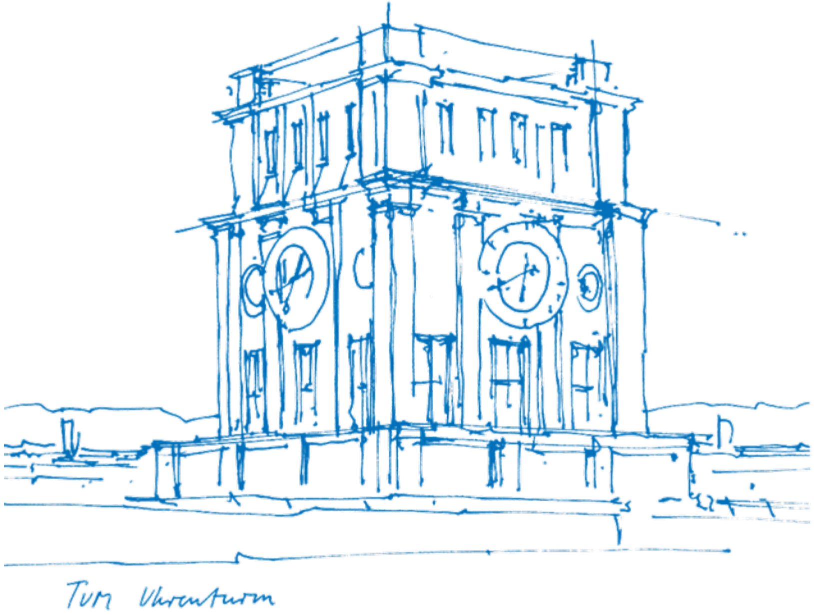
Parallel Programming Tutorial -Dependency Analysis

M.Sc. Andreas Wilhelm
Technichal University Munich
June 19, 2017



Solution for Assignment 5

Solution for Assignment 5 - Sections

- Parallelize the companytree algorithm with OpenMP sections
- Set the correct number of threads
  - use the runtime function `omp_set_num_threads`
- Enable nested parallelism
  - use the runtime function `omp_set_nested`
- Avoid oversubscription
  - Option 1: Use conditional parallelism by utilizing `omp_get_level`
  - Option 2: Use `omp_set_max_active_levels`

```c
#define MAX_NESTING_LEVEL 8

void parallel_traverse(tree *node) {
  if (node != NULL) {
    node->work_hours = compute_workHours(node->data);
    top_work_hours[node->id] = node->work_hours;

    #pragma omp parallel sections {
      #pragma omp section
      parallel_traverse(node->right);
      #pragma omp section
      parallel_traverse(node->left);
    }
  }
}

void traverse(tree *node, int numThreads) {
  omp_set_num_threads( numThreads );
  omp_set_nested( 1 );
  omp_set_max_active_levels( MAX_NESTING_LEVEL );

  parallel_traverse(node);
}
```

Solution for Assignment 5 - Tasks

- Parallelize the companytree algorithm with OpenMP tasks
- Set the correct number of threads
  - use the clause `num_threads`
- Use a single thread to start the algorithm
- Optimization: Create only one task per recursion

```
void parallel_traverse(tree *node) {
        if (node != NULL) {

          #pragma omp task
          parallel_traverse(node->right);
          parallel_traverse(node->left);

          node->work_hours = compute_workHours(node->data);
          top_work_hours[node->id] = node->work_hours;
  }
}

void traverse(tree *node, int numThreads) {
        #pragma omp parallel num_threads( numThreads )
        {
                #pragma omp single
                parallel_traverse(node);
        }
}
```

(Data) Dependency Analysis

Dependence Notation

- S1 and S2 are statements

Type	Meaning	Symbol	Alternative Symbols	Example
True dependence	RAW	S1 δ^t S2	δ, δ^f	S1: x=1 S2: y=x
Antidependence	WAR	S1 δ^a S2	δ^{-1}	S1: y=x S2: x=1
Output dependence	WAW	S1 δ^o S2		S1: x=1 S2: x=2

- RAW = ”read after write“
- WAR = ”write after read“
- WAW = ”write after write“

Iteration Vector

```
for (i1 = 1; i1 < N1; i1++) {
   for (i2 = 1; i2 < N2; i2++) {
      ...
      for(in = 1; in < Nn; in++) {
         S:      ...
      }
      ...
   }
}
```

- The iteration vector for a statement S in the loop is given by $\vec{i} = (i_1, i_2, \ldots, i_n)$ where i_k, $(1 \leq \mathrm{k} \leq n)$, represents the iteration number for the loop at nesting level k.
- The set of all possible iteration vectors for S is called *iteration space*.

