

Precedence Relations in the OpenMP Programming Model

M. Gonzalez, J. Oliver, X. Martorell,
E. Ayguadé, J. Labarta and N. Navarro

European Center for Parallelism of Barcelona
Technical University of Catalunya

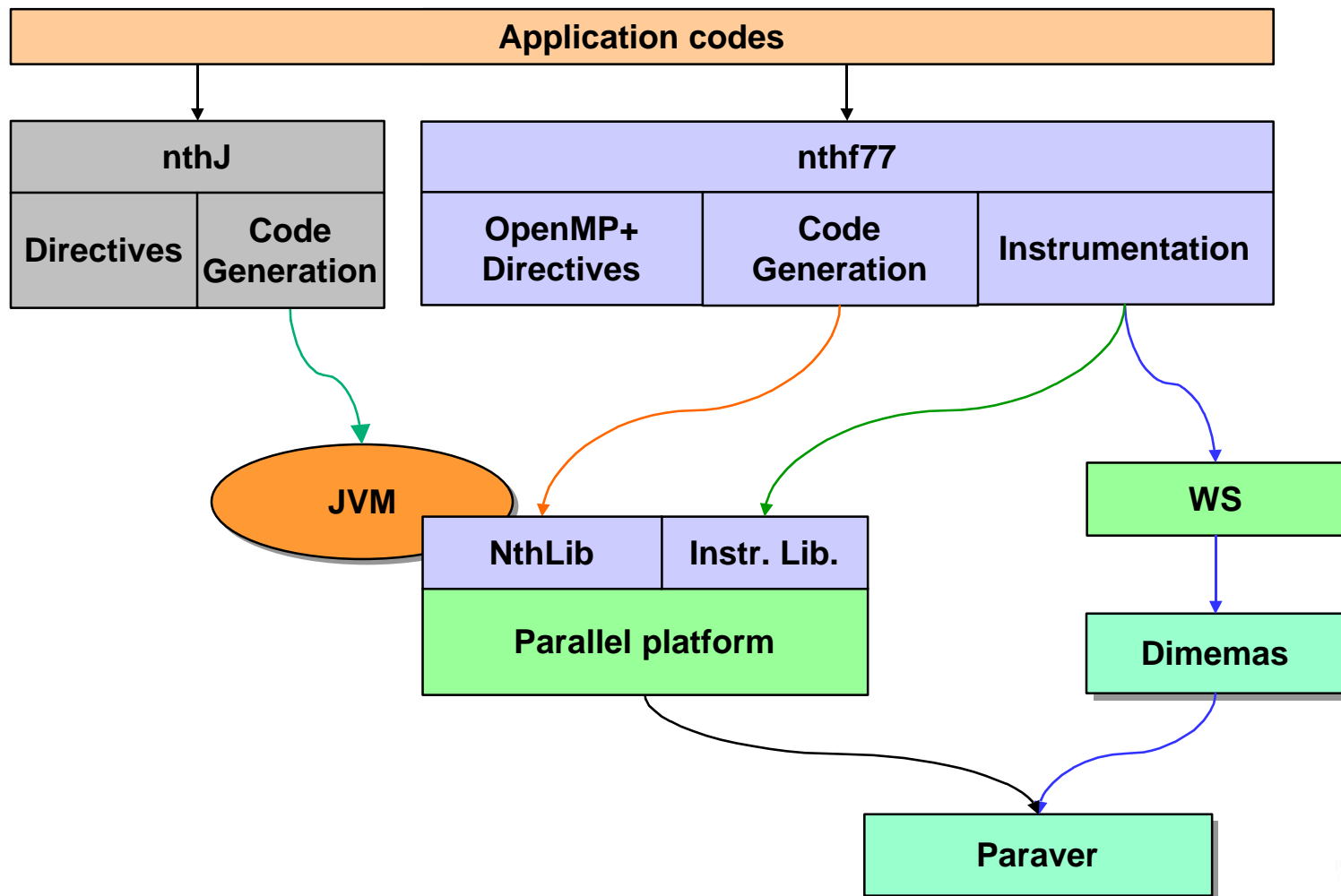


Talk outline

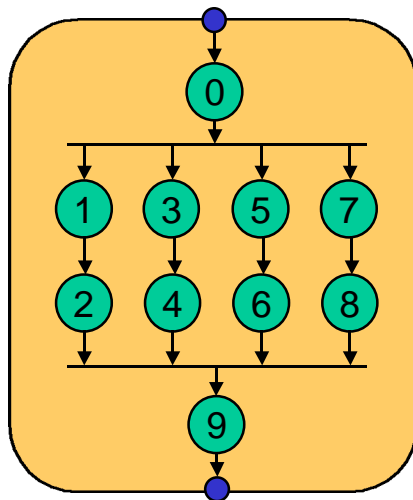
- **NanosCompiler environment**
- **Precedence relations in OpenMP**
- **OpenMP extensions**
- **An example / Initial experimentation**
- **Conclusions**



