OpenMP Doacross Loops Case Study

November 14, 2017
Gabriele Jost and Henry Jin
Outline

• Background
  – The OpenMP doacross concept

• LU-OMP implementations
  – Different algorithms
  – Manual synchronization vs doacross

• Performance Analysis:
  – Synchronization
  – Work scheduling
  – Compilers

• Summary and Conclusions
• OpenMP 4.0:
  - **ordered** clause for the work sharing construct
  - **ordered** construct to enclose a structured block within the loop body
    - Statements in the structured block are executed in lexical iteration order
• OpenMP 4.5 **depend** clause for the **ordered** construct:
  - The **depend** clause is used to express dependences on earlier iterations via **sink** and **source** arguments
  - The **ordered** construct is placed within the loop body
  - Used to specify cross-iteration dependences

**Doacross Concept**

```c
#pragma omp for ordered(1)
for (i=1; i<n; i++)
...
#pragma omp ordered depend(sink: i-1)
b(i) = foe (a[i]), b[i-1]);
#pragma omp ordered depend(source)
...
```
Exploiting Parallelism with Doacross

• Hyperplane or pipeline algorithms lend themselves to doacross parallelism
• Example: Computational Fluid Dynamics (CFD) code OVERFLOW
  - LU matrix decomposition with Symmetric Gauss-Seidel (LU-SGS) line sweeping for the implicit portion of a time step
The NPB LU Pseudo-Application

• The code employs SSOR to solve a 7-band, block-diagonal matrix
• The method is factorized into Lower and Upper solver steps
• Both solver steps carry dependences in each spatial dimension
  – prevents straightforward OpenMP parallelization

```
...  
  do k=kst, kend  
    do j=jst, jend  
      do i=ist, iend  
        v(i,j,k)=v(i,j,k)+a*v(i-1,j,k)+b*v(i,j-1,k)  
          + c*v(i,j,k-1) + d  
      enddo
    enddo
  enddo
...  
```
LU Thread Manual Pipelining vs Doacross

- Using OpenMP Doacross
  - Place the `parallel` construct on k loop
  - Place the workshare and the `ordered` clause on the j loop
  - Place `ordered` constructs with `depend` clause in the j loop body
  - The j-loop is now ordered according to the dependences specified

```c
!$omp parallel
do k=kst, kend
  call sync_left(nx,ny,nz,rsd)
!$omp do schedule(static)
do j=jst, jend
  call jacld(a,b,c,d,j,k)
  call blts(a,b,c,d,rsd,j,k)
enddo
!$omp enddo nowait
  call sync_right(nx,ny,nz,rsd)
enddo
```

```c
subroutine blts(a,b,c,d,rsd,j,k)
do i = ist, iend
  do m = 1, 5
    ... v(m,i,j,k) = v(m,i,j,k) -omega*(ldy(m,1,i)*v(m,i-1,j,k) + v(m,i,j-1,k) ...) 
  end do
end do
```

Where is i?
- i-loop is hidden in jacld, blts
- i-loop in jacld is vectorizable

```c
!$omp parallel
do k=kst, kend
!$omp do schedule(static) ordered(1)
do j=jst, jend
!$omp ordered depend(sink:j-1)
  call jacld(a,b,c,d,j,k)
  call blts(a,b,c,d,rsd,j,k)
!$omp ordered depend(source)
enddo
!$omp enddo nowait
```

```c
sync_left
```

```c
sync_right
```

Where is i?
- i-loop is hidden in jacld, blts
- i-loop in jacld is vectorizable
LU 2D Hyperplane Manual Implementation

- Hyperplane $l$ contains all points where $l = j + k$
- Points on a hyperplane can be computed independently
- Transform loop over $(j, k)$ into a loop over $(l, jk)$, where $jk$ iterates over the points within the hyperplane
- No explicit thread-to-thread synchronization is required
- Requires restructuring of the loop

```c
!$omp parallel private (j,l,k)
do l=1st, lend
!$omp do schedule(static)
do jk =max(l-jend,jst), min(l-2,jend)
j = jk
k = l - jk
call jacld(a,b,c,d,j,k)
call blts(a,b,c,d,rsd,j,k)
enddo
!$omp enddo
enddo
```

Implicit barrier synchronization
LU 2D Hyperplane Doacross Implementation

- Maintains the original code structure
  - Extend dependences across 2 dimensions
  - Work sharing on $k$ loop with `ordered (2)` clause
  - `ordered` construct with 2 sinks

```c
!$omp do schedule(static,1) ordered(2)
do  k=kst, kend
  do  j=jst, jend
  !$omp ordered depend(sink:k-1,j) &
  !$omp&    depend(sink:k,j-1)
    call jaclld(a,b,c,d,j,k)
    call blts(a,b,c,d,rsd,j,k)
  !$omp ordered depend(source)
  enddo
enddo
```

- Best work balancing achieved using chunk size 1
- Only dependences are specified
- Work schedule dependent on compiler generated synchronization and runtime library
LU 3D Hyperplane Doacross

- The nested loops run within the original loop bounds
- Place `ordered(3)` clause on the triply nested loop
- Place `ordered` clause with 3 sinks in the loop body

```c
!$omp do schedule(static,1) ordered(3)
   do k = 2, nz -1
      do j = jst, jend
         do i = ist, iend
            !$omp ordered depend(sink: k-1,j,i) depend(sink: k,j-1,i)
            !$omp& depend(sink: k,j,i-1)
            call jacld(... i, j, k)
            call blts( isiz1, isiz2, isiz3,
                  > nx, ny,nz,omega,crsd,a,b,c,d,i,j,k)
            !$omp ordered depend(source)
            end do
            end do
      end do
   end do
end do
```
Evaluation Environment

