Aufgabenblatt 5

Aufgabe 1

Datei: partdiff.c
Zeit: ca. 7h, davon 5h Fehlersuche & Recherche

Aufgabe 2

Datei: partdiff-b.c
Methoden für Zeilen- und Spaltenweise Aufteilung heißen calculateRow und calculateCol

Aufgabe 4

Messung 1:

Standardaufteilung
Benchmark mit hyperfine --warmup 1 --min-runs 3 --parameter-scan n 1 24 'srun -p vl-parcio -c {n} ./partdiff {n} 2 4096 2 2 45'

Threads	Berechnungsdauer	Speedup
1	845.136 s ± 0.071 s	/
2	298.934 s ± 0.037 s	2,83
3	183.686 s ± 0.019 s	4,6
4	150.998 s ± 1.289 s	5,6
5	118.393 s ± 0.017 s	7,1
6	100.873 s ± 0.030 s	8,37
7	86.672 s ± 2.106 s	9,75
8	76.162 s ± 0.427 s	11,1
9	68.170 s ± 0.393 s	12,4
10	61.512 s ± 0.001 s	13,73
11	56.700 s ± 0.014 s	14,9
12	52.831 s ± 0.008 s	16
13	49.279 s ± 0.273 s	17,15
14	46.525 s ± 0.380 s	18,16
15	43.384 s ± 0.171 s	19,48
16	41.392 s ± 0.219 s	20,42
17	39.060 s ± 0.134 s	21,6
18	38.072 s ± 0.616 s	22,2
19	36.033 s ± 0.484 s	23,45
20	34.707 s ± 0.244 s	24,35
21	33.005 s ± 0.076 s	25,6

Threads	Berechnungsdauer	Speedup
22	32.074 s ± 0.308 s	26,34
23	30.840 s ± 0.110 s	27,4
24	29.760 s ± 0.057 s	28,4

Mit dem parallelen Programm und 24 Threads ergab sich gegenüber dem seriellen ein Speedup von 28,4. Allgemein lässt sich sagen, dass
für dieses Intervall eine Erhöhung der Threadanzahl die Laufzeit verringert. Auffällig ist jedoch, dass die Unterschiede der Laufzeiten
immer kleiner werden je mehr Threads genutzt werden. Zwischen 2 und 3 Threads ist der Unterschied 115 s, zwischen 10 und 11 nur noch
4,7 s und zwischen 23 und 24 Threads beträgt die Differenz der Laufzeiten gerade einmal 1,08 s. Erklären lässt sich dies durch die
steigende Synchronisationsarbeit bei steigender Threadanzahl. Diese "dämpft" den Speedup, wodurch die Differenzen in der höheren
Threadanzahl nicht mehr so groß, wie am Anfang sind.
Da die anderen Datenaufteilungen bei uns keine korrekte Matrix mehr berechnet haben, können wir nur Vermutungen zu den Unterschieden aufstellen.
Weil C row-major-order nutzt, wäre wahrscheinlich die Zeilenvariante schneller als die Spalten- oder Elementvariante, welche durch
ineffiziente Speicherzugriffe an Zeit einbüßen müssten.

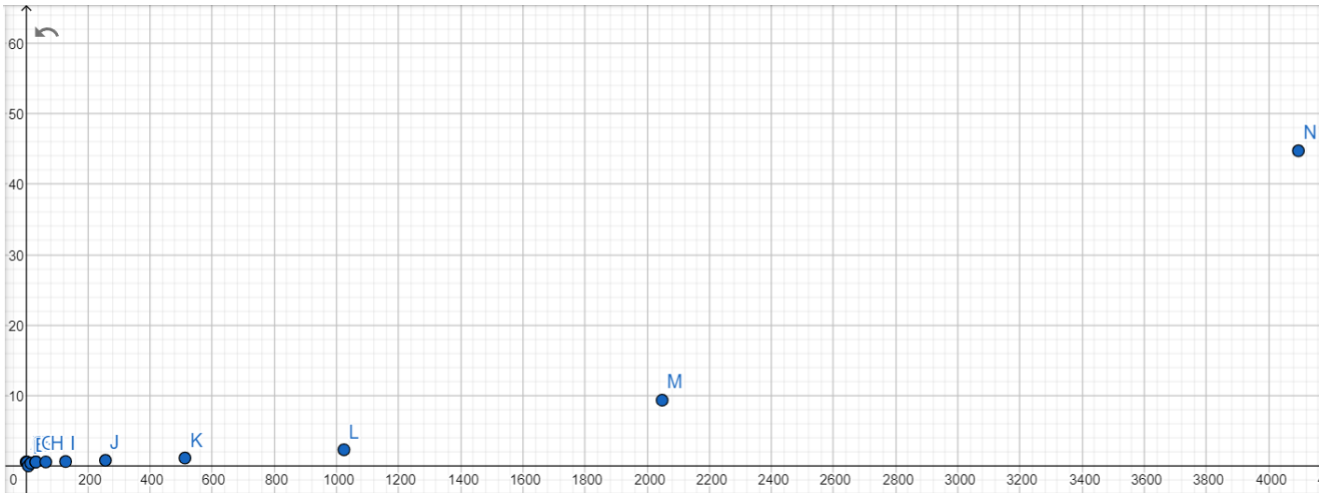
Messung 2:

