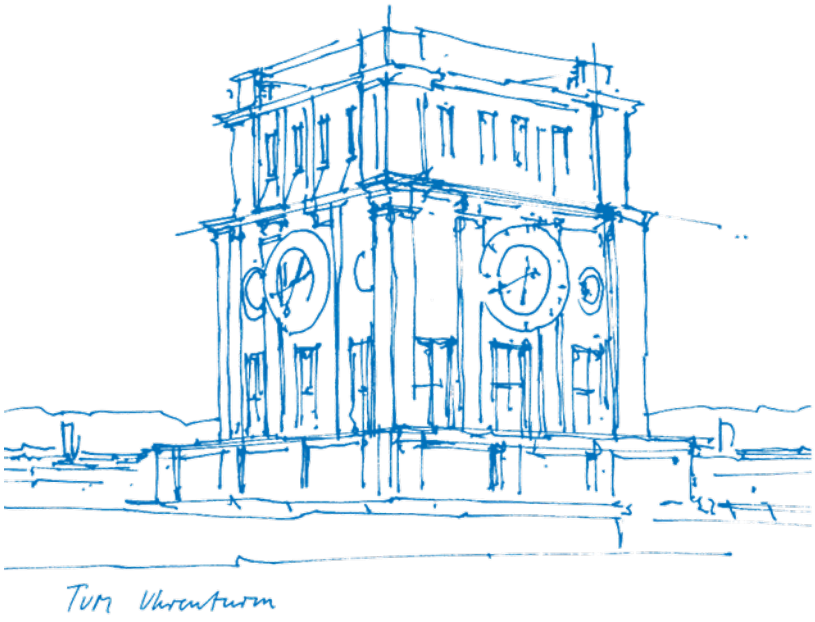
Parallel Programming Tutorial - Loop Transformations

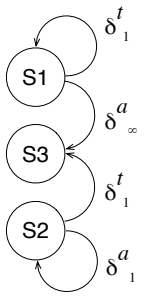
M.Sc. Andreas Wilhelm
Technichal University Munich
June 13, 2016



Solution for Assignment 6

Assignment 6 - Exercise 1

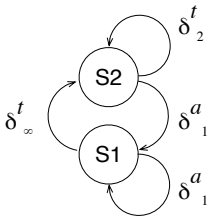
```
    for (i=1; i<n; i++) {
S1:   A(i+1) = A(i) + 2*B(i)
S2:   C(i+1) = C(i+2)
S3:   B(i)   = C(i-1)
    }
```



Source	Sink	Type	Dist.Vector	Dir.Vector
S1:A(i+1)	S1:A(i)	True	(1)	(<)
S1:B(i)	S2:B(i)	Anti	(0)	(=)
S2:C(i+1)	S3:C(i-1)	True	(2)	(<)
S2:C(i+2)	S2:C(i+1)	Anti	(1)	(<)

Assignment 6 - Exercise 2

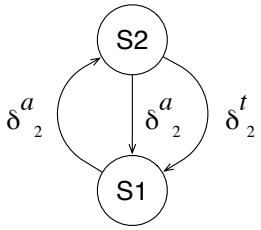
```
  for (i=1; i<n; i++) {
    for (j=1; j<m; j++) {
S1:   B(i-1,j+1) = B(i,i+2) + A(i,j+1)
S2:   A(i,j+1)   = A(i,j) * B(i+1,j)
    }
  }
```



Source	Sink	Type	Dist.Vector	Dir.Vector
S2:A(i,j+1)	S2:A(i,j)	True	(0,1)	(=,<)
S2:A(i,j+1)	S1:A(i,j+1)	True	(0,0)	(=,=)
S2:B(i+1,j)	S1:B(i-1,j+1)	Anti	(2,-1)	(<,>)
S1:B(i,j+2)	S1:B(i-1,j+1)	Anti	(1,1)	(<,<)

Assignment 6 - Exercise 3

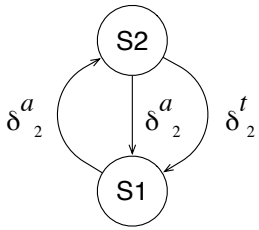
```
  for (i=1; i<3; i++) {
    for (j=1; j<3; j++) {
S1:   B(2*i,j) = A(i,3-j)
S2:   A(i,j)   = B(i+2,j+1)
    }
  }
```