NanosCompiler environment



Nested parallelism

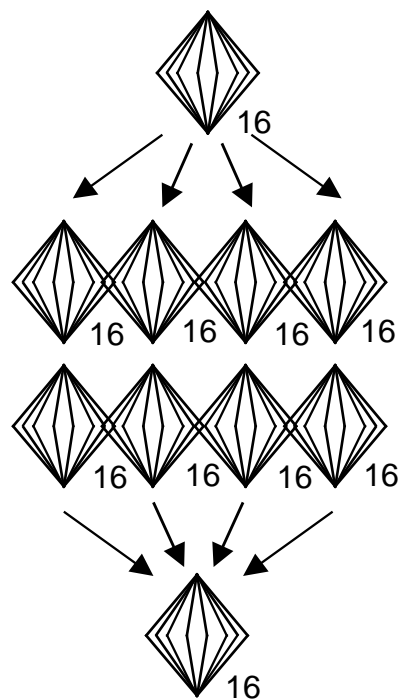


Single level

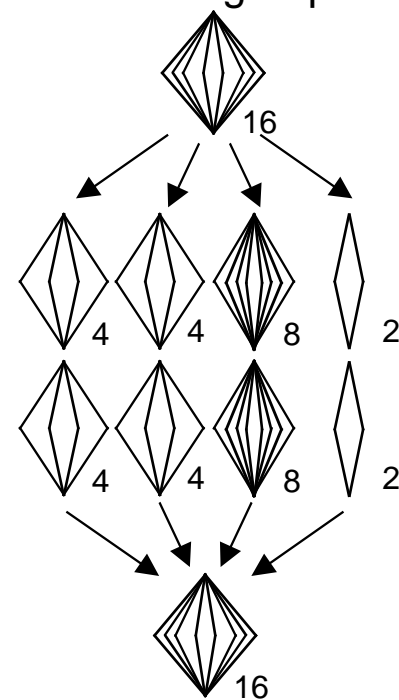


Multiple levels

All-to-all



Thread groups



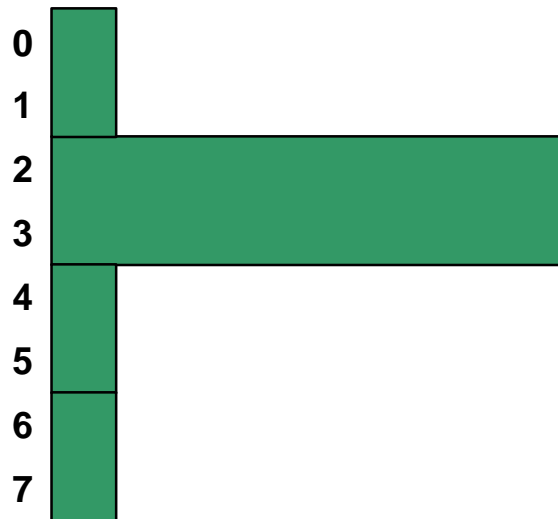
Loop is executed on P processors

P



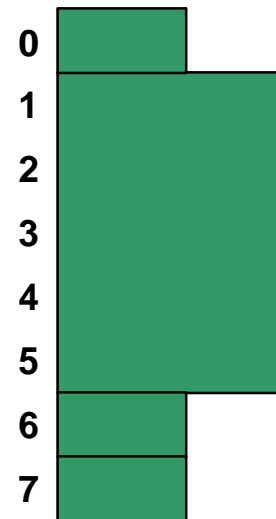
Thread groups

GROUPS(nfrag)



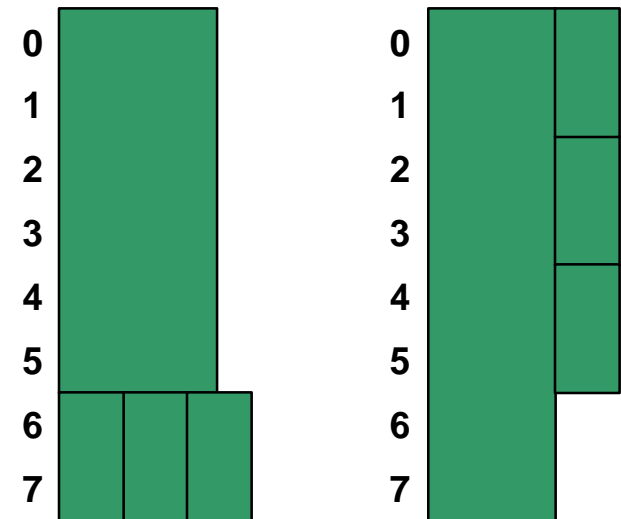
nfrag=4

GROUPS(nfrag, weight)



weight={1,4,1,1}

GROUPS(nfrag, who, howmany)



who={6,0,6,6}

howmany={2,6,2,2}

```
C$OMP PARALLEL DO PRIVATE(II)
C$OMP& GROUPS(nfrag)
  DO II = 1, nfrag
    ...
    call compute(..., work(II), ...)
    ...
  END DO
C$OMP END PARALLEL DO
```

[ewomp99, lcpc00]



