt werden können. Abgesehen von solchen Einschränkungen ist es aber zum Beispiel möglich, eine Grafikkarte mit einem Standard C++-Programm anzusprechen, was die Lernkurve für die Entwicklung solcher Programme deutlich senkt.

Ein weiterer Teil von SYCL, mit dem man sich dennoch vertraut machen muss, sind die Memory Models. Die Memory Models dienen dazu, Daten zwischen den an der Ausführung des Programms beteiligten Geräten zu verschieben. In dieser Arbeit wird dafür das Buffer-Accessor Modell verwendet. Beim Buffer-Accessor Modell werden Daten vom Host-Gerät, üblicherweise der PC des Entwicklers, beziehungsweise dessen CPU, über einen Buffer bereit gestellt. Die anderen an der Ausführung beteiligten Geräte, zum Beispiel eine Grafikkarte, können dann mittels eines Accessors auf diese Buffer zugreifen, um die Daten auf die Grafikkarte zu kopieren. Man muss beachten, dass man innerhalb eines Kernels nur Zugriff auf die Daten hat, auf die man Vorher mit einem Accessor zugegriffen hat. Desweiteren hat man keine Garantie dafür, wann Änderungen, die ein externes Gerät an den Daten in diesem Buffer vornimmt, wieder mit dem Host synchronisiert werden, sodass die neuen Daten vom Buffer zurück in die ursprüngliche Datenstruktur kopiert werden. Es gibt Wege, den Transfer von Daten zu bestimmten Zeitpunkten zu erzwingen, jedoch trägt man dann als Entwickler die Verantwortung für unnötige Transfers von Daten, die die Ausführung des Programms verlangsamen. Beschränkt man sich auf die Nutzung der Buffer und Accessor, wird SYCL den Spielraum bei der Ausführung von Datenverschiebungen bestmöglich nutzen, damit es zu keinen Verzögerungen kommt.

4 Implementierung

In diesem Kapitel wird zunächst erkärt, was die Ideen zur Implementierung des Bottom-Up Barnes-Hut Algorithmus waren und wie der Algorithmus zur Erstellung des Baumes entworfen wurde. Anschließend wird für alle relevanten Programmteile erklärt, wie die Implementierung konkret umgesetzt wurde und welche Einschränkungen dabei zu beachten waren.

4.1 Theorie hinter der Implementierung und Herangehensweise

Für die Implementierung eines Bottom-Up Barnes-Hut Algorithmus muss man anders als beim ursprünglichen Algorithmus vorgehen, da das Bottom-Up Prinzip nicht mit dem eigentlichen Verfahren des Barnes-Hut Algorithmus zu vereinbaren ist. Da man das Erstellen des Barnes-Hut Baumes bei den Blättern beginnt, ist es nicht möglich, neue Körper anhand der bereits vorhandenen Knoten einzusortieren. Hierfür wäre eine vorhandene Teilstruktur des Baumes notwendig, sodass man für den neuen Körper entscheiden kann, in welche Oktanten der vorhandenen Knoten er hineinpasst und in welchen Kindknoten er dementsprechend hinzugefügt werden muss. Bei einem Bottom-Up Ansatz werden die inneren Knoten auf Basis der Blattknoten, welche bereits alle Körper enthalten, erstellt, sodass für eine korrekte Anordnung der Körper im Baum eine andere Strategie gewählt werden muss. Wie in Kapitel 3 bereits angesprochen, müssen die Körper der Simulation bereits vor der Erstellung des Baumes räumlich sortiert werden, falls dieser Bottom-Up erzeugt werden soll. Damit die Sortierung der Körper sinnvoll ist, muss sie einige Annahmen erfüllen. Konkret sollen Körper, die näher zusammen einsortiert werden, auch räumlich einen geringeren Abstand voneinander haben. Dies lässt sich nicht mit hundertprozentiger Korrektheit machen, ohne die Abstände zwischen allen Körpern zu bestimmen. Da dies eine Aufgabe mit quadratischer Komplexität ist, ist es jedoch nicht sinnvoll, alle Abstände als Vorbereitung für die Erstellung des Barnes-Hut Baumes zu bestimmen, da die Komplexität des gesamten Vorgangs, also der Erstellung des Barnes-Hut Baumes zusammen mit der Beschleunigungsberechnung, geringer als quadratisch sein soll. Die Sortierung, die hier durchgeführt wird, muss diese Bedingung also nur näherungsweise erfüllen.

Deshalb soll nun die Hilbert-Kurve als Grundlage für die Sortierung verwendet werden. Betrachtet man die Eigenschaften der Hilbert-Kurve, ist sie mit den richtigen Parametern in der Lage, die Bedingungen hinreichend gut zu erfüllen. Die Hilbert-Kurve reduziert einen dreidimensionalen Raum auf eine Dimension, indem sie den Raum in einem bestimmten Muster durchquert, sodass sie jeden Punkt im Raum einmal besucht. Die Punkte kann man dann anstatt mit ihren dreidimensionalen Koordinaten mit der Stelle identifizieren, auf der sie auf der Hilbert-Kurve liegen, also ihrem Hilbert-Index. Wählt man den Grad der Kurve (und damit auch die Anzahl von Punkten auf der Hilbert-Kurve) im Verhältnis zur Anzahl der Körper in der N-Körper Simulation entsprechend, kann man ein hinreichend gutes Ergebnis erzielen, bei dem die Fälle, wo sich die Kurve wieder einem Bereich im Raum annähert, wo sie bereits war, und somit Körper weiter voneinander entfernte

Hilbert-Indizes zugewiesen bekommen, obwohl sie in Wahrheit nahe zusammen liegen, möglichst wenig ins Gewicht fallen. Für die Sortierung sollen alle Körper in der Simulation einen Hilbert-Index bekommen. Anschließend werden sie anhand ihrer Indizes aufsteigend angeordnet. Damit gelten sie dann als näherungsweise sortiert.

Die durch die Hilbert-Indizes zustande gekommene Sortierung der Körper wird dann zur Grundlage der Anordnung der Körper im Bottom-Up Barnes-Hut Baum, indem alle Körper in der selben Reihenfolge in Blattknoten eingefügt werden. Wurde für jeden Körper im Datensatz ein Blattknoten erstellt, wird der Baum eine Ebene nach der anderen nach oben hin bis zur Wurzel aufgebaut, wobei alle Knoten auf der selben Ebene parallel erzeugt werden können. Prinzipbedingt bietet der Bottom-Up Ansatz einige Vorteile gegenüber Top-Down. Einerseits ist der fertige Baum ausbalanciert, es dauert also gleich lang, um zu allen Blattknoten zu iterieren. Bei einem Top-Down Barnes-Hut Baum kann es zwischen den Blattknoten große Unterschiede dabei geben, wie tief sie im Baum liegen. Der Top-Down Baum besteht für den gleichen Datensatz normalerweise auch aus deutlich mehr Knoten als der Bottom-Up Baum. Dadurch dauert die Erzeugung des Baums Top-Down im Normalfall länger (siehe Kapitel 5).

4.2 Sortieralgorithmus auf Basis von Hilbert-Indizes

Damit die Sortierung anhand der Hilbert-Kurve auch effizient auf Grafikkarten durchgeführt werden kann, wird ein iterativer Algorithmus benötigt, mit dem die Hilbert-Indizes für alle Körper parallel und unabhängig voneinander bestimmt werden können. Deshalb wird nicht eine gesamte Hilbertkurve berechnet, sondern nur die Indizes der Körper des Datensatzes, die sie auf einer hypothetischen Hilbertkurve (aber mit festen Eigenschaften) haben würden, wenn man sie durch den Raum ziehen würde. Der Vorteil dieser Methode ist, dass sich für eine Hilbertkurve mit gegebenen Parametern diese Indizes für alle Körper unabhängig voneinander, parallel, und ohne Rekursion berechnen lassen (im Gegensatz zu einer vollständigen Hilbert-Kurve), sodass sie sich für die Ausführung auf einer Grafikkarte eignet. Zur Berechnung der Indizes wird der C-Algorithmus von Doug Moore[1] verwendet. Da dieser Algorithmus mit bit-shift Operationen arbeitet, unterstützt er lediglich positive Koordinaten in ganzzahliger Form. Die Koordinaten der gegebenen Datensätze müssen also so umgeformt werden, dass es auch ohne Nachkommastellen genügend verschiedene Punkte gibt, damit nicht zu viele Körper gleiche Hilbert-Indizes zugewiesen bekommen. Hierfür werden die Koordinaten aus den Datensätzen zunächst so verschoben, dass sie alle im positiven Bereich liegen, und anschließend mit dem festen Faktor 1000 skaliert. Dieser Faktor wurde gewählt, da sich aus Erfahrungswerten ergeben hat, dass er sich am besten für die räumliche Sortierung der Körper mit den gegebenen Datensätzen eignet. Je nachdem, wie groß die absoluten Werte der Koordinaten eines Datensatzes sind, und wie dicht die Körper in einem Datensatz beieinander liegen, kann der optimale Faktor anders ausfallen. Die Qualität der Sortierung kann mit verschiedenen Skalierungsfaktoren aufgrund des „hin und her“ der Hilbert-Kurve schwanken, Je nachdem wo genau sie relativ zu den Körpern im Datensatz verläuft.

Die transformierten Koordinaten können nun in den Algorithmus zur Berechnung der Hilbert-Indizes eingegeben werden. Hierbei muss der Grad der Hilbert-Kurve entsprechend gewählt werden, damit sie genügend mögliche Indizes für die Anzahl der Körper zur Verfügung stellt. Es kann toleriert

[1] https://github.com/adishavit/hilbert

werden, wenn manchmal derselbe Index an mehrere Körper vergeben wird, jedoch sollte dies nicht zu oft vorkommen, da sich sonst aus den Indizes nicht mehr die räumliche Lage der Körper zueinander ablesen lässt. Der Algorithmus weist nun jedem Körper einen Index auf der definierten Hilbert-Kurve zu. Die Indizes werden in ein Array gespeichert, jeder Index jeweils an die Stelle, an der der zugehörige Körper im Datensatz liegt, damit Hilbert-Index und Körper einander zugeordnet werden können.

Die Hilbert-Indizes sollen nun aufsteigend sortiert werden. Da Sortieren auf Grafikkarten nicht effizient möglich ist, muss die Datenstruktur mit den Indizes hierfür auf den CPU-Speicher übertragen werden. Hier kann das Sortieren einfach auf der CPU mit einem Aufruf von std::sort() erledigt werden. Jedoch sollen nicht lediglich die Hilbert-Indizes aufsteigend sortiert werden, sondern jegliche Verschiebungen der Hilbert-Indizes sollen analog mit einer Liste von IDs der Körper durchgeführt werden, sodass als Endergebnis eine räumlich sortierte Liste aller Körper steht (dargestellt durch ihre IDs). Hierfür wurde ein eigener Comparator definiert, der anschließend beim Aufruf von std::sort() übergeben wurde.

