Static Non-concurrency Analysis of OpenMP Programs

Yuan Lin
Sun Microsystems, Inc.

“What's Wrong with This Code?”

```
#pragma omp parallel for
for (i=0; i<n; i++) {
      for (j=0; j<n; j++) {
            a[i][j] = i+j;
      }
}
```

"What's Wrong with This Code?"

```
#pragma omp parallel for
for (i=0; i<n; i++) {
        for (j=0; j<n; j++) {
                a[i][j] = i+j;
        }
}
```

- j is 'shared'.
- The reads and writes of j by different threads may cause data races.
- The code may not produce the same result as its sequential version does.

Static OpenMP Error Checking in Sun Studio Compilers

- Static data race detection and scope checking
- Use the -vpara/-xvpara option

```
> cc -xopenmp -xO3 -xvpara t.c

"t.c", line 1: Warning: inappropriate scoping
   variable 'j' may be scoped inappropriately as 'shared'
   . read at line 3 and write at line 3 may cause
     data race
```

Concurrent Execution

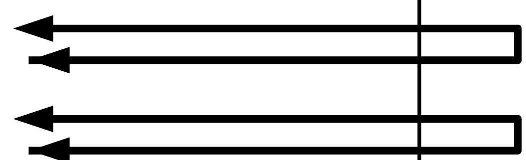
- Concurrency is where the execution order of two statements is not enforced.
- Non-concurrency is where the execution order of two statements is enforced.
- Concurrent execution is a necessary condition of causing data race.
- If two statements will never be executed concurrently, then they will not cause data race.

Examples: concurrent vs. non-concurrent

```
#pragma omp parallel
{
        a = 1;
        b = 2;
}
```

Examples: concurrent vs. non-concurrent

```
#pragma omp parallel
{
        a = 1;
        b = 2;
}
```



Examples: concurrent vs. non-concurrent

```
#pragma omp parallel
{
   a = 1;
   #pragma omp barrier
   b = 2;
}
```

Examples: concurrent vs. non-concurrent

```
#pragma omp parallel
{
    a = 1;
    #pragma omp barrier
    b = 2;
}
```

Examples: concurrent vs. non-concurrent

```
#pragma omp parallel
{
    a = 1;
    #pragma omp barrier
    b = 2;
}
```

Examples: concurrent vs. non-concurrent

```
#pragma omp parallel
{
   #pragma omp master
   a = 1;
   #pragma omp master
   b = 2;
}
```

Examples:
concurrent vs. non-concurrent

```
#pragma omp parallel
{
   #pragma omp master
   a = 1;
   #pragma omp master
   b = 2;
}
```

Examples: concurrent vs. non-concurrent

```
#pragma omp parallel
{
   #pragma omp single nowait
   a = 1;
   #pragma omp single
   b = 2;
}
```

Examples: concurrent vs. non-concurrent

```
#pragma omp parallel
{
   #pragma omp single nowait
   a = 1;
   #pragma omp single
   b = 2;
}
```

Goal

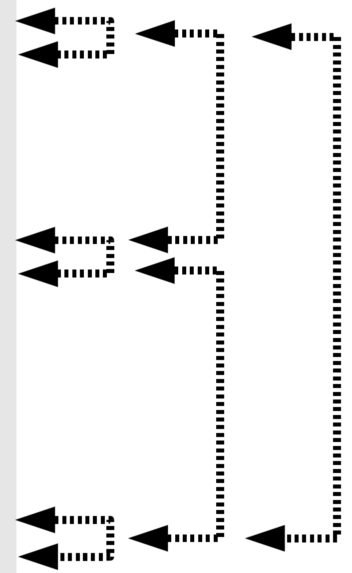
- Detect non-concurrency statically
  - > at compile time,
  - > whether two statements in a parallel construct
  - > will NOT be executed concurrently
  - > by different threads in the team for the parallel region.
- Allow underestimation of real non-concurrency
  - > When the method fails, the two statements may, but need not execute concurrently.

How to Detect Non-concurrency?

```
#pragma omp parallel
{
    a = ...;
    #pragma omp single
    {
      a = ...;
    }
    #pragma omp for
    for (i=0; i<n; i++){
        b[i] = a;
    }
}
```

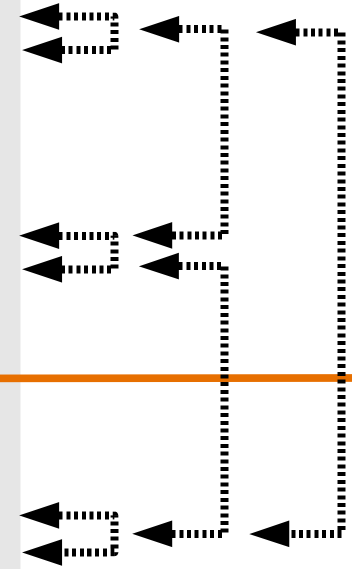
How to Detect Non-concurrency?

```
#pragma omp parallel
{
    a = ...;
    #pragma omp single
    {
      a = ...;
    }
    #pragma omp for
    for (i=0; i<n; i++){
        b[i] = a;
    }
}
```



How to Detect Non-concurrency?

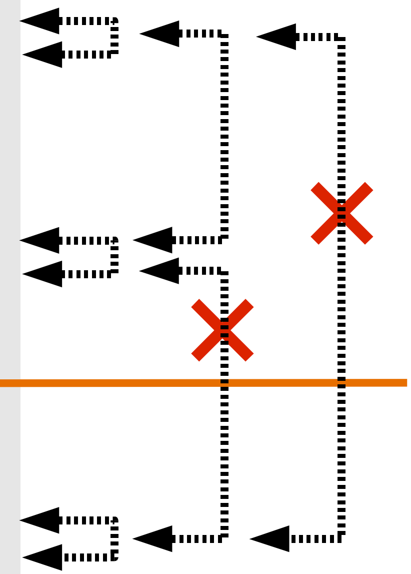
```
#pragma omp parallel
{
    a = ...;
    #pragma omp single
    {
      a = ...;
    }
    #pragma omp for
    for (i=0; i<n; i++){
        b[i] = a;
    }
}
```



Step 1: phase partitioning based on barriers

How to Detect Non-concurrency?

