Exercise 3

Team: Summit

Description: Investigate the following given code examples along with their task.

```
void copy(double *x, double *y)
{
        for (int i = 0; i < SIZE; i++)
        {
                x[i] = y[i];
        }
}
```

1. **Can this code be safely parallelized manually?**

   Yes, for example with

   #pragma omp for

   Can this code be safely parallelized by the compiler?
   Yes, there's no problem. I tried it with the compiler flag -O1 and -O3 and got a very good speedup.

2. Normalize the following loop nest.

```
for (int i = 4; i <= N; i += 7)
{
        for (int j = 0; j <= N; j += 3)
        {
                A[i] = 0;
        }
}
```

Normalized:

```
for (int i = 0; i <= (N - 4)/7; i++)
{
        A[(7*i + 4)] = 0;
}
```

The speedup is enormous from 16sec. to 0,008 sec. But we must be aware that (N – 5)/7 results in an Integer, otherwise we can become conflicts due segmentation fault.

3. Does the following code excerpt hold any dependencies?

```
for (int i = 1; i < N; i++)
{
        for (int j = 1; j < M; j++)
        {
                for (int k = 1; k < L; k++)
                {
                        a[i + 1][j][k - 1] = a[i][j][k] + 5;
                }
        }
}
```

Yes. The nested loops use the overlaying variables.
If not, how would you parallelize it?
If yes, what are the distance and direction vectors?

Distance vectors: captures the "shape" of dependences, but not the particular source and sink. **Direction vectors**: captures the "direction" of dependences, but not the particular shape.

S1 has a true dependence on itself.

e.g S1[1,1,2] & S1[2,1,1] for a(2,1,1)

Distance Vector = (1,0,-1)

Direction Vector = (<,=,>)