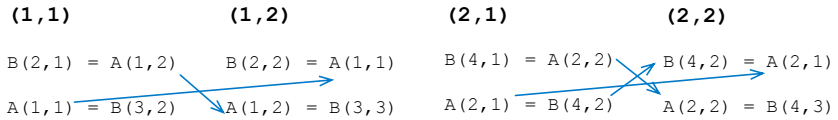


Source	Sink	Type	Dist.Vector	Dir.Vector
S2:A(i,j)	S1:A(i,3-j)	True	(0,1)	(=,<)
S1:A(i,3-j)	S2:A(i,j)	Anti	(0,1)	(=,<)
S2:B(i+2,j+1)	S1:B(2*i,j)	Anti	(0,1)	(=,<)

Assignment 6 - Exercise 3

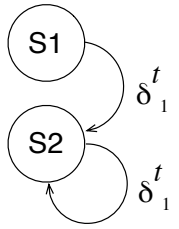
```
  for (i=1; i<3; i++) {
    for (j=1; j<3; j++) {
S1:   B(2*i,j) = A(i,3-j)
S2:   A(i,j)   = B(i+2,j+1)
    }
  }
```





Additional Exercise 1

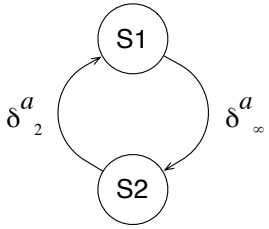
```
  for (i=1; i<n; i++) {
    for (j=1; j<m; j++) {
S1:   A(i,j) = B(i,j)
S2:   C(i,j) = C(i-1, j) + A(i-1,j+1)
    }
  }
```



Source	Sink	Type	Dist.Vector	Dir.Vector
S1:A(i,j)	S2:A(i-1,j+1)	True	(1,-1)	(<,>)
S2:C(i,j)	S2:C(i-1,j)	True	(1,0)	(<,=)

Additional Exercise 2

```
  for (i=2; i<n; i++) {
    for (j=2; j<m; j++) {
S1:   A(i+1,j) = B(2*i,j)
S2:   B(2*i,j) = C(i-1, j) + A(i+1,j+1)
    }
  }
```



Source	Sink	Type	Dist.Vector	Dir.Vector
S2:A(i+1,j+1)	S1:A(i+1,j)	Anti	(0,1)	(=,<)
S1:B(2*i,j)	S2:B(2*i,j)	Anti	(0,0)	(=,=)