Example: multiblock code

```
C Initialize blocks and compute vector work
...
C Solve within each block
10   tres=0.0
C$OMP PARALLEL DO SCHEDULE(STATIC) GROUPS(nblock,work)
C$OMP& REDUCTION(+:tres) PRIVATE(res)
    do iblock=1, nblock
        call solve(a(loc(iblock)),nx(iblock),ny(iblock),nz(iblock),res,tol)
        tres=tres+res
    enddo
C$OMP END PARALLEL DO
C Perform inter-block interactions
...
    if (tres.gt.tol) goto 10
...

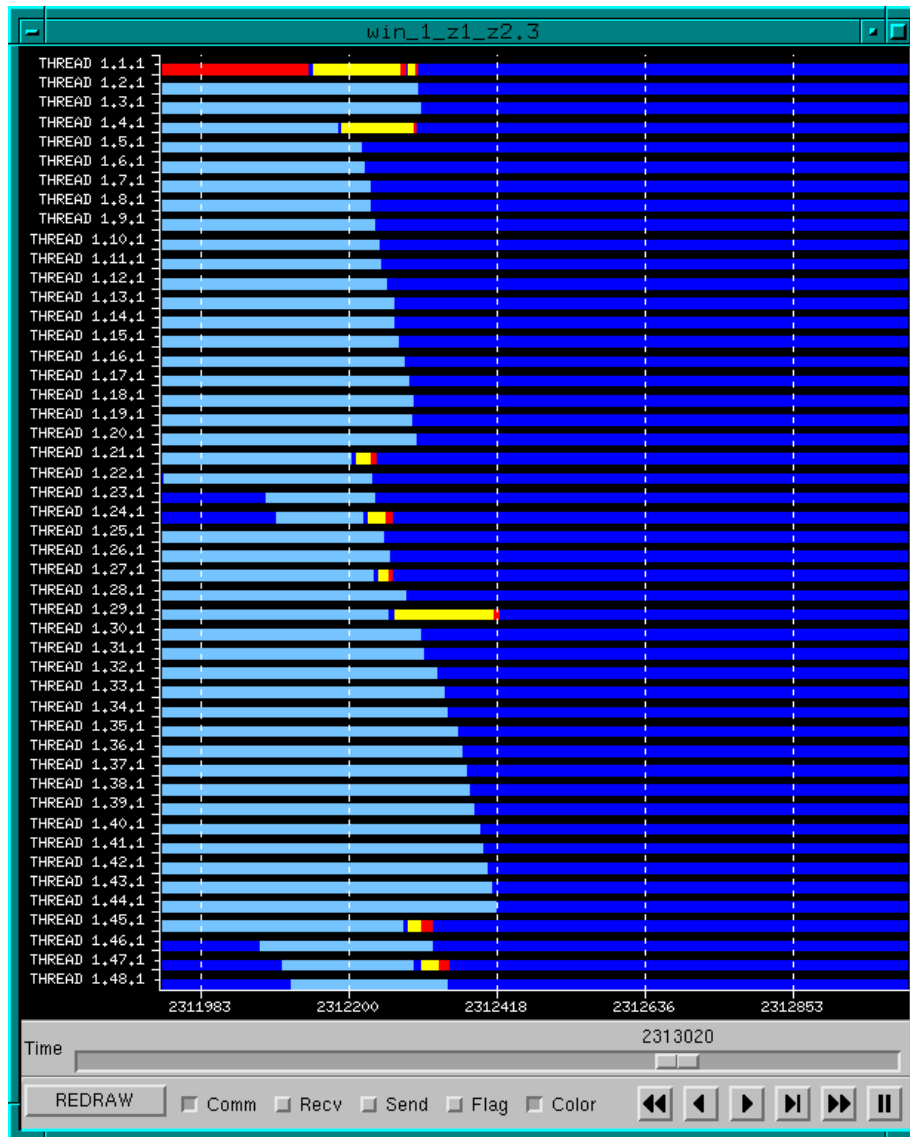
```

```
subroutine solve(t,nx,ny,nz,tres,tol)
...
    res=0.0
C$OMP PARALLEL DO SCHEDULE(STATIC) REDUCTION(+:res)
C$OMP& PRIVATE(i,j,k)
    do k=1, nz
        do j=1, ny
            do i=1, nx
                t(i,j,k)=(told(i,j,k-1)+told(i,j,k+1)+
+                       told(i,j-1,k)+told(i,j+1,k)+
+                       told(i-1,j,k)+told(i+1,j,k)+
+                       told(i,j,k)*6.0)/12.0
                res=res+(t(i,j,k)-told(i,j,k))**2
            enddo
        enddo
    enddo
C$OMP END PARALLEL DO
...
end

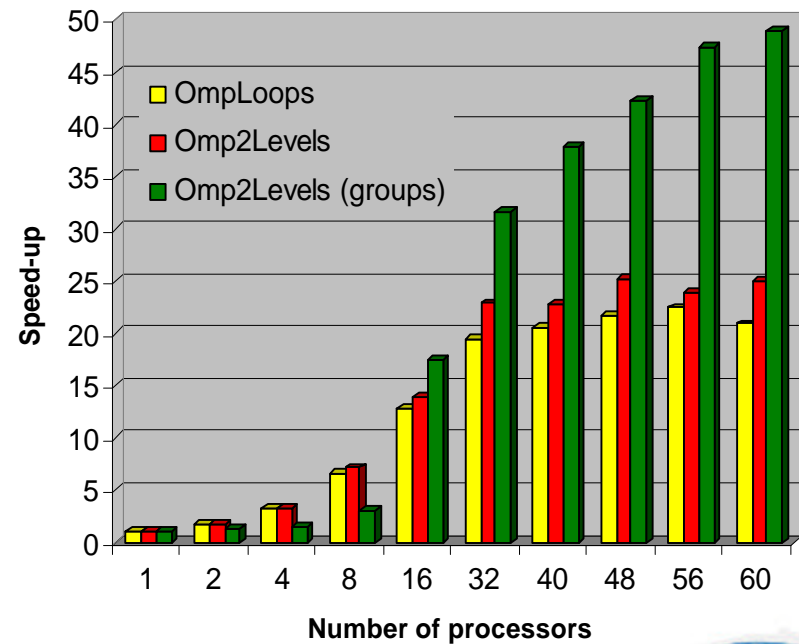
```



Example: multiblock code



GROUPS(8,weight)
 weight[8]={1,8,1,1,1,8,1,1}



Precedence relations

■ Objective:

- Less rigid parallelism definition
 - ✓ fork, do completely independent computation, and join
- Precedence relations among work parcelled out by work-sharing constructs (sections and loops)
 - ✓ Naming work-sharing constructs
 - ✓ Specification of predecessor/successor relationships
- Pipelined execution model
- Coarse-grain tasks

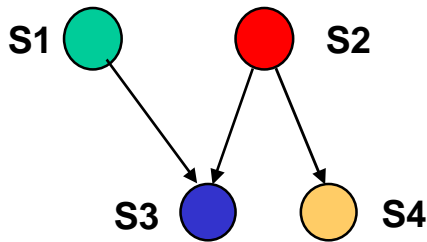
■ Single level parallelism

■ Multiple levels of parallelism with thread groups

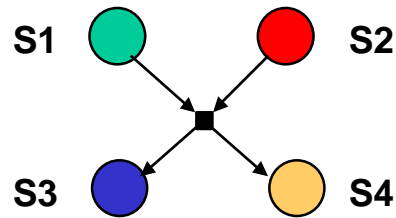


Precedence relations

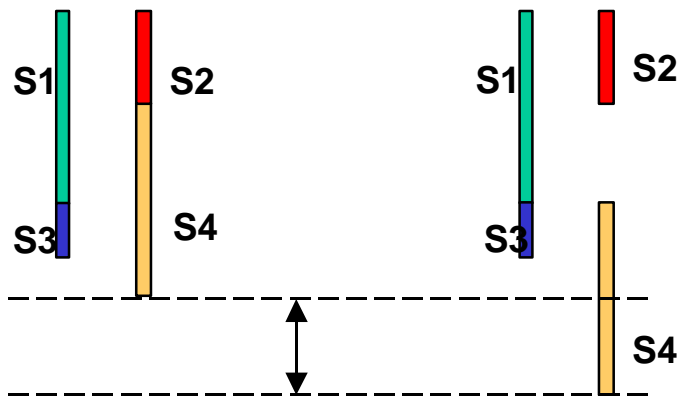
■ Single-level parallelism



Task Graph



OpenMP Task Graph



!\$OMP PARALLEL SECTIONS

!\$OMP SECTION



!\$OMP SECTION



!\$OMP END PARALLEL SECTIONS

!\$OMP PARALLEL SECTIONS

!\$OMP SECTION



!\$OMP SECTION

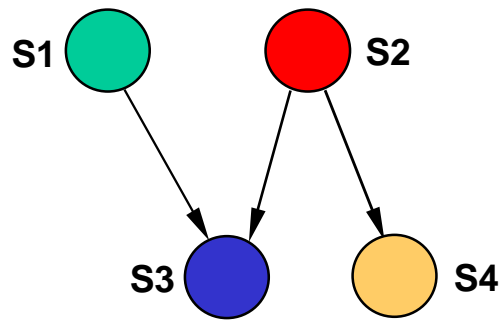


!\$OMP END PARALLEL SECTIONS



Precedence relations

■ Single-level parallelism



Task Graph

```
!$OMP PARALLEL SECTIONS
```

```
!$OMP SECTION NAME(S1) SUCC( S3 )
```



```
!$OMP SECTION NAME(S2) SUCC( S3, S4 )
```



```
!$OMP SECTION NAME(S3) PRED ( S1, S2 )
```



```
!$OMP SECTION NAME(S4) PRED( S2 )
```



```
!$OMP END PARALLEL SECTIONS
```