Iteration Vector - Example

```
for (i = 1; i < 3; i++) {
   for (j = 1; j < 4; j++) {
      S:  ...
   }
}
```

- The iteration space of statement S is $\{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3)\}$

Data Dependence

Informal Definition

There is a data dependence from statement S_1 to statement S_2 (S_2 dependes on S_1), if and only if (1) both statements access the same memory location and at least one of them writes to it, and (2) there is a feasible run-time execution path from S_1 to S_2.

Formal Definition

$\exists M, S_1, S_2, \vec{i}, \vec{j}$:

1. $(\vec{i} < \vec{j})$[1]or $(\vec{i} = \vec{j}$ and there is a path from S_1 to S_2) [2][3]
2. S_1 and S_2 access M on $\vec{i}$ and $\vec{j}$, respectively
3. One of these accesses is a write

[1] called *loop-carried dependence*

[2] called *loop-independent dependence*

[3] The operations $<$ and $=$ are defined componentwise from left to right.

Distance Vector

Definition

- Suppose there is a dependence from statement S_1 on iteration $\overrightarrow{i}$ of a loop nest to statement S_2 on iteration $\overrightarrow{j}$
- The distance vector is defined as $d(\overrightarrow{i}, \overrightarrow{j})_k = \left[d(\overrightarrow{i}, \overrightarrow{j})_1, \ldots, d(\overrightarrow{i}, \overrightarrow{j})_N\right]$,
  where $d(\overrightarrow{i}, \overrightarrow{j})_k = j_k - i_k$.

Example

The distance vector for the dependence $S[(2,2,2)]\ \delta^t\ S[(3,1,2)]$ of the following loop nest is $(1,-1,0)$.

```
for (i = 1; i < N; i++) {
   for (j = 1; j < M; j++) {
      for (k = 1; k < L; k++) {
         S:    A(i + 1, j - 1, k) = A(i,j,k)
      }
   }
}
```

Direction Vector

Definition

- Suppose there is a dependence from statement S_1 on iteration $\overrightarrow{i}$ of a loop nest to statement S_2 on iteration $\overrightarrow{j}$
- Direction vector $D(\overrightarrow{i}, \overrightarrow{j})_k = \begin{cases} \text{``}<\text{''}, & d(i,j)_k > 0 \\ \text{``}=\text{''}, & d(i,j)_k = 0 \\ \text{``}>\text{''}, & d(i,j)_k < 0 \end{cases}$

Example

The direction vector for the dependence $S[(2,2,2)]\ \delta^t\ S[(3,1,2)]$ of the following example is $(<,>,=)$.

```
for (i = 1; i < N; i++) {
   for (j = 1; j < M; j++) {
      for (k = 1; k < L; k++) {
         S:    A(i + 1,j - 1,k) = A(i,j,k)
      }
   }
}
```

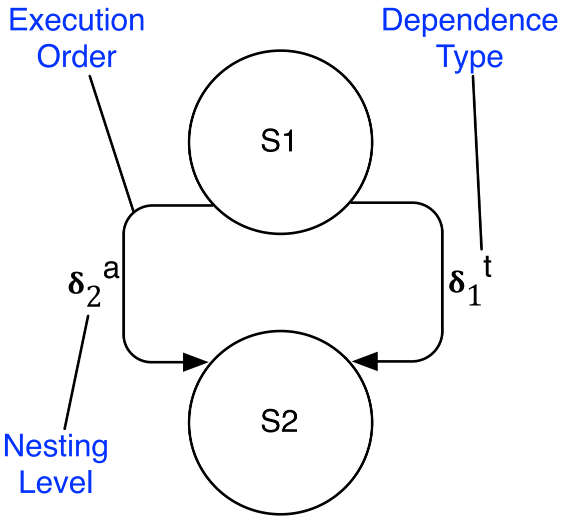
The **level** of a loop-carried dependence is the index of the leftmost non-“$=$” of $D(i,j)$.

Dependence Graphs

- Nodes: The statements of a program
- Edges: The dependences between the statements from the first executed statement to the following one
- Each edge is labeled with the dependence type and the nesting level

Example

```
for (i = 1; i < N; i++) {
   for (j = 1; j < M; j++) {
      S1:    A(i + 1,j) = B(i,j + 1)
      S2:    B(i,j) = A(i,j)
   }
}
```



Example 1

- Give the dependence graph for the following loop.

```
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)
}
```

- Give the distance and direction vectors for the loop-carried dependencies.

