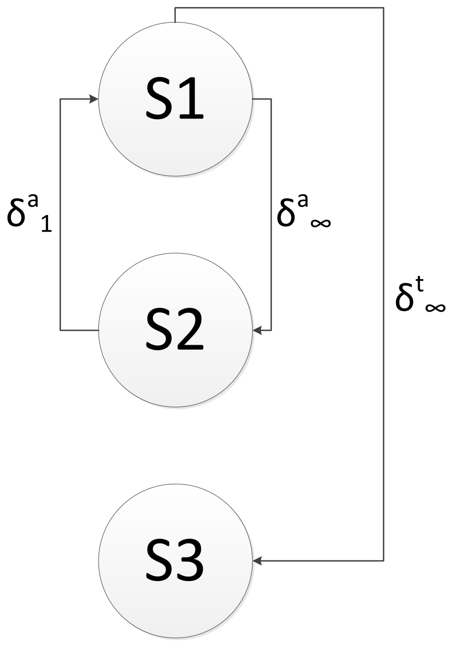
Apply loop distribution to the code given below. Distribute the loop as far as possible. Which loop can be parallelized?

```
         for (i=0;i<n;i++){
    S1:    B(i)=A(i)
    S2:    A(i)=A(i)+B(i+1)
    S3:    C(i)=2*B(i)
         }
```

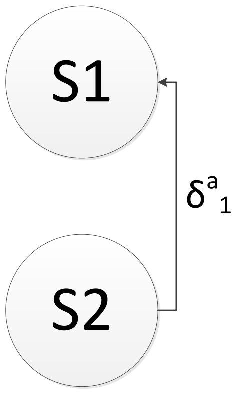


Loop Distribution:

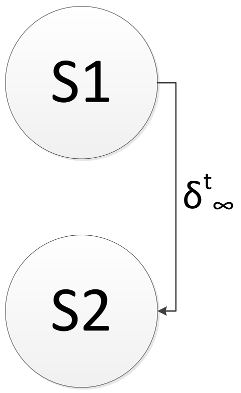
```
         for (i=0;i<n;i++){
    S1:    B(i)=A(i)
    S2:    A(i)=A(i)+B(i+1)
         }

         #pragma omp parallel for
         for (i=0;i<n;i++){
    S3:    C(i)=2*B(i)
         }
```

Apply loop alignment to

```
           for (i=0;i<n;i++){
     S1:     B(i)=D(i)
     S2:     A(i)=A(i)+B(i+1)
           }
```

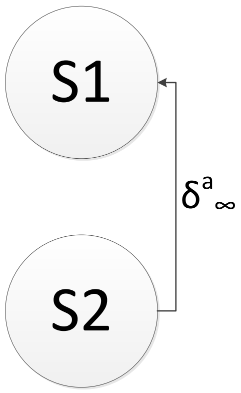


```
           for (i=-1;i<n;i++){
     S1:     if(i < n-1) B(i+1)=D(i+1)
     S2:     if(i >= 0)  A(i)=A(i)+B(i+1)
           }
```



Interchange statements to keep initial dependency.

```
           for (i=-1;i<n;i++){
     S2:     if(i >= 0)  A(i)=A(i)+B(i+1)
     S1:     if(i < n-1) B(i+1)=D(i+1)
           }
```



Peel off first and last loop iteration

```
     i = -1
     S2:     if(i >= 0)  A(i)=A(i)+B(i+1) //always false
     S1:     if(i < n-1) B(i+1)=D(i+1)  //always true

           for (i=0;i<n-1;i++){
     S2:     if(i >= 0)  A(i)=A(i)+B(i+1) // always true
     S1:     if(i < n-1) B(i+1)=D(i+1)  // always true    }

     i = n-1 (n-1 is the last iteration of i < n not n !!!)
     S2:     if(i >= 0)  A(i)=A(i)+B(i+1) //always true
     S1:     if(i < n-1) B(i+1)=D(i+1)  // always false
```

Delete unnecessary if conditions and use loop counter (i) inside arrays

```
     B(0)=D(0)
           for (i=0;i<n-1;i++){
     S2:     A(i)=A(i)+B(i+1)
     S1:     B(i+1)=D(i+1)     }
     A(n-1)=A(n-1)+B(n-1 + 1)
```