Precedence relations

- PRED/SUCC clauses can appear at any point

!\$OMP PARALLEL SECTIONS

!\$OMP SECTION **NAME(S1)**



!\$OMP SUCC(S3)



!\$OMP SECTION **NAME(S2)** SUCC(S3, S4)



!\$OMP SECTION **NAME(S3)** PRED (S2)



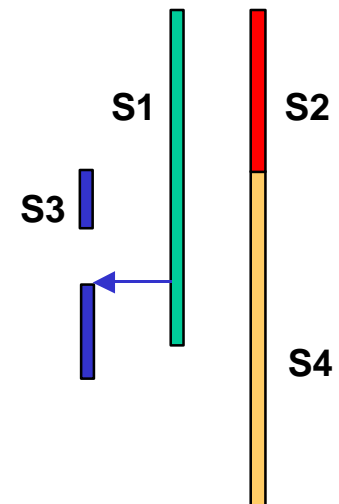
!\$OMP PRED (S1)



!\$OMP SECTION **NAME(S4)** PRED(S2)



!\$OMP END PARALLEL SECTIONS



Precedence relations

■ Multilevel parallelism

```
!$OMP PARALLEL DO SCHEDULE(STATIC,1)
```

```
  do KK=1, N
```

```
!$OMP PARALLEL SECTIONS
```

```
!$OMP SECTION NAME( S1:KK ) SUCC( S3:KK )
```



```
!$OMP SECTION NAME( S2:KK ) SUCC( S3:KK, S4:KK )
```



```
!$OMP SECTION NAME( S3:KK ) PRED( S1:KK, S2:KK )
```



```
!$OMP SECTION NAME( S4:KK )
```

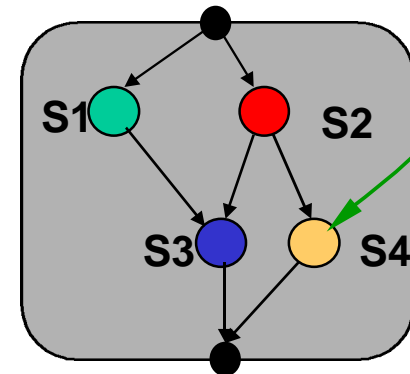
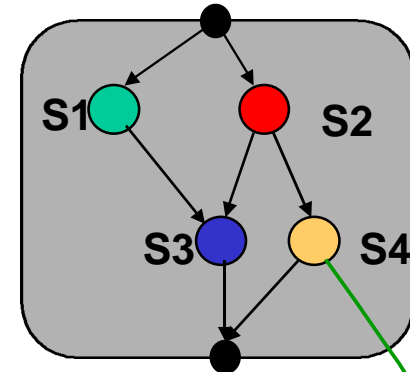
```
!$OMP& PRED( S2:KK, S4:KK-1 ) SUCC(S4:KK+1)
```



```
!$OMP END PARALLEL SECTIONS
```

```
  enddo
```

```
!$OMP END PARALLEL DO
```



OpenMP extensions

■ NAME

- Defines a name space for the tasks coming out from a work-sharing construct

```
!$OMP ( SECTION | DO | SINGLE ) NAME( name_ident [:expr ] )
```

■ SUCC and PRED

- Specifies a successor/predecessor relationship between two named tasks

```
[ !$OMP ] SUCC( task_id [, task_id]* ) IF ( expr )
```

```
[ !$OMP ] PRED( task_id [, task_id]* ) IF ( expr )
```

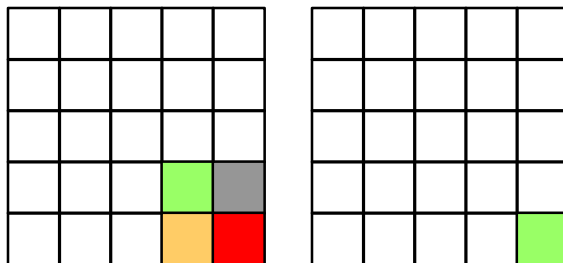
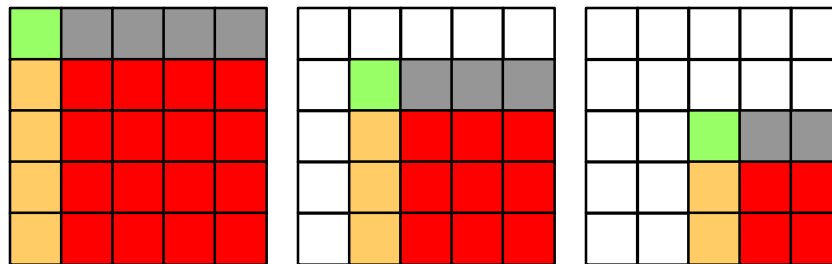
```
task_id = ( name_ident [:expr]* ) | ( name_ident [:expr]*, expr )
```



Another example: blocked LU

Multiple levels of parallelism:

- Intra-block loop-level parallelism: lu0, fwd, bdiv, bmod
- Inter-block parallelism



```
do kk=1,NB
```

```
  call lu0(A,kk)
```

```
  do jj=kk+1,NB
    call fwd(A,kk,jj)
  enddo
```

```
  do ii=kk+1,NB
```

```
    call bdiv(A,ii,kk)
```

```
    do jj=kk+1,NB
      call bmod(A,ii,jj,kk)
    enddo
```

```
  enddo
```