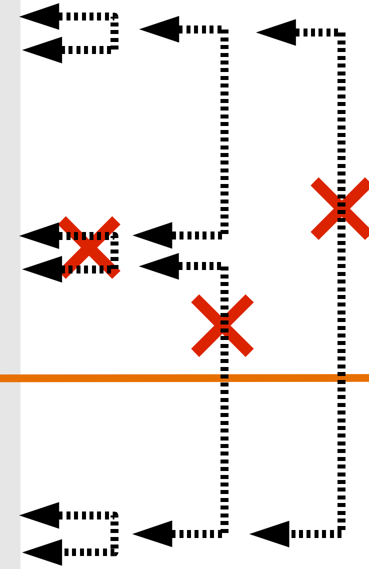
```
#pragma omp parallel
{
    a = ...;
    #pragma omp single
    {
      a = ...;
    }
    #pragma omp for
    for (i=0; i<n; i++){
        b[i] = a;
    }
}
```



Step 1: phase partitioning based on barriers

How to Detect Non-concurrency?

```
#pragma omp parallel
{
    a = ...;
    #pragma omp single
    {
      a = ...;
    }
    #pragma omp for
    for (i=0; i<n; i++){
        b[i] = a;
    }
}
```



Step 2: check within a phase based on the semantics of OMP constructs

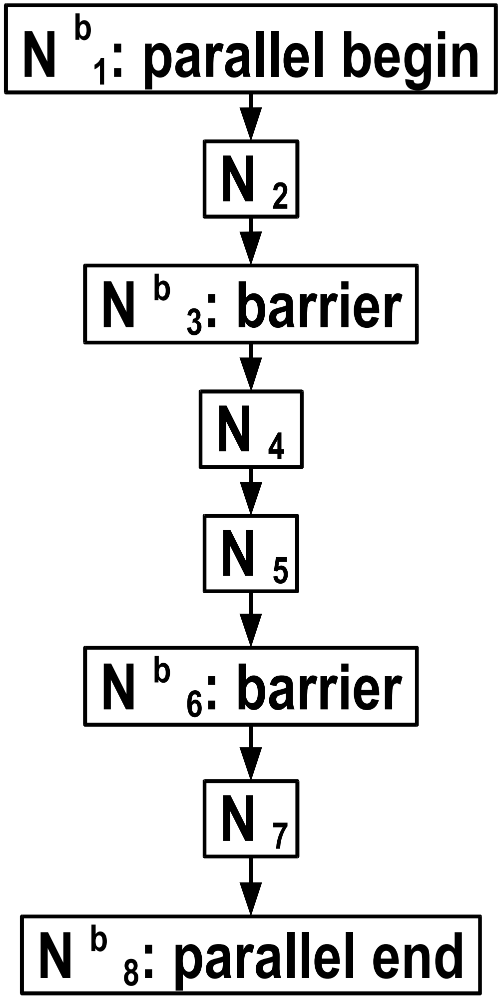
How to Detect Non-concurrency?

```
#pragma omp parallel
{
    a = ...;
    #pragma omp single
    {
      a = ...;
    }
    #pragma omp for
    for (i=0; i<n; i++){
          b[i] = a;
    }
}
```

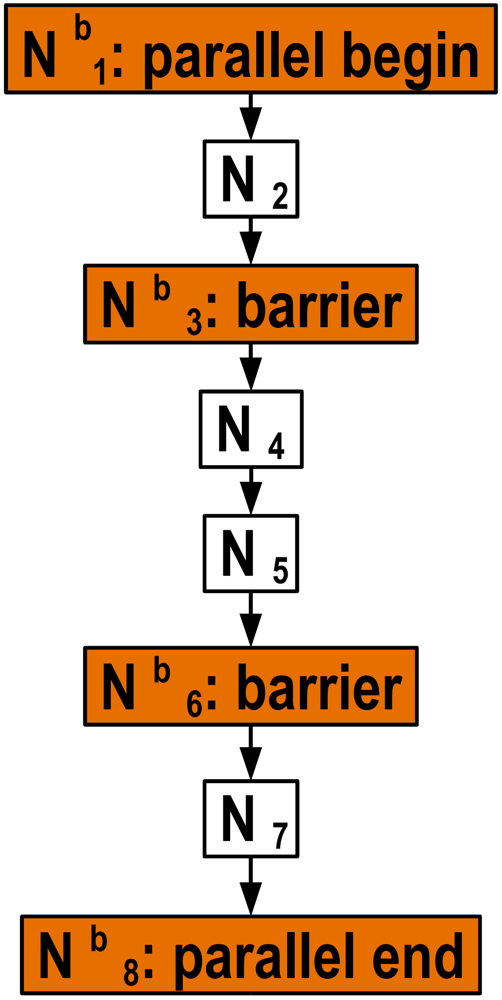
Two Steps

- Phase partitioning
  - > Two statements that are NOT in any common phase will not be executed concurrently.
- Detecting non-concurrency within a phase
  - > Use the semantics of OMP constructs to decide whether two statements within a phase will be executed concurrently.