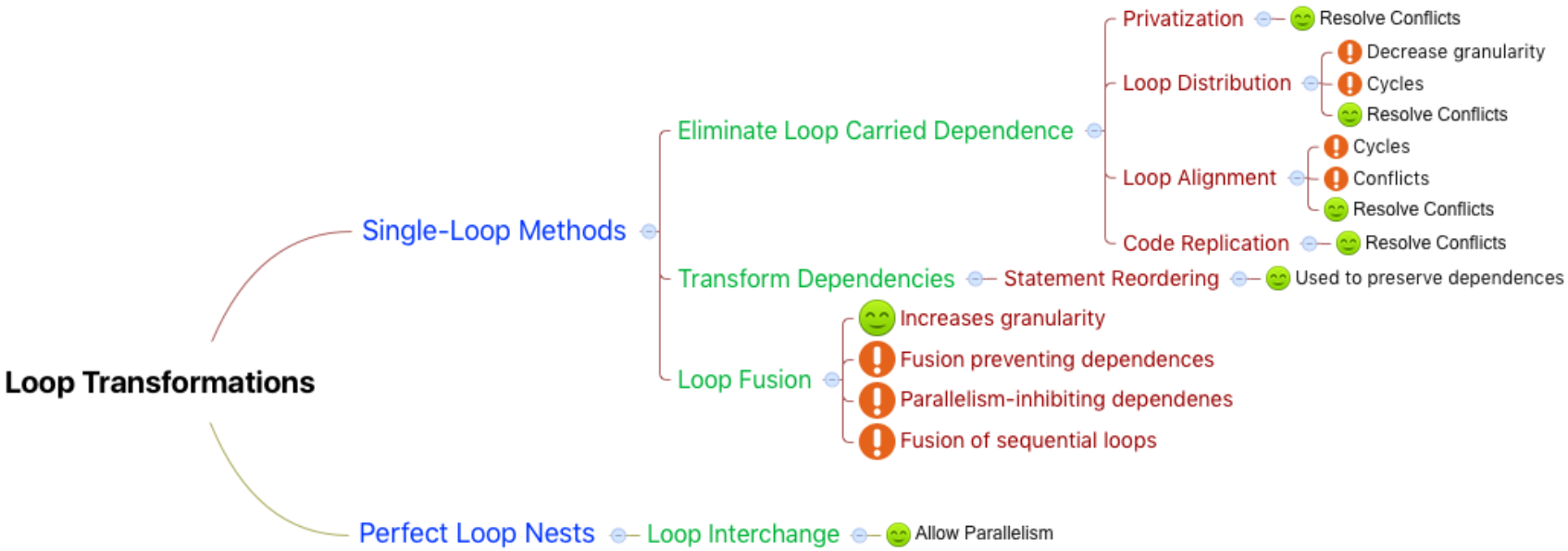
Loop Transformations

Transformations

Theorem

Any reordering transformation that preserves every dependence in a program preserves the meaning of that program.

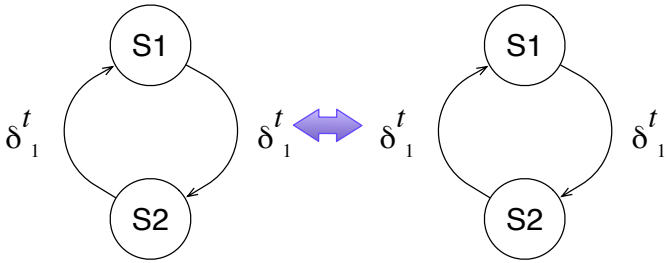
Transformations - Mindmap



Statement Reordering

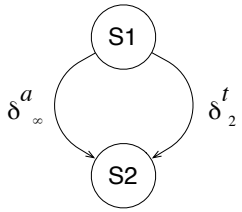
```
  for (i=1; i<10; i++) {
S1:   A(i+1) = F(i)
S2:   F(i+1) = A(i)
  }
```

```
  for (i=1; i<10; i++) {
S2:   F(i+1) = A(i)
S1:   A(i+1) = F(i)
  }
```



Loop Distribution I

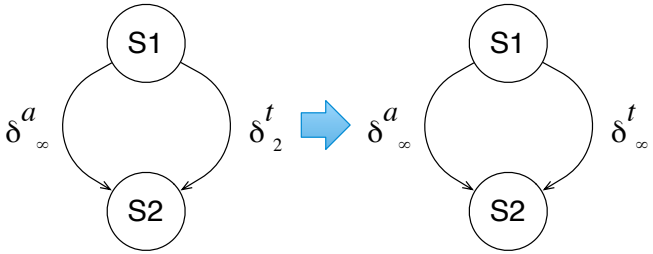
```
  for (i=1; i<n; i++) {
    for (j=1; j<m; j++) {
S1:   A(i,j) = B(i,j)
S2:   B(i,j) = A(i,j-1)
    }
  }
```



Loop Distribution I

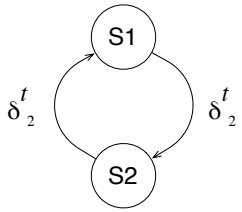
```
  for (i=1; i<n; i++) {
    for (j=1; j<m; j++) {
S1:   A(i,j) = B(i,j)
S2:   B(i,j) = A(i,j-1)
    }
  }
```

```
  for (i=1; i<n; i++) {
    for (j=1; j<m; j++) {
S1:   A(i,j) = B(i,j)
    }
    for (j=1; j<m; j++) {
S2:   B(i,j) = A(i,j-1)
    }
  }
```



