Fallstudie Teilchendynamik:

In dieser Fallstudie sollen Sie die in der Vorlesung und den Praktika vermittelten Techniken zur Laufzeitoptimierung auf das Problem einer Teilchensimulation anwenden.

Dazu optimieren Sie in Gruppenarbeit den Programmcode und dokumentieren Ihre Ergebnisse in einer Ausarbeitung. **Spätester Abgabetermin für Ihre Ausarbeitung und Ihres Quellcodes ist der 26.06.2020. Pro Gruppe geben Sie eine Ausarbeitung ab.**

Sie können bis zum **15.06.** eine Vorabversion Ihrer Ausarbeitung und Ihres Codes abgeben, um dazu formatives Feedback zu erhalten. Die Vorabversion wird also nicht bewertet, sondern das Feedback soll Ihnen helfen Ihre Ausarbeitung zu verbessern.

Im Folgenden sind die Arbeitsaufträge zur Optimierung und deren Dokumentation aufgeführt.

Am Ende des Dokuments finden Sie die Bewertungskriterien.

Problembeschreibung:

In der Fallstudie soll eine Simulation von vielen Teilchen, welche über Gravitations- oder Coulombkraft miteinander wechselwirken, optimiert werden. Die Simulation läuft in diskreten Zeitschritten (dt) ab. Zunächst wird für ein Teilchen berechnet, welche Kraft von den anderen Teilchen des Systems auf dieses ausgeübt wird. Anschließend wird daraus die Änderung der Geschwindigkeit bestimmt. Diese Schritte werden für alle Teilchen des Systems durchgeführt, so dass dann die neuen Geschwindigkeiten für alle Teilchen vorliegen.
Mit diesen aktualisierten Geschwindigkeiten wird dann die neue Position der Teilchen nach einem Zeitschritt (dt) berechnet. Danach kann der nächste Zeitschritt ausgeführt werden.
Solche Simulationen werden unter anderem in der Astrophysik zur Berechnung von Galaxie-Entwicklungen und zur Berechnung der Verteilung von dunkler Materie verwendet. Daneben spielen Sie auch eine Rolle in der Simulation von Molekülstrukturen. Der vorliegende Code ist algorithmisch nicht optimal. Die Komplexität des Algorithmus ist $O(n^2)$. In der Praxis werden solche Berechnungen für viele Milliarden Teilchen durchgeführt. Darum werden dann Algorithmen eingesetzt, welche Vereinfachungen vornehmen, so dass sich die Komplexität auf $O(n \log n)$ verringert. Der hier verwendete Algorithmus ist jedoch eine gutes Anwendungsbeispiel, um die verschiedenen Optimierungstechniken zu üben. Sie sollen daher den

verwendeten Algorithmus als gegeben annehmen und dessen Implementierung hinsichtlich der Laufzeit optimieren.
Um die Überprüfung der Korrektheit der Ergebnisse zu vereinfachen wird in der vorliegenden Simulation nur ein Zeitschritt ausgeführt. Dieser allerdings mehrmals, um Schwankungen und Starteffekte in den Laufzeitmessungen durch Mittelung abzuschwächen.

Arbeitsaufträge:

Die Arbeitsaufträge enthalten sowohl Aufgabenteile zur Optimierung als auch zu deren Dokumentation in Ihrer Ausarbeitung. Sowohl die eigentlichen Optimierungen, als auch deren Dokumentation gehen entsprechend dem unten angegebenen Bewertungsschlüssel in die Note der Ausarbeitung ein.

Es reicht aus, wenn Sie die Ergebnisse für eines Ihrer Rechnersysteme angeben und für diese Optimierungen vornehmen.

Verwenden Sie als Fließkommazahl-Modell das IEEE 754 konforme Modell *precise*.

Die Optimierungen bauen aufeinander auf. Führen Sie also die Optimierungen Schritt für Schritt durch und modifizieren Sie, wenn nicht anders angegeben, Ihren bestehenden Code weiter.