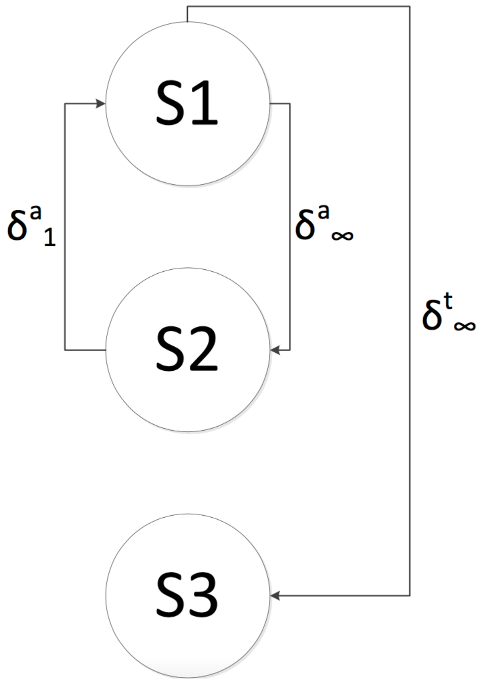
Source	Sink	Dep.Type	Dist. Vector	Dir. Vector	
...	...	...	...	...	...

- Source, Sink: Specify the references in the form *S1:B(i)*
- Type: Loop-independent (l-i) or loop-carried dependence (l-c)
- Dep.Type: True-, Anti-, or Output-Dependence
- Vectors: n-Tuples where n is the depth of the loop nest

Solution for Example 1

```
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)
}
```

Source	Sink	Dep.Type	Dist. Vector	Dir. Vector
S2: B(i + 1)	S1: B(i)	a	(1)	(<)



Example 2

- Give the dependence graph for the following loop.

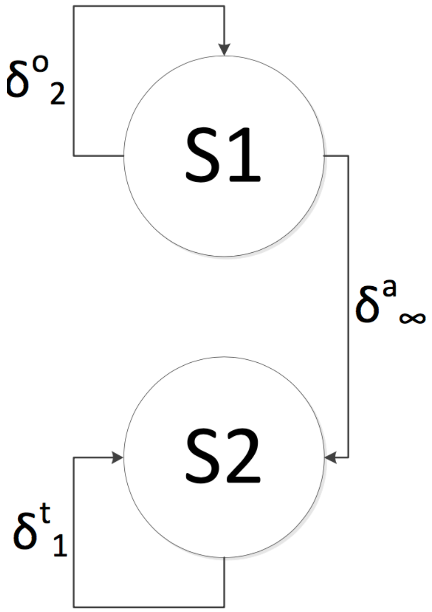
```
for (i = 1; i < N; i++) {
   for (j = 1; j < M; j++) {
      S1:   A(i)   = B(i,j)
      S2:   B(i,j) = B(i - 1,2 * j)
   }
}
```

- Give the distance and direction vectors for the dependencies.

Solution for Example 2

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

Source	Sink	Dep.Type	Dist. Vector	Dir. Vector
S1: A(i)	S1: A(i)	o	(0,*)	(=, *)
S2: B(i, j)	S2: B(i-1, 2*j)	t	(1,-j)	(<, >)



Example 3

- Give the dependence graph for the following loop.

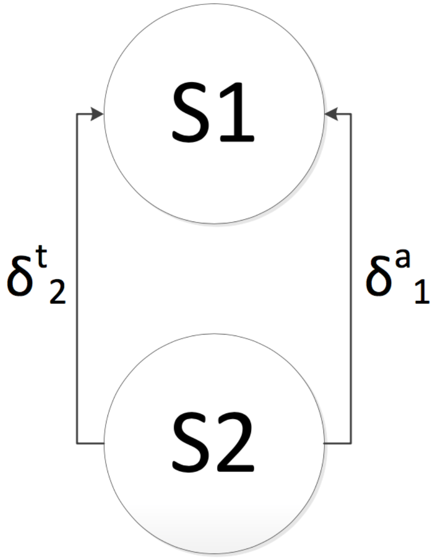
```
for (i = 0; i < N; i++) {
   for (j = 0; j < M; j++) {
      S1:   B(i - 1,j) = C(i,j - 2)
      S2:   C(i,j)    = 2 * B(i,j + 1)
   }
}
```

- Give the distance and direction vectors for the loop-carried dependencies.

Solution for Example 3

```
for (i = 0; i < N; i++) {
    for (j = 0; j < M; j++) {
        S1:   B(i - 1,j) = C(i,j - 2)
        S2:   C(i,j)    = 2 * B(i,j + 1)
    }
}
```

Source	Sink	Dep.Type	Dist. Vector	Dir. Vector
S2: B(i, j+1)	S1: B(i-1, j)	a	(1,1)	(<, <)
S2: C(i, j)	S1: C(i, j-2)	t	(0,2)	(=, <)



Assignment 6

Assignment 6: Dependence Analysis

1. Download the exercise sheet in libreoffice calc format (.ods)
2. Open the exercise sheet (with enabled macros)
3. Find all dependences of the exercises and fill out the sheet
   - This time: fill out information about all dependences, not only the loop-carried dependences
4. Press the "Save as CSV" button to save the information to a .csv file on the same direcotry
5. Upload this .csv file at the submission page