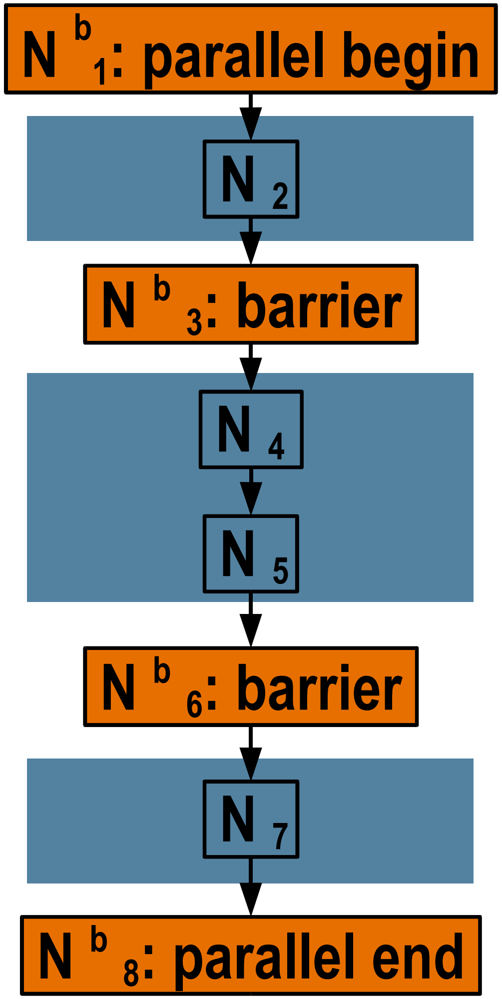
Phase Partitioning - Example 1



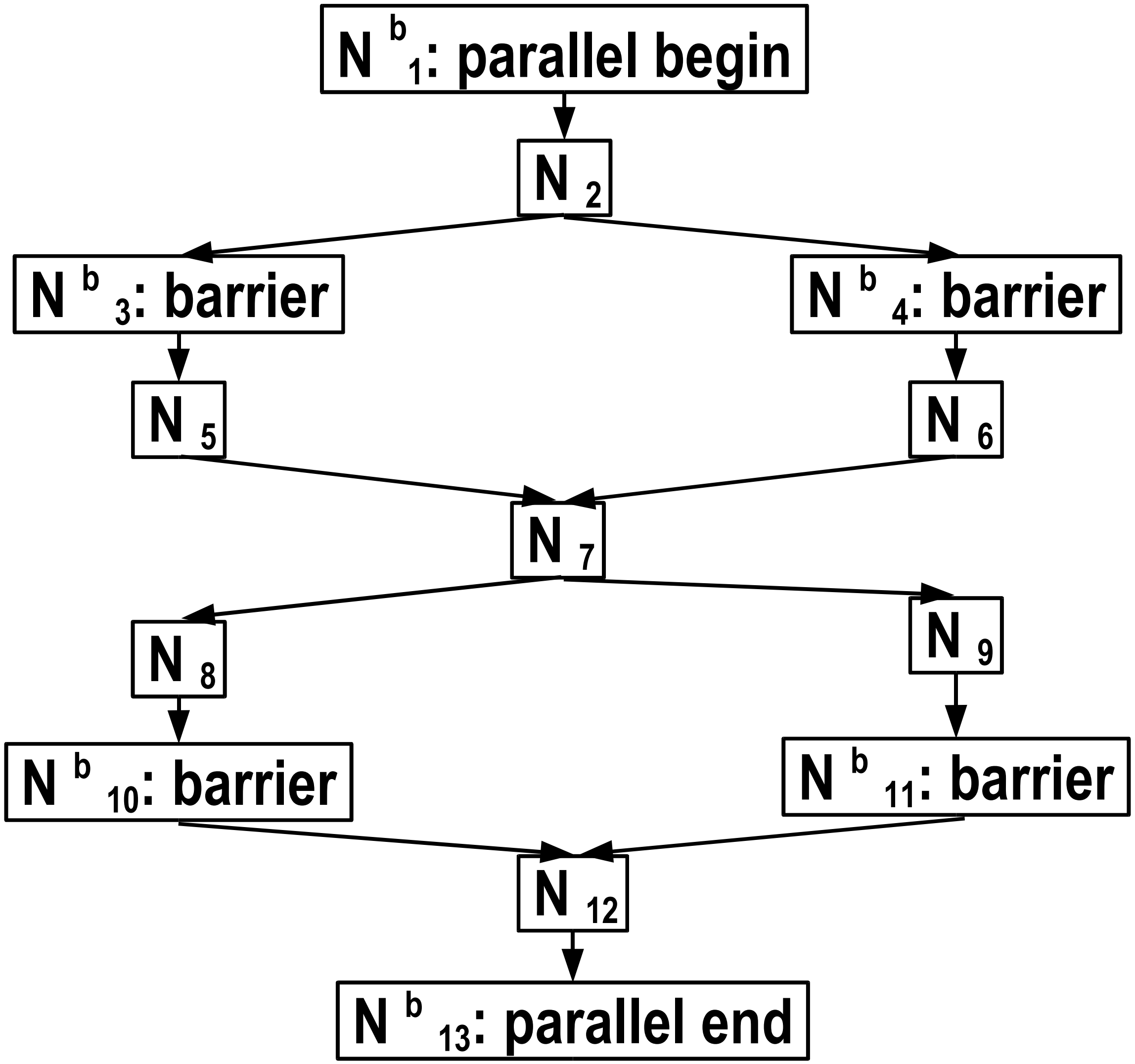
Phase Partitioning - Example 1



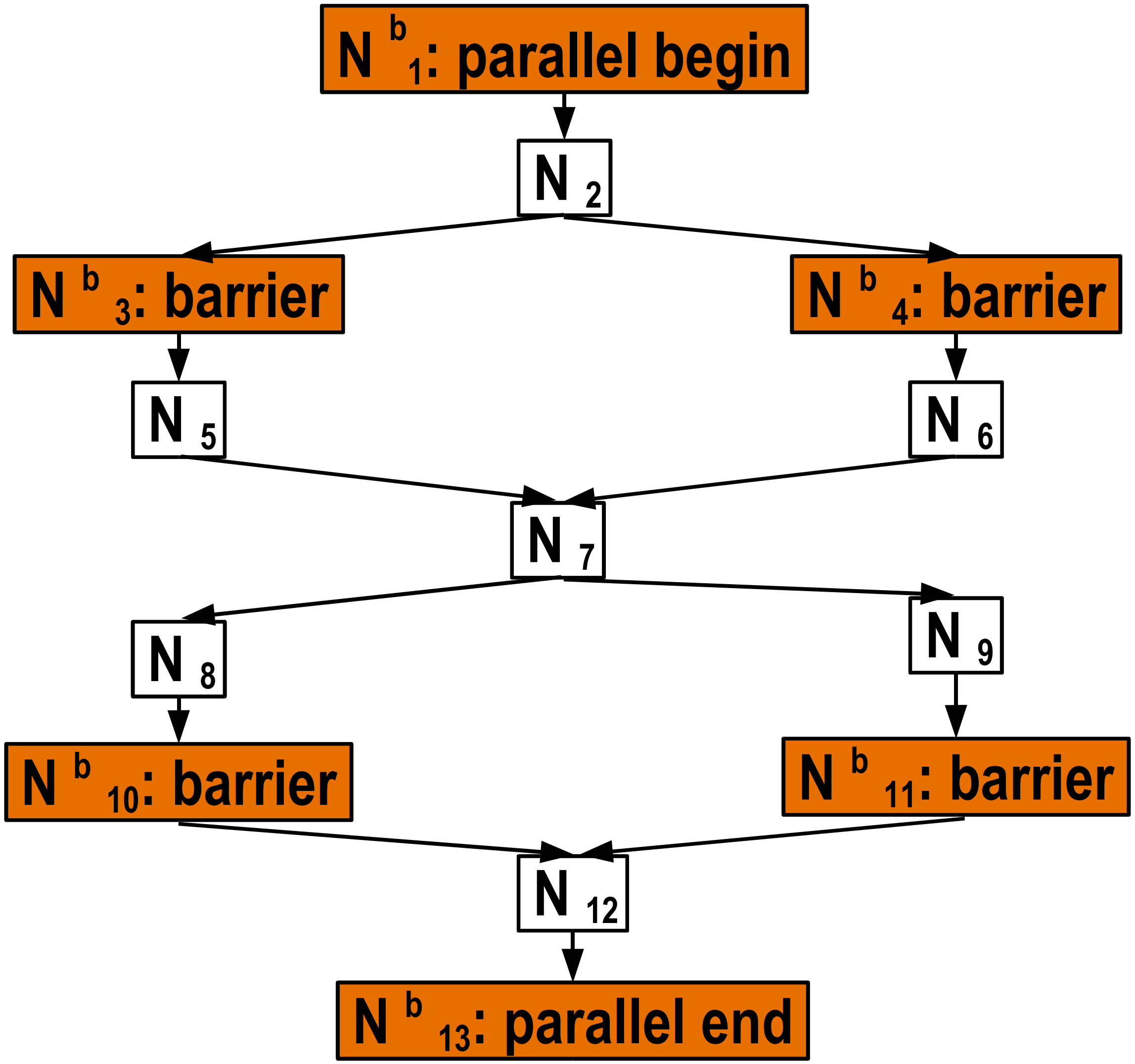
Phase Partitioning - Example 1