There is also a second solution. Instead of shifting statement S1 to the right, S1 is shifted to the left. The code below shows you the different solutions. The code below is available as source on the homepage.

```
// compile with: gcc -std=c99 -fopenmp -o loop_alignment loop_alignment.c

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>

#define N 30

void unaligned(int *a, int *b, int *d, int n)
{
	for(int i=0; i < n; i++)
	{
		b[i] = d[i];
		a[i] = a[i] + b[i + 1];
	}
}

void aligned_right_shift(int *a, int *b, int *d, int n)
{
	b[0] = d[0];

	#pragma omp parallel for
	for(int i=0; i < n-1; i++)
	{
		a[i] = a[i] + b[i + 1];
		b[i + 1] = d[i + 1];
	}

	a[n-1] = a[n-1] + b[n-1 + 1];
}

void aligned_left_shift(int *a, int *b, int *d, int n)
{
	b[0] = d[0];

	#pragma omp parallel for
	for(int i=1; i < n; i++)
	{
		a[i-1] = a[i-1] + b[i];
		b[i] = d[i];
	}

	a[n-1] = a[n-1] + b[n-1 + 1];
```

```
}

void print_array(int *array, int n)
{
	for(int i=0; i<n; i++)
	{
		printf("%d2 ", array[i]);
	}
	printf("\n");
}

int main ()
{
	int a0[N+1];
	int b0[N+1];
	int d0[N+1];

	int a1[N+1];
	int b1[N+1];
	int d1[N+1];

	int a2[N+1];
	int b2[N+1];
	int d2[N+1];

	int random;

	/* initialize random seed: */
	srand ( time(NULL) );

	for(int i=0; i<N+1; i++)
	{
		random = rand() % 10;
		a0[i] = random;
		a1[i] = random;
		a2[i] = random;

		random = rand() % 10;
		b0[i] = random;
		b1[i] = random;
		b2[i] = random;

		random = rand() % 10;
		d0[i] = random;
		d1[i] = random;
		d2[i] = random;
	}
	unaligned(a0, b0, d0, N);
	aligned_right_shift(a1, b1, d1, N);
	aligned_left_shift(a2, b2, d2, N);

	printf("a0, a1, a2\n");
	print_array(a0, N+1);
	print_array(a1, N+1);
	print_array(a2, N+1);

	printf("\nb0, b1, b2\n");
	print_array(b0, N+1);
	print_array(b1, N+1);
	print_array(b2, N+1);

	printf("\nd0, d1, d2\n");
	print_array(d0, N+1);
	print_array(d1, N+1);
	print_array(d2, N+1);

}
```

```
user@pc:~$ export OMP_NUM_THREADS=4
user@pc:~$ ./loop_alignment
a0, a1, a2
112 12 32 82 12 172 112 82 82 32 62 112 82 102 92 102 172 92 142 52 82 92 32 62 152 122 12 142 112 52 72
112 12 32 82 12 172 112 82 82 32 62 112 82 102 92 102 172 92 142 52 82 92 32 62 152 122 12 142 112 52 72
112 12 32 82 12 172 112 82 82 32 62 112 82 102 92 102 172 92 142 52 82 92 32 62 152 122 12 142 112 52 72

b0, b1, b2
22 72 62 82 02 62 12 22 72 52 72 02 22 52 52 52 52 22 02 42 62 12 72 52 42 62 62 52 32 32 42
22 72 62 82 02 62 12 22 72 52 72 02 22 52 52 52 52 22 02 42 62 12 72 52 42 62 62 52 32 32 42
22 72 62 82 02 62 12 22 72 52 72 02 22 52 52 52 52 22 02 42 62 12 72 52 42 62 62 52 32 32 42

d0, d1, d2
22 72 62 82 02 62 12 22 72 52 72 02 22 52 52 52 52 22 02 42 62 12 72 52 42 62 62 52 32 32 52
22 72 62 82 02 62 12 22 72 52 72 02 22 52 52 52 52 22 02 42 62 12 72 52 42 62 62 52 32 32 52
22 72 62 82 02 62 12 22 72 52 72 02 22 52 52 52 52 22 02 42 62 12 72 52 42 62 62 52 32 32 52
```