Hinweis zu den Laufzeitmessungen:

- Sollten Ihre Messungen erst in späteren Durchläufen stabil werden, können Sie die Anzahl der Messungen und diejenigen, die verworfen werden, selbst abändern. Dokumentieren Sie dies auf jeden Fall in Ihrer Ausarbeitung.
- Wenn sich auch dann keine stabile Laufzeit ergibt, z.B. etwa jede zweite Messung deutlich schlechter ist, mitteln Sie über die besseren Ergebnisse. Dokumentieren Sie auch dies bitte in Ihrer Ausarbeitung.
- Sollte die Laufzeit ggf. im Zuge Ihrer Optimierungen zu kurz sein, um noch eine sinnvolle Messung zu ermöglichen, können Sie auch die Problemgröße anpassen. Dokumentieren Sie auch dies bitte in Ihrer Ausarbeitung und halten Sie Rücksprache mit einem Betreuer.

1. Erstellen einer Baseline

a. Baseline erstellen: Kompilieren Sie den Code mit für die Baseline sinnvollen Compiler-Flags und führen Sie Laufzeitmessungen durch, um eine Baseline der Performance und Laufzeit zu erhalten, mit der Sie Ihre weiteren Ergebnisse vergleichen können.
b. Dokumentieren Sie Ihr Ergebnis in Ihrer Ausarbeitung. Geben Sie den verwendeten Prozessor und Compiler inklusive der verwendeten Flags an. Begründen Sie die Auswahl der verwendeten Compiler-Flags.

2. Testroutine zur Überprüfung der Korrektheit der Ergebnisse

a. Erstellen Sie eine Testroutine mit der Sie die Ergebnisse Ihrer späteren Optimierungen auf Korrektheit überprüfen können.
Hinweise:
   - Die Testroutine muss nicht auf Laufzeit optimiert sein. Sie sollten Sie aber so in Ihr Programm integrieren, dass sie einfach deaktiviert werden kann, um so auch Laufzeitmessungen ohne Überprüfung vornehmen zu können.
     In späteren Aufgabenteilen sollen Sie die Problemgröße (Anzahl der Teilchen) über den Parameter *`nrOfParticles`* erhöhen. Da sich dadurch der Code nicht ändert, brauchen Sie dann eine ggf. zeitaufwändige Überprüfung nicht vorzunehmen.
   - Für Korrektheit der Ergebnisse reicht es, wenn die Koordinaten und Geschwindigkeiten der Teilchen in numerischen Grenzen mit der Referenzimplementierung übereinstimmen. In der vorliegenden Anwendung bedeutet dies, dass der Betrag der Differenz jeweils kleiner als absolut 0.001 oder 0.1% des jeweiligen Referenzwertes sein muss.

b. Erklären Sie die Arbeitsweise Ihrer Testroutine und beschreiben Sie mögliche alternative Vorgehen. Begründen Sie, warum Sie sich für den gewählten Ansatz entschieden haben.

3. Skalare Optimierung

a. Führen Sie eine Strength-Reduction der Berechnungen in der Funktion `MoveParticles()` durch, indem Sie aufwändige Berechnungen durch weniger aufwändige ersetzen. Überprüfen Sie auch die Datentypen der verwendeten Zahlkonstanten und passen Sie diese ggf. an um die Performance zu optimieren.
   Überprüfen Sie die Korrektheit Ihrer Ergebnisse mit Ihrer Testroutine. Kommentieren Sie die Änderungen im Quellcode und fügen Sie diesen Ihrer Abgabe hinzu.
b. Führen Sie eine Laufzeitmessung durch und dokumentieren Sie in Ihrer Ausarbeitung die Veränderung der Laufzeit und der Performance in GFLOPs.