Listing 4.1 Comparator für das Sortieren der Paare aus Hilbert-Index und Körper-ID anhand der Indizes

```
struct pairComparator {
    bool operator()(const std::pair<bitmask_t, int>& left,
             const std::pair<bitmask_t, int>& right) {
             return left.first < right.first;
    }
};
```

Damit die Hilbert-Indizes und die IDs der Körper gleichzeitig sortiert werden können, wurde eine Datenstruktur erstellt, die Paare aus jeweils einem Hilbert-Index und der zugehörigen Körper-ID beinhaltet. Der Comparator in Listing 4.1 vergleicht in dieser Datenstruktur nun die Paare ausschließlich auf Basis ihrer Hilbert-Indizes (der Hilbert-Index ist immer an der ersten Stelle des Paares). Ist dabei der Hilbert-Index des linken Paares größer als der des rechten Paares, werden die gesamten Paare vertauscht. So werden die Hilbert-Indizes aufsteigend sortiert, wobei gleichzeitig die Körper in die richtige Reihenfolge gebracht werden. Anschließend können die IDs der Körper aus der gemeinsamen Datenstruktur entnommen werden und weiter für die Erstellung des Barnes-Hut Baumes verwendet werden. Um die Performance des Sortiervorgangs zu verbessern, wurden die Schleifenaufrufe, mit denen Daten zwischen Datenstrukturen verschoben werden, mit `#pragma omp parallel for` parallelisiert. Da die Sortierung auf der CPU ausgeführt wird, nimmt dieser Schritt trotzdem einen beträchtlichen Anteil der Laufzeit in Anspruch, in etwa so lang wie die Generierung des Barnes-Hut Baumes (siehe Kapitel 5).

4.3 Bottom-Up Baumerstellung

Ist der Sortiervorgang abgeschlossen, werden die Körper in genau der Reihenfolge in Blattknoten eingefügt. Wurden für alle Körper Blattknoten erzeugt, beginnt der Aufbau der nächsthöheren Ebene des Baumes. Dabei werden immer 8 aufeinanderfolgende Blattknoten zu den Kindknoten eines

inneren Knotens, der über sie eingefügt wird. Die eigentliche Idee des Barnes-Hut Baumes, dass jeder Kindknoten einen Oktanten des Bereichs des Elternknotens darstellt, wird hierbei komplett ignoriert. Diese Vorgabe einzuhalten ist Bottom-Up nicht möglich, da jeder innere Knoten erst nach den Blattknoten eingefügt wird, und dann Blattknoten umsortiert oder geteilt und Top-Down erweitert werden müssten, was sehr viel Zeit in Anspruch nehmen und das Bottom-Up Prinzip verletzen würde. Die durch die Hilbert-Indizes vorgegebene Reihenfolge der Blattknoten zu übernehmen, ist also deutlich praktikabler. Wurde für alle Blattknoten ein Elternknoten eingefügt, werden im nächsten Schritt diese inneren Knoten wieder zu jeweils 8 Kindknoten eines höherliegenden Knoten zusammengefasst. Dieser Vorgang wird wiederholt, bis in einem Schritt ein einziger Elterknoten für alle Knoten einer Ebene ausreicht, dies ist dann die Wurzel.

Da bei der Erstellung jedes inneren Knoten beim Bottom-Up Verfahren bereits alle Kindknoten von Anfang an bekannt sind, kann man direkt während der Erstellung des Baumes die Massen und Massezentren jedes Pseudoknoten berechnen. Beim Top-Down Verfahren muss hierfür nochmal durch den gesamten Baum iteriert werden, nachdem er vollständig erstellt wurde. Ein weiterer Unterschied betrifft die Berechnung der Kantenlängen für die inneren Knoten. Die Kantenlängen der inneren Knoten stellen ein Maß für die Größe des Bereichs dar, den ein innerer Knoten mit seinen Kindknoten abdeckt. Im Top-Down Verfahren beträgt die Kantenlänge einfach die Hälfte der Kantenlänge des Elternknotens, wobei die Kantenlänge der Wurzel der Kantenlänge der AABB entspricht. Da im Bottom-Up Verfahren ein innerer Knoten bei seiner Erstellung noch keinen Elternknoten hat, muss die Kantenlänge hier anders berechnet werden. Für die inneren Knoten, deren Kindknoten die Blattknoten sind, werden die Koordinaten der in den Blattknoten enthaltenen Körpern miteinander verglichen. Es werden jeweils die kleinsten und größten X-, Y- und Z-Koordinaten abgespeichert. Anschließend werden drei Kantenlängen berechnet, indem jeweils die kleinste X-Koordinate von der größten abgezogen wird, analog für die anderen Achsen. Die größte dieser drei Kantenlängen wird zur Kantenlänge des inneren Knotens. Für innere Knoten, deren Kindknoten ebenfalls innere Knoten sind, werden die abgespeicherten kleinsten und größten X-, Y- und Z-Koordinaten miteinander verglichen. Es werden von allen diesen Werten die minimalen und maximalen Koordinaten bestimmt und anschließend für jede Achse das Minimum vom Maximum subtrahiert. Daraus ergibt sich die Kantenlänge für den neuen inneren Knoten. Dieser Vorgang wird bis zur Wurzel fortgeführt.

Anders als beim Top-Down Verfahren ist die Kantenlänge der Wurzel beim Bottom-Up Verfahren jedoch nicht gleich der Kantenlänge der AABB. Bei der Berechnung der AABB werden nach der Bestimmung der maximalen Kantenlänge die minimalen und maximalen Koordinaten jener Achsen, auf der die größte Kantenlänge nicht lag, so verändert, dass die Kantenlänge auf allen Achsen gleich ist, und sich somit ein quadratischer Bereich ergibt. Beim Bottom-Up Verfahren entfällt dieser Schritt. Durch diese Anpassung der minimalen und maximalen Koordinaten vergrößert sich der Bereich, und weil beim Bottom-Up Verfahren weitere Kantenlängen auf der Grundlage dieses Bereichs berechnet werden, würde diese Vergrößerung mehrfach stattfinden und sich aufsummieren. Am Ende hätte die Wurzel eine Kantenlänge, die deutlich über der Kantenlänge der AABB liegt. Daher werden die Bereiche in ihrer rechteckigen Form belassen, um realistischere (und kleinere) Kantenlängen für die Beschleunigungsberechnung zu erhalten.

Als weitere Optimierung des Bottom-Up Verfahrens wurde die Möglichkeit eingeführt, mehr als 8 Kindknoten pro innerem Knoten einzufügen. Da der Grund für die Anzahl von 8 Kindknoten, nämlich dass jeder Kindknoten einen räumlichen Oktanten darstellt, beim Bottom-Up Verfahren nicht mehr existiert, kann diese Optimierung ohne Weiteres eingeführt werden. Ob jeweils 8 oder

16 (oder mehr) aufeinanderfolgende Körper aus der vorsortierten Liste in die Kindknoten eines einzelnen inneren Knoten eingefügt werden, macht für die Korrektheit keinen Unterschied. Es müssen lediglich die Anzahl der Durchläufe in den inneren Schleifen des Algorithmus, die über die Kindknoten iterieren, erhöht und die Anzahl der eingefügten inneren Knoten verringert werden. Dies ist leicht zu parametrisieren, sodass man je nach Bedarf die Anzahl der Kindknoten pro innerem Knoten bei jeder Ausführung des Programms einstellen kann. Das kann vor allem bei großen Datensätzen für die Laufzeit von Vorteil sein, da so in großen Mengen von dicht beieinander liegenden Körpern mehr Körper auf einmal mit einem einzigen Pseudoknoten dargestellt werden können. Die Anzahl der bei der Beschleunigungsberechnung zu bearbeitenden Knoten kann also verringert werden.

4.4 Beschleunigungsberechnung

Der Bottom-Up Baum wurde auf eine Art und Weise programmiert, dass er sich möglichst ähnlich zum Top-Down Baum verhält, um Änderungen am restlichen Programm zu vermeiden. Nichtsdestotrotz gibt es einige prinzipielle Unterschiede, die Änderungen an der Beschleunigungsberechnung notwendig machten. So gibt es für den Top-Down Baum eine Datenstruktur, die die Körper in-order gemäß ihrer Lage im Baum sortiert, um die Beschleunigungsberechnung dahingehend zu optimieren, dass Körper, die gleichzeitig bearbeitet werden, eher in der gleichen Tiefe im Baum liegen. Da die Körper im Bottom-Up Baum alle auf derselben Ebene liegen, kann diese Datenstruktur hier ignoriert werden. Ein weiterer Unterschied ist der Index der Wurzel. Die Beschleunigungsberechnung beginnt mit dem Hinzufügen der Wurzel auf den Stack, und anschließend (falls vorhanden) mit dem Hinzufügen ihrer Kindknoten. Da beim Top-Down Baum die Wurzel immer als erstes hinzugefügt wird, war hier die ID des ersten Knotens fest auf 0 kodiert. Beim Bottom-Up Baum wird die Wurzel jedoch zuletzt hinzugefügt, die ID des ersten Knotens auf dem Stack muss also dynamisch aus der Anzahl der Knoten bestimmt werden. Eine weitere Änderung an der Beschleunigungsberechnung war erforderlich, um mehr als 8 Kindknoten pro innerem Knoten im Bottom-Up Baum zu ermöglichen. Da es diesen Fall bei den Top-Down Bäumen nie gibt, erwartet der Beschleunigungs-Kernel immer eine feste Anzahl von 8 Kindknoten. Für den Bottom-Up Baum wurde dies in eine Schleife umprogrammiert, deren Anzahl an Durchläufen durch die vom Nutzer gewählte Anzahl an Kindknoten bestimmt wird.

5 Auswertung

In diesem Kapitel werden nun die mit dem neuen Algorithmus durchgeführten Tests beschrieben. Die gemessenen Laufzeiten werden mit den Laufzeiten der Top-Down Implementierung verglichen. Außerdem wird die Auswirkung verschiedener Parameter auf die Performance des neuen Algorithmus untersucht. Anschließend werden für die Beobachtungen mögliche Erklärungen gegeben.

