Exploit Multi-Level Parallelism in OpenMP

Henry Jin
hjin@nas.nasa.gov
NASA Advanced Supercomputing Division
NASA Ames Research Center

Multi-Level Parallelism (MLP)

- Single level
  - performance limited by single dimension (loop iteration) size
- Multi level
  - increased space for parallelization
  - reduced surface-to-volume ratio
    - potentially improve performance

```
do K=1,20
   do J=1,30
   . . .
   enddo
enddo
```

one level, 20 iters
two level, 20x30 iters

Support of MLP in OpenMP

- Nested OpenMP
  - defined in the standard, supported in a limited number of commercial compilers (e.g. IBM XL compiler, Intel 8 compiler)
  - research projects
    - NanosCompiler – with additional extension
    - OmniCompiler
  - cannot avoid synchronization at the end of inner parallel regions

- OpenMP extensions
  - SGI '**`NEST`**' clause
    - for perfectly nested loops

- Task-based parallelism
  - Intel/KAI work
    - dynamic nature

A Better Approach?

- Question: Is there a more efficient way to exploit MLP?
  - Avoid using nested OpenMP (synchronization issue)
  - Handle more general cases
    - not just perfect loop nests
  - A light weighted approach

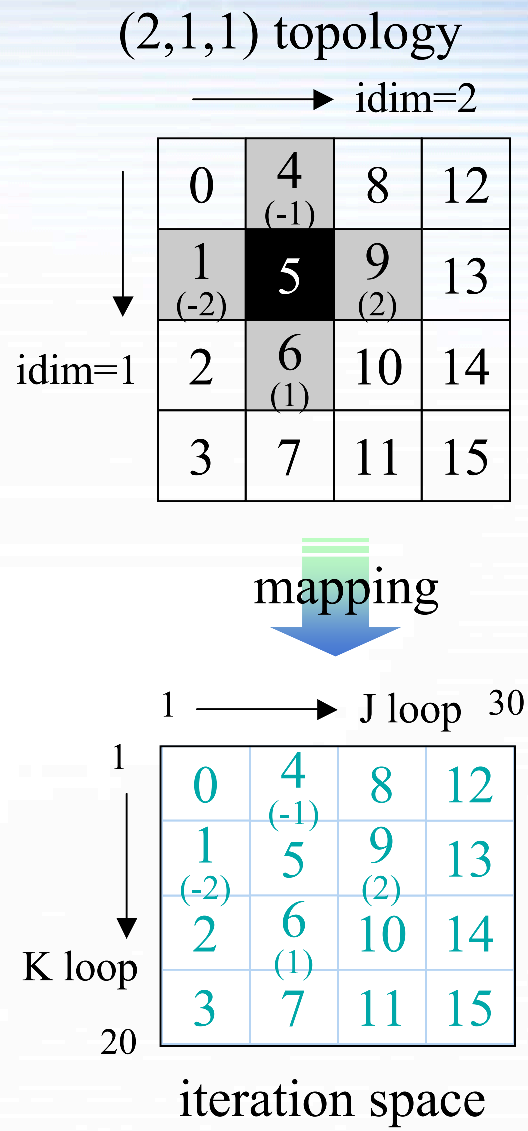
- A proposed work on exploiting multidimensional parallelism (MOMP directives)
  - Presented by H. Jin and G. Jost at WOMPEI 2003 (LNCS2858, p.511)

Thread Topology

- **`TMAP(ndim[,shape])`**
  - define a thread topology for a team of threads
- **`MDO(idim[,gilow,gihigh])`**
  - bind a thread topological dimension to a loop

```
!$OMP MDO(1)
   DO K=1,20
!$OMP MDO(2)
      DO J=1,30
```

- **`TSIGNAL(idir[,idir])`**
  `TWAIT  (idir[,idir])`
  - synchronize between two threads
    idir – direction of a neighboring thread