4. Vektorisierung I

a. Überprüfen Sie den Vektorisierungs-Report. Die Schleife in `MoveParticles()` über den index **`j`** wird wegen Datenabhängigkeiten nicht vektorisiert. Trennen Sie die Schleife in einen vektorisierbaren und einen nicht-vektorisierbaren Teil auf und vektorisieren Sie so möglichst viele der Berechnungen. Überprüfen Sie den Vektorisierungsreport, ob die Schleife tatsächlich vektorisiert wurde. Passen Sie ggf. die Compiler-Flags an, um eine optimale Performance zu erhalten.
   Überprüfen Sie die Korrektheit Ihrer Ergebnisse mit Ihrer Testroutine. Kommentieren Sie die Änderungen im Quellcode und fügen Sie diesen inklusive der Report-Dateien Ihrer Abgabe hinzu.
b. Führen Sie eine Laufzeitmessung durch und dokumentieren Sie in Ihrer Ausarbeitung die Veränderung der Laufzeit und der Performance in GFLOPs.
   Hinweis: Im nächsten Aufgabenteil werden Sie die Vektorisierung mit entsprechenden pragmas vornehmen. Die Änderungen, die Sie zum Aufteilen der Schleifen vornehmen, verwenden Sie also **nicht** für die weiteren Optimierungsschritte.

5. Vektorisierung II mit Speicheroptimierung

In den folgenden Teilen verwenden Sie bitte Ihren Code von Aufgabenteil 3, also eine Version ohne Vektorisierung in der Schleife über **`j`** aber mit skalarer Optimierung.

a. Fügen Sie ein oder mehrere pragmas ein, welche die Datenabhängigkeiten überschreiben und so zu einer Vektorisierung der Schleife über **`j`** führen. Überprüfen Sie die Korrektheit der Lösung.
b. Führen Sie eine Laufzeitmessung durch und dokumentieren Sie welche Änderungen im Code Sie vorgenommen haben. Sie brauchen dieses Zwischenergebnis nicht hochzuladen. Dokumentieren Sie aber in jedem Fall die Veränderung der Laufzeit und der Performance in GFLOPs.
c. Überprüfen Sie den Vektorreport. In der entsprechenden Schleife kommen nicht optimale Speicherzugriffe (non-unit strided loads) vor. Optimieren Sie die Datenstruktur und das Programm so, dass unit-stride Zugriffe erfolgen.
d. Überprüfen Sie erneut den Vektorreport. Sie sollten jetzt feststellen, dass beim Laden der Vektorregister unaligned-Zugriffe vorkommen, und außerdem Peel-Loops angelegt werden. Sorgen Sie mit optimierter Speicherallokation und entsprechenden Zusicherungen dafür, dass dieser Overhead nicht mehr anfällt.

Hinweis: Auch wenn der Compiler Ihnen hier ggf. Arbeit abnimmt, fügen Sie bitte trotzdem die entsprechenden Zusicherungen in Ihren Code ein.
Überprüfen Sie die Korrektheit der Lösung und fügen Sie Ihren kommentierten Quellcode inklusive der Report-Dateien Ihrer Abgabe hinzu.

e. Führen Sie eine Laufzeitmessung durch und dokumentieren Sie in Ihrer Ausarbeitung die Veränderung der Laufzeit und der Performance in GFLOPs.
f. Beschreiben Sie die Veränderungen am Code, die Sie in Teil c) und d) vorgenommen haben. Begründen Sie warum diese Änderungen zu einer Verbesserung der Laufzeit führen, oder - falls dies bei Ihnen nicht der Fall sein sollte - führen sollten.

