Zur Interpretation der Ergebnisse müssen zuallererst Störfaktoren für die Auswertung und ungestörte Ablauf zusammengefasst werden. Hierbei handelt es sich wahrscheinlich primär um zwei Gruppen:

1. Zufällige Zugriffe des Betriebssystems auf Prozessor oder Arbeitsspeicher – dies wäre nur vermeidbar bei einer Ausführung außerhalb eines Betriebssystems, welches im Rahmen der Studienarbeit soweit nicht möglich ist. Die Schwankungen, welche hierdurch auftreten, sollten klein, jedoch nicht vernachlässigbar sein.
2. Zufällige Zugriffe von Benutzeranwendungen auf Prozessor oder Arbeitsspeicher – da auf dem Studiencomputer lediglich TeamViewer parallel aktiv ist und dies einen gleichmäßigen Einfluss auf die Ergebnisse haben, wodurch dies vernachlässigbar sein sollte.

Aus diesen Störfaktoren lässt sich schließen, dass für eine genauere Interpretation der Ergebnisse mit verhältnismäßig hohem Rechenaufwand geeigneter sind, da die Abweichungen sich dort prozentual gesehen weniger bemerkbar machen als Ergebnisse mit wenig Rechenaufwand.

Als allererstes ist zu erkennen, dass bei den Datensätzen „Times1“ und „Times2“ sowohl der Anteil „dynamic“ und „guided“ länger benötigen als „static“ – eine mögliche Erklärung hierbei liegt in der Art wie diese „Schedules“ funktionieren. Während bei „static“ davon ausgegangen wird, dass alle Threads gleichzeitig beenden und dementsprechend wenig Overhead reserviert wird, ist dies bei „dynamic“ und „guided“ nicht der Fall ist. Da der Overhead bei dieser Aufgabe nicht benötigt wird ist die statische Herangehensweise bei dieser Aufgabe somit klar die effizienteste Variante. Ein erwartetes Ergebnis, da die „static-schedule“ für Codeabschnitte mit identisch aufwändigen Iterationen ist, welches hier klar der Fall ist. „Guided“ als leichte Abschwächung von „static“ mit ein wenig Overhead liegt in diesem Fall – wie erwartet – kurz hinter den Ausführungszeiten von „static“. „Dynamic“ liegt am Ende der Messdaten, da „dynamic“ auf nicht identisch aufwändigen Iterationen ausgelegt ist und dieser Typ besonders viel Overhead reserviert, welches hier ineffizient ist und dementsprechend das Programm verlangsamt.

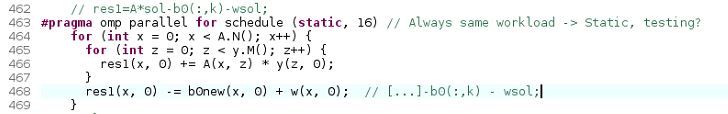
Bei den Visualisierungen mit hohem Zeitaufwand ist zu erkennen, dass bei „static“ und „Guided“, der Effekt einer Erhöhung der Threadanzahl bedeutend stärker ausfällt als die Vergrößerung des Chunks, während bei „dynamic“ die Vergrößerung des Chunks eine stärkere Rolle spielt als bei den anderen beiden „schedules“. Allgemein ist einfach zu verstehen, dass wenn mehr Aufgaben gleichzeitig abgearbeitet werden können (Hervorgerufen durch eine Erhöhung der Anzahl der Threads) das Programm bedeutend schneller läuft. Eine Erhöhung des Chunks verringert die Zeit, welches das Programm versucht Daten aus dem Arbeitsspeicher abzufragen und in den Prozessor zu kopieren, was vor allem bei „schedules“, bei welchen die Threads alle gleichzeitig die Daten abfragen (und dementsprechend wenig Overhead), weniger eine Rolle spielt. Die Begründung hierin liegt in der Arbeitsweise eines Prozessors, da dieser bereits bekannte Aufgaben parallel mit abarbeiten kann und weniger Code geladen werden muss, da die Aufgabe sich nur in kleinen Punkten unterscheidet. Hieraus ergibt sich, dass wenn bei der „dynamic-schedule“ nicht alle Threads gleichzeitig auf den Arbeitsspeicher zugreifen, die Chunk-Größe eine bedeutend größere Rolle spielt muss, als dies bei „static“ und „guided“ der Fall ist.

