Parallele und verteilte Systeme
Übung 1

Peter Dunn, Marie Trojan, Fionn Erickson
Matrikelnummern: 121530, 121067, 121588
Bauhaus-Universität Weimar

19.11.2020

Teil 1. Das erste parallele Programm

1.

siehe num_threads.cpp Datei

3.

pragma omp parallel num_threads(4)
Number of threads: 4
This task took 0.000105 seconds

pragma omp parallel num_threads(10)
Number of threads: 10
This task took 0.000298 seconds

pragma omp parallel num_threads(100)
Number of threads: 100
This task took 0.002389 seconds

pragma omp parallel num_threads(1000)
Number of threads: 1000
This task took 0.019866 seconds

Mit der Anzahl der Threads steigt ebenso die Anzahl der Sekunden, welche die Ausführung des Programms benötigten. Dies ist der Fall weil dieser Prozess nicht an verschiedenen threads aufgeteilt wird, sondern von jeden thread wiederhohlt wird, und somit der Aufwand auf den Rechner, und die durchnschittszeit fuer alle threads insgesammt steigt.

Teil 2. Matrix-Multiplikation

1.

Die Matrix Multiplikation dürfte am längsten von allen for-Schleifen dauern, und die Berechnung ändert die Elemente an sich nicht, also ist die Reihenfolge der for-Schleife egal, und somit kann diese for-Schleife durch den Befehl “pragma omp parallel for collapse(3)öhne Probleme parallel beschleunigt werden. Das Speed-up ist jedoch nur bei größeren Matrizen sichtbar. (Beispielsweise bei 3, 4, 6 ist die serielle Version schneller, jedoch ist bei Matrizen mit de Dimensionen 300, 400, 600 die parallele Version schneller. (um größere Matrizen zu testen, hat die geschickte version den print der Matrizen ausgeschaltet.)

2.

```
useradd@DESKTOP-H7P7MNO:/mnt/f/UniStuff/Sem3/PvS1/pvs-uebung1$ nano matmult.cpp
useradd@DESKTOP-H7P7MNO:/mnt/f/UniStuff/Sem3/PvS1/pvs-uebung1$ g++ -fopenmp matmult.cpp -o matmult
useradd@DESKTOP-H7P7MNO:/mnt/f/UniStuff/Sem3/PvS1/pvs-uebung1$ ./matmult 3 4 6
Matrix sizes C[3][6] = A[3][4] x B[4][6]
Perform matrix multiplication...
Matrix multiplication took 0.000001 seconds
Matrix A:
   3.0      6.0      7.0      5.0
   3.0      5.0      6.0      2.0
   9.0      1.0      2.0      7.0
Matrix B:
   0.0      9.0      3.0      6.0      0.0      6.0
   2.0      6.0      1.0      8.0      7.0      9.0
   2.0      0.0      2.0      3.0      7.0      5.0
   9.0      2.0      2.0      8.0      9.0      7.0
Matrix C:
  71.0     73.0     39.0    127.0    136.0    142.0
  40.0     61.0     30.0     92.0     95.0    107.0
  69.0    101.0     46.0    124.0     84.0    122.0

Done.
double free or corruption (out)
Aborted
useradd@DESKTOP-H7P7MNO:/mnt/f/UniStuff/Sem3/PvS1/pvs-uebung1$ nano matmult.cpp
useradd@DESKTOP-H7P7MNO:/mnt/f/UniStuff/Sem3/PvS1/pvs-uebung1$ g++ -fopenmp matmult.cpp -o matmult
useradd@DESKTOP-H7P7MNO:/mnt/f/UniStuff/Sem3/PvS1/pvs-uebung1$ ./matmult 3 4 6
Matrix sizes C[3][6] = A[3][4] x B[4][6]
Perform matrix multiplication...
Matrix multiplication took 0.000086 seconds
Matrix A:
   3.0      6.0      7.0      5.0
   3.0      5.0      6.0      2.0
   9.0      1.0      2.0      7.0
Matrix B:
   0.0      9.0      3.0      6.0      0.0      6.0
   2.0      6.0      1.0      8.0      7.0      9.0
   2.0      0.0      2.0      3.0      7.0      5.0
   9.0      2.0      2.0      8.0      9.0      7.0
Matrix C:
  71.0     73.0     39.0    127.0     91.0    142.0
  40.0     61.0     30.0     92.0     95.0    107.0
  69.0    101.0     46.0    124.0     84.0    122.0

Done.
double free or corruption (out)
Aborted
```

Oben sieht man ein Beispiel von der Durchführung, erst mal ohne Parallelisierung, danach mit Parallelisierung (der Zeit unterschied ist wie zuvor gesagt wegen der Größe der Matrizen). An kleinen Matrizen bemerkt man das die Parallelisierung nicht die Werte oder die Reihenfolge der Matrizen manipuliert. (die Zufalls werte für die Matrizen sind seeded, also werden bei gleichen Dimensionen die selben Matrizen produziert)

3.

Speedup = sequenzielle Laufzeit / parallelisierte Laufzeit
Beispiel: < 1000 >< 1000 >< 1000 >

```
marie@bee:~/Schreibtisch/3. Semester/PVS/pvs-uebung1$ g++ -fopenmp -Wall -o mat
mult matmult.cpp && ./matmult 1000 1000 1000
Matrix sizes C[1000][1000] = A[1000][1000] x B[1000][1000]
Perform matrix multiplication...

sequenzieller Durchlauf
Matrix multiplication took 3.935101 seconds

Done.
Speicherzugriffsfehler (Speicherabzug geschrieben)
marie@bee:~/Schreibtisch/3. Semester/PVS/pvs-uebung1$ g++ -fopenmp -Wall -o mat
mult matmult.cpp && ./matmult 1000 1000 1000
Matrix sizes C[1000][1000] = A[1000][1000] x B[1000][1000]
Perform matrix multiplication...

paralellisierter Durchlauf
Matrix multiplication took 0.320862 seconds

Done.
Speicherzugriffsfehler (Speicherabzug geschrieben)
marie@bee:~/Schreibtisch/3. Semester/PVS/pvs-uebung1$
```

Speedup = 3.935101/0.320862
Speedup = 12.26415406

4.

Für weiteren Speedup könnten die for-Schleifen in der alloc_mat und init_mat Funktionen ähnlicher Weise parallelisiert werden. Diese Befehle an sich sind jedoch extrem schnell, und die Beschleunigung von der Parallelisierung ist geringer als der Slowdown der von der Kommunikation der threads produziert wird, also wird eine Parallelisierung von diesen beiden for-Schleifen im generellen Fall nicht hilfreich sein. (Vielleicht könnte es bei ausreichend großen Matrizen ein Speedup verursachen, aber die Größe dieser Matrizen ist mehr als was unsere Heimrechner innerhalb von einem realistisch machbaren Zeitraum berechnen könnten, und würde selbst da kein ansatzweise so großen Speedup wie die Parallelisierung der Matrizenmultiplikation produzieren.)