6. Thread-Parallelisierung

a. In diesem Arbeitsschritt führen Sie Thread-Parallelisierungen auf mehreren Kernen aus. Führen Sie zunächst eine Parallelisierung der Schleife in `MoveParticles()` über den index **`j`** durch. Nutzen Sie hierbei die maximal mögliche Anzahl an Threads. Stellen Sie mit Ihrer Testroutine sicher, dass Sie korrekte Ergebnisse erhalten und sorgen Sie dafür, dass die Schleife weiterhin auch vektorisiert wird. Kommentieren Sie die Änderungen im Quellcode und fügen Sie diesen Ihrer Abgabe hinzu.
b. Führen Sie eine Laufzeitmessung durch und dokumentieren Sie in Ihrer Ausarbeitung die Veränderung der Laufzeit und der Performance in GFLOPs.
c. Führen Sie nun eine Parallelisierung der Schleife in `MoveParticles()` über den index **`i`** durch und nutzen Sie zunächst alle verfügbaren Threads. Die innere Schleife über den index **`j`**, wird nun nicht mehr parallelisiert sondern wie in Aufgabenteil 5b) vektorisiert.
Stellen Sie mit Ihrer Testroutine sicher, dass Sie korrekte Ergebnisse erhalten und optimieren Sie das Scheduling der Parallelisierung. Kommentieren Sie die Änderungen im Quellcode und fügen Sie diese Ihre Abgabe hinzu.
d. Führen Sie eine Laufzeitmessung durch und dokumentieren Sie in Ihrer Ausarbeitung die Veränderung der Laufzeit und der Performance in GFLOPs. Geben Sie an, welches Scheduling Sie verwendet haben und begründen Sie Ihre Wahl.
e. Führen Sie auch Laufzeitmessungen mit weniger Threads (z.B. 1,2,4) aus und vergleichen Sie die Performance dieser Messungen miteinander. Stellen Sie dar, wie gut Ihre Implementierung nun mit der Anzahl an Threads skaliert. Erstellen Sie dazu ein Speedup-Diagramm. Erklären Sie den sich ergebenden Verlauf des Speedups. Berücksichtigen Sie dabei auch Effekte wie Hyper-threading.

f. Beschreiben und begründen Sie die Unterschiede der Performance der beiden unterschiedlichen Parallelisierungen die Sie in Aufgabenteil a) und c) durchgeführt haben. Erklären Sie wodurch der Performance-Unterschied verursacht wird.

7. Skalierung der Problemgröße

a. Ermitteln Sie die Cache-Größen Ihres Prozessors und berechnen Sie den Speicherbedarf für die bisher verwendete Problemgröße (16384 Teilchen). Typischerweise sollte der Speicherbedarf des Problems kleiner als die L3-Cachegröße sein. Berechnen Sie ab welcher Problemgröße der Speicherbedarf die Cachegröße übersteigt. Stellen Sie Ihre Berechnungen nachvollziehbar in der Ausarbeitung dar.

b. Führen Sie eine Laufzeitmessung mit einer Problemgröße durch, welche sicher nicht mehr in den L3-Cache passt. Die Laufzeit kann dafür relativ groß werden und Sie können die Überprüfung der Ergebnisse deaktivieren, um Laufzeit zu sparen. Dokumentieren Sie Ihre Ergebnisse und vergleichen Sie die erreichte Performance in GFLOPS mit der in Aufgabe 6 erreichten Leistung.

c. Führen Sie ein Loop-Tiling wie in der Vorlesung erklärt durch, um die Anzahl an Cache-Hits bei großen Problemgrößen zu erhöhen. Führen Sie das Tiling für die Schleife mit index **i** durch, d.h. die Schleifenreihenfolge ist dann:

```
for (int ii = 0; …) {
   …
   for (int j = 0; …) {
          …
          for (int i = ii; …) {
          …
          }
          …
   }
   …
}
```

Überprüfen Sie mithilfe Ihrer Testmethode und der ursprünglichen Problemgröße, ob Ihre Implementierung korrekte Ergebnisse liefert. Die Tile-Größe ist ein Tuningparameter. Versuchen Sie diesen zu optimieren.
Kommentieren Sie die Änderungen im Quellcode und fügen Sie diesen Ihrer Abgabe hinzu.