5.1 Versuchsaufbau

Die folgenden Ergebnisse wurden, falls nicht anders angegeben, auf einem System mit einer NVIDA GeForce RTX 3080 Grafikkarte erzielt. Für den Vergleich zwischen verschiedenen Systemen wurden Systeme mit den Grafikkarten NVIDIA Geforce RTX 3090, NVIDA Tesla P100, nachfolgend Tesla genannt, und NVIDIA Quadro GP100, nachfolgend Quadro genannt, verwendet. Für das Erstellen des Testprogramms wurde hipSYCL verwendet. Was die Parameter des Programms betrifft, wurden die nachfolgend nicht genannten Parameter immer auf ihren Standardwerten belassen, die auch in der Dokumentation des Programms zu finden sind. Konkret beträgt die Dauer eines Simulationsschrittes eine Stunde in der internen Simulationszeit, und die Anzahl der Work Items beträgt 640. Als Datensätze wurden unterschiedlich detaillierte Abbildungen des Sonnensystems aus der NASA JPL Small Body Database mit jeweils 178, 999, 2678, 12648, 19054, 138723 oder 1216869 Körpern verwendet. Die interne Gesamtzeit für die Simulation wurde abhängig von der Größe des Datensatzes auf 30 Tage für alle Datensätze bis 2678 Körpern, 7 Tage für die Datensätze mit 12648 und 19054 Körpern, und einen Tag für die Datensätze mit 138723 und 1216869 Körpern gesetzt. Die nachfolgend dargestellten Laufzeiten beziehen sich jeweils auf die durchschnittliche Dauer eines einzelnen Simulationsschritts, die sich aus dem Durchschnitt der Laufzeiten aller Simulationsschritte einer einzelnen Simulation ergibt. Alle Laufzeiten werden in Millisekunden angegeben.

5.2 Laufzeiten und Messwerte

5.2.1 Top Down verglichen mit Bottom Up

Gesamtlaufzeit von Simulationsschritten

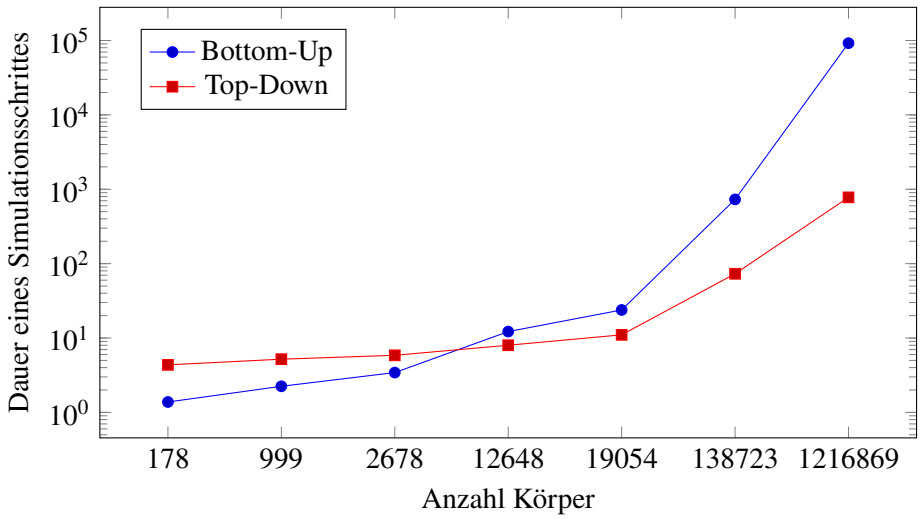

Abbildung 5.1: Vergleich der Laufzeiten des Bottom-Up Algorithmus und des Top-Down Algorithmus bei verschieden großen Datensätzen und gleichbleibendem Theta von 0,6. Zu sehen ist die durchschnittliche Dauer eines einzelnen Simulationsschrittes in Millisekunden im Verhältnis zur Anzahl der Körper im Datensatz. Der Bottom-Up Algorithmus ist bei kleineren Datensätzen schneller, skaliert jedoch schlechter und fällt daher hinter den Top-Down Algorithmus zurück.

Mit diesem Schaubild kann man einen ersten Eindruck für die Performance des Bottom-Up Ansatzes im Vergleich zum herkömmlichen Top-Down Barnes-Hut Algorithmus gewinnen. In Abbildung 5.1 wird die durchschnittliche Laufzeit eines einzelnen Simulationsschrittes ins Verhältnis zur Anzahl an Körpern im verwendeten Datensatz gesetzt. Die Laufzeiten wurden alle mit einem Theta von 0,6 erzielt. Man kann sehen, dass Bottom-Up bei den drei kleinsten Datensätzen einen Vorteil gegenüber Top-Down hat, aber dann schlechter skaliert. Während sich bei den mittelgroßen Datensätzen mit 10.000 bis 20.000 Körpern die Laufzeiten noch im selben Rahmen bewegen, kann Bottom-Up bei den zwei größten Datensätzen nicht mehr mithalten und wird um jeweils eine beziehungsweise zwei Größenordnungen langsamer. Dies ist natürlich ein enttäuschendes Ergebnis, jedoch lässt sich mit einer genaueren Untersuchung der Laufzeiten der einzelnen Programmteile die Ursache für die schlechte Skalierung besser eingrenzen.

Vergleich der Laufzeiten einzelner Programmteile

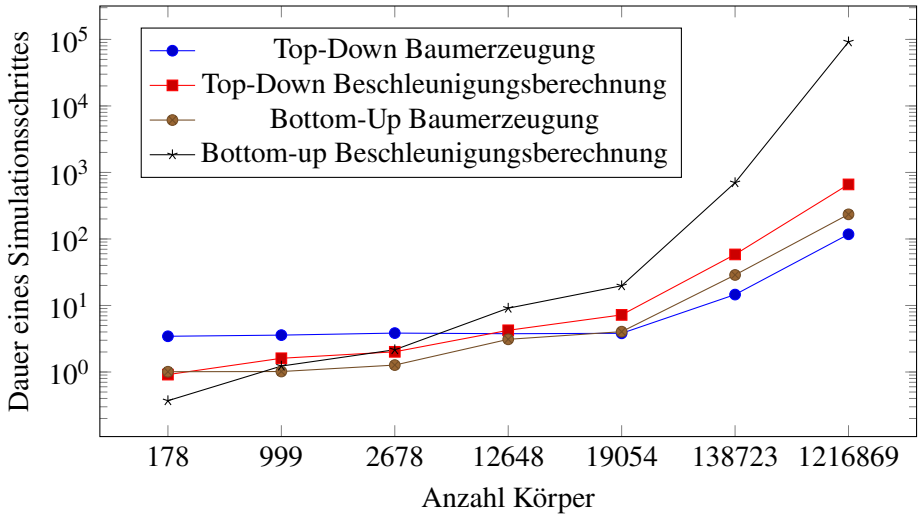

Abbildung 5.2: Vergleich der Laufzeiten der zwei großen Programmhälften Beschleunigungsberechnung und Baumerstellung, sowohl beim Bottom-Up Algorithmus als auch beim Top-Down Algorithmus, bei verschieden großen Datensätzen und gleichbleibendem Theta von 0,6. Zu sehen ist die durchschnittliche Dauer der jeweiligen Programmteile innerhalb eines einzelnen Simulationsschrittes, angegeben in Millisekunden, im Verhältnis zur Anzahl der Körper im Datensatz. Man kann erkennen, dass der Hauptgrund für die schlechte Skalierung des Bottom-Up Algorithmus innerhalb der Beschleunigungsberechnung liegt, da diese bei großen Datensätzen mit Abstand den größten Anteil der Laufzeit einnimmt.

In Abbildung 5.2 wird der Vergleich zwischen Bottom-Up und Top-Down etwas vertieft, um die genaue Ursache für die schlechte Skalierung des Bottom-Up Algorithmus eingrenzen zu können. Die Gesamtlaufzeiten der Simulationsschritte aus Abbildung 5.1 wurden in diesem Schaubild auf die zwei gro0en Programmteile aufgeteilt, die Baumerzeugung (inklusive Berechnung von Schwerpunkten, Hilbert-Indizes, Sortierung, und eigentlicher Erstellung der Baumstruktur) und den Beschleunigungskernel, in dem der fertige Barnes-Hut Baum für die Berechnung des nächsten Simulationsschrittes verwendet wird. Die Laufzeiten wurden alle wieder mit einem Theta von 0,6 gemessen. Es wird deutlich, dass die Beschleunigungsberechnung den größten Anteil an der schlechten Skalierfähigkeit des Bottom-Up Algorithmus hat. Während sie für die drei kleinen Datensätze noch schneller beziehungsweise gleich schnell ist wie bei Top-Down, wird sie dann deutlich langsamer und ist beim größten Datensatz schlussendlich um mehrere Größenordnungen langsamer als beim Top-Down Algorithmus. Der übrige Teil des Programms skaliert bei Bottom-Up zwar auch schlechter als bei Top-Down, jedoch längst nicht in demselben Maß wie die Beschleunigungsberechnung.

Ein paar weitere Metriken geben einen Einblick, was zu dieser schlechten Performance der Beschleunigungsberechnung führen könnte. So beträgt für den größten Datensatz die Anzahl bearbeiteter Knoten während der Beschleunigungsberechnung pro Simulationsschritt beim Bottom-Up Algorithmus knapp 120.000 Knoten, während beim Top-Down Algorithmus unter den selben

Gegebenheiten lediglich etwa 1200 Knoten für die Beschleunigungsberechnung bearbeitet werden. Dieser Faktor entspricht in etwa dem Faktor, um den die Beschleunigungsberechnung beim Bottom-Up Algorithmus langsamer ist als beim Top-Down Algorithmus. Der gewünschte Effekt des Barnes-Hut Algorithmus, die Anzahl der notwendigen Berechnungen bei der Beschleunigungsberechnung zu verringern, ist in der neuen Bottom-Up Variante also deutlich ineffektiver. Dazu kann man noch eine weitere Metrik einführen. Betrachtet man die durchschnittliche Kantenlänge aller inneren Knoten der Barnes-Hut Bäume, so beträgt sie beim Bottom-Up Algorithmus circa 0,95, während die durchschnittliche Kantenlänge im Top-Down Barnes-Hut Baum lediglich 0,04 beträgt. Diese Werte beziehen sich auf die Differenzen der Koordinaten der Körper im Koordinatensystem des Datensatzes. Zur Referenz, die meisten Körper in den Datensätzen befinden sich im Bereich zwischen -30 und 30 auf den Koordinatenachsen. Bei so einem großen Unterschied können im Top-Down Verfahren dann natürlich viel mehr Pseudoknoten für die Beschleunigungsberechnung eingesetzt werden, während beim Bottom-Up Verfahren in viel mehr Fällen die Beschleunigung für einzelne Körper berechnet werden muss.