Loop Distribution II - Cycle

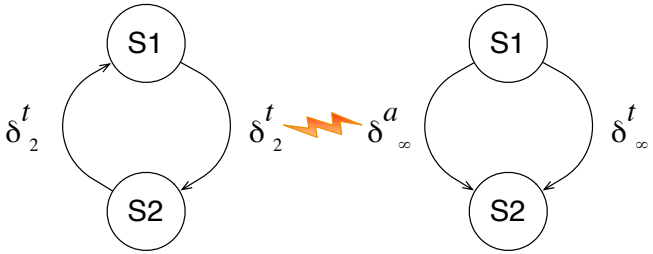
```
  for (i=1; i<n; i++) {
    for (j=1; j<m; j++) {
S1:   A(i,j) = B(i,j)
S2:   B(i,j+1) = A(i,j-1)
    }
  }
```



Loop Distribution II - Cycle

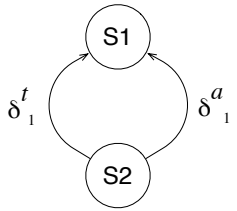
```
  for (i=1; i<n; i++) {
    for (j=1; j<m; j++) {
S1:   A(i,j) = B(i,j)
S2:   B(i,j+1) = A(i,j-1)
    }
  }
```

```
1    for (i=1; i<n; i++) {
2      for (j=1; j<m; j++) {
3  S1:   A(i,j) = B(i,j)
4      }
5      for (j=1; j<m; j++) {
6  S2:   B(i,j+1) = A(i,j-1)
7      }
8    }
```



Loop Alignment I

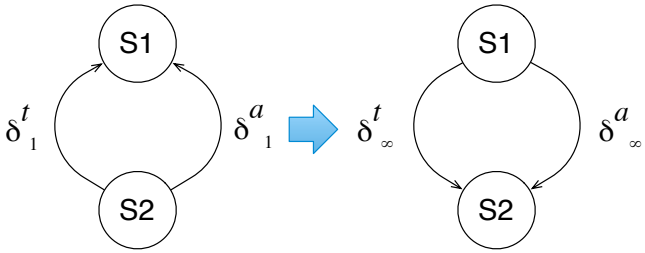
```
  for (i=1; i<n; i++) {
S1:   A(i)   = B(i)
S2:   B(i+1) = A(i+1)
  }
```



Loop Alignment I

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

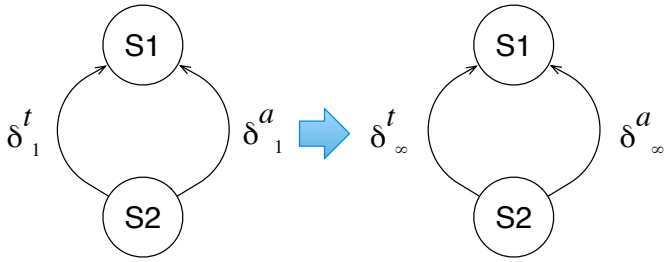
```
1    for (i=1; i<n+1; i++) {
2  S1:   if (i<n) A(i) = B(i)
3  S2:   if (i>1) B(i) = A(i)
4    }
```



Loop Alignment I - Peeling Off Executions

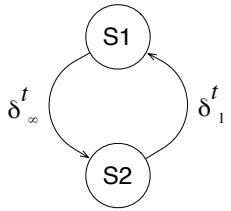
```
  for (i=1; i<n; i++) {
S1:   A(i)   = B(i)
S2:   B(i+1) = A(i+1)
  }
```

```
1    A(1) = B(1)
2    for (i=2; i<n; i++) {
3  S2:   B(i) = A(i)
4  S1:   A(i) = B(i)
5    }
6    B(n) = A(n)
```



Loop Alignment II - Cycle

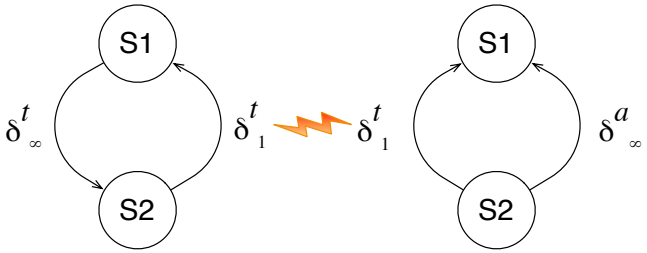
```
  for (i=1; i<n; i++) {
S1:   A(i)   = B(i)
S2:   B(i+1) = A(i)
  }
```