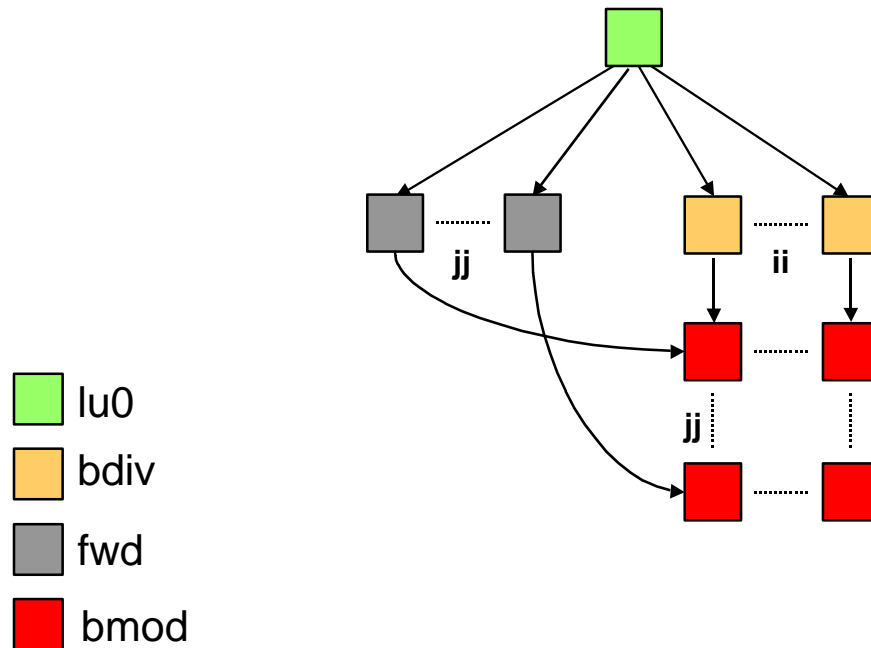
```
enddo
```



Another example: blocked LU

■ Inter-block parallelism

- Inside iteration kk



```
do kk=1,NB
```

```
  call lu0(A,kk)
```

```
  do jj=kk+1,NB
    call fwd(A,kk,jj)
  enddo
```

```
  do ii=kk+1,NB
```

```
    call bdiv(A,ii,kk)
```

```
    do jj=kk+1,NB
      call bmod(A,ii,jj,kk)
    enddo
```

```
  enddo
```

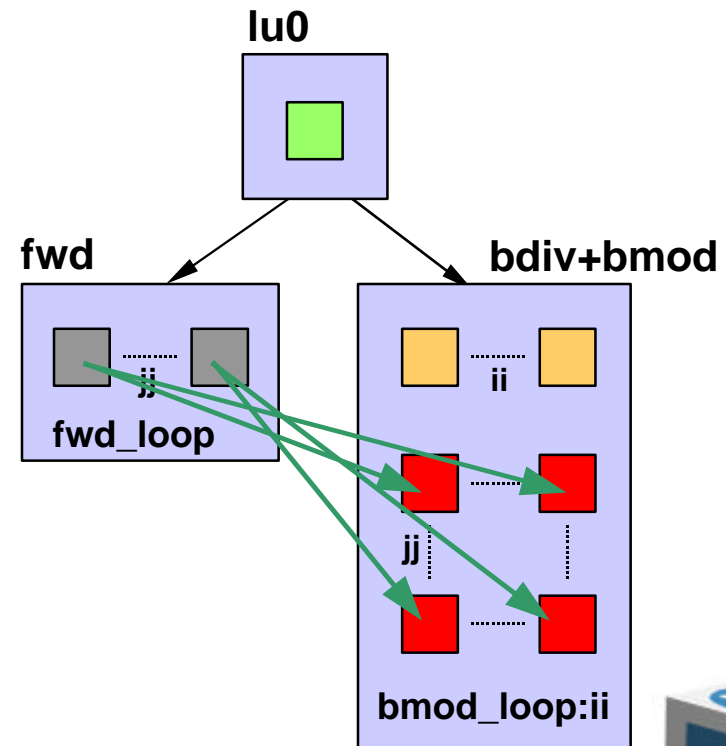
```
enddo
```



Another example: blocked LU

■ Task definition

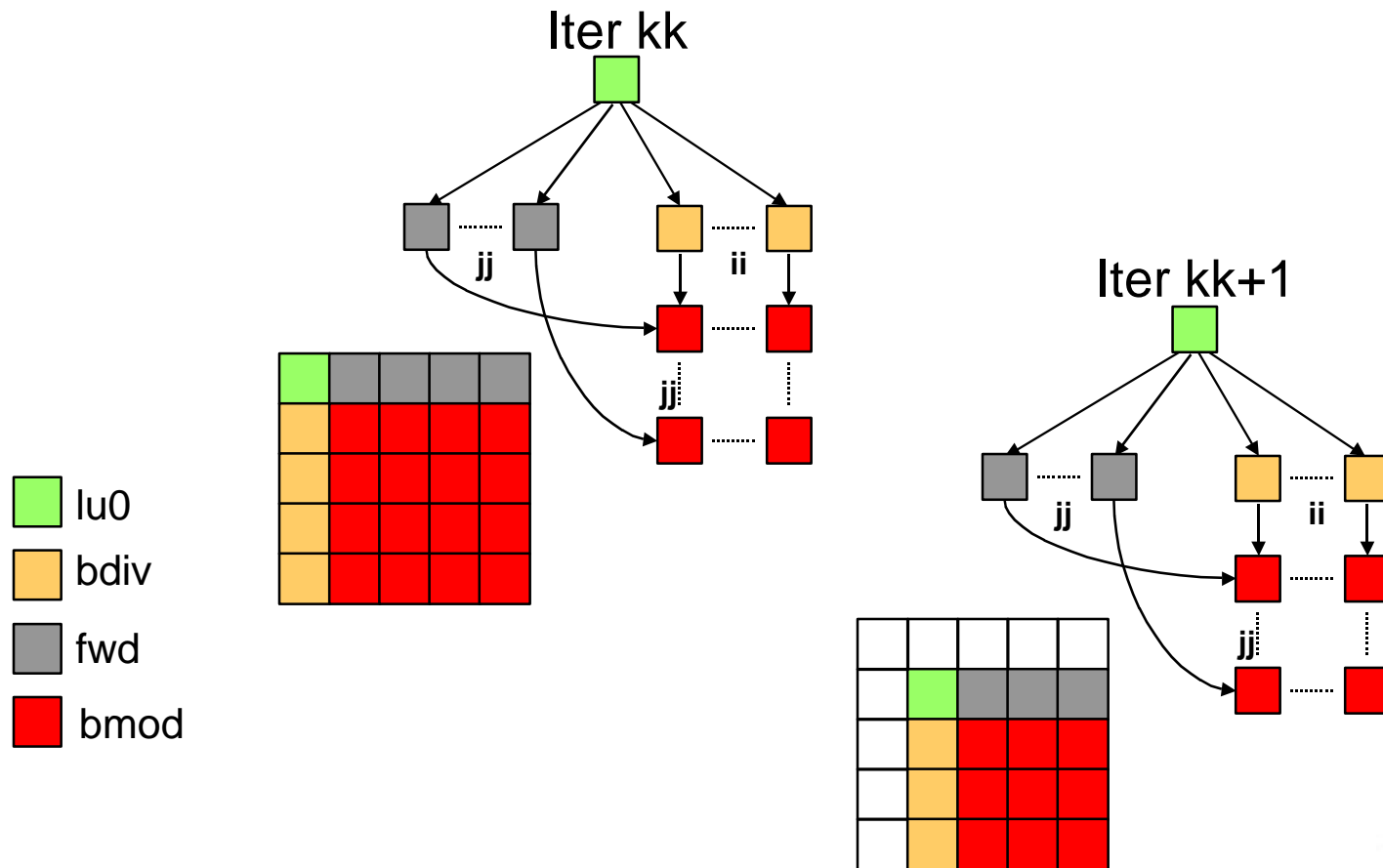
```
do kk=1,NB
  !$OMP PARALLEL SECTIONS
  !$OMP SECTION NAME(lu0) SUCC(fwd,bdiv+bmod)
    call lu0(A,kk)
  !$OMP SECTION NAME(fwd) PRED(lu0)
  !$OMP PARALLEL DO NAME(fwd_loop)
    do jj=kk+1,NB
      call fwd(A,kk,jj)
    !$OMP SUCC(bmod_loop:ALL,jj)
    enddo
  !$OMP SECTION NAME(bdiv+bmod) PRED(lu0)
  !$OMP PARALLEL DO
    do ii=kk+1,NB
      call bdiv(A,ii,kk)
    !$PARALLEL DO NAME(bmod_loop:ii)
      do jj=kk+1,NB
    !$OMP PRED(fwd_loop,jj)
      call bmod(A,ii,jj,kk)
      enddo
    enddo
  !$OMP END PARALLEL SECTIONS
enddo
```