d. Führen Sie erneut eine Laufzeitmessung mit einer Problemgröße durch, welche sicher nicht mehr in den L3-Cache passt. Wie in Teil a) können Sie die Überprüfung der Ergebnisse deaktivieren, um Laufzeit zu sparen. Dokumentieren Sie Ihre Ergebnisse und vergleichen Sie die erreichte Performance in GFLOPS mit der in Teil a) erreichten Leistung. Begründen Sie Ihre Wahl der Tile-Größe. Leiten Sie die Tile-Größe, wenn möglich, aus den Eigenschaften Ihrer Plattform her.

8. Abschluss

a. Stellen Sie die Ergebnisse Ihrer Optimierungen übersichtlich etwa in Form einer Tabelle oder eines Diagramms dar. Kommentieren Sie, wie groß der Einfluss der einzelnen Schritte ist. Stellen Sie Hypothesen auf, welche die jeweiligen Änderungen bezüglich der Performance erklären können.
b. Vergleichen Sie die Ergebnisse der Performance in GFLOP/s (Baseline und der Optimierung mit der besten Performance) mit der von Ihnen berechneten Peak Performance und der maximalen Performance, welche Ihr System im Linpack-Benchmark erreicht hat. Achten Sie darauf, dass Sie die Berechnung der Peak-Performance nachvollziehbar darstellen.
   Kommentieren Sie, wie sich die erreichten Performancewerte zu den theoretischen und im Linpack gemessenen Werten verhalten.

Optionale Zusatzaufgaben:

1. Welche weiteren Optimierungsschritte sehen Sie und welche Laufzeitverbesserungen erwarten Sie von diesen? Setzen Sie diese falls möglich um und bestimmen Sie die Performance.
2. Nutzen Sie MPI, um die Berechnung auf mehrere Prozesse aufzuteilen. Mit MPI wäre dann eine Aufteilung auf mehrere Rechner möglich. Das Datenset darf für alle Prozesse vorliegen, aber die Berechnung der Teilchen sollte auf mehrere Prozesse aufgeteilt werden. Versuchen Sie die Kommunikation zwischen den Prozessen zu minimieren.
   Messen Sie die Performance Ihrer Implementierung.

Bewertungskriterien für die Ausarbeitung:

Ihre Ausarbeitung stellt einen Teil der Prüfungsleistung dar und geht zu 50% in die Benotung des Moduls ein. Die anderen 50% ergeben sich aus der mündlichen Prüfung. Zur Erreichung der maximalen Punktzahl sind die optionalen Aufgabenteile nicht nötig. Mit Ihnen können jedoch andere Teile ausgeglichen werden.

Zur Bewertung wird der folgende Bewertungsmaßstab herangezogen.

A) Technischer Teil (50%) Vollständige Bearbeitung der Optimierungs-Schritte, Qualität des Codes inklusive Kommentierung	-	o	+	++	+++	Gesamt
Baseline (1a)						
Testprogramm (2a)						
Skalare Optimierung (3a)						
Vektorisierung I (4a)						
Vektorisierung II (5a,5c,5d)						
Thread-Parallelisierung (6a,6c)						
Skalierung der Problemgröße (7c)						

B) Beschreibung und Erläuterung der Ergebnisse (30%)	-	o	+	++	+++	Gesamt
Baseline (1b)						
Testprogramm (2b)						
Skalare Optimierung (3b)						
Vektorisierung I (4b)						
Vektorisierung II (5b,5e,5f)						
Thread-Parallelisierung (6b,6d,6e,6f)						
Skalierung der Problemgröße (7a, 7b, 7d)						
Abschluss (8a, 8b)						

C) Darstellungsleistung(20%)	-	o	+	++	+++	Gesamt
Sprache (Ausdrucksweise, Grammatik, Rechtschreibung)						
Layout und Textbild						