./partdiff 24 2 i 2 2 45

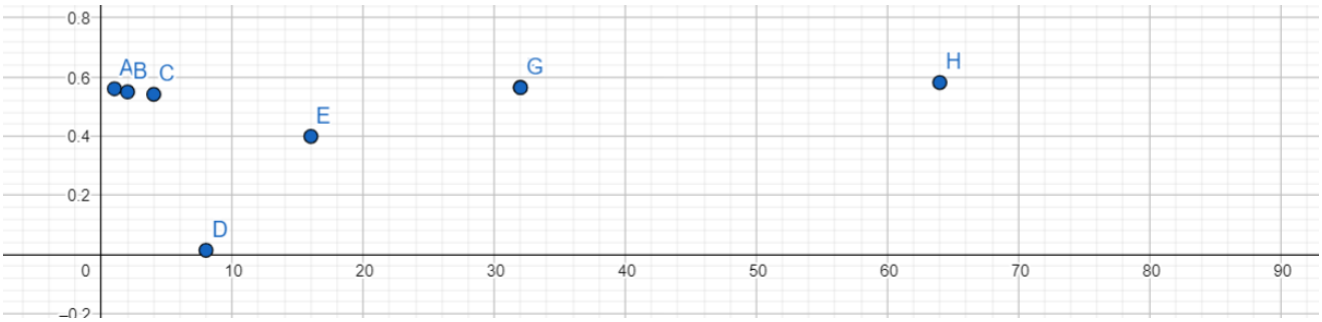
Interlines	Berechnungsdauer
4096	44.736s
2048	9.331s
1024	2.306s
512	1.289s
256	0.802s
128	0.630s
64	0.581s
32	0.564s
16	0.399s
8	0.014s
4	0.541s
2	0.549s
1	0.560s

```
NodeName=ant19 Arch=x86_64 CoresPerSocket=24
   CPUAlloc=48 CPUTot=48 CPULoad=2.89
   AvailableFeatures=(null)
   ActiveFeatures=(null)
   Gres=(null)
   NodeAddr=ant19 NodeHostName=ant19 Version=20.11.8
   OS=Linux 4.18.0-348.2.1.el8_5.x86_64 #1 SMP Tue Nov 16 14:42:35 UTC 2021
```

```
   RealMemory=120536 AllocMem=98304 FreeMem=110242 Sockets=1 Boards=1
   State=ALLOCATED ThreadsPerCore=2 TmpDisk=0 Weight=1 Owner=N/A MCS_label=N/A
   Partitions=vl-parcio
   BootTime=2021-11-19T11:06:22 SlurmdStartTime=2021-11-19T11:07:12
   CfgTRES=cpu=48,mem=120536M,billing=48
   AllocTRES=cpu=48,mem=96G
   CapWatts=n/a
   CurrentWatts=0 AveWatts=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
   Comment=(null)
```



x = Anzahl Interlines
y = Zeit in s



Interpretation:
Der Rechenaufwand steigt im Endeffekt linear zu der Anzahl der Interlines. Da diese jedoch exponentiell wächst 2^x, sieht die Berechnungszeit auch exponentiell wachsend aus.
Der Aufwand der Berechnung, steigt somit linear zur größe der Matrix an. Dieser ist somit in der calculate Funktion $O(n^2)$, durch die verschachtelten for-Schleifen, welche durch jedes Element n durchlaufen.
Weiterhin kann man sagen, dass bei weiter wachsenden Interlines, die Berechnungszeit ebenfalls um ein vielfaches steigern würde, was dann selbst paralellisiert sehr lange dauern wird.