Dokumentation des Codes:

Aus den obigen Beobachtungen ergeben sich folgende Zusammenhänge für die Parallelisierung des BEM-Codes:

Allgemeine Fälle

1. Schleifen für Matrix-Vektor-/Vektor-Vektor-Produkte und einfache Zuweisungen: Hierfür wird der „scheduling-type“ „static“ gewählt, da der Rechenaufwand pro Iteration nahezu identisch bleibt. Somit wird kaum Overhead benötigt und die Situation von „Times1“ und „Times2“ lässt sich optimal übertragen.



Beispiel eines einfachen Matrix-Vektor-Produkt; Entnommen aus BEM, Release vom 12.06.2020



Beispiel einer einfachen Zuweisungsschleife; Entnommen aus BEM, Release von 12.06.2020

1. Schleifen mit indirekter Indexierung: (Bild von Zeile 265+) Hierfür wird der „scheduling-type“ „guided“ verwendet, da durch die indirekte Indexierung lediglich ein verhältnismäßig kleiner zeitlicher Unterschied zwischen den einzelnen Threads entsteht, somit auch nur ein geringer Overhead benötigt wird.
2. „Reductions“ für parallele Schleifen: Zur Verwendung von benutzerdefinierten Reduktionsdirektiven müssen diese Sonderfälle zuerst definiert werden. Hieraus resultierend mussten zwei Reduktionsdeklarationen erstellt werden (eine für Vektoren mit Ganzzahlen und eine für größere Gleichkommazahlen).



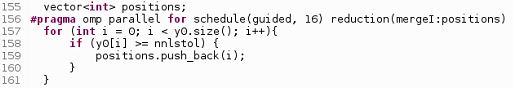
Deklaration für Vektoren mit Integer-Werten; Entnommen aus BEM, Release von 13.06.2020 (Zeilenumbruch zur besseren Lesbarkeit)



Deklaration für Vektoren mit Integer-Werten; Entnommen aus BEM, Release von 13.06.2020 (Zeilenumbruch zur besseren Lesbarkeit)

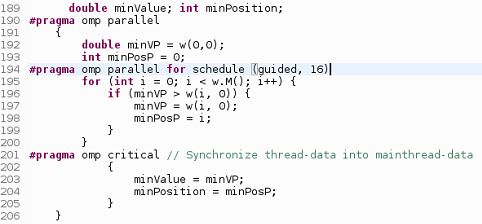
Sonderfälle

1. Find-Befehl (aus MatLab übersetzt): Für einen find-Befehl sollte der „scheduling-type“ entweder „dynamic“ oder „guided“ sein, je nach Anwendungsgebiet. In diesem Fall ergab sich nach Laufzeitmessungen „guided“, da dies den gewünschten geringeren Overhead reservierte und effizienter nutzte. Hier vereinfachte eine „reduction“ den Code erheblich, da diese die neuen Indexwerte in der richtigen Reihenfolge aus den einzelnen Threads in den Mainthread reduziert und somit eine Menge Codezeilen und Ausführungszeit spart.