Die Ursache für diese schlechten Werte liegt einerseits an der Methode, die für die Vorsortierung der Körper verwendet wurde. Wie bereits erwähnt hat das Muster der Hilbert-Kurve zufolge, dass sie sich in ihrem Verlauf immer wieder Bereichen annähert, die sie bereits abgedeckt hat. Liegen dann Körper auf den gegenüberliegenden Seiten der Lücke, die die Hilbert-Kurve zwischen diesen Bereichen lässt, dann werden sie unter Umständen durch das gewählte Sortierverfahren mit gehörigem Abstand voneinander einsortiert, obwohl sie räumlich dicht beieinander liegen. Passiert dies zu oft, bleiben unter Umständen nicht mehr genug richtig sortierte, nah beieinander liegende Körper übrig, um eine Gruppe von Kindknoten für einen einzelnen Elternknoten zu bilden, die dicht genug ist, dass sie in vielen Fällen mit dem Pseudoknoten vereinfacht werden kann. Wenn es in jeder Gruppe von Kindknoten einen Kindknoten gibt, der etwas weiter von den anderen entfernt ist, reicht dies bereits aus, um die Kantenlänge des Elternknoten so zu vergrößern, dass häufig für alle Kindknoten die Beschleunigungsberechnungen einzeln durchgeführt werden müssen.

Ein weiterer Faktor, der das Problem noch verschärft, ist einer, der eigentlich auch einer der Vorteile des Bottom-Up Ansatzes sein sollte. Dadurch, dass der Bottom-Up Barnes-Hut Baum komplett ausbalanciert ist, hat jeder innere Knoten zwangsläufig 8 Kindknoten (außer einem, falls die Anzahl der Körper nicht durch 8 teilbar ist). Das heißt er hat 8 Kindknoten, egal, ob es an dieser Stelle 8 Körper gibt, die dicht beieinander liegen, oder nicht. Selbst, wenn die Vorsortierung der Körper einwandfrei funktioniert, heißt das nur, dass jedem inneren Knoten die 8 nächsten Körper als Kindknoten zugeteilt werden. Diese können dennoch so weit auseinander liegen, dass es nie dazu kommt, dass der Pseudoknoten für eine Beschleunigungsberechnung verwendet wird. Beim Top-Down Algorithmus ist es deutlich flexibler, da dort wirklich nur die Körper Kindknoten eines inneren Knoten werden, die sich auch im entsprechenden Bereich befinden. Dabei muss es sich auch nicht immer zwangsläufig um 8 Körper handeln. Selbst wenn an bestimmten Stellen nur 4 Körper nah genug zusammen sind, um problemlos als Pseudoknoten zusammengefasst zu werden, würde dies beim Top-Down Algorithmus geschehen, und es würden drei Berechnungen eingespart werden. Beim Bottom-Up Algorithmus würden zu den vier dicht zusammen liegenden Körpern noch weitere 4 als demselben Elternknoten hinzugefügt werden. Liegen diese dann nicht mehr so dicht zusammen, müssen häufiger alle 8 Berechnungen einzeln ausgeführt werden. Durch die standardmäßige Zuteilung von 8 Kindknoten zu jedem Elternknoten ist der Bottom-Up Algorithmus etwas starr, und eine Vereinfachung wird nur in den Fällen vorgenommen, wo mindestens 8 Körper dicht zusammen liegen. Alle anderen Fälle, die bei Top-Down zu einer Vereinfachung in kleinerem Maße führen würden, werden von Bottom-Up überhaupt nicht optimiert.

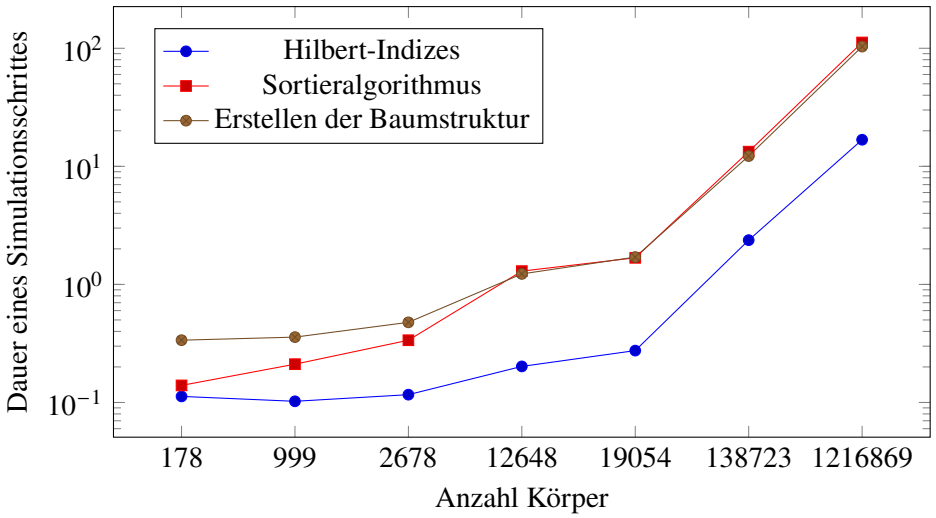

Abbildung 5.3: Vergleich der Laufzeiten der Programmabschnitte Berechnung der Hilbert-Indizes, Sortieren der Körper anhand der Indizes, und eigentliche Generierung der Baumstruktur innerhalb der Erstellung des Bottom-Up Barnes-Hut Baumes, bei verschieden großen Datensätzen und gleichbleibendem Theta von 0,6. Zu sehen ist die durchschnittliche Dauer der einzelnen Abschnitte innerhalb eines Simulationsschrittes. Die Laufzeiten werden in Millisekunden angegeben. In dieser Abbildung stehen sie im Verhältnis zur Anzahl der Körper im jeweiligen Datensatz. Während die Berechnung der Hilbert-Indizes keine große Rolle für die Laufzeit der Baumerstellung spielt, dauern die Ausführung des Sortieralgorithmus und die restliche Arbeit zur Erstellung des Baumes (inklusive Berechnung der Schwerpunkte und Massen) in etwa gleich lang.

In Abbildung 5.3 wird der Teil „Baumerzeugung“ aus Abbildung 5.2 noch einmal weiter unterteilt. Zu sehen sind die durchschnittlichen Laufzeiten der Programmteile zur Berechnung der Hilbert-Indizes, zum Sortieren der Körper anhand der Indizes und zum eigentlichen Erstellen der Baumstruktur, im Verhältnis zur Größe des verwendeten Datensatzes bei einem Theta von 0,6. Diese Laufzeiten sind dabei alles Teile der Gesamtlaufzeit eines einzelnen Simulationsschrittes. Laut den Messwerten hat die Berechnung der Hilbert-Indizes den kleinsten Einfluss auf die Dauer der Baumerstellung. Sie läuft für den größten Datensatz um etwa eine Größenordnung schneller als die übrigen beiden Programmteile. Eine weitere Optimierung an dieser Stelle scheint also nicht notwendig. Die Laufzeiten der Erstellung der Baumstruktur und des Sortieralgorithmus hingegen betragen beim größten Datensatz beide in etwa 100 Millisekunden. Dies ist vor allem dann interessant, wenn man sich das Verhältnis zur Dauer der Baumerstellung im Top-Down Verfahren in Erinnerung ruft. Dort hat der gesamte Vorgang, inklusive Berechnung der Massen und Schwerpunkte für den größten Datensatz im Schnitt etwa 120 Millisekunden gedauert. Lässt man den Sortieralgorithmus außen vor, wäre das Erstellen der Baumstruktur mit knapp unter 100 Millisekunden dann tatsächlich etwas schneller. Jedoch wird die Sortierung gebraucht, damit bei der Generierung des Baumes das aufwendige Einsortieren der Körper entfallen kann. Betrachtet man, dass die Körper bei der Baumgenerierung überhaupt nicht mehr sortiert werden müssen, sondern lediglich eingefügt werden, ist die Laufzeit für diesen Vorgang überraschend hoch. Eine mögliche Erklärung könnten die Schleifen sein, die für jeden inneren Knoten Berechnungen mit all seinen Kindknoten anstellen,

ohne dafür weitere Paralellität einzuführen. Der Einfluss dieser Schleifen auf die Laufzeit der Baumgenerierung wird in einem späteren Teil dieses Kapitels nochmal etwas ersichtlicher. Aber selbst wenn die Erstellung des Baumes überhaupt keine Zeit in Anspruch nehmen würde, wäre hier aufgrund der Laufzeit des Sortieralgorithmus trotzdem keine Zeitersparnis im Vergleich zum Top-Down Verfahren möglich. Da eine Vorsortierung der Körper für den Bottom-Up Ansatz notwendig ist, Sortieren aber nicht für die Ausführung auf Grafikkarten geeignet ist, ist es schwierig hier eine Empfehlung für die Verbesserung der Performance zu geben.

5.2.2 Verschiedene Theta

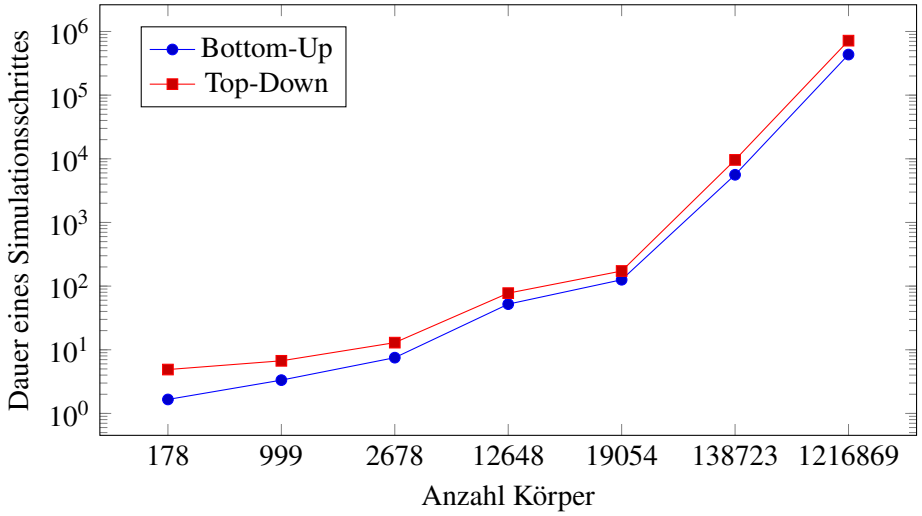

Abbildung 5.4: Vergleich der Laufzeiten des Bottom-Up Algorithmus und des Top-Down Algorithmus bei verschieden großen Datensätzen und gleichbleibendem Theta von 0. Zu sehen ist die durchschnittliche Dauer eines einzelnen Simulationsschrittes in Millisekunden im Verhältnis zur Anzahl der Körper im Datensatz. Hier ist der Bottom-Up Algorithmus durchgehend etwas schneller als der Top-Down Algorithmus.

