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

```plaintext
    do K=1,20
        do J=1,30
            ...
            enddo
        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(\text{ndim[,shape]})**
  - define a thread topology for a team of threads

- **MDO(\text{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])**
  - synchronize between two threads
  - \text{idir} – direction of a neighboring thread

- **TWAIT (idir[,idir])**

(2,1,1) topology

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
```

idim=2

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
```

idim=1

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>11</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
```

idim=1

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
```

mapping

iteration space J loop

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
```

iteration space K loop

```
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
```
# Coding Comparison

<table>
<thead>
<tr>
<th>Code with MOMP directives</th>
<th>OpenMP with Nanos extensions</th>
<th>OpenMP with SGI extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>!$OMP PARALLEL</td>
<td>!$OMP PARALLEL</td>
<td>!$OMP PARALLEL DO</td>
</tr>
<tr>
<td>!$OMP&amp; TMAP(2,NZ,0)</td>
<td>!$OMP&amp; GROUPS(NZ)</td>
<td>!$SGI+NEST(K,J)</td>
</tr>
<tr>
<td>!$OMP MDO(1)</td>
<td>!$OMP DO</td>
<td>DO K=1,NZ</td>
</tr>
<tr>
<td>DO K=1,NZ</td>
<td>DO K=1,NZ</td>
<td>DO J=1,NY</td>
</tr>
<tr>
<td>ZETA = K*0.1</td>
<td>ZETA = K*0.1</td>
<td>do more work</td>
</tr>
<tr>
<td>!$OMP MDO(2)</td>
<td>!$OMP PARALLEL DO</td>
<td>ENDDO</td>
</tr>
<tr>
<td>DO J=1,NY</td>
<td>!$OMP MDO(2)</td>
<td>ENDDO</td>
</tr>
<tr>
<td>do more work</td>
<td>DO J=1,NY</td>
<td>do more work</td>
</tr>
<tr>
<td>ENDDO</td>
<td>ENDDO</td>
<td>ENDDO</td>
</tr>
<tr>
<td>!$OMP END PARALLEL</td>
<td>!$OMP END PARALLEL DO</td>
<td>!$OMP END PARALLEL DO</td>
</tr>
<tr>
<td>ENDVO</td>
<td>ENDDO</td>
<td>ENDDO</td>
</tr>
<tr>
<td>!$OMP END PARALLEL</td>
<td></td>
<td>!$OMP END PARALLEL DO</td>
</tr>
</tbody>
</table>

**OpenMP with Nanos extensions**

```c
!$OMP PARALLEL
!$OMP& GROUPS(NZ)
!$OMP MDO(2)
DO J=1,NY
  do more work
ENDDO
!$OMP MDO(1)
DO K=1,NZ
  do more work
ENDDO
!$OMP END PARALLEL DO
```

**OpenMP with SGI extensions**

```c
!$OMP PARALLEL DO
!$OMP& 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)

2-D pipelining (4×4 cpus)

```plaintext
 !$OMP PARALLEL TMAP(1)
   DO 10 K=2,NZ-1
 !$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

 !$OMP PARALLEL TMAP(2,1,1)
   DO 10 K=2,NZ-1
 !$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