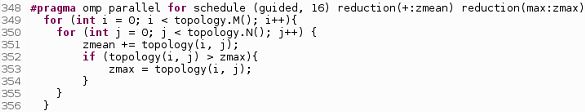
Implementierung eines Find-Befehls äquivalent zur Funktionsweise der Software MatLab; Entnommen aus BEM, Release vom 16.08.2020

1. Suche nach einem Werteminimum mit Positionsangabe: Für die Suche nach einem Werteminimum und dessen Position ergibt sich der „scheduling-type“ zu „guided“ oder „dynamic“, je nach Komplexität der Aufgabenstellung. In diesem Fall ergibt sich nach Laufzeitmessungen „…“, da dies den gewünschten Overhead reservierte und effizienter nutzte. Bei dieser Art von Aufgabe ergab sich auch ein vollständig neues Muster für die Parallelisierung, da ein Vergleich über bereits synchronisierte Variablen (über die Definition im Parallel-Konstrukt) langwierige „critical“-Blöcke spart. Somit wird die Suche über die beiden synchronisierten Variablen mit Startwerten durchgeführt, welche später in den Mainthread gemerged werden, indem die Werte der beiden privaten Variablen auf die eigentlich außerhalb definierten Variablen in einem „critical“-Block übertragen wird, sobald die Suche abgeschlossen ist. (TODO)



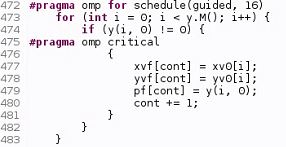
Parallele Suche eines Minimums mit gekoppelter Indexsuche des Minimums; Entnommen aus BEM, Release vom 16.08.2020

1. Summen- und Maximumbildung über „reduction“-Direktive: Für die Bildung einer Summe und der Suche nach einem Maximum (entkoppelt von einer parallelen Suche nach dem dazugehörigen Index), kann eine Parallelisierung mithilfe von „guided“ und zwei „reduction“-Direktiven verwendet werden. Zwar ergaben die Laufzeitmessungen für „static“ und „guided“ ähnliche Werte bei einer kleinen Datenmenge, jedoch wurde in diesem Fall „guided“ für den Allgemeinfall verwendet – aufgrund einer bessern Skalierung bei größer werdenden Problemen.



Beispiel für einen Algorithmus zum Aufsummieren und der Suche eines Maximums der Elemente der Matrix „topology“; Entnommen aus BEM, Release vom 16.08.2020

1. Einfügen in Vektoren mit abhängigem Index: Für die strikte Einfügung von Werten in einen Vektor wird zuallererst ein „scheduling-type“ „guided“ benötigt, da diese Art von Problemen ein wenig mehr Overhead benötigt. Da die Einfügung der Werte strikt erfolgen soll, darf also jederzeit nur ein Thread auf den Codeblock zugreifen, welcher die Zuweisungen ausführt – welches zur Konsequenz hat, dass die Codeperformance zwar zunimmt, aber nicht so stark skaliert wie dies bei der Parallelisierung eines Matrix-Vektor-Produkt der Fall wäre.



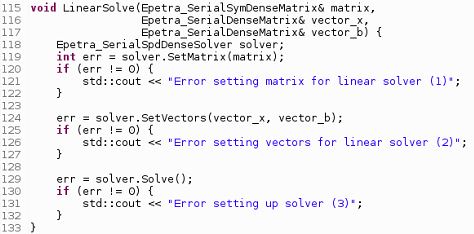
Beispiel für einen Algorithmus für die strikt indexbasierte Einfügung von Werten in einen Vektor; Entnommen aus BEM, Release vom 16.08.2020

1. Nicht parallele Regionen 1: (Veränderung von Arraylängen, …) (TODO)

C:\Users\Henrik Bartsch\AppData\Local\Microsoft\Windows\INetCache\Content.Word\NotParallel_Allocations and Implementations.jpg

Beispiel für einen nicht parallelisierbaren Codeabschnitt; Entnommen aus BEM, Release vom 16.08.2020

1. Nicht parallele Regionen 2: (LinearSolve) (TODO)



Beispiel für einen nicht parallelisierbaren Codeabschnitt; Entnommen aus BEM, Release vom 16.08.2020