Loop Alignment II - Cycle

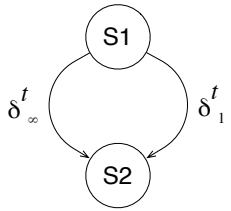
```
  for (i=1; i<n; i++) {
S1:   A(i)   = B(i)
S2:   B(i+1) = A(i)
  }
```

```
1    for (i=1; i<n+1; i++) {
2  S1:   if (i<n) A(i) = B(i)
3  S2:   if (i>1) B(i) = A(i-1)
4    }
```



Loop Alignment III - Conflict

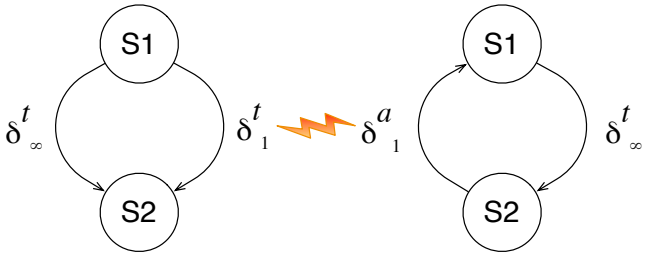
```
  for (i=1; i<n; i++) {
S1: A(i) = B(i)
S2: C(i) = A(i) + A(i-1)
  }
```



Loop Alignment III - Conflict

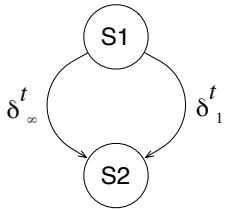
```
  for (i=1; i<n; i++) {
S1: A(i) = B(i)
S2: C(i) = A(i) + A(i-1)
  }
```

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



Code Replication

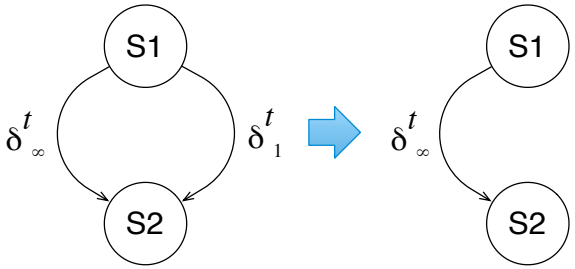
```
    for (i=1; i<n; i++) {
S1:   A(i) = B(i)
S2:   C(i) = A(i) + A(i-1)
    }
```



Code Replication

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

```
    for (i=1; i<n; i++) {
      private(T)
S1:   A(i)   = B(i)
      if (i=1) T = A(0)
      else     T = B(i-1)
S2:   C(i) = A(i) + T
    }
```



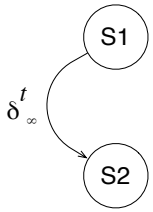
Transformations

Theorem

Alignment, replication, and statement reordering are sufficient to eliminate all carried dependences in a single loop that contains no cycles and in which the distance of each dependence is a constant independent of the loop index.

Loop Fusion I

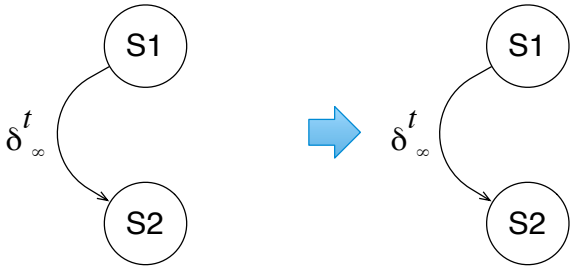
```
    for (i=1; i<n; i++) {
S1:   A(i) = B(i+1)
    }
    for (i=1; i<n; i++) {
S2:   C(i) = A(i) + B(i)
    }
```



Loop Fusion I

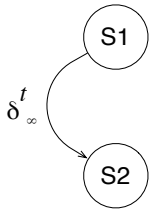
```
    for (i=1; i<n; i++) {
S1:   A(i) = B(i+1)
    }
    for (i=1; i<n; i++) {
S2:   C(i) = A(i) + B(i)
    }
```

