Jules Siemssen und Gero Balsen

Assignment 1  
Parallele Algorithmen – Multicore Praktikum

Prof. Dr. Thomas Umland – Hochschule Bremerhaven

Die Gruppe besteht aus den Studenten:

Gero Balsen 33462

Jules Siemssen

Im Folgenden werden die Fragestellungen zum ersten Assignment im Kurs „Parallele Algorithmen – Multicore-Praktikum“ beantwortet und mit grafischen Darstellungen der Messergebnisse dargestellt.

Dabei sollten die Messungen auf einem Serversystem mit 40 physikalischen und 80 logischen Kernen stattfinden. Weiterhin ist das System mit 128GB Ram ausgestattet.

Der Test selbst bestand aus zwei Abschnitten. In beiden Abschnitten wurde während der Laufzeit mit einem Odd Even Merge Splitting Algorithmus sortiert. Lediglich die Vorsortierung wurde geändert um den Effekt dieser auf die allgemeine Laufzeit zu ermitteln. Im ersten Abschnitt wurde das System mit 60.000.000 unsortierten Zufallszahlen konfrontiert. Dabei wurden die Zufallszahlen aus dem Wertebereich einer Integer-Zahl aus Java gewählt. Zur Vorsortierung wurde die Arrays.Sort() Methode genutzt, anschließend sortierten die Threads untereinander mit einer Abwandlung des Merge-Sorts. Dabei wurde bei der Threadanzahl 1 bis 10 die Anzahl in einer Schritten erhöht, bei der Threadanzahl 10 bis 20 in zweier Schritten und ab 20 Threads in fünfer Schritten die Anzahl der zu nutzenden Threads erhöht, sodass wir einen sequentiellen Test mit einem Thread als Basis haben und ab 2 Threads bis 80 voranschreiten. Sollte jedoch die Anzahl der Zahlen nicht durch die Anzahl der Threads ohne Rest teilbar sein, wurde dieser Schritt ausgelassen. Dabei wird jede Stufe fünfmal getestet, und das Ergebnissen ist das ermittelte Minimum der fünf Durchläufe. Die Ergebnisse hierzu findet man in Abbildung 1 - Arrays.Sort.

Im zweiten Abschnitt des Tests wurde die Vorsortierung durch einen selbstimplementierten Bubblesort ersetzt. Des Weiteren wurde die Anzahl der Zufallszahlen reduziert, sodass das System nur noch 60.000 Zahlen sortieren musste. Die Ergebnisse hierzu findet man in Abbildung 4 - Bubblesort.

Wenn man nun die beiden Varianten anhand der Grafiken betrachtet sieht man eindeutig, dass die Bubblesort-Variante wesentlich mehr Optimierungspotential durch den Einsatz von mehr Threads besitzt. Dies erkennt man vor allem an der Effizienz, die im Gegensatz zur Arrays.Sort-Variante Werte von über 1 annimmt. Dies weißt auf einen wesentlich schlechteren Sortieralgorithmus in der Vorsortierung hin. Auch, dass im Gegensatz zur Bubblesort-Variante die Arrays.Sort-Variante schnell an die Grenzen der Optimierung durch Einsatz von mehr Threads spricht für diese Theorie.

All diese Grafen zeigen deutlich, dass die Bubblesort-Variante eine eindeutig schlechte Möglichkeit zur Vorsortierung ist. Die geschätzte Laufzeit hierfür beträgt , wohingegen der von Java bereits implementierte Sortieralgorithmus hinter Arrays.Sort eine Laufzeit von ca. besitzt.

# Abbildungsverzeichnis

[Abbildung 1 - Arrays.Sort 1](file:///C:\Users\André\Google%20Drive\Multicore\Assignment01-Auswertung.docx#_Toc498522514)

[Abbildung 2 - Beschleunigung Arrays.Sort 1](file:///C:\Users\André\Google%20Drive\Multicore\Assignment01-Auswertung.docx#_Toc498522515)

[Abbildung 3 - Effizienz Arrays.Sort 1](file:///C:\Users\André\Google%20Drive\Multicore\Assignment01-Auswertung.docx#_Toc498522516)

[Abbildung 4 - Bubblesort 1](file:///C:\Users\André\Google%20Drive\Multicore\Assignment01-Auswertung.docx#_Toc498522517)

[Abbildung 5 - Beschleunigung Bubblesort 1](file:///C:\Users\André\Google%20Drive\Multicore\Assignment01-Auswertung.docx#_Toc498522518)

[Abbildung 6 - Effizienz Bubblesort 1](file:///C:\Users\André\Google%20Drive\Multicore\Assignment01-Auswertung.docx#_Toc498522519)

Abbildung 1 - Arrays.Sort

Abbildung 2 - Beschleunigung Arrays.Sort

Abbildung 3 - Effizienz Arrays.Sort

Abbildung 4 - Bubblesort

Abbildung 5 - Beschleunigung Bubblesort

Abbildung 6 - Effizienz Bubblesort