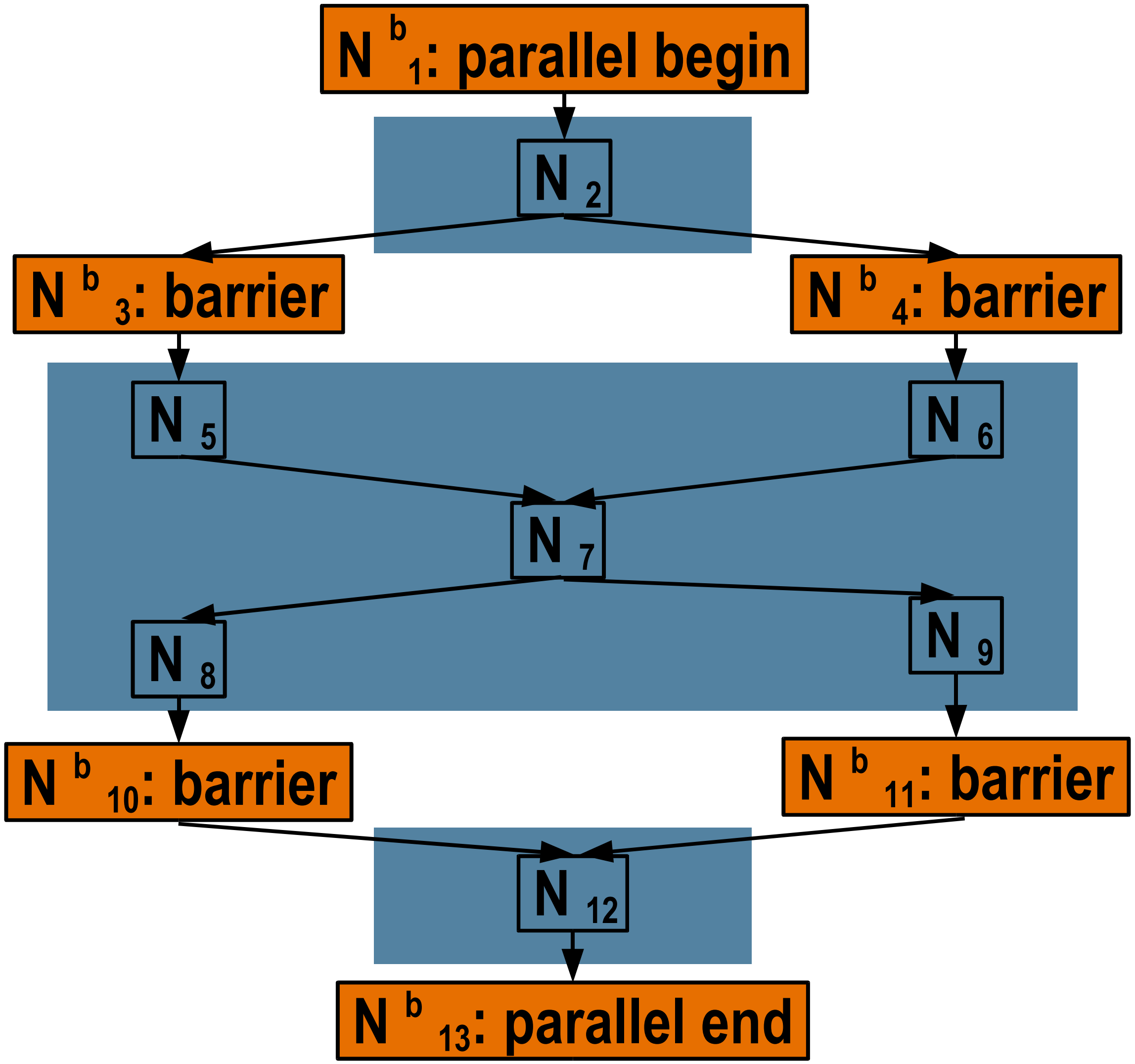
Phase Partitioning - Example 2



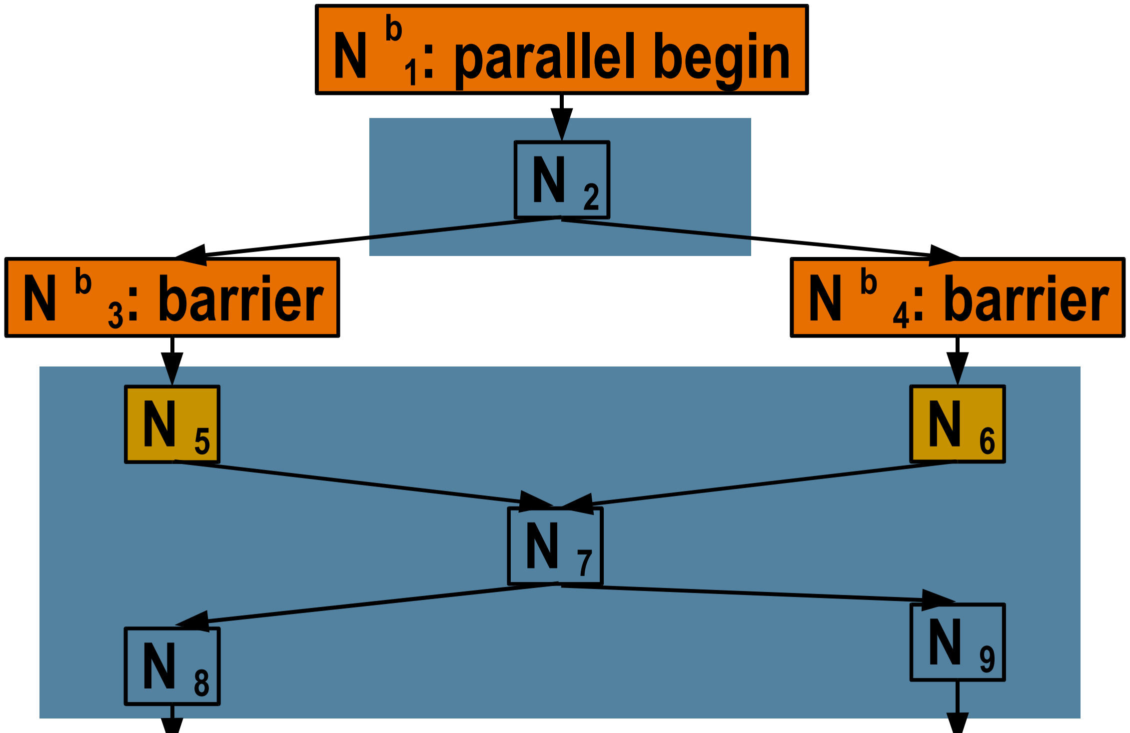
Phase Partitioning - Example 2



Phase Partitioning - Example 2