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



Loop Fusion II - Fusion preventing Dependency

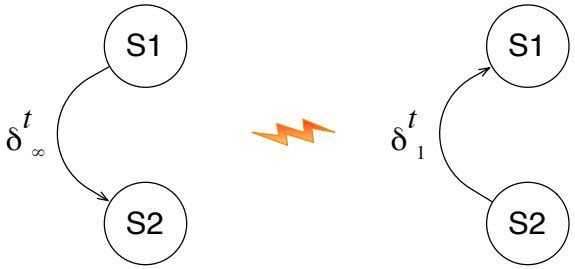
```
    for (i=1; i<n; i++) {
S1:   A(i) = B(i+1)
    }
    for (i=1; i<n; i++) {
S2:   C(i) = A(i+1) + B(i)
    }
```



Loop Fusion II - Fusion preventing Dependency

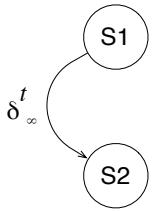
```
    for (i=1; i<n; i++) {
S1:   A(i) = B(i+1)
    }
    for (i=1; i<n; i++) {
S2:   C(i) = A(i+1) + B(i)
    }
```

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



Loop Fusion III - Parallelism inhibiting Dependency

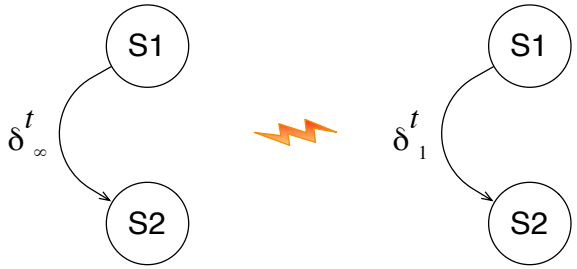
```
    for (i=1; i<n; i++) {
S1:   A(i+1) = B(i+1)
    }
    for (i=1; i<n; i++) {
S2:   C(i) = A(i) + B(i)
    }
```



Loop Fusion III - Parallelism inhibiting Dependency

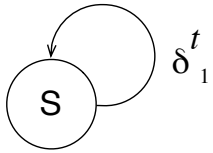
```
    for (i=1; i<n; i++) {
S1:   A(i+1) = B(i+1)
    }
    for (i=1; i<n; i++) {
S2:   C(i) = A(i) + B(i)
    }
```

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

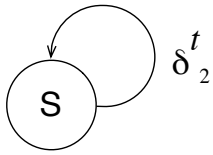


Loop Interchange

```
  for (i=1; i<n; i++) {
    for(j=1; j<m; j++) {
S:    A(i+1,j) = A(i,j) + B(i,j)
    }
  }
```



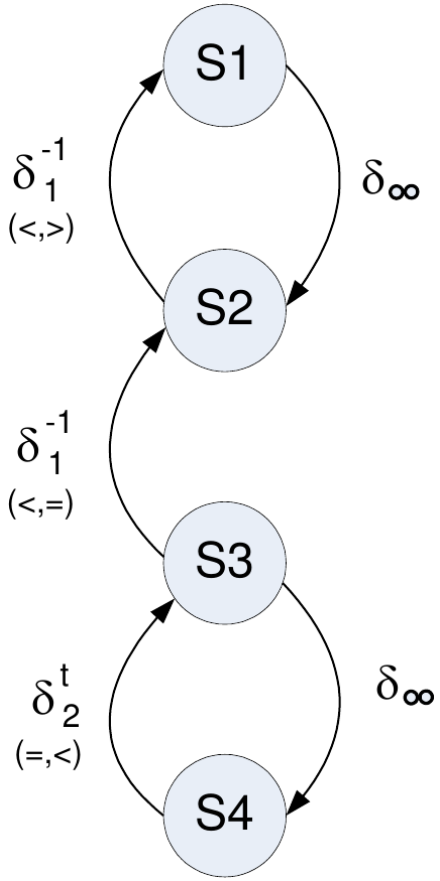

```
  for (j=1; j<m; j++) {
    for(i=1; i<n; i++) {
S:    A(i+1,j) = A(i,j) + B(i,j)
    }
  }
```