In Abbildung 5.4 wird nun die Auswirkung eines anderen Theta-Werts auf die Laufzeit des Bottom-Up Algorithmus mit der Auswirkung, die es auf den Top-Down Algorithmus hat, verglichen. Zu sehen ist die durchschnittliche Dauer eines einzelnen Simulationsschrittes in Millisekunden im Verhältnis zur Größe des verwendeten Datensatzes, diesmal bei einem Theta von 0. Ein Theta von 0 macht in der Praxis keinen Sinn, da man so den zusätzlichen Aufwand für das Erstellen und Traversieren einer Baumstruktur ohne jegliche Vorteile hat. Jedoch können diese Laufzeiten hier einen weiteren Einblick in das Verhalten des Bottom-Up Algorithmus im Vergleich zum Top-Down Algorithmus geben. Da bei einem Theta von 0 nie ein Pseudoknoten für eine Berechnung verwendet wird, werden immer alle Berechnungen für alle Körper einzeln durchgeführt. Das hat zur Folge, dass man weiß, dass sowohl der Top-Down Algorithmus als auch der Bottom-Up Algorithmus bei der Beschleunigungsberechnung immer genau gleich viele Berechnungen durchführen. Da der Bottom-Up Algorithmus in Abbildung 5.4 bei allen Datensätzen kontinuierlich schneller ist als Top-Down, kann man daraus also schlussfolgern, dass er die selbe Anzahl an Berechnungen in einem

kürzeren Zeitraum durchführen kann. Das bedeutet, dass der Bottom-Up Algorithmus während der Beschleunigungsberechnung durch die vorteilhaftere Struktur des Baumes tatsächlich Zeit sparen kann. Von der Baumerstellung kann die Zeitersparnis nicht kommen, da sie vom Theta unabhängig ist und wir bereits festgestellt haben, dass Bottom-Up zumindest für große Datensätze hier langsamer ist als Top-Down. Mit dem Ergebnis aus Abbildung 5.4 können wir also schlussfolgern, dass der Bottom-Up Algorithmus durchaus das Potential hat, schneller als der Top-Down Algorithmus zu sein. Dafür müsste er aber wenigstens ungefähr gleich viele Knoten bearbeiten wie Top-Down, und zwar auch bei einem Theta größer als 0.

Mit Theta gleich 1 wurden ebenfalls Daten der Algorithmen erhoben, jedoch ähnelt das Ergebnis stark dem Ergebnis bei Theta 0,6. Bottom-Up ist schenller bei kleinen Datensätzen, skaliert dann aber schlecht und verliert schließlich den Anschluss zu Top-Down. Die Erkenntnis, die man hieraus mitnehmen kann, ist, dass Bottom-Up von größeren Theta nicht merklich mehr profitiert als Top-Down. Somit ist dies keine Möglichkeit, den Bottom-Up Algorithmus konkurrenzfähiger zu machen. Es wurden ebenfalls Vergleiche der Laufzeit mit verschieden großen Datensätzen relativ zu einem Theta von 0 angestellt, um den Effekt eines größeren Theta auf die Laufzeit bei unterschiedlichen Datensätzen sowohl Top-Down als auch Bottom-Up zu untersuchen. Die Ergebnisse haben jedoch nur die bisherigen Erkenntnisse bestätigt, dass Bottom-Up bei kleinen Datensätzen schneller, bei mittleren Datensätzen in etwa gleich schnell und bei großen Datensätzen deutlich langsamer ist als Top-Down. Es wurde ebenfalls bestätigt, das über diese prinzipiellen Laufzeitvorteile hinaus keiner der beiden Algoritmen mehr von bestimmten Theta-Einstellungen profitiert als der andere.

5.2.3 Anzahl an Kindknoten

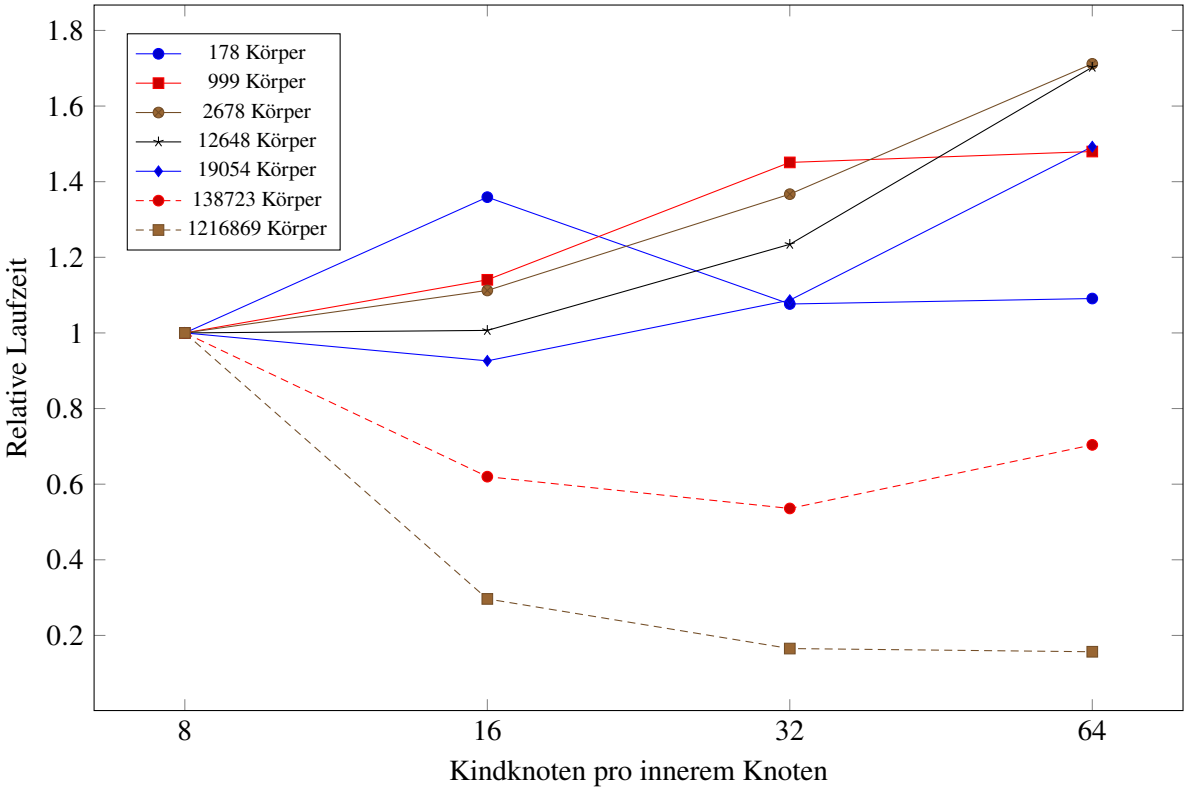

Abbildung 5.5: Vergleich der Laufzeiten des Bottom-Up Algorithmus bei unterschiedlicher Anzahl an Kindern pro Knoten und verschieden großen Datensätzen, bei gleichbleibendem Theta von 0,6. Die Laufzeiten werden relativ zur Laufzeit mit dem jeweiligen Datensatz und 8 Kindknoten pro innerem Knoten angegeben. Zur Bestimmung der Laufzeit dient dabei die durchschnittliche Dauer eines Simulationsschrittes. Während bei kleinen bis mittleren Datensätzen eine größere Anzahl an Kindknoten eine Verlangsamung bewirkt, kann sie bei großen Datensätzen die Laufzeit auf ein Fünftel reduzieren.

In Abbildung 5.5 wird die Laufzeit des Bottom-Up Algorithmus in ein Verhältnis zu der Anzahl an Kindknoten pro innerem Knoten gesetzt, wobei ein Graph jeweils die Entwicklung der Laufzeit für einen bestimmten Datensatz wiederspiegelt. Die Laufzeit wird relativ zur Laufzeit bei 8 Kindknoten angegeben. Die zugrundeliegenden Daten zur Bestimmung der Laufzeit sind die durchschnittliche Dauer eines einzelnen Simulationsschritts (ein Simulationsschritt entspricht einer Stunde in der Simulation), wobei die Simulation bei den Datensätzen bis 2678 Körpern 30 Tage, bei den Datensätzen bis 19054 Körpern 7 Tage und bei allen größeren Datensätzen einen Tag simulierte, um einen möglichst guten Durchschnittswert bei überschaubarer Gesamtlaufzeit zu erreichen.

Aus Abbildung 5.5 lässt sich nun die optimale Anzahl an Kindknoten für die verschieden großen Datensätze ablesen. Für die Datensätze mit 178 und 999 Körpern folgen die Laufzeiten nicht ganz demselben Trend wie bei den übrigen Datensätzen, was sich aber auf die geringe absolute Dauer der Simulationsschritte zurückführen lässt (1 bis 3 Millisekunden). Die Laufzeiten können also

im Laufe der Simulation stark von der Lage einiger Körper und von allgemeinen Schwankungen beeinflusst werden. Ansonsten ist der Algorithmus bei den Datensätzen bis 2678 Körpern mit 8 Kindknoten deutlich am schnellsten, und eine größere Anzahl an Kindknoten erzeugt ausschließlich eine Verschlechterung der Laufzeit (bis zu 70% langsamer). Bei dem Datensatz mit 12648 Körpern wird ein Wendepunkt erreicht, da sich hier die Laufzeit sowohl mit 8 als auch mit 16 Kindknoten auf beinahe demselben Niveau befindet. Für den Datensatz mit 19054 Körpern ist es, wenn auch nur leicht, von Vorteil, den Algorithmus mit 16 Kindknoten pro innerem Knoten auszuführen. An den Laufzeiten des Algorithmus mit dem Datensatz mit 138723 Knoten kann man nun gut erkennen, wohin sich der Trend bewegt. Die Durchläufe mit mehr als 8 Kindknoten sind um bis zu 40% schneller, und auch wenn die beste Laufzeit bei 32 Kindknoten erreicht wird, ist auch die Simulation mit 64 Kindknoten immer noch deutlich schneller als die mit den standardmäßigen 8 Kindknoten. Man kann hieraus also schließen, dass für noch größere Datensätze (mit ähnlicher räumlicher Verteilung der Körper) der Algorithmus mit noch mehr Kindknoten pro innerem Knoten seine beste Laufzeit erzielen würde. Die Erhöhung der Anzahl der Kindknoten ist also ein adäquates Mittel, um den Bottom-Up Barnes-Hut Algorithmus besser skalieren zu lassen.