Another example: blocked LU

■ Inter-block parallelism

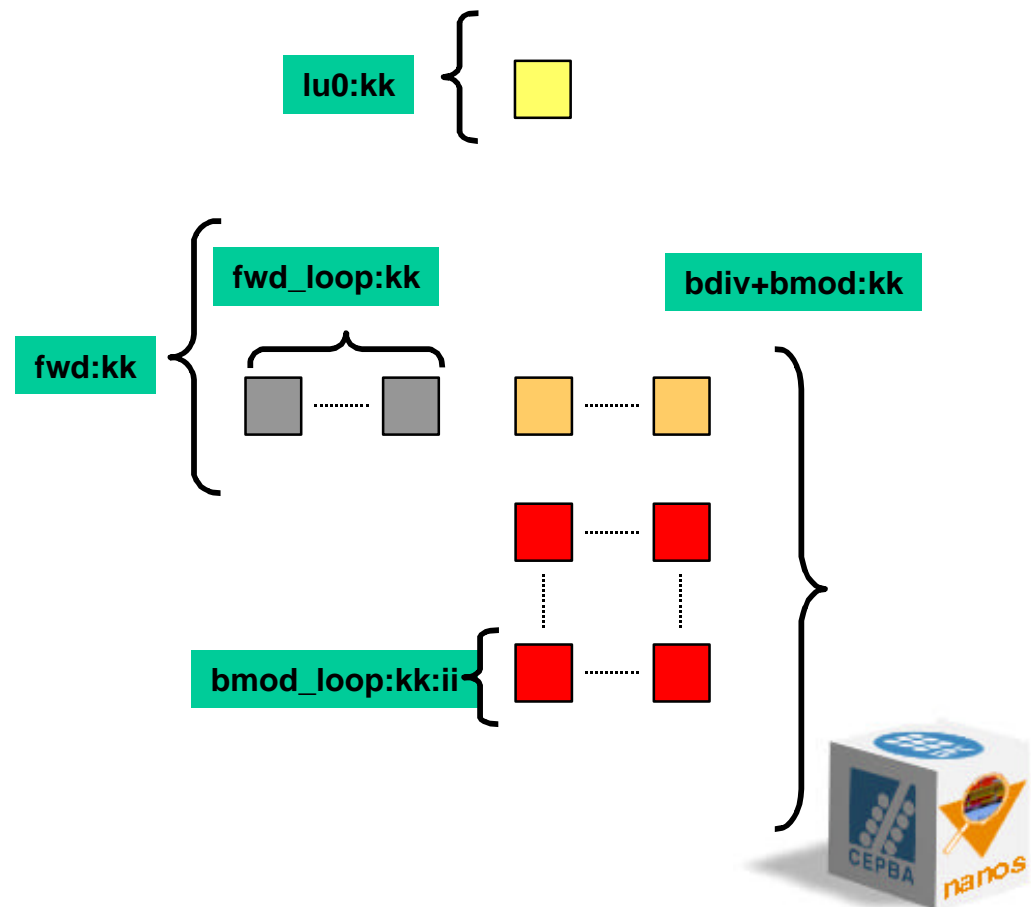
- Pipelining across successive iterations kk



Another example: blocked LU

■ Task definition

```
!$OMP PARALLEL DO
  do kk=1,NB
!$OMP PARALLEL SECTIONS
  !$OMP NAME(lu0:kk)
    call lu0(A,kk)
  !$OMP NAME(fwd:kk)
  !$OMP PARALLEL DO NAME(fwd_loop:kk)
    do jj=kk+1,NB
      call fwd(A,kk,jj)
    enddo
  !$OMP NAME(bdiv+bmod:kk)
  !$OMP PARALLEL DO
    do ii=kk+1,NB
      call bdiv(A,ii,kk)
  !$PARALLEL DO NAME(bmod_loop:kk:ii)
    do jj=kk+1,NB
      call bmod(A,ii,jj,kk)
    enddo
  enddo
!$OMP END PARALLEL SECTIONS
enddo
```

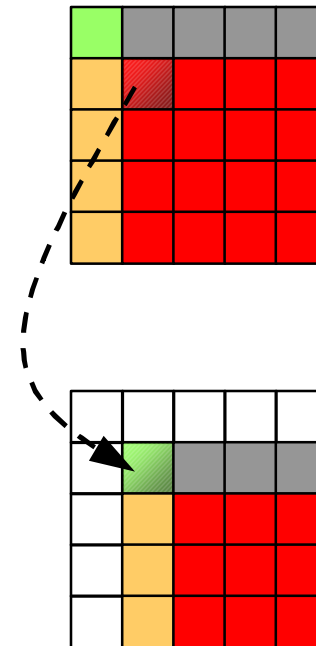


Another example: blocked LU

■ Precedence definition (bmod \rightarrow lu0)

```
!$OMP PARALLEL DO
  do kk=1,NB
!$OMP PARALLEL SECTIONS
  !$OMP NAME( lu0:kk )
  !$OMP &PRED( bmod_loop:kk-1:kk , jj ) IF(kk.gt.1)
    call lu0(A,kk)

!$OMP NAME( bdiv+bmod:kk )
!$OMP PARALLEL DO
  do ii=kk+1,NB
    call bdiv(A,ii,kk)
!$PARALLEL DO NAME( bmod_loop:kk:ii )
    do jj=kk+1,NB
      call bmod(A,ii,jj,kk)
!$OMP &SUCC( lu0:kk+1 ) IF( jj.eq.kk+1 .and. ii.eq.kk+1 )
    enddo
  enddo
!$OMP END PARALLEL SECTIONS
enddo
```