Exam 2010 Q4

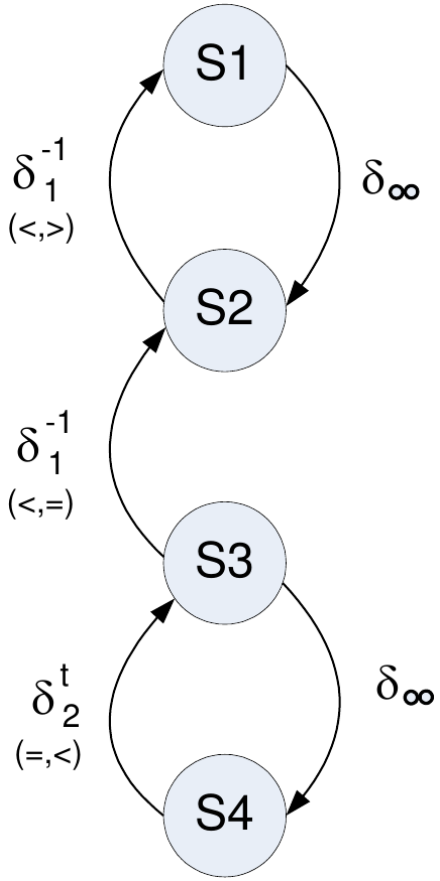
Apply loop distribution to the following loop nest. Distribute it as far as possible.Other transformations might help you. Mark loops with OMP FOR that can be run in parallel.

```
    for (i=...)
      for (j=...)
      {
S1:   ...
S2:   ...
S3:   ...
S4:   ...
      }
```



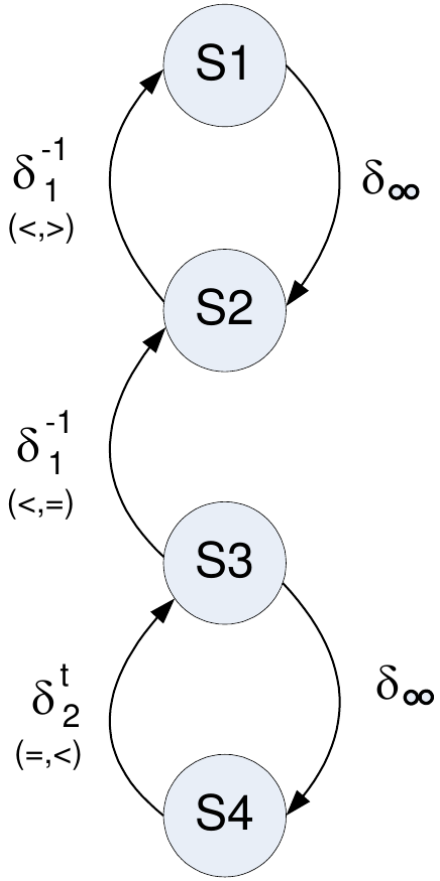
Exam 2010 Q4: Solution Step 1

```
    for (i=...)
      for (j=...)
      {
S3:   ...
S4:   ...
      }

    for (i=...)
      for (j=...)
      {
S1:   ...
S2:   ...
      }
```



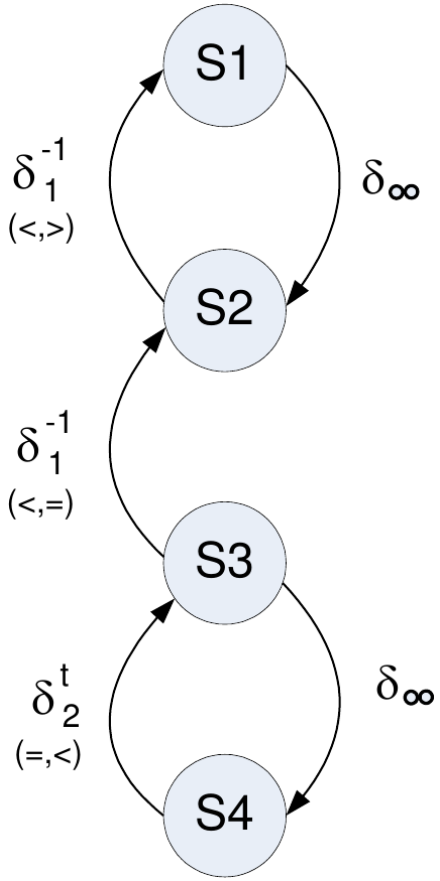
Exam 2010 Q4: Solution Step 2

```
    for (j=...)
      for (i=...)
      {
S3:   ...
S4:   ...
      }

    for (i=...)
      for (j=...)
      {
S1:   ...
S2:   ...
      }
```