Der Grund für diese Beobachtungen liegt genauso wie der Ursprung der schlechten Skalierung des Bottom-Up Algorithmus in der Beschleunigungsberechnung. Wir haben bereits festgestellt, dass der Bottom-Up Algorithmus so langsam ist, weil er beim Berechnen der Beschleunigungen ein Vielfaches an Berechnungen durchführt. Ein Grund hierfür war, dass die Anzahl der Kindknoten immer starr auf 8 festgesetzt ist. Somit hatte ein innerer Knoten auch 8 Kindknoten, wenn es keine 8 dicht zusammenliegenden Körper gab. Dadurch konnte die Anzahl der notwendigen Berechnungen nicht so effektiv reduziert werden. Erhöht man nun die Anzahl der Kindknoten auf 64, wird dieser Fall zwar deutlich verschärft, jedoch können die Fälle, in denen eine große Menge von Körpern dicht zusammen liegt, viel mehr Zeit einsparen, da immer 64 Berechnungen auf einmal eingespart werden können. Dazu passt auch, dass diese Optimierung besonders erfolgreich bei den größeren Datensätzen ist. Dies hat nicht zwingend etwas mit der reinen Anzahl von Körpern zu tun, sondern vor allem mit de Dichte. Die Körper aller Datensätze, die in diesen Testläufen betrachtet werden, sind immer über den gleichen Bereich verteilt. Es folgt also logischerweise, dass die Dichte an Körpern beim Datensatz mit über einer Million Körper viel höher ist als beim Datensatz mit 1000. Vor allem befinden sich die meisten der Million Körper in einem Asteroidengürtel, also nochmal in einem Bruchteil des Bereichs, der vom gesamten Datensatz abgedeckt wird. Das heißt, auch wenn es mit 64 Kindknoten pro innerem Knoten viele Pseudoknoten gibt, die nie in der Beschleunigungsberechnung verwendet werden, kann in den dichtesten Regionen des Datensatzes die deshalb verlorene Zeit mühelos wieder eingespart werden. Die Werte zur Anzahl bearbeiteter Knoten und zur durchschnittlichen Kantenlänge bestätigen dies. Im Gegnsatz zu den 120.000 Körpern, die beim größten Datensatz vom Bottom-Up Algorithmus mit 8 Kindknoten bearbeitet werden, sind es mit 64 Kindknoten nur noch rund 30.000 Körper. Die durchschnittliche Kantenlänge der inneren Knoten hat sich beinahe halbiert und liegt mit 64 Kindknoten nur noch bei 0,54. Diese Werte sind zwar immer noch deutlich schlechter als beim Top-Down Algorithmus, zeigen aber trotzdem ein Potential zur Verbesserung des Bottom-Up Algorithmus, vor allem für bestimmt geartete Datensätze, auf.

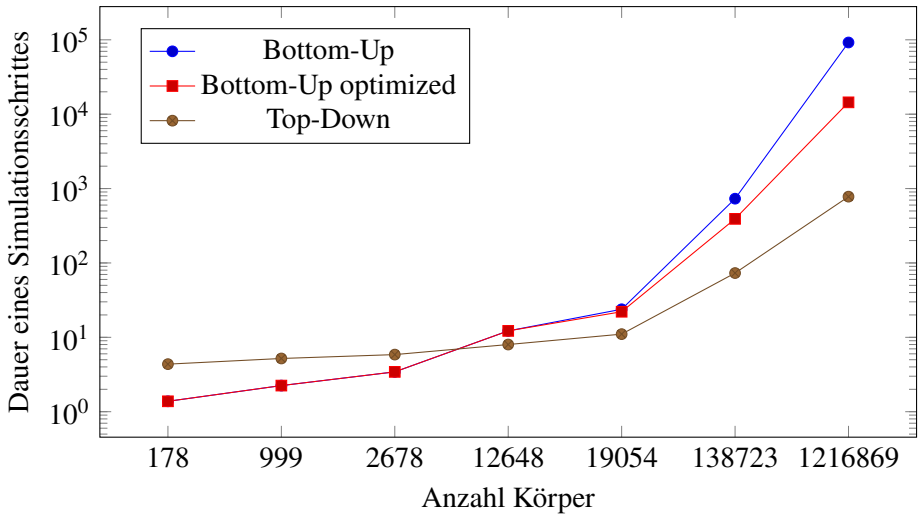

Abbildung 5.6: Vergleich der Laufzeiten des Bottom-Up Algorithmus, des Top-Down Algorithmus und des Bottom-Up Algorithmus mit optimaler Anzahl an Kindknoten bei verschieden großen Datensätzen und gleichbleibendem Theta von 0,6. Konkret sind das 8 Kindknoten für die Datensätze mit 178, 999, 2678 und 12648 Körpern, 16 Kindknoten für den Datensatz mit 19054 Körpern, 32 Kindknoten für den Datensatz mit 138723 Körpern und 64 Kindknoten für den Datensatz mit 1216869 Körpern. Zu sehen ist die durchschnittliche Dauer eines einzelnen Simulationsschrittes in Millisekunden im Verhältnis zur Anzahl der Körper im Datensatz. Die Erhöhung der Anzahl der Kindknoten kann Bottom-Up nicht ausreichend beschleunigen, um vergleichbar so gut zu skalieren wie Top-Down.

In Abbildung 5.6 wird der nun beschleunigte Bottom-Up Algorithmus mit dem vorherigen Zustand und dem Top-Down Algorithmus verglichen. Der verbesserte Bottom-Up Algorithmus verwendet für jeden Datensatz die optimale Anzahl an Kindknoten, also 8 Kindknoten für die Datensätze mit 178, 999, 2678 und 12648 Körpern, 16 Kindknoten für den Datensatz mit 19054 Körpern, 32 Kindknoten für den Datensatz mit 138723 Körpern und 64 Kindknoten für den Datensatz mit 1216869 Körpern. Die Laufzeiten beziehen sich wie immer auf die durchschnittliche Laufzeit eines einzelnen Simulationsschritts in Millisekunden, bei einem Theta von 0,6. Man kann gut erkennen, dass der optimierte Bottom-Up Algorithmus für große Datensätze eine beträchtliche Beschleunigung erfahren hat, jedoch wird das schenll von dem immer noch riesigen Zeitabstand, den auch der optimierte Bottom-Up Algorithmus zum Top-Down Algorithmus hat, relativiert. Um konkrete Zahlen zu nennen, mit dem größten Datensatz und 64 Kindknoten benötigt der Bottom-Up Algorithmus durchschnittlich 15.000 Millisekunden pro Simulationsschritt, was zwar mehr als 5 mal schneller ist als mit 8 Kindknoten, jedoch benötigt der Top-Down Algoritmus mit dem gleichen Theta und dem gleichen Datensatz lediglich 700 Millisekunden pro Simulationsschritt. Es scheint sehr unwahrscheinlich, dass man diese Lücke ohne weiteres schließen kann, jedoch ist das Potential der Optimierung mit den Kindknoten noch nicht voll ausgeschöpft. Im aktuellen Stand des Programms wird die Erhöhung der Anzahl der Kindknoten so umgesetzt, dass die Schleifen, die vorher über 8 Knoten iterierten, nun über eine beliebige Anzahl an Kindknoten iterieren können. Dies hat den bereits erwähnten Vorteil der weiteren Reduzierung der Anzahl der Berechnungen, jedoch verlangsamen Schleifen über 64 Elemente natürlich die Ausführung des

Programms für jedes einzelne Work-Item. Dies gilt sowohl für die Erstellung des Baumes als auch für die Beschleunigungsberechnung, da beide im Moment noch schleifen über die Kindknoten enthalten. Es wäre interessant, sich über einen Ansatz Gedanken zu machen, der entweder die Anzahl der Schleifen reduziert oder eine parallele Ausführung der Schleifen ermöglicht, falls genügend Work-Items zur Verfügung stehen. Dies könnte die Laufzeit noch einmal beträchtlich senken, falls eine weitere Senkung der Anzahl der Berechnung nicht in genügendem Maße erfolgen kann.

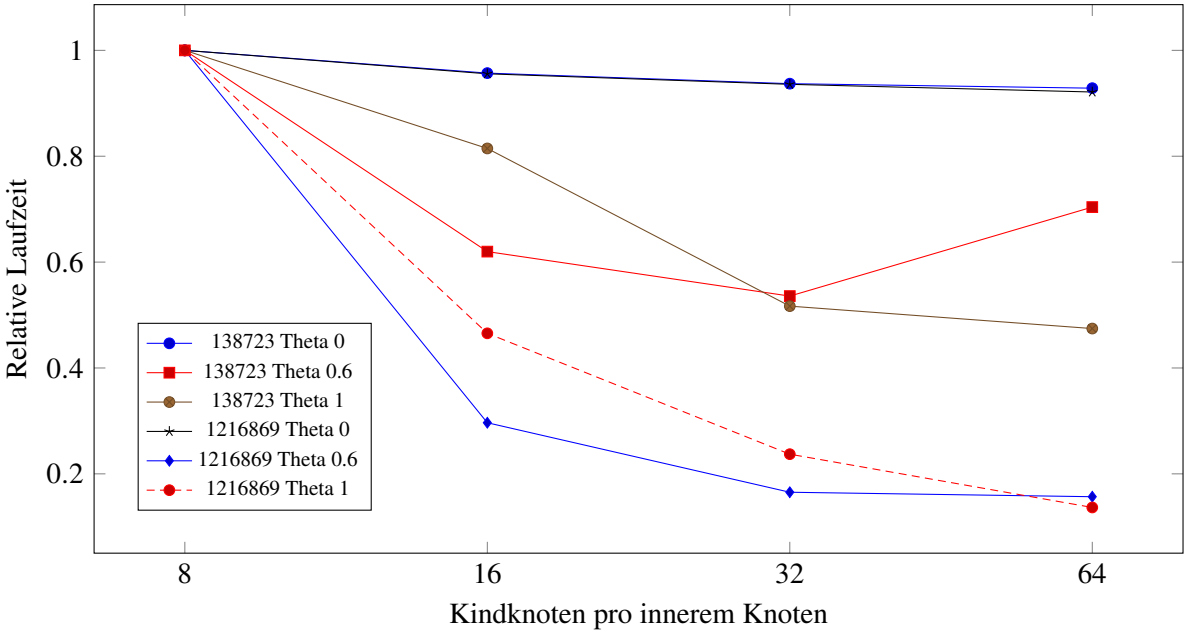

Abbildung 5.7: Vergleich der Laufzeiten des Bottom-Up Algorithmus bei unterschiedlicher Anzahl an Kindern pro Knoten und 2 verschieden großen Datensätzen, mit diversen Thetas. Die Laufzeiten werden relativ zur Laufzeit mit dem jeweiligen Datensatz, dem jeweiligen Theta und 8 Kindknoten pro innerem Knoten angegeben. Zur Bestimmung der Laufzeit dient dabei die durchschnittliche Dauer eines Simulationsschrittes. Bei einem Theta von 0 wirkt sich die Erhöhung der Anzahl der Kindknoten fast gar nicht auf die Laufzeit aus. Bei einem Theta von 1 ist die Verringerung der Laufzeit vergleichbar wie bei einem Theta von 0,6.