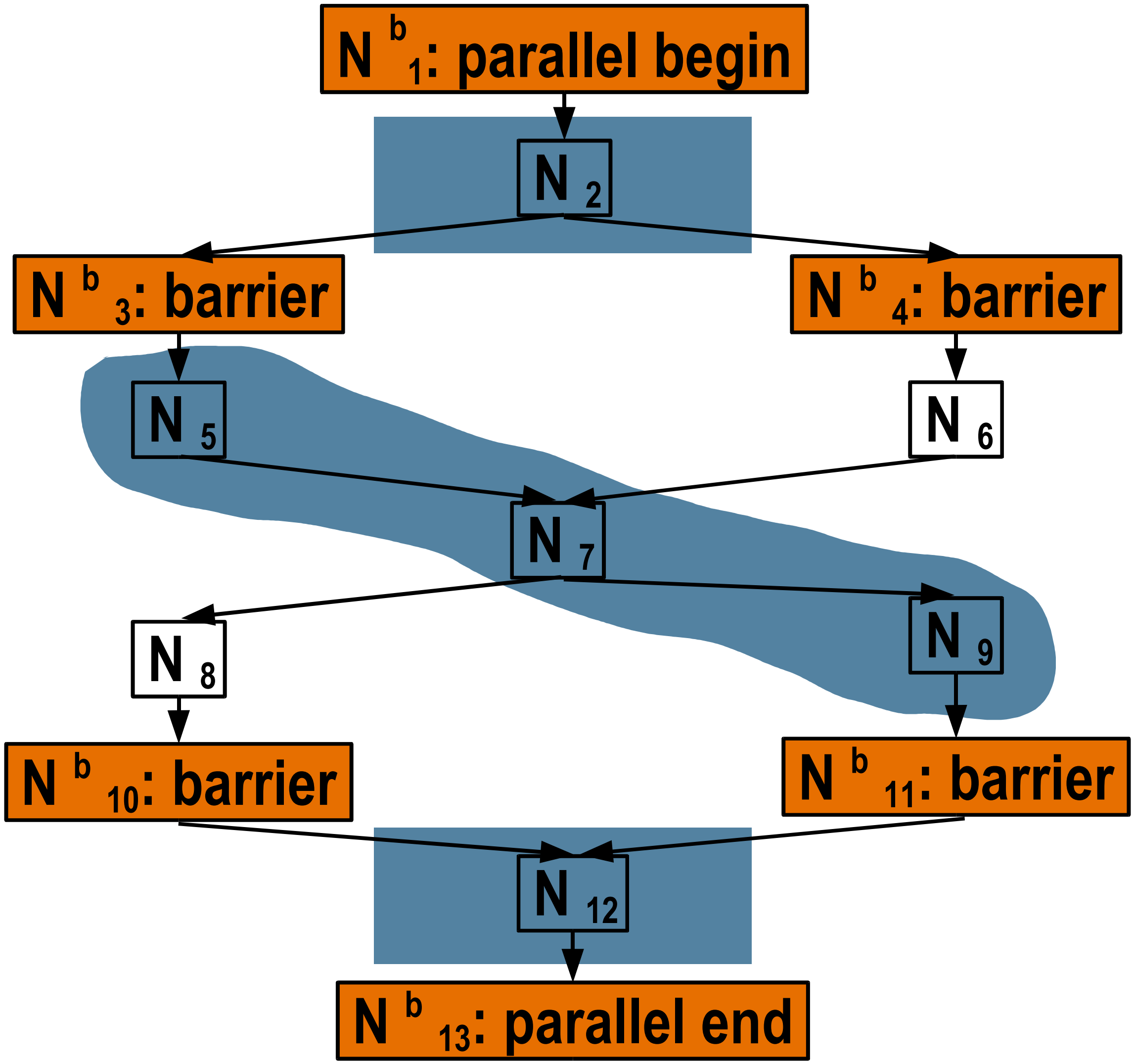


Phase Partitioning - Example 2

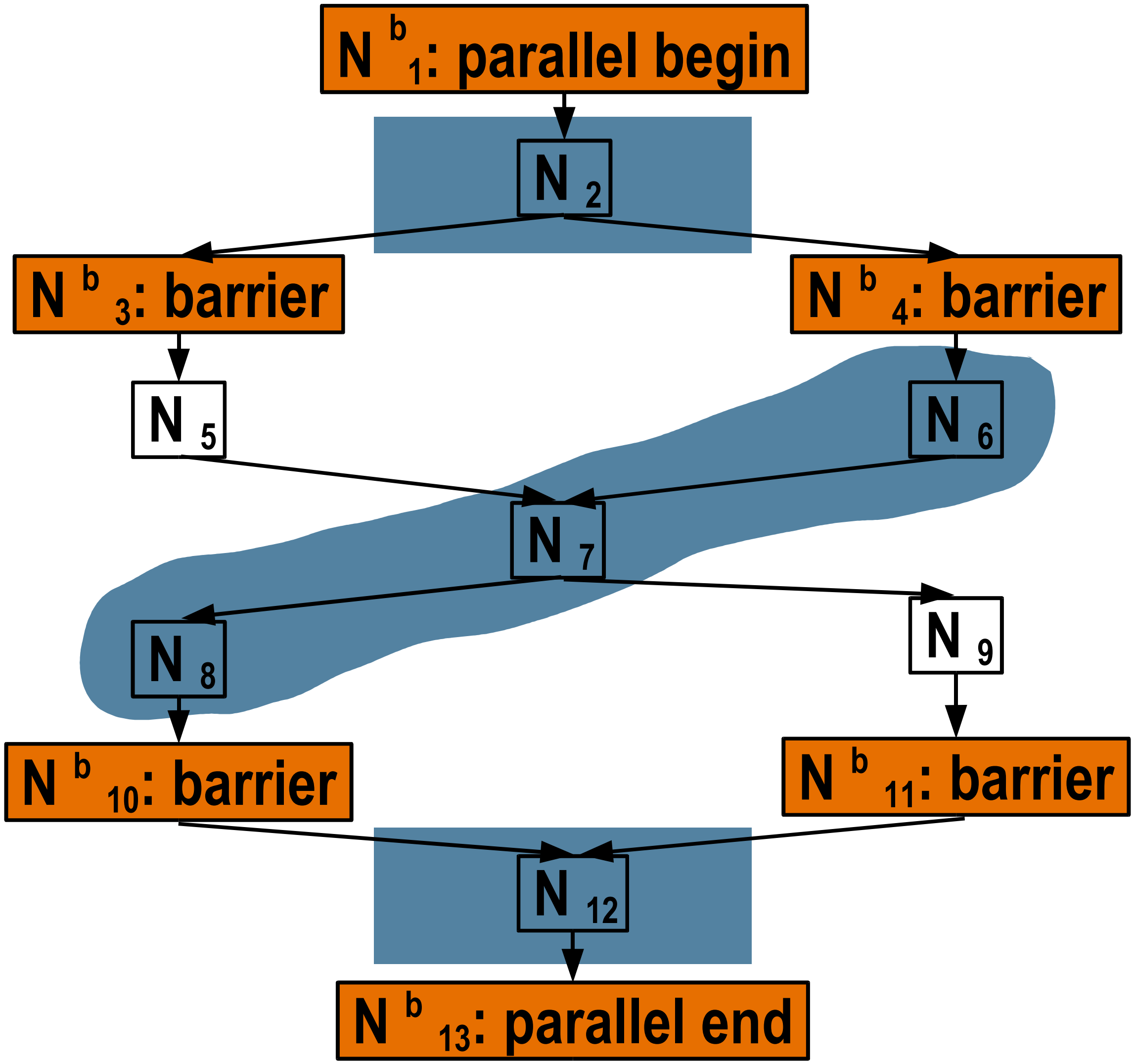


But N_5 and N_6 will not be executed concurrently, because only one of them will be executed by the team.