• Pleiades Super Computer and a KNL cluster at NASA Ames Research Center
• Intel Xeon Broadwell dual E5-2680v4 (2.4GHz) with 28 cores
• Intel Xeon Phi ™ with CPU 7230 (1.3 GHz) with 64 cores
• GNU gcc 7.1 : gfortran provides OpenMP 4.5 support
  – -O3 -fopenmp -mavx2 -g for Xeon Broadwell
  – -O3 -mavx512f -fopenmp -g on KNL
• Intel ifort version 2017.1.132
  – -O3 -ipo -axCORE-AVX2 -qopenmp -g for Xeon Broadwell
  – -O3 -xmic-avx512 -ipo -g for KNL
• Performance analysis:
  – op_scope:
    • Low overhead profiling tool for OpenMP and MPI; licensed software
  – Paraver 4.6:
    • Flexible performance analysis tool distributed by the Barcelona Supercomputer Center
Class LU Class C Performance with GNU GCC

- Observations:
  - manual and doac performance is similar!
  - doac-hp2 slightly out-performs manual-hp2
  - Performance behavior on Broadwell and KNL is similar
  - hp2 performs better than pipelined execution
  - hp3 performs poorly:
    - Index calculations
    - Lack of vectorization

Size LU Class C
nx=ny=nz=162
## manual-pipe vs doac-pipe on Xeon Broadwell

### pipe-manual.C.x secs

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Elap:</td>
<td>30.71</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User:</td>
<td>767.40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sys:</td>
<td>67.48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread:</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total:</td>
<td>33.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.88</td>
<td>rhs__omp_fn.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.33</td>
<td>blts_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.05</td>
<td>buts_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.83</td>
<td>jacu_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.79</td>
<td>jacld_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.70</td>
<td>+sync_left</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.88</td>
<td>ssor__omp_fn.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.35</td>
<td>libgomp.so.1.0.0::gomp_team_barrier_wait_end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.07</td>
<td>exact_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.03</td>
<td>erhs__omp_fn.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.02</td>
<td>+sync_right</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>libgomp.so.1.0.0::gomp_barrier_wait_end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### doac-pipe.C.x secs

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Elap:</td>
<td>30.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User:</td>
<td>768.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sys:</td>
<td>67.93</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread:</td>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Total:</td>
<td>33.88</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Symbol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9.83</td>
<td>rhs__omp_fn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.33</td>
<td>blts_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.24</td>
<td>buts_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.67</td>
<td>jacl_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.48</td>
<td>jacu_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.89</td>
<td>+libgomp.so.1.0.0::GOMP_doacross_wait</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.89</td>
<td>ssor__omp_fn.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>libgomp.so.1.0.0::gomp_team_barrier_wait_end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.07</td>
<td>exact_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.03</td>
<td>erhs__omp_fn.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>+libgomp.so.1.0.0::GOMP_doacross_post</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>libgomp.so.1.0.0::gomp_barrier_wait_end</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **op_scope** profile for Class C; GNU GCC 7.1
- Time in seconds based on CPU_CLK_UNHALTED
- Displayed is the most time consuming thread
- Observation:
  - GOMP doacross synchronization time very close time in synchronization routines
Paraver Timelines for pipelined LU

doac-pipe

- Class A on 8 threads
- Paraver performance analysis tool timeline
  - Vertical axis indicates time
  - Horizontal axis indicates thread ID
- Time spent in `blts` (dark blue), `buts` (white), tracing disabled (light blue)
- Traced are all even iterations
- Clearly detectable pipeline for both implementations

manual-pipe
2D Hyperplane Work Schedule Comparison on KNL

- Class C, GNU gcc 7.1/gfortran
- Observations:
  - doac-hp2 with chunk size 1 performance is similar to manual-hp2
  - doac-hp2 static with chunk size > 1 performs poorly
  - doac-hp2 static(schedule,1) performance boost for 162 threads!
- Similar observations on Xeon Broadwell

Chunk size 1?
GNU gcc/gfortran vs Intel ifort Performance

- **version.A** indicates gcc performance, **version_intel.A** indicates ifort performance
- Observations:
  - gfortran and ifort yield similar performance for manual synchronization
  - Synchronization problem in ifort for doac-pipe implementations
- Similar Observations for Class C
Summary

• We compared different implementations of the NPB-OMP Benchmark LU
  - Manual synchronization vs OpenMP 4.5 doacross
  - Performance vs productivity
  - Compiler support

• OpenMP 4.5 provides ease of use
  - does not require restructuring of the original code

• Performance of manual synchronization vs doacross is comparable
  - doacross performance depends on choosing appropriate scheduling
    o schedule (static,1) worked best for our LU benchmark

○ Drawbacks
  - Correctness and performance depends a lot on the quality of the compiler
  - Performance analysis and debugging is difficult, due to the dependence on compiler transformation and OMP runtime library

• Good news
  - gcc 7.1 provides full support (Fortran and C) and yields acceptable performance!
References

- **OpenMP Doacross:**

- **OpenMP 4.5 Specifications**

- **OpenMP 4.5 Examples**

- **Paraver**
  - [https://tools.bsc.es/paraver](https://tools.bsc.es/paraver)

- **op_scope**

- **NAS Parallel Benchmarks:**
  - [https://www.nas.nasa.gov/publications/npb.html](https://www.nas.nasa.gov/publications/npb.html)
• Overflow LU-SGS:

• Pleiades Super Computer
  - https://www.nas.nasa.gov/hecc/resources/pleiades.html