Aufgabe 5: Leistungsanalyse

16.11.2019

Auch wenn es nur gefordert war die sequentielle Version von Partdiff mit einer PThread Parallelisierten Variante zu vergleichen, werden vier Programme verglichen: Partdiff, die sequentielle Version, Partdiff-OpenMP, mit OpenMP parallelisiert (die Version aus der Abgabe voriger Woche), Partdiff-Posix, eine naive mit PThread parallelisierte Variante wo in jeder Iteration Threads erzeugt und vereinigt ("gejoined") werden, und Partdiff-Posix-Keepalive, wo nur am Anfang PThreads erzeugt, aber erst nach der Rechnung vereinigt werden. Im folgenden wird die Standardabweichung durch

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n-1}}$$

mit $n = 3$ berechnet.

Erklärung Alle parallelisierten Varianten von Partdiff zeigen einen erheblichen Speedup. Es fällt jedoch in 7 auf das Partdiff-Posix ab 10 Threads stark an Wachstum verliert und ab 12 Threads sogar langsamer wird. Dies lässt sich vor allem darauf zurückzuführen, dass die Erzeugung und Löschung (Joinen) von Threads in jeder Iteration einen beträchtlichen Overhead hat, im Vergleich zu Partdiff-OpenMP und Partdiff-Posix-Keepalive, wo die Threads nicht in jeder Iteration neu erzeugt werden. Es zeichnet sich schon ab einer Anzahl von 3 Threads ab, das der Zeitgewinn gegenüber den anderen beiden Varianten geringer ist und immer langsamer wächst, bis ab 11 Threads Partdiff-Posix noch langsamer macht. In Abbildung 6, 7 und 8 ist gut erkennbar, das die Laufzeiten und Speedup von Partdiff-Posix-Keepalive und Partdiff-OpenMP sich kaum unterscheiden. Ein Mensch kann also bei simplen Programmen eine ähnliche Leistungssteigerung durch manuelle Parallelisierung erreichen, wie OpenMP. In Abbildung 8 zeigt sich, günstiger ist am Anfang Threads zu erzeugen und über Mutexes und Bedingungsvariablen die Arbeit über den Rechenzeitraum auf die Threads zu verteilen, als in jeder Iteration einem neuen Thread die Arbeit zuzuweisen und dann auf ebenjene Threads zu warten. So kann man eine Leistungssteigerung von bis zu 50% bei 12 Threads gegenüber Partdiff-Posix messen.

Parameter	
Iterationen:	1500
Interlines:	512
Störfunktion:	2
Verfahren:	Jacobi

Abbildung 1: Parameter für Partdiff

Programm	Ausfuehrungsknoten
Partdiff	west2
Partdiff-Posix	west4
Partdiff-Posix-Keepalive	west1
Partdiff-OpenMP	west2

Abbildung 2: Knoten auf dem Cluster, auf der die Messungen für die jeweiligen Programme gemacht wurden

Partdiff	σ
790.903	0.497

Abbildung 3: Mittlere Ausführungszeit in Sekunden von Partdiff mit der Standardabweichung σ

Threads	Partdiff-Posix	σ	Partdiff-Posix-Keepalive	σ	Partdiff-OpenMP	σ
1	794.36	0.467	783.09	0.064	782.29	0.071
2	402.18	0.007	392.043	0.2	393.147	0.33
3	272.41	0.255	264.033	0.038	263.08	0.191
4	206.887	0.002	198.11	0.219	198.28	0.113
5	167.96	0.113	158.55	0.092	158.5	0.092
6	141.31	0.028	131.877	0.054	131.953	0.016
7	122.68	0.064	113.267	0.073	113.36	0.049
8	108.11	0.042	99.177	0.054	99.05	0.021
9	96.467	0.026	88.733	0.047	88.26	0.057
10	87.71	0.177	79.713	0.005	79.763	0.045
11	86.083	0.49	72.863	0.038	72.61	0.12
12	100.613	0.766	66.893	0.026	66.963	0.158

Abbildung 4: Mittlere Ausführungszeiten in Sekunden gefolgt von ihrer jeweiligen Standardabweichung σ