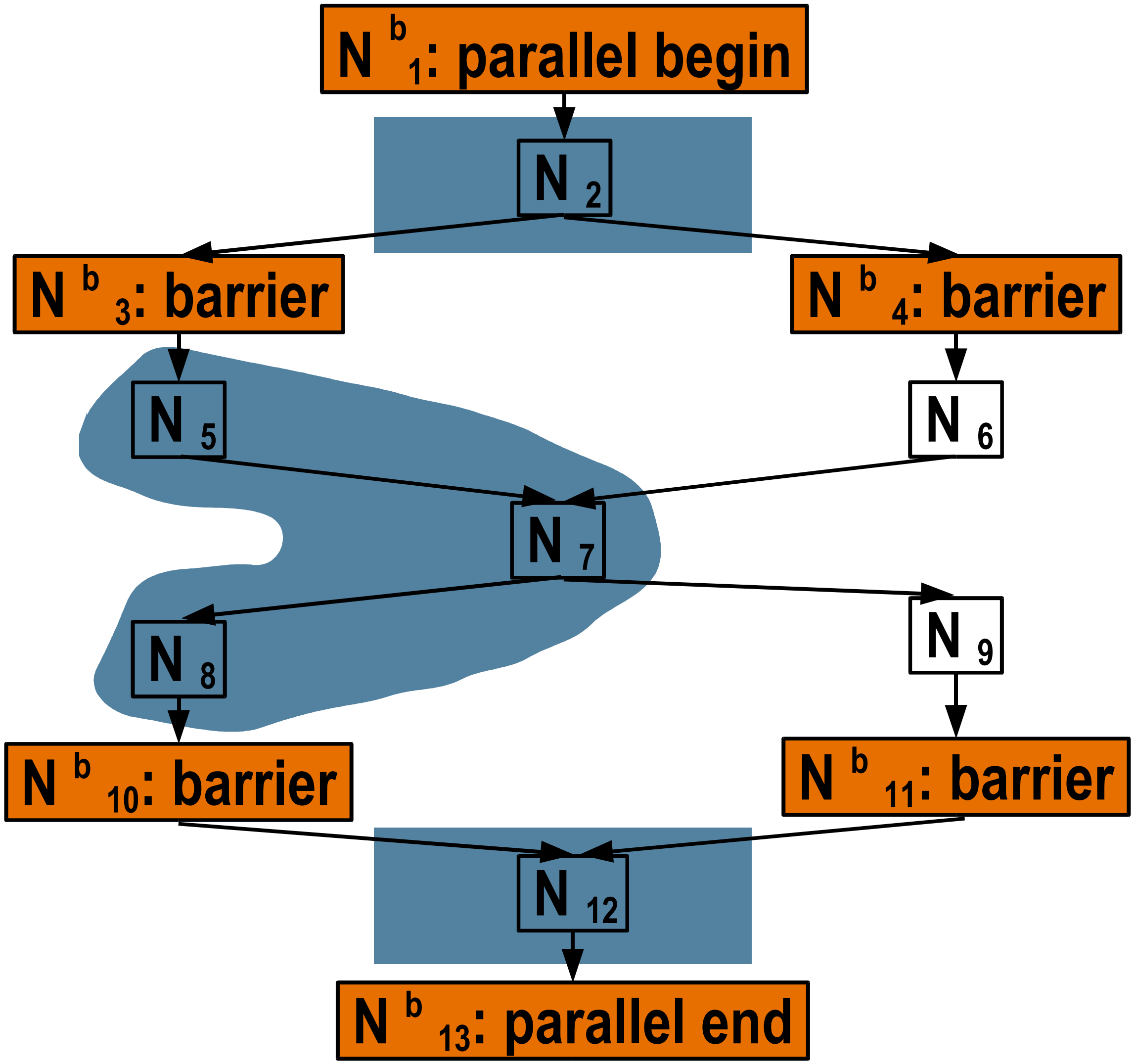
Phase Partitioning - Example 2



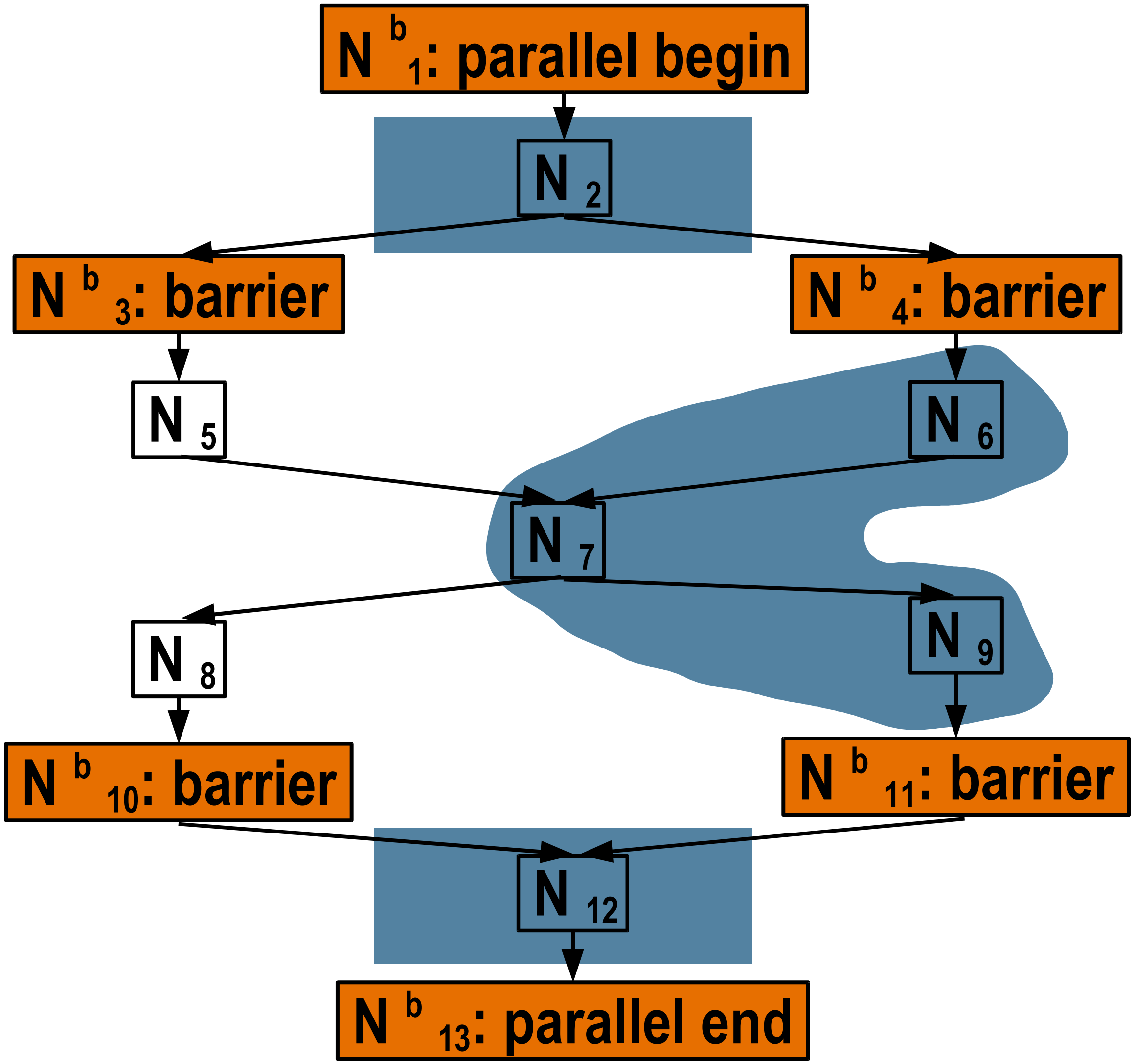
Phase Partitioning - Example 2



Phase Partitioning - Example 2



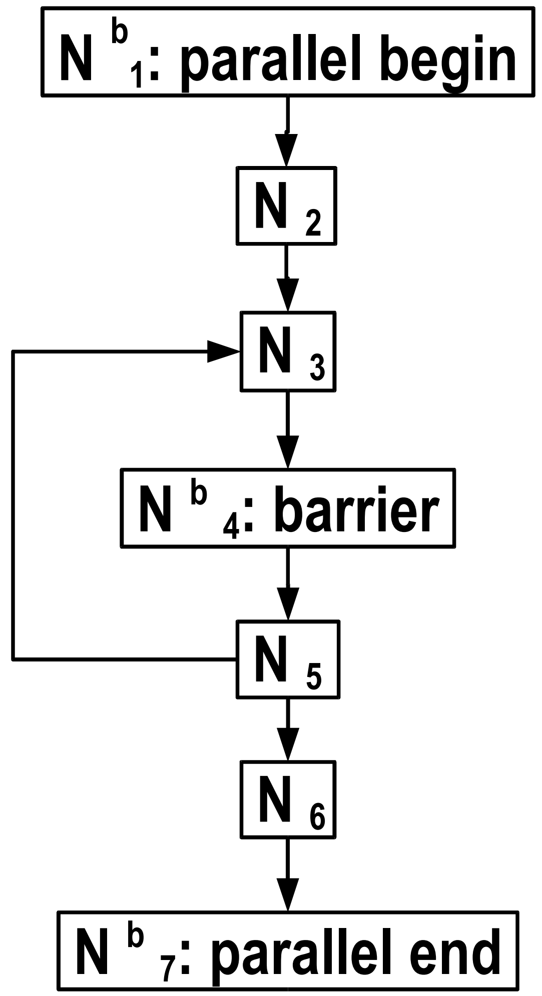
Phase Partitioning - Example 2



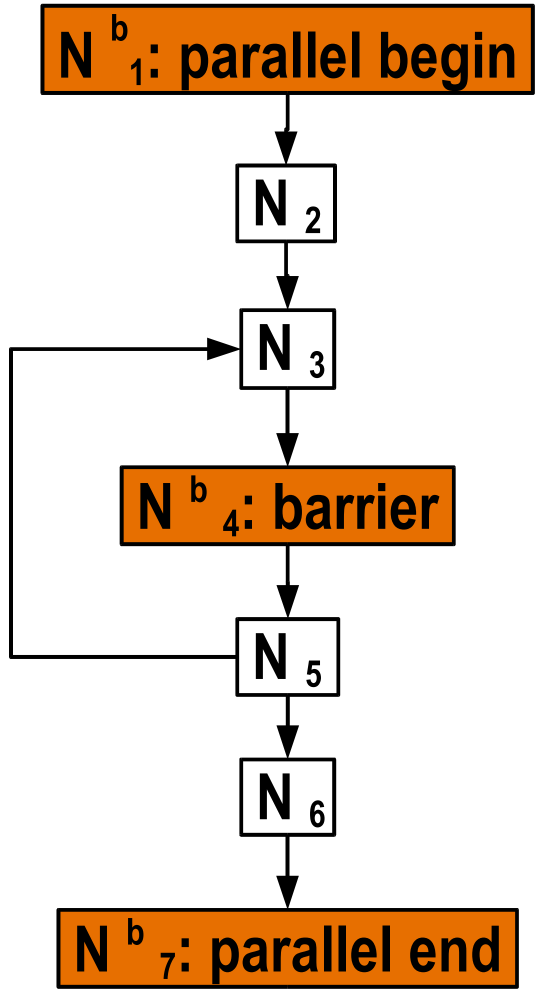
Phase Partitioning

- A phase (*bar1, bar2*) consists of a sequence of nodes along all barrier free paths that starts at barrier node *bar1* and ends at barrier node *bar2* in the same parallel construct.
- If two nodes in a parallel region do not share any phase, then they will not be executed concurrently by different threads in the team that executes the parallel region.
- Phases can be computed by performing two passes of depth-first-search on the OpenMP control flow graph.