Mit Abbildung 5.7 soll die optimierte Variante noch genauer auf ihre Performance mit verschiedenen Parametern untersucht werden. Zu sehen sind die Laufzeiten des Bottom-Up Algorithmus mit den zwei größten Datensätzen und verschieden vielen Kindknoten. Dabei wird nun aber auch der Parameter Theta manipuliert. Die Laufzeiten werden immer relativ zum Bottom-Up Algorithmus mit dem jeweiligen Datensatz und dem jeweiligen Theta, aber nur 8 Kindknoten angegeben. Die Laufzeiten beziehen sich wie immer auf die durchschnittliche Dauer eines einzelnen Simulationsschrittes. Mit Abbildung 5.7 kann untersucht werden, durch welchen Effekt genau die Erhöhung der Anzahl der Kindknoten eine deutliche Verringerung der Laufzeit erreicht. Während bei einem Theta von 0,6 und 1 die Laufzeiten mit der Erhöhung der Anzahl von Kindknoten so reagieren wie erwartet, wird bei einem Theta von 0 durch diese Optimierung kaum Zeit gespart. Das lässt darauf schließen, dass der Hauptgrund für die Zeitersparnis tatsächlich die Reduzierung in der

Anzahl der Berechnungen bei der Beschleunigungsberechnung ist, und nicht eine anderweitige beschleunigung des Codes durch mehr Kindknoten. Da bei einem Theta von 0 immer für alle Körper die Beschleunigungsberechnungen durchgeführt werden, kann die Anzahl der Berechnungen auch nicht durch etwaige Optimierungen verringert werden. Da bei einem Theta von 0 mehr Kindknoten fast gar keine Zeitersparnis im Vergleich zu den anderen Thetas bringen, kann man ziemlich sicher ausschließen, dass die höhere Anzahl an Kindknoten noch positive Effekte an anderen Stellen des Programms hat. Man kann aber auch beobachten, dass bei einem Theta von 1 die Zeitersparnis mit einer großen Zahl von Kindknoten noch einmal verbessert scheint. Dies kann man damit erklären, dass bei einem Theta von 0,6 noch nicht alle Pseudoknoten, die 64 dicht beieinander liegende Körper als Kindknoten haben, in vereinfachter Form für die Beschleunigungsberechnung verwendet werden konnten. Falls durch die Erhöhung des Thetas noch einmal einige solcher Pseudoknoten dazukommen, hat das durch die große Zahl an Kindknoten pro Pseudoknoten recht schnell eine große Verringerung der Laufzeit zur Folge. Bei dem Datensatz mit 1216869 Körpern scheint das Potential für Vereinfachung jedoch auch schon bei einem Theta von 0,6 relativ ausgeschöpft zu sein, sodass sich die relative Zeitersparnis bei einem Theta von 1 im gleichen Bereich bewegt. Dies ist nochmal ein Beleg dafür, dass die Effektivität dieser Optimierung und auch des Bottom-Up Barnes-Hut Algorithmus an sich stark von der Art und Form der verwendeten Datensätze abhängt.

5.2.4 Verschiedene Systeme

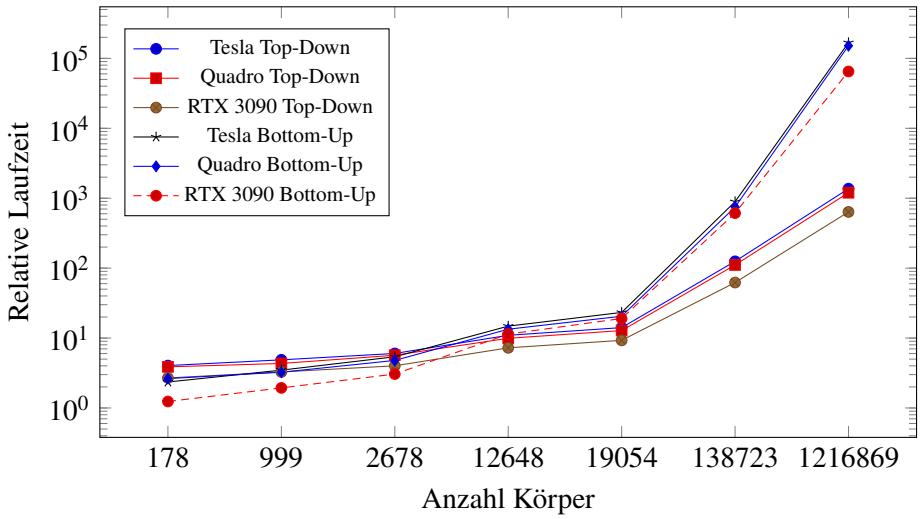

Abbildung 5.8: Vergleich der Laufzeiten des Top-Down und des Bottom-Up Algorithmus auf verschiedenen Grafikkarten, bei verschieden großen Datensätzen und gleichbleibendem Theta von 0,6. Die Laufzeiten in Millisekunden angegeben. Zur Bestimmung der Laufzeit dient dabei die durchschnittliche Dauer eines Simulationsschrittes. Im Allgemeinen kann man für alle Grafikkarten das jeweils erwartete Verhalten für die beiden Algorithmen beobachten. Etwas überraschend ist dabei, dass die Tesla und Quadro schlechter abschneiden als die RTX 3090, obwohl sie eigentlich eine deutlich bessere Rechenleistung für Fließkommarechnungen bieten.

In Abbildung 5.8 werden die Laufzeiten sowohl des Top-Down Algorithmus als auch des Bottom-Up Algorithmus auf verschiedenen Grafikkarten veranschaulicht. Die Laufzeiten beziehen sich dabei auf die durchschnittliche Laufzeit eines einzelnen Simulationsschrittes in Millisekunden. Die Ergebnisse wurden mit einem Theta von 0,6 erzielt. Für den Bottom-Up Algorithmus wurden hier zunächst immer 8 Kindknoten pro innerem Knoten verwendet. Als erstes wird deutlich, dass die RTX 3090 sowohl für den Top-Down als auch für den Bottom-Up Algorithmus am besten geeignet ist. Sie ist um eine deutliche Marge schneller als die Quadro und Tesla, welche etwas näher zusammenliegen. Hier liegt jedoch die Quadro durchweg vorne. Das Bottom-Up für kleine Datensätze schneller ist, aber dann schlechter skaliert, ist auch auf diesen Grafikkarten der Fall. Dass die Tesla und Quadro sich als langsamer als normale Verbrauchergrafikkarten herausstellen, ist bemerkenswert. Diese Grafikkarten sind auf Fließkommarechnungen ausgelegt und bieten hierfür deutlich mehr Kapazität als die Verbrauchergrafikkarten. Auch wenn bei der N-Körper Simulation eine Vielzahl von Fließkommaberechnungen ausgeführt werden, scheint das bei diesen Algorithmen nicht der entscheidende Faktor über die Laufzeit zu sein.

Ein möglicher Grund hierfür könnte der jeweilige Speicher der Grafikkarten sein. Ein Punkt, wo die RTX 3090 und die RTX 3080, die übrigens auch schneller als die Tesla und die Quadro ist, einen Vorteil haben, ist bei der Speichergeschwindigkeit. Die Übertragungsgeschwindigkeit in der Spitze ist um ein Vielfaches höher beim GDDR6X Speicher der RTX Grafikkarten als beim HBM2 Speicher der Tesla und Quadro. Der Unterschied der Laufzeiten zwischen den Grafikkarten ist zwar nicht genauso groß, aber man kann vermuten, dass zumindest in regelmäßigen Abständen von beiden Algorithmen auf eine Art und Weise zugegriffen wird, die beim HBM2 Speicher immer wieder zu einer Verzögerung im Vergleich zum GDDR6X Speicher führt. Das lässt noch einmal den Punkt aufkommen, dass die Speicherverwaltung der Algorithmen vielleicht auch optimierbar ist und zu einer besseren Performance vor allem auf spezialisierten Rechengrafikkarten führen könnte.

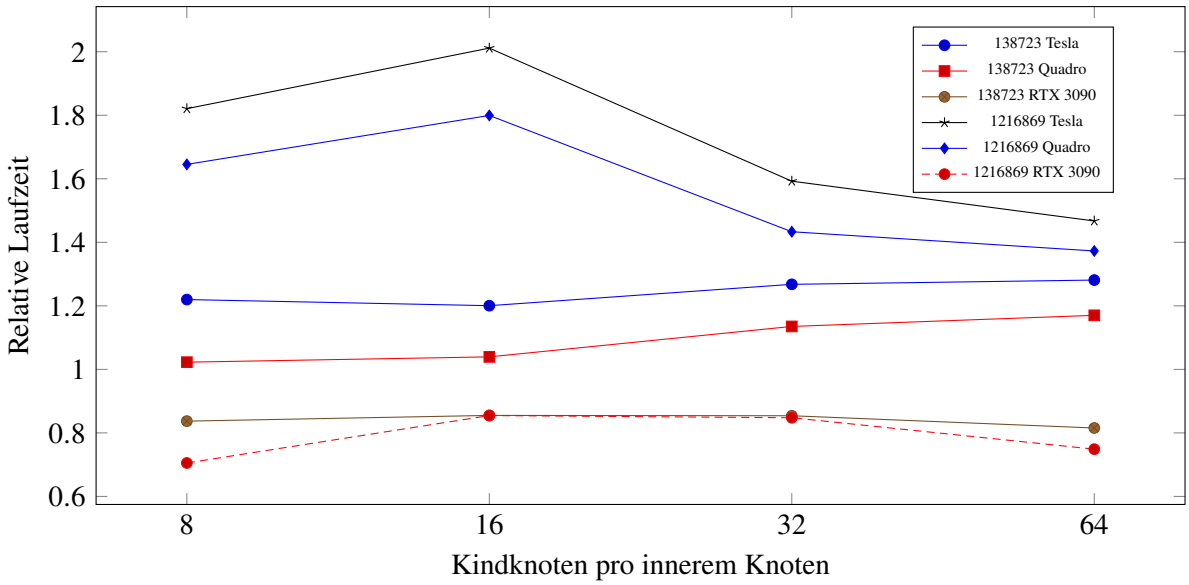

Abbildung 5.9: Vergleich der Laufzeiten des Bottom-Up Algorithmus bei unterschiedlicher Anzahl an Kindern pro Knoten und zwei verschieden großen Datensätzen, bei gleichbleibendem Theta von 0,6. Die Laufzeiten werden relativ zur Laufzeit mit dem jeweiligen Datensatz auf einer RTX 3080 angegeben. Zur Bestimmung der Laufzeit dient dabei die durchschnittliche Dauer eines Simulationsschrittes. Die RTX 3090 skaliert mit einer größeren Anzahl von Kindknoten praktisch identisch zur RTX 3080. Für die Tesla und Quadro gilt bei mittelgroßen Datensätzen das Gleiche. Bei großen Datensätzen profitieren sie mehr von einer größeren Anzahl an Kindknoten als eine RTX 3080, bleiben jedoch ungefähr 50% langsamer.