Threads	Partdiff-Posix	Partdiff-Posix-Keepalive	Partdiff-OpenMP
1	0.996	1.01	1.011
2	1.967	2.017	2.012
3	2.903	2.995	3.006
4	3.823	3.992	3.989
5	4.709	4.988	4.99
6	5.597	5.997	5.994
7	6.447	6.983	6.977
8	7.316	7.975	7.985
9	8.199	8.913	8.961
10	9.017	9.922	9.916
11	9.188	10.855	10.892
12	7.861	11.823	11.811

Abbildung 5: Speedup Tabelle. Bezieht sich auf das sequentielle Partdiff

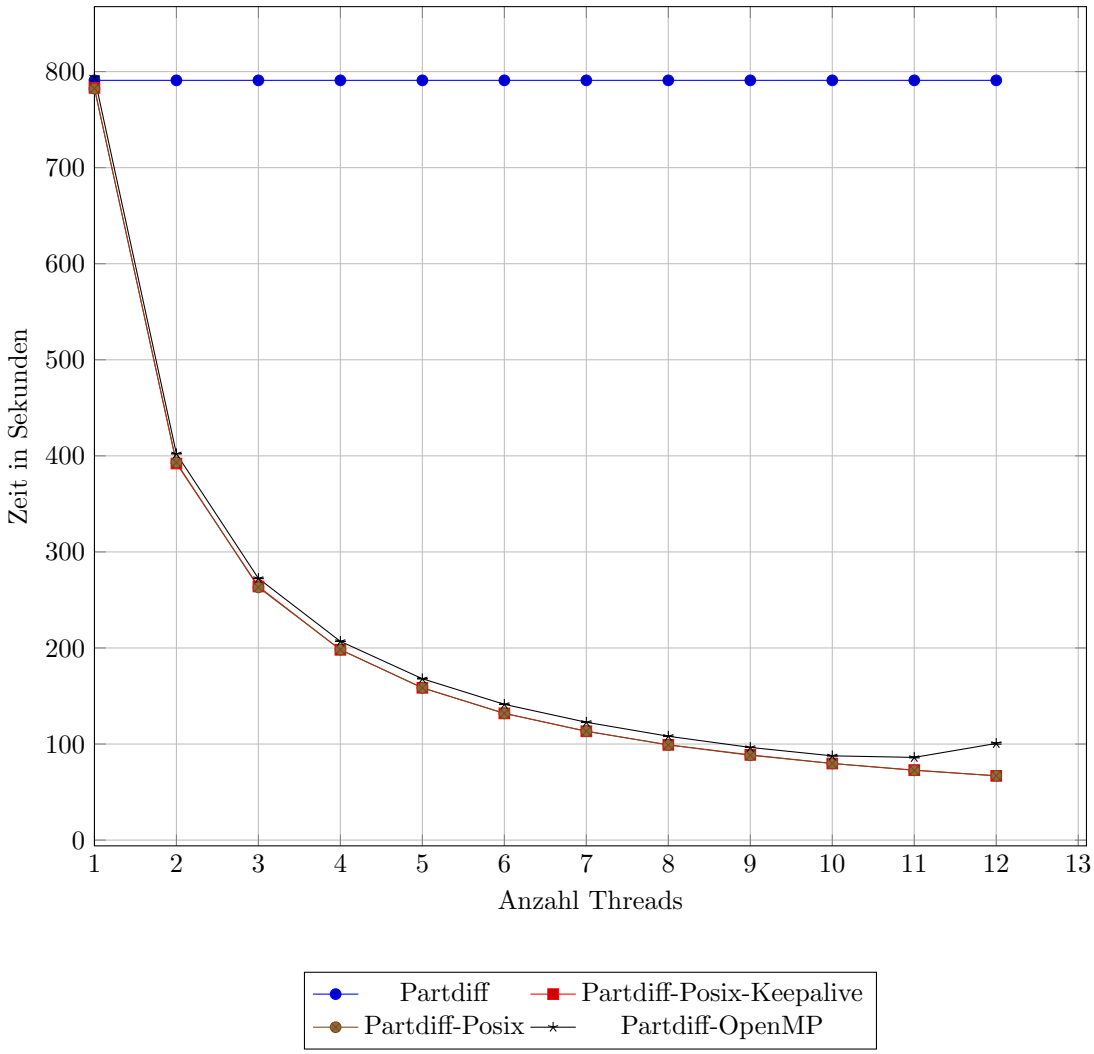

Abbildung 6: Vergleich der Laufzeiten aller vier Programme mit ihrer jeweiligen Standardabweichung

