Parallel Computing, WS 2019

Assignment 2

Nachname, Vorname:	KhachidzeTornike
Matrikelnummer:	01469313
E-Mail-Adresse:	a01469313@unet.univie.ac.at
Datum:	06.12.2019

Parallelization strategy

```
A[i][j]= ((f1(A[i-1][j-1])+
           f1(A[i][j-1])+
           f1(A[i-1][j]))*
           f1(A[i-1][j-1]+A[i][j-1]+A[i-1][j]))
           % 1000;
if (A[i][j]==0) A[i][j]=999;
```

Meine Parallelisierungsstrategie besteht darin, das Array diagonal zu Iterieren, da wir Abhängigkeiten von Elementen haben. Darüberhinaus ist das Hauptziel, die Abhängigkeit loszuwerden.

```
#pragma omp parallel if (N>500) default(none) num_threads(16) shared(parallelArray) private (i,j)
  {

    for(i = 1; i < N; i++)
    {
      #pragma omp for
        for(j = 1; j <= i; j++)
        {
           parallelArray[j][i-j+1] = ((f1(parallelArray[j-1][i-j+1-1])+
                                       f1(parallelArray[j][i-j+1-1])+
                                       f1(parallelArray[j-1][i-j+1]))*
                                       f1(parallelArray[j-1][i-j+1-1]+parallelArray[j][i-j+1-1]+parallelArray[j-1][i-j+1]))
                                       % 1000;

          if (parallelArray[j][i-j+1]==0) parallelArray[j][i-j+1]=999;
        }
    }

    for(i = 1; i < N; i++)
    {
      #pragma omp for
      for(j = N - 1; j >= i; j--)
      {
         parallelArray[N-j+i-1][j] = ((f1(parallelArray[N-j+i-1-1][j-1])+
                                       f1(parallelArray[N-j+i-1][j-1])+
                                       f1(parallelArray[N-j+i-1-1][j]))*
                                       f1(parallelArray[N-j+i-1-1][j-1]+parallelArray[N-j+i-1][j-1]+parallelArray[N-j+i-1-1][j]))
                                       % 1000;

        if (parallelArray[N-j+i-1][j]==0) parallelArray[N-j+i-1][j]=999;
      }
    }
```

Hier in diesem Code wird Array entweder sequential oder parallel gefüllt.

OpenMP pragmas

Meine implementierung basiert auf C++.

```
#pragma omp parallel if (N>500) default(none) num_threads(16) shared(parallelArray) private (i,j)
```

pragma omp parallel – Parallelkonstrukt.
if(N>500) – um Program etweder sequential oder parallel auszuführen.
default(none) - wir definieren alle Parameter selbst.
num_threads(16) - optimale Anzahl von Threads.
schared(parallelArray) - Array ist sichtbar für alle threads.
private(i,j) - Schleifen haben privaten Variablen (i und j).
pragma omp for - for schleife parallel mit parameter von parallel construct.

Best execution times
g++-7.2.0 -O3 -std=c++11 -fopenmp -mcmodel=large stencil.cpp && ./a.out 25 48 3000 2000 25673 41983
icc -O3 -std=c++11 -fopenmp -mcmodel=large stencil.cpp && ./a.out 25 48 3000 2000 25673 41983

g++ parallel	20236 ms
g++ sequential	224033 ms
icc parallel	23021 ms
icc sequential	248075 ms

Speed-ups

g++ sequential - g++ parallel → **11,07**
icc sequential - g++ parallel → **12,25**
icc sequential - icc parallel → **10,77**

Monitoring

```
[a01469313@cora05 ~]$ top -U a01469313
```

```
[a01469313@cora05 assignment2]$ g++-7.2.0 -O3 -std=c++11 -fopenmp -mcmodel=large stencil.cpp
[a01469313@cora05 assignment2]$ ./stencil
```

```
top - 15:21:55 up 188 days,  1:36,  2 users,  load average: 6.55, 3.55, 1.78
Tasks: 401 total,   2 running, 399 sleeping,   0 stopped,   0 zombie
Cpu(s): 99.9%us,  0.1%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  24597076k total,  9179104k used, 15417972k free,   124044k buffers
Swap: 12369916k total,    15388k used, 12354528k free,   294324k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
19698 a0146931  20   0 9205m 7.9g 1848 R 1599.9 33.7   3:25.78 stencil
19715 a0146931  20   0 19548 1640 1056 R  0.7  0.0   0:00.12 top
18633 a0146931  20   0 99.8m 1976  976 S  0.0  0.0   0:00.09 sshd
18634 a0146931  20   0  109m 1996 1560 S  0.0  0.0   0:00.12 bash
19482 a0146931  20   0 99.8m 1972  976 S  0.0  0.0   0:00.03 sshd
19483 a0146931  20   0  109m 1960 1544 S  0.0  0.0   0:00.04 bash
```

Gcc experiences

Langsam

hier wird mit verschiedenen threads ausprobiert. Das beste Ergebnis zeigt sich mit 16 Threads.

Thread	Time
16	23021 ms
14	27295 ms
12	30243 ms
10	36710 ms
8	39387 ms
6	44350 ms
4	64456 ms