In Abbildung 5.9 wird nun zuletzt noch die Auswirkung der Optimierung mit mehreren Kindknoten pro innerem Knoten auf verschiedene Grafikkarten untersucht. Die Laufzeiten beziehen sich auf den Bottom-Up Algorithmus bei den zwei größten Datensätzen, verschieden vielen Kindknoten und einem Theta von 0,6. Die Laufzeiten sind relativ zu den entsprechenden Laufzeiten mit dem gleichen Datensatz und der selben Anzahl an Kindknoten auf einer RTX 3080. Wie zuvor bereits gesehen, bestätigt sich hier nochmal, dass die Verbrauchergrafikkarten der RTX Reihe schneller sind als die Quadro und Tesla, wobei die RTX 3090 noch einmal um durchschnittlich 20% schneller ist als die RTX 3080. Was die Skalierung der Laufzeiten mit den Anzahlen an Kindknoten angeht, sind die Grafikkarten hier größtenteils gleichauf. Sie haben relativ zueinander immer die gleichen Geschwindigkeitsabstände für einen bestimmten Satz von Parametern. Was jedoch zu beobachten ist, ist dass die Quadro und Tesla Grafikkarten Schwierigkeiten mit dem größten Datensatz bei 8 und 16 Kindknoten pro innerem Knoten haben. Hier sind sie deutlich langsamer als sonst, die Tesla bis zu 2 mal langsamer als die RTX 3080. Geht man davon aus, dass diese Anomalie ebenfalls vom Speicher dieser Grafikkarten ausgelöst wird, würde das bedeuten, dass die problematischen Speicherzugriffe durch eine größere Anzahl an Kindknoten verringert werden können, weshalb die Quadro und Tesla mit 32 und 64 Kindknoten beim größten Datensatz überdurchschnittlich viel aufholen. Das könnte daran liegen, dass mit 64 Kindknoten die zu bearbeitenden Kindknoten bei der Beschleunigungsberechnung in größeren Gruppen geladen und

dann abgearbeitet werden können, was diesen Grafikkarten wohl mehr liegt als kleinere, regelmäßige Speicherzugriffe. Die Verbesserung reicht jedoch nicht aus, um die eigentlich bessere Performance bei der Fließkommaberechnung wirklich zur Geltung kommen zu lassen.

5.3 Fazit

Insgesamt sind die Laufzeiten ein ernüchterndes Ergebnis für den Bottom-Up Ansatz. Für große Datensätze scheint der Algorithmus prinzipbedingt in allen Teilen des Programms langsamer zu sein, sei es bei der Baumerstellung wegen der langen Laufzeit des Sortieralgorithmus, oder bei der Beschleunigungsberechnung aufgrund von viel mehr notwendigen Berechnungen als beim Top-Down Algorithmus, hauptsächlich verschuldet durch die nicht optimale Vprortierung der Körper mittels der Hilbert-Kurve, und die starre Vorgabe, dass jeder innere Knoten gleich viele Kindknoten haben muss, die zur Gruppierung von Körpern führt, die nicht wirklich dicht beieinander liegen. Einige Lichtblicke sind zwar die bessere Performance bei kleinen Datensätzen, jedoch handelt es sich hierbei um kein wirkliches Anwendungsgebiet, und dass Bottom-Up mehr Knoten pro Zeit abarbeiten kann ist nicht besonders nützlich, da die Anzahl an notwendigen Berechnungen die von Top-Down um mehr als eine Größenordnung übersteigt. Es gibt zwar einige Optimierungsmöglichkeiten, wie die Anzahl von Kindknoten pro innerem Knoten zu erhöhen, diese Optimierungen hatten auch Erfolg und haben die Laufzeit des Algorithmus deutlich reduziert, jedoch war es bei weitem nicht ausreichend, um mit dem Top-Down Algorithmus zu konkurrieren. Um mit Top-Down mithalten zu können, müssten deutlich bessere Methoden zur Vorsortierung der Körper und vielleicht auch zur Strukturierung des Baums ausgemacht werden, um nicht pauschal eine feste Anzahl an Körpern zu gruppieren, unabhängig von ihrer Lage zueinander. Eine weitere Erkenntnis ist, dass der Bottom-Up Algorithmus je nach Begebenheit der verwendeten Datensätze stark in seiner Performanz schwanken kann. So ist der Grund für die Zeitersparnis durch mehr Kindknoten nicht zuletzt die in den großen Datensätzen dicht mit Körpern besetzten Regionen. Das gleiche gilt für die Sortierung der Körper mit der Hilbert-Kurve, sie ist situationsabhängig und muss für jeden Anwendungsfall angepasst werden. Deswegen ist es vermutlich schwer, eine Lösung für den Bottom-Up Barnes-Hut Algoritmus zu finden, die ihn allgemein leistungsfähig machen wird.

6 Zusammenfassung

In dieser Arbeit wurde untersucht, ob die Erstellung eine Barnes-Hut Baumes im Bottom-Up Prinzip durch die theoretisch höhere Parallelität und den dadurch entstehenden, besser ausbalancierten Baum Zeit sowohl bei der Baumgenerierung als auch bei der Beschleunigungsberechnung einsparen könnte. Die Erstellung des Baumes komplett entgegen der ursrpünglichen Idee des Barnes-Hut Algorithmus erweißt sich jedoch als nicht trivial und bringt viele Tücken mit sich. Da die eigentliche Methode zum Einsortieren von Körpern in einen Barnes-Hut Baum entfällt, musste eine provisorische Sortierung der Körper durchgeführt werden, um vorneweg ohne Hilfe des Baumes an sich eine näherungsweise räumlich sortierte Liste der Körper zu erstellen. Diese Sortiermethode erwieß sich jedoch als deutlich schlechter als die eigentliche Barnes-Hut Methode des Top-Down Algorithmus. Die Struktur des Baumes ist auch nicht mehr so gut an die Beschaffenheit des betrachteten Datensatzes angepasst, weshalb die Leistung des Algorithmus in verschiedenen Anwendungen stärker schwankt. Vor allem aber ist die Anzahl der benötigten Berechnungen zur Beschleunigungsberechnung stark gestiegen, was auf die zwei eben genannten Punkte zurückzuführen ist. Der Barnes-Hut Algorithmus ist daher Bottom-Up im Moment nicht konkurrenzfähig.

Ausblick

Es gibt einige Überlegungen, wie man auf dem jetzigen Stand aufbauen könnte, zum Beispiel durch einen anderen Ansatz zur Vorsortierung der Körper, eine weitere Untersuchung der Funktionalität mit mehr als 8 Kindknoten, wo man unter anderem den Code neu strukturieren könnte, um die Anzahl an Schleifen zu verringern, oder eine Möglichkeit schaffen, die Schleifen zu parallelisieren. Dadurch könnte die Laufzeit des Algorithmus, zumindest bei dichten Datensätzen, noch einmal verringert werden. Der Abstand zum Top-Down Algorithmus ist aber immer noch groß. Eine weitere mögliche Überlegung wäre, eine neue Metrik als die Kantenlängen der inneren Knoten dafür zu verwenden, zu entscheiden ob ein Pseudoknoten für eine Berechnung verwendet wird oder nicht. Man könnte argumentieren, dass sich der Baum in seiner Struktur nun so stark vom ursprünglichen Barnes-Hut baum unterscheidet, dass die Metrik nicht mehr geeignet ist. Das würde auch nochmal umständliche Berechnungen aus dem Code entfernen, die nur durchgeführt werden, um diese Metrik zu erhalten, da sie ja nicht mehr einfach von den Elternknoten erschlossen werden kann. Dabei muss man dann aber aufpassen, dass man nicht zu sehr die Bedeutung des Parameters Theta aufweicht, da man sonst natürlich eine Laufzeitverbesserung erreichen könnte, aber auf Kosten der Präzision.

Literaturverzeichnis

[BH86] J. Barnes, P. Hut. „A hierarchical O (N log N) force-calculation algorithm“. In: *nature* 324.6096 (1986), S. 446–449 (zitiert auf S. 9, 14).

[FG23] Y. Faqir-Rhazoui, C. Garcia. „Exploring Heterogeneous Computing Environments: A Preliminary Analysis of Python and SYCL Performance“. In: *Conference on Cloud Computing, Big Data & Emerging Topics*. Springer. 2023, S. 3–16 (zitiert auf S. 11).

[NHP07] L. Nylons, M. Harris, J. Prins. „Fast n-body simulation with cuda“. In: *GPU gems* 3 (2007), S. 62–66 (zitiert auf S. 11).

[SHT+95] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, J. Hennessy. „Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-Hut, fast multipole, and radiosity“. In: *Journal of Parallel and Distributed Computing* 27.2 (1995), S. 118–141 (zitiert auf S. 11).

[Thü23] T. Thüring. „Comparison of different n-body algorithms on various hardware platforms using SYCL“. University of Stuttgart, 5. Apr. 2023 (zitiert auf S. 9).

[YB11] R. Yokota, L. A. Barba. „Treecode and fast multipole method for N-body simulation with CUDA“. In: *GPU Computing Gems Emerald Edition*. Elsevier, 2011, S. 113–132 (zitiert auf S. 11).

Alle URLs wurden zuletzt am 04. 10. 2023 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift