Hochleistungsrechnen

Tim Welge, Benjamin Wolf, Enrico Milutzki

12.11.2016

1 Parallelisierung mit OpenMP (120 Punkte)

```
Funktionsaufruf:
$ ./partdiff-openmp 12 2 512 2 2 64
```

	1.Durchlauf	2.Durchlauf	3.Durchlauf
Keine Paralellisierung	35,854sec	35,776sec	35,804sec
Paralellisiert	3,502sec	4,188sec	4,272sec

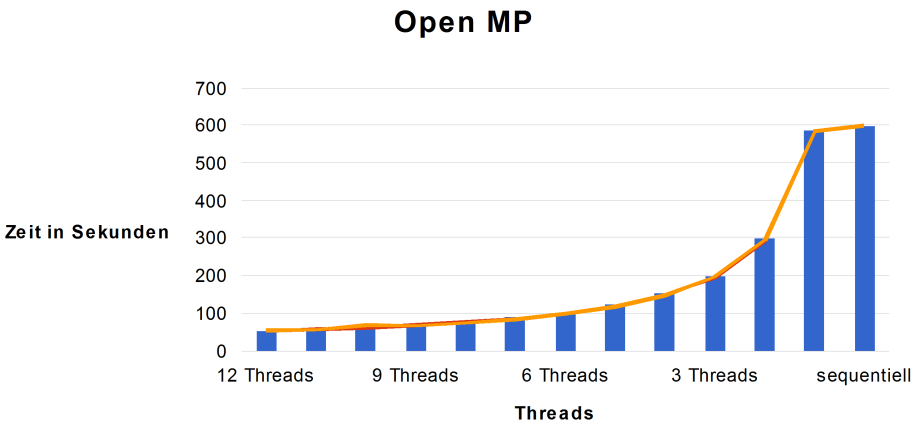
```
Funktionsaufruf:
$ ./partdiff-openmp 12 2 512 2 1 1e-6
```

	1.Durchlauf	2.Durchlauf	3.Durchlauf
Keine Paralellisierung	0,677sec	0,681sec	0,677sec
Paralellisiert	0,171sec	0,165sec	0,187sec

```
Zeit:
165min, davon ca.45min für Fehlersuche
```

2 Leistungsanalyse (120 Punkte)

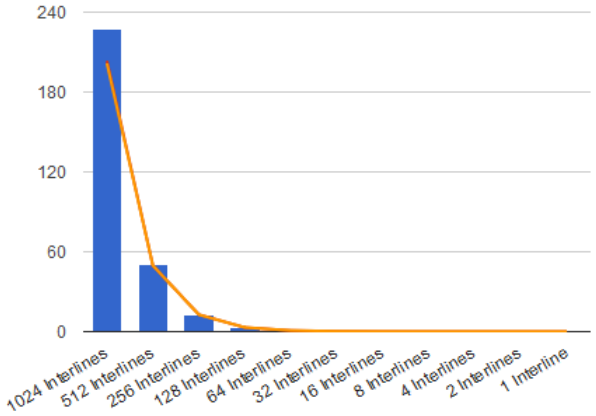
Messung 1



Threads	1. Durchlauf	2. Durchlauf	3.Durchlauf	Mittelwert		1. node	2. node	3. node
12 Threads	51,637	51,091	53,268	51,99866667		abu1	abu1	abu1
11 Threads	55,173	56,381	54,457	55,337		abu1	abu1	abu1
10 Threads	61,361	60,822	67,793	63,32533333		abu1	abu1	abu1
9 Threads	68,623	67,697	66	67,45533333		abu1	abu1	abu2
8 Threads	73,522	74,084	73,644	73,75		abu1	abu2	abu3
7 Threads	86,344	84,041	83,839	84,74133333		abu1	abu2	abu3
6 Threads	98,875	97,858	97,554	98,09566667		abu1	abu2	abu3
5 Threads	120,171	117,296	117,159	118,2086667		abu1	abu2	abu3
4 Threads	151,237	147,561	146,251	148,3496667		abu1	abu2	abu3
3 Threads	195,596	194,974	196,947	195,839		abu1	abu2	abu3
2 Threads	297,425	292,264	292,72	294,1363333		abu1	abu2	abu3
1 Thread	584,779	584,766	585,027	584,8573333		abu1	abu2	abu3
sequentiell	599,298	598,865	598,529	598,8973333		abu1	abu2	abu3
Funktionsaufruf:	$ ./partdiff-openmp n 2 512 2 2 1024							
	n = Anzahl der Threads							

Man sieht deutlich, dass das Programm von der Parallelisierung profitiert. Durch das Verdoppeln der benutzten Threads verringert sich die benötigte Zeit um etwa die Hälfte. Das sieht man am Besten, wenn man die Zeiten von 1 Thread und 2 Threads vergleicht, oder von 2 Threads und 4 Threads. Jedoch nimmt der Speedupfaktor mit zunehmender Threadzahl ab. Dafür könnte OpenMP verantwortlich sein, da auch hier ein Scheduling durchgeführt werden muss, da mit mehreren Threads auch mehr Kontextwechsel nötig werden und so irgendwann die Effizienz abnimmt.

Messung 2



Interlines	1. Durchlauf	2. Durchlauf	3.Durchlauf	Mittelwert
1024 Interlines	227,837	203,453	201,63	210,9733333
512 Interlines	50,412	49,607	49,644	49,88766667
256 Interlines	12,561	12,79	12,695	12,682
128 Interlines	3,107	3,099	3,125	3,110333333
64 Interlines	0,883	0,841	0,881	0,8683333333
32 Interlines	0,266	0,243	0,255	0,2546666667
16 Interlines	0,085	0,081	0,092	0,086
8 Interlines	0,043	0,049	0,039	0,04366666667
4 Interlines	0,019	0,028	0,058	0,035
2 Interlines	0,032	0,022	0,047	0,03366666667
1 Interline	0,023	0,018	0,017	0,01933333333
Rechnerknoten	abu1	abu2	abu3	

Mit 1024 Interlines ist die Rechenzeit, die benötigt wird, die das Programm benötigt, um zu terminieren besonders hoch. Das könnte daran liegen, dass bei einer Matrixgröße von 1024 die Matrix nicht mehr in den Cache passt und damit nicht mehr schnell adressiert werden kann. Im Gegensatz dazu sieht man, dass ab 8 Interlines nicht mehr die Matrixgröße eine Rolle spielt sondern nur noch die verfügbare Rechenpower der einzelnen Kerne, die die Berechnung ausführen. So kan