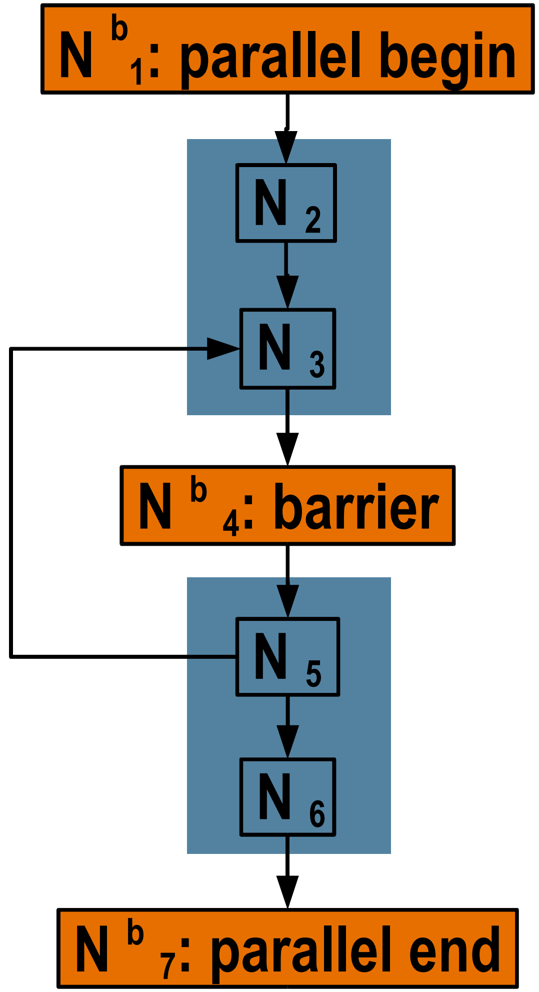
Phase Partitioning - Example 3



Phase Partitioning - Example 3



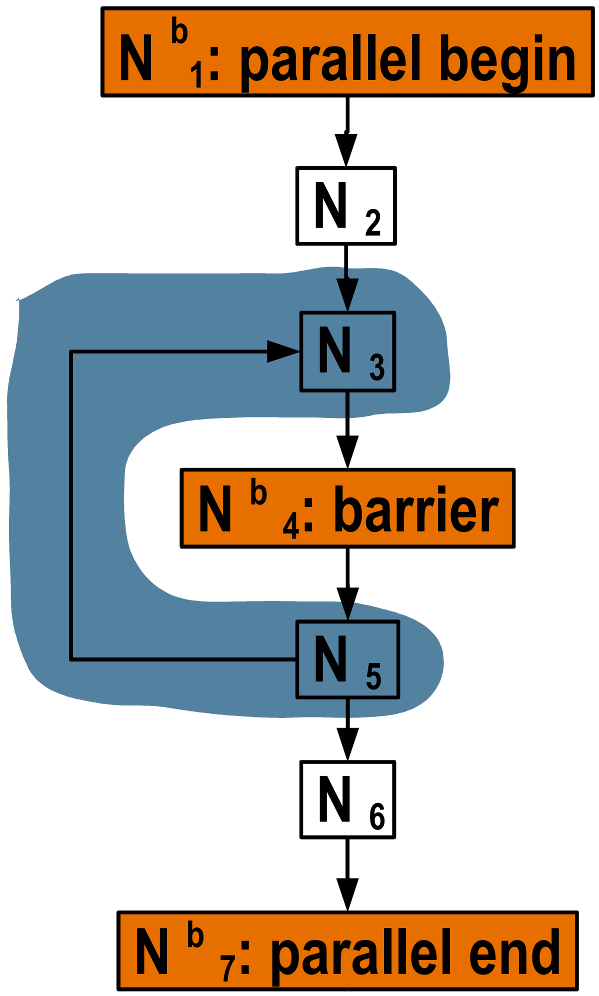
Phase Partitioning - Example 3



$(N^b{}_1, N^b{}_4) = \{N_2, N_3\}$

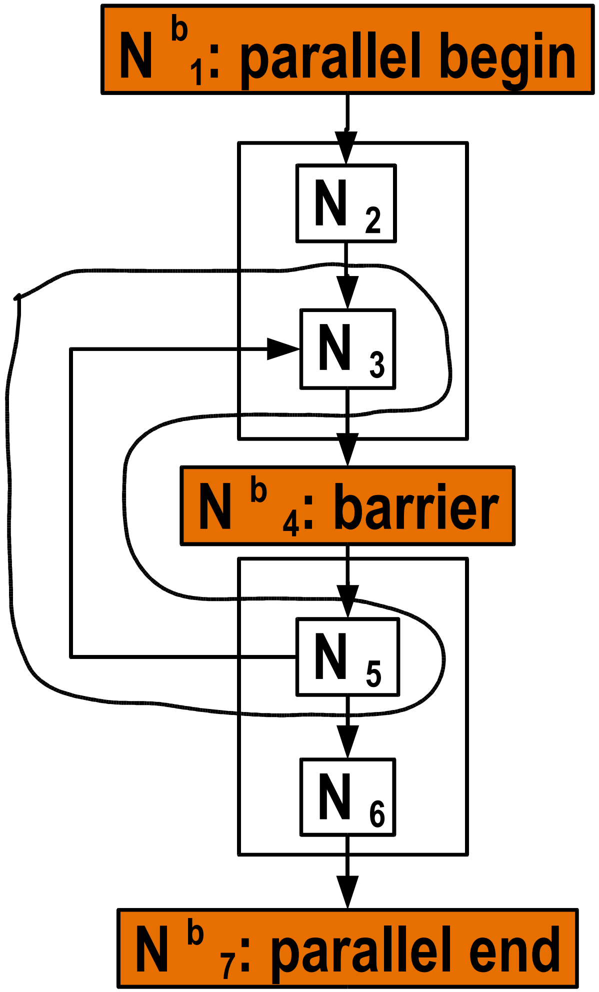
$(N^b{}_4, N^b{}_7) = \{N_5, N_6\}$

Phase Partitioning - Example 3



$(N^b_4, N^b_4) = \{N_3, N_5\}$

Phase Partitioning - Example 3



$(N^b_1, N^b_4) = \{N_2, N_3\}$

$(N^b_4, N^b_4) = \{N_3, N_5\}$

$(N^b_4, N^b_7) = \{N_5, N_6\}$

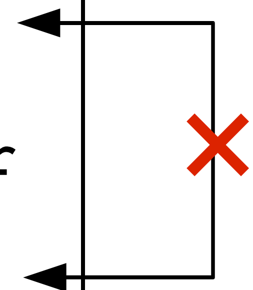
Detection within One Phase

- N1 and N2 are two nodes that appear in the same phase.
- Find the sufficient conditions under which N1 and N2 will not be executed concurrently.
- Structure analysis based on the semantics of OpenMP constructs
  - > MASTER
  - > ORDERED
  - > SINGLE
  - > (CRITICAL is not considered)

Detection within One Phase - MASTER

- N1 and N2 are in MASTER constructs that are bound to the same parallel construct.

```
#pragma omp parallel
{
    #pragma omp master
    {
            a = 1;
    }
    #pragma omp master
    {
            a = 2;
    }
}
```



Detection within One Phase - ORDERED

- N1 and N2 are in ORDERED constructs that are bound to the same DO/FOR construct.

```
#pragma omp parallel
{
    #pragma omp for ordered nowait
    for (i=1; i<n; i++) {
        #pragma omp ordered
        a = 1;
    }
    #pragma omp for ordered
    for (i=1; i<n; i++) {
        #pragma omp ordered
        a = 2;
    }
}
```

Detection within One Phase - ORDERED

- N1 and N2 are in ORDERED constructs that are bound to the same DO/FOR construct.

```
#pragma omp parallel
{
   #pragma omp for ordered nowait
   for (i=1; i<n; i++) {
       #pragma omp ordered
       a = 1;
   }
   #pragma omp for ordered
   for (i=1; i<n; i++) {
       #pragma omp ordered
       a = 2;
   }
}
```

Detection within One Phase - SINGLE

- N1 and N2 are in the same SINGLE construct, and at least one of the following is true,
  - > the SINGLE construct is not in any loop within the parallel region.
  - > the SINGLE construct is in a loop within the parallel region, and there is no barrier free path from the SINGLE end directive node to the header of the immediately enclosing loop.
  - > the SINGLE construct is in a loop within the parallel region, and there is no barrier free path from the header of the immediately enclosing loop to the SINGLE begin directive node.

Detection within One Phase - SINGLE

```
#pragma omp parallel
{
   #pragma omp single nowait
   {
         a = 1;
         a = 2;
   }
}
```

Detection within One Phase - SINGLE

```
#pragma omp parallel
{
   for (i=1; i<n; i++) {
       #pragma omp single nowait
       {
          a = 1;
          a = 2;
       }
}
```

Related Work

- T.E. Jeremiassen and S.J. Eggers: *Static analysis of barrier synchronization in explicitly parallel programs*. (PACT 1994)
- S. Satoh, K. Kusano, and M. Sato: *Compiler optimization techniques for OpenMP programs*. (EWOMP 2000)
- Static analysis of data races (not a complete list)
  - V. Balasundaram and K. Kennedy
  - D. Callahan and K. Kennedy
  - P. Emrath and D. Padua
  - R. Netzer and S. Ghosh

Summary

- Concurrency is a necessary condition for data races.
- A compile-time analysis technique that can detect non-concurrency in OpenMP programs is presented.
  - Phase partitioning
  - Detecting non-concurrency between statements that do not share any common phase
  - Detecting on-concurrency between statements that share a common phase

Possible Research Topics

- Inter-procedural non-concurrency analysis in OpenMP
- Hybrid static and runtime data race detection for OpenMP programs
- Optimizations for OpenMP programs

Static Non-concurrency Analysis of OpenMP Programs

Yuan Lin
yuan.lin@sun.com
Sun Microsystems, Inc.