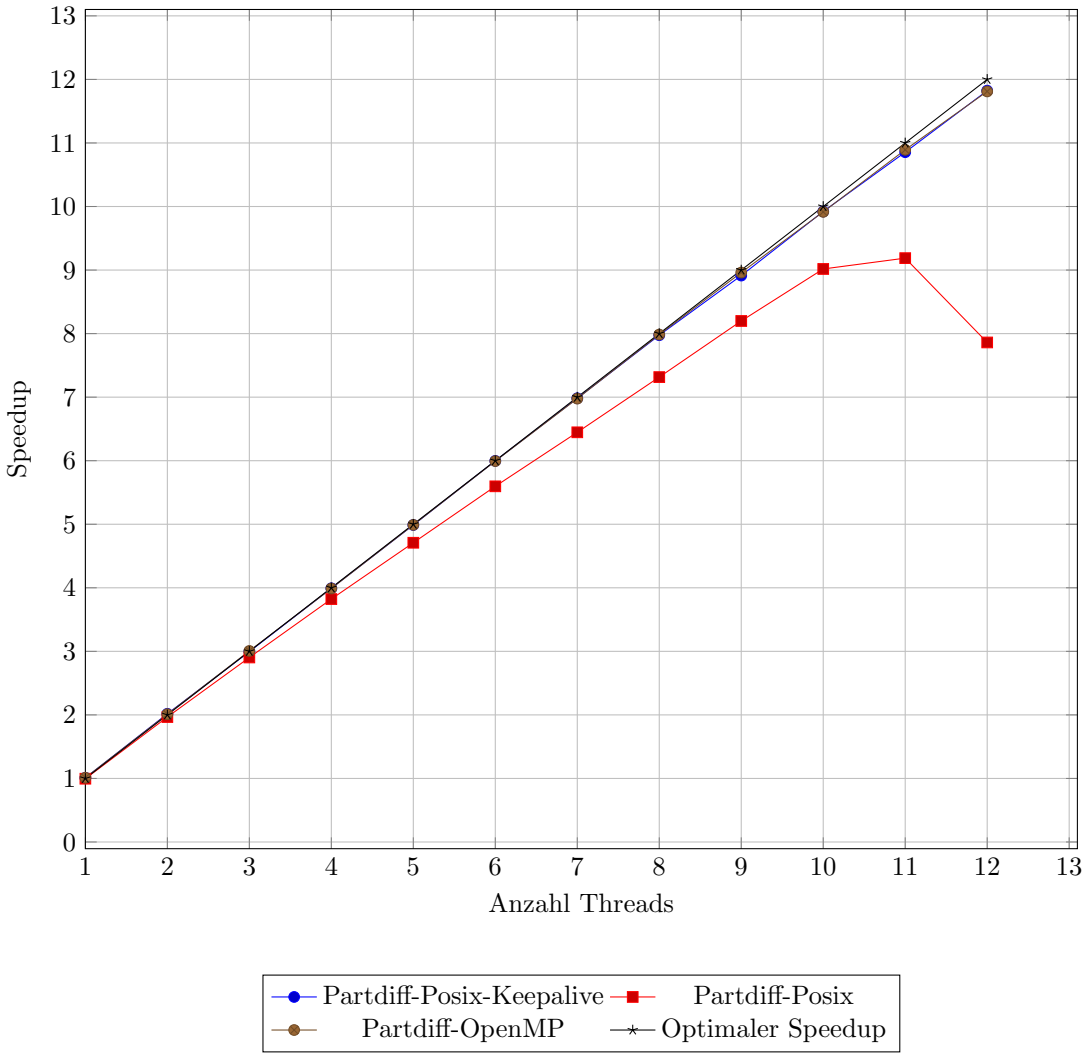

Abbildung 7: Speedup Graph mit optimalen Speedup als Vergleich. Bezieht sich auf das sequentielle Partdiff