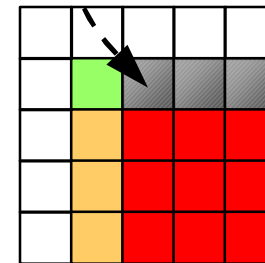
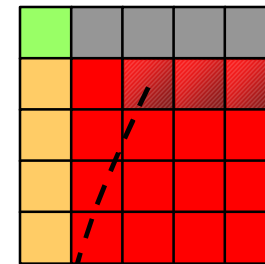


Another example: blocked LU

■ Precedence definition (bmod \rightarrow fwd)

```
!$OMP NAME(fwd:kk) PRED (lu0:kk)
!$OMP PARALLEL DO NAME(fwd_loop:kk)
do jj=kk+1,NB
!$OMP PRED ( bmod_loop:kk-1, jj ) IF(kk.gt.1)
  call fwd(A,kk,jj)
enddo

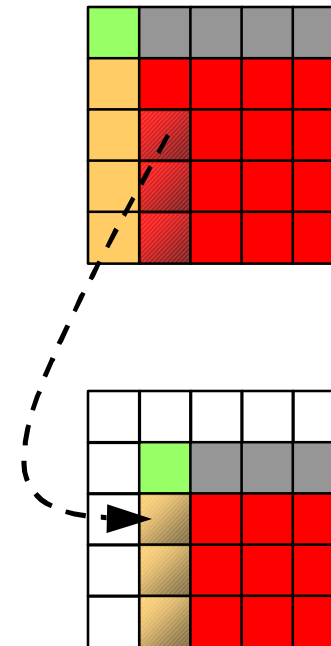
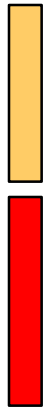
!$OMP NAME( bdiv+bmod:kk ) PRED( lu0:kk )
!$OMP PARALLEL DO
do ii=kk+1,NB
  call bdiv(A,ii,kk)
!$PARALLEL DO NAME( bmod_loop:kk:ii )
  do jj=kk+1,NB
    call bmod(A,ii,jj,kk)
!$OMP SUCC( fwd_loop:kk+1, jj ) IF(ii.eq.kk+1)
  enddo
enddo
```



Another example: blocked LU

■ Precedence definition (bmod \rightarrow bdiv)

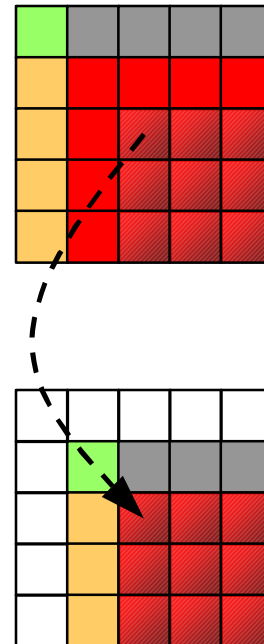

```
!$OMP NAME(bdiv+bmod:kk)
!$OMP PARALLEL DO
  do ii=kk+1,NB
!$OMP PRED( bmod_loop:kk-1:kk, ii ) IF(kk.gt.1)
    call bdiv(A,ii,kk)
!$PARALLEL DO NAME( bmod_loop:kk:ii )
    do jj=kk+1,NB
      call bmod(A,ii,jj,kk)
!$OMP SUCC( bdiv+bmod:kk+1 ) IF(jj.eq.kk+1)
    enddo
  enddo
enddo
```