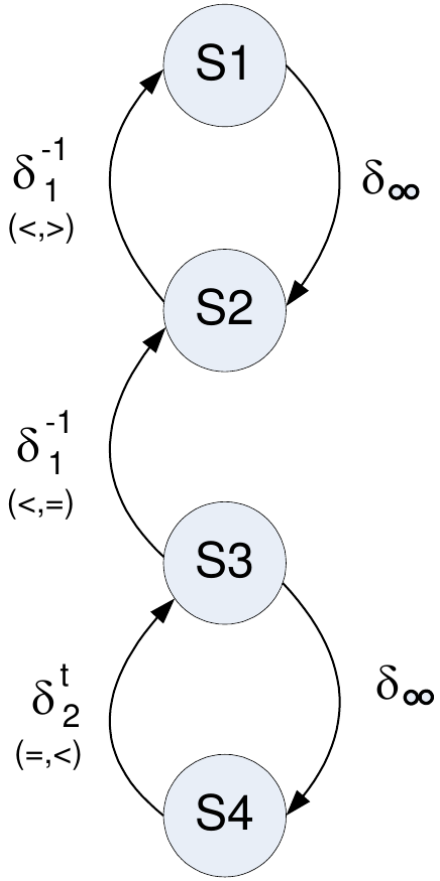
Exam 2010 Q4: Solution Step 3

```
    for (j=...)
    {
      for (i=...)
S3:   ...
      for (i=...)
S4:   ...
    }

    for (i=...)
      for (j=...)
      {
S1:   ...
S2:   ...
      }
```



Exam 2010 Q4: Solution Step 4

```
    for (j=...)
    {
      #pragma omp parallel for
      for (i=...)
S3:   ...
      #pragma omp parallel for
      for (i=...)
S4:   ...
    }
    for (i=...)
      for (j=...)
      {
S1:   ...
S2:   ...
      }
```



Exam 2012 Q7

Transform the following loop into a parallel loop using loop alignment. Do not distribute the loop. The generated code should not have if-statements in the loop body.

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

Exam 2012 Q7: Solution Step 1

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

Shift S1 right: i+1

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

Exam 2012 Q7: Solution Step 2

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

Shift S1 right: i+1

```
      i=1
S1:   if(i < n-1) B(i+1) = A(i+1)
S2:   if(i > 1  ) C(i) = C(i) + B(i+1)
      for(i=2; i<n-1; i++) {
S1:   if(i < n-1) B(i+1) = A(i+1)
S2:   if(i > 1  ) C(i) = C(i) + B(i+1)
      }
      i=n-1
S1:   if(i < n-1) B(i+1) = A(i+1)
S2:   if(i > 1  ) C(i) = C(i) + B(i+1)
```

Exam 2012 Q7: Solution Step 3

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

Shift S1 right: i+1

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

Exam 2012 Q7: Solution Step 4

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

Shift S1 right: i+1

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

Exam 2012 Q7: Solution Step 4

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

Shift S1 right: i+1

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

Assignment 7 -9

Assignment 7-9: Loop Transformations

1. Assignment 7
   - Apply loop distribution to the loop in `loop_fission_seq.c`
   - Distribute it as far as possible, other transformations may help
   - Parallelize the loop with OpenMP in `loop_fission_par.c` and upload it
2. Assignment 8
   - Apply loop alignment to the loop in `loop_alignment_seq.c`
   - Do not distribute the loop
   - Parallelize with OpenMP in `loop_alignment_par.c` and upload it
3. Assignment 9
   - Apply loop fusion to the loop in `loop_fusion_seq.c`
   - Parallelize the loop with OpenMP in `loop_fusion_par.c` and upload it