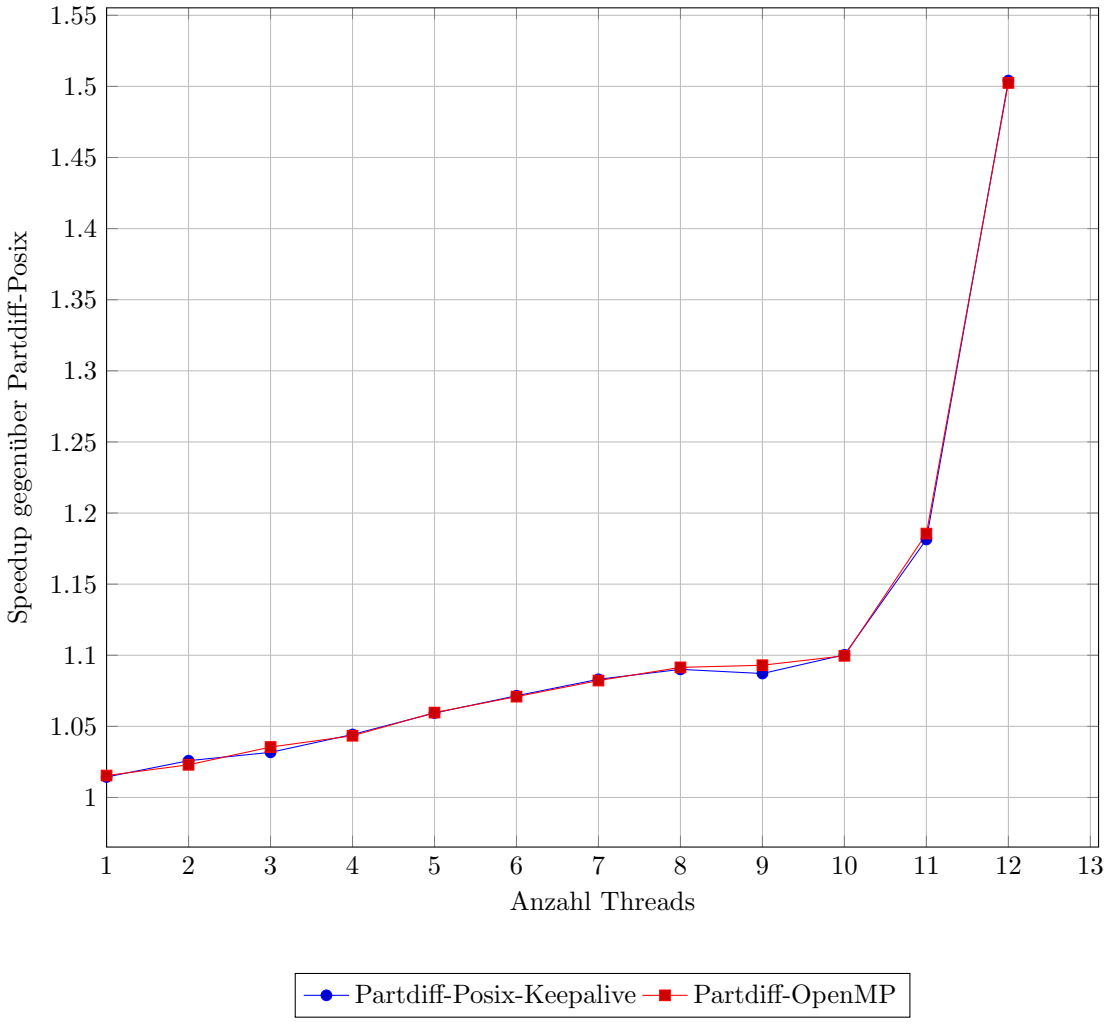

Abbildung 8: Speedup Graph. Bezieht sich auf Partdiff-Posix