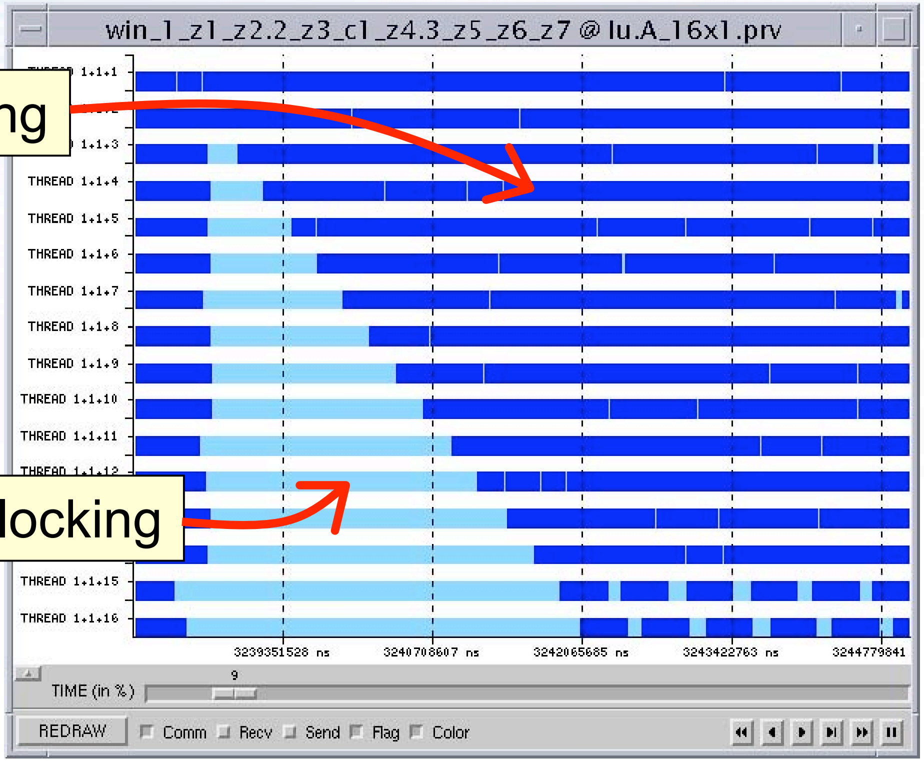


Coding Comparison

Code with MOMP directives	OpenMP with Nanos extensions	OpenMP with SGI extensions
!$OMP PARALLEL !$OMP& TMAP(2,NZ,0) !$OMP MDO(1) DO K=1,NZ *ZETA = K*0.1* !$OMP MDO(2) DO J=1,NY *do more work* ENDDO ENDDO !$OMP END PARALLEL	!$OMP PARALLEL !$OMP& GROUPS(NZ) !$OMP DO DO K=1,NZ *ZETA = K*0.1* !$OMP PARALLEL DO DO J=1,NY *do more work* ENDDO !$OMP END PARALLEL DO ENDDO !$OMP END PARALLEL	!$OMP PARALLEL DO !$SGI+NEST(K,J) DO K=1,NZ DO J=1,NY *do more work* ENDDO ENDDO !$OMP END PARALLEL DO

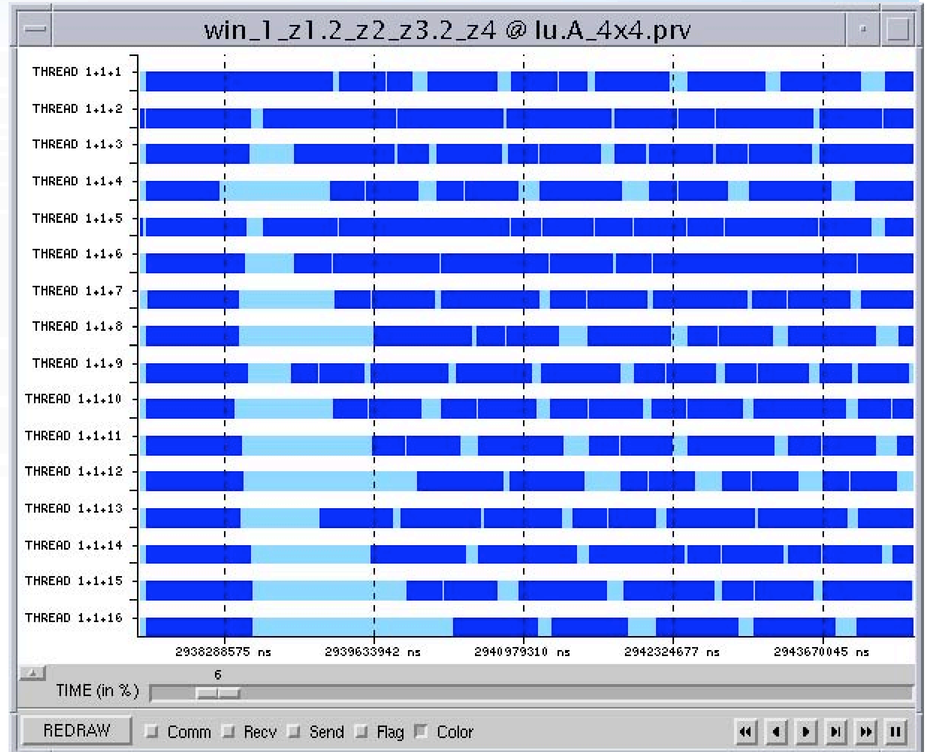
LU-MOMP: 1-D vs. 2-D Pipelining

1-D pipelining (16 cpus)




```
!$OMP PARALLEL TMAP(1)
    DO 10 K=2,NZ-1
!$OMP TWAIT(-1)
!$OMP MDO(1)
      DO 20 J=2,NY-1
        V(..,J,K)=V(..,J-1,K)+..
 20   CONTINUE
!$OMP TSIGNAL(1)
 10 CONTINUE
```

2-D pipelining (4×4 cpus)




```
!$OMP PARALLEL TMAP(2,1,1)
    DO 10 K=2,NZ-1
!$OMP TWAIT(-1,-2)
!$OMP MDO(1)
      DO 20 J=2,NY-1
!$OMP MDO(2)
      DO 20 I=2,NX-1
        V(I,J,K)=V(I-1,J,K)+..
 20   CONTINUE
!$OMP TSIGNAL(1,2)
 10 CONTINUE
```

Items in the Wish List

- Make “NOWAIT” between loop nests more useful
- Uniform runtime control
  - e.g. master/slave stacksize
  - a method to clean up threads (opposite to creating threads)
- Synchronization among a subset of threads
  - notion of “subteam”
  - point-to-point synchronization
- Thread topology
  - Poster by B. Chapman, L Huang, H. Jin, G. Jost, and B. de Supinski