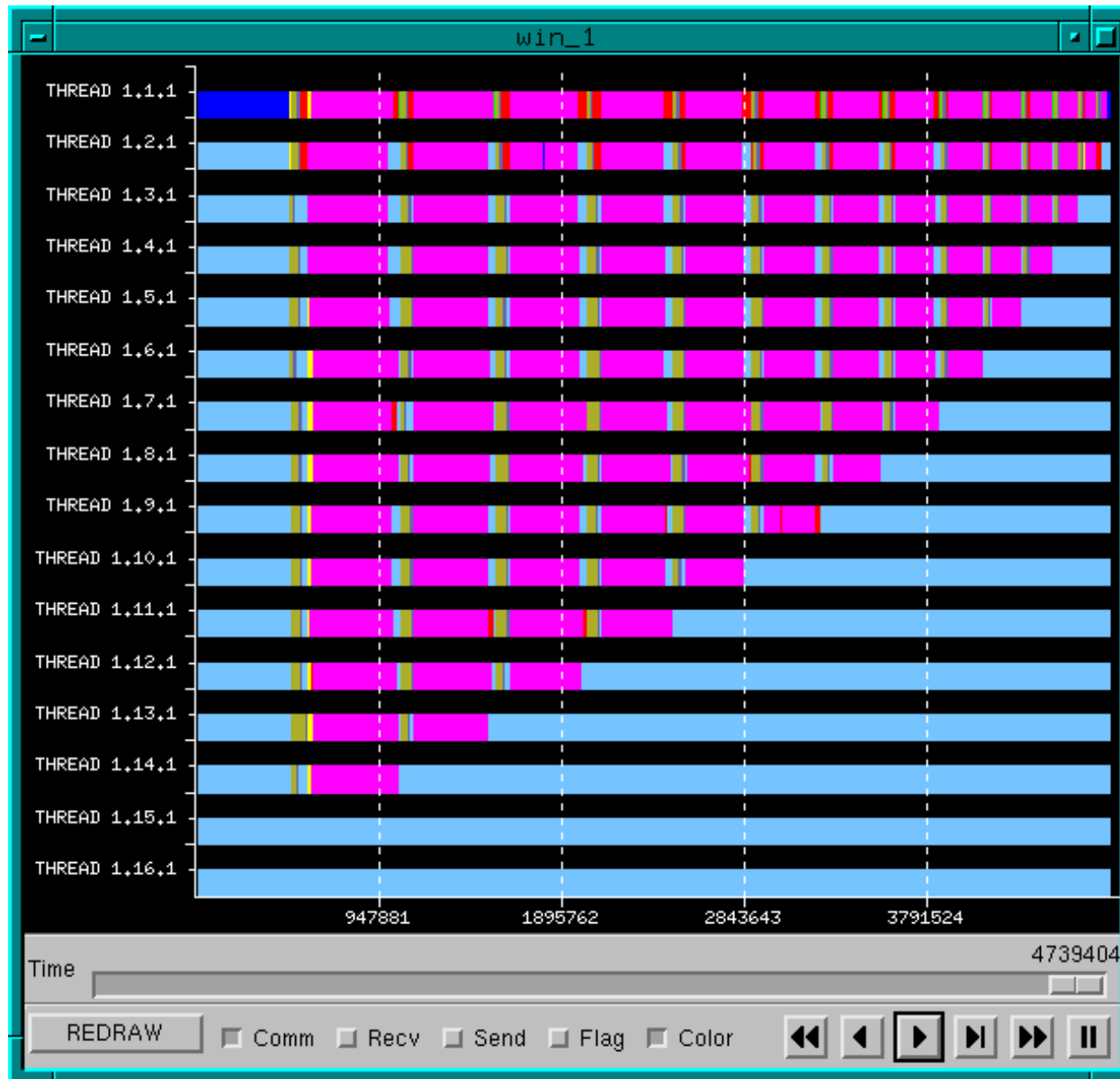
Another example: blocked LU

■ Precedence definition (bmod \rightarrow bmod)

```
!$OMP NAME(bdiv+bmod:kk)
!$OMP PARALLEL DO
  do ii=kk+1,NB
    call bdiv(A,ii,kk)
!$PARALLEL DO NAME(bmod_loop:kk:ii)
  do jj=kk+1,NB
!$OMP PRED( bmod_loop:kk-1:ii, jj ) IF( kk.gt.1)
    call bmod(A,ii,jj,kk)
!$OMP SUCC( bmod_loop:kk+1:ii, jj ) IF(jj.gt.kk+1 .and. ii.gt.kk+1)
  enddo
enddo
```



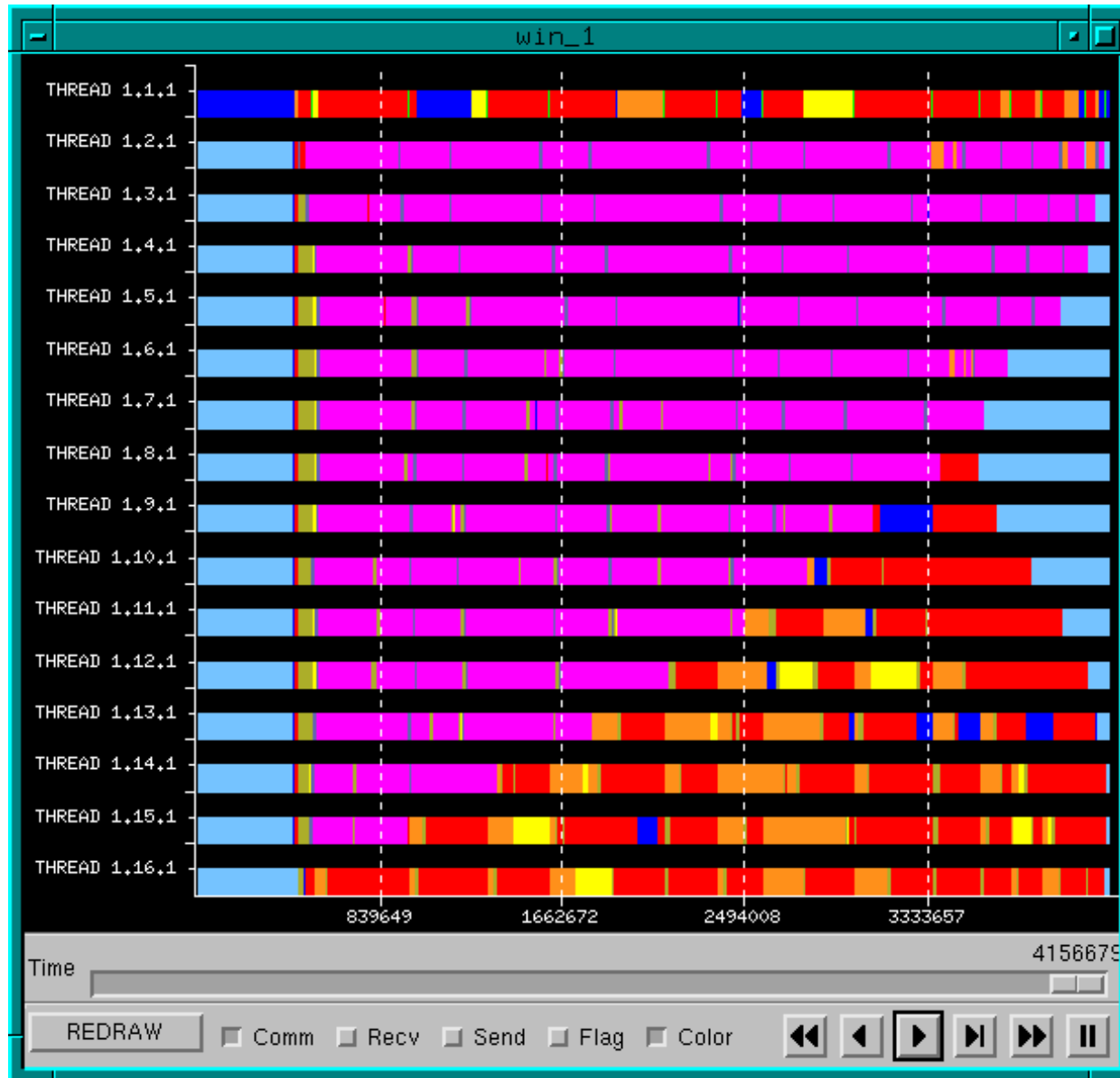
Another example: blocked LU



No precedences



Another example: blocked LU



With precedences



Conclusions

- **Extensions to OpenMP to specify precedence relations among tasks**
- **Framework of multiple levels of parallelism**
- **Implementation issues:**
 - Based on the use of synchronization through shared-memory
 - Mechanisms in the thread library to queue tasks blocked in a precedence (timeout)
 - In progress in the NanosCompiler
- **Promising results and applicability**

