National Aeronautics and Space Administration

OpenMP Doacross Loops Case Study

November 14, 2017

Gabriele Jost and Henry Jin

www.nasa.gov

Outline

- Background
  - The OpenMP doacross concept
- LU-OMP implementations
  - Different algorithms
  - Manual synchronization vs doacross
- Performance Analysis:
  - Synchronization
  - Work scheduling
  - Compilers
- Summary and Conclusions

Background

- OpenMP 4.0:
  - **ordered** clause for the work sharing construct
  - **ordered** construct to enclose a structured block within the loop body
    - Statments 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

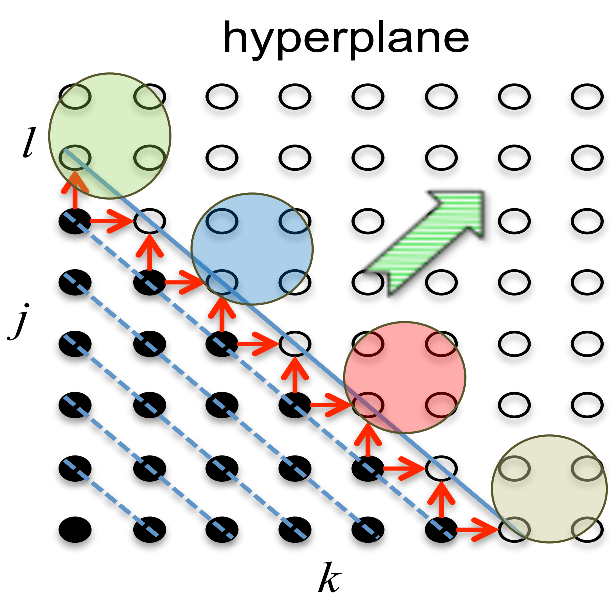
Wait point for completion of iteration i-1

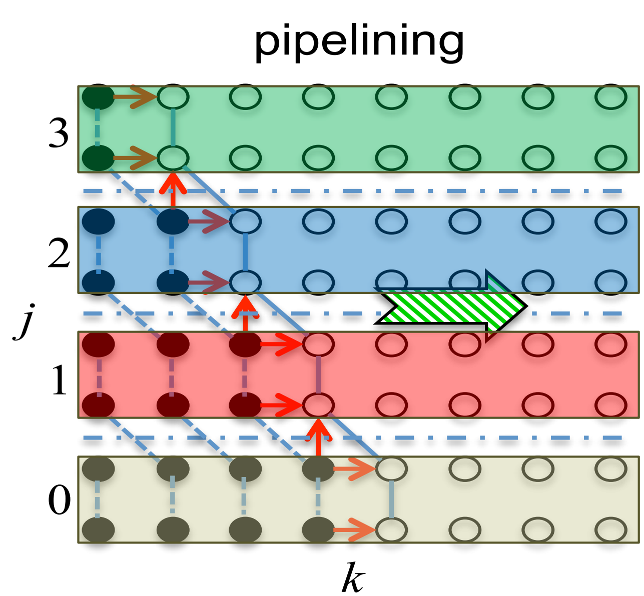
Signal completion of iteration i

```
#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 **L**ower and **U**pper 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

manual

```
!$omp parallel
do k=kst, kend
  call sync_left(nx,ny,nz,rsd)
 !$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
```

```
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
```

doac

```
!$omp parallel
do k=kst, kend
 !$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
enddo
```

sync_left

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

```
!$omp parallel private (j,l,k)
 do l=lst, lend
!$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`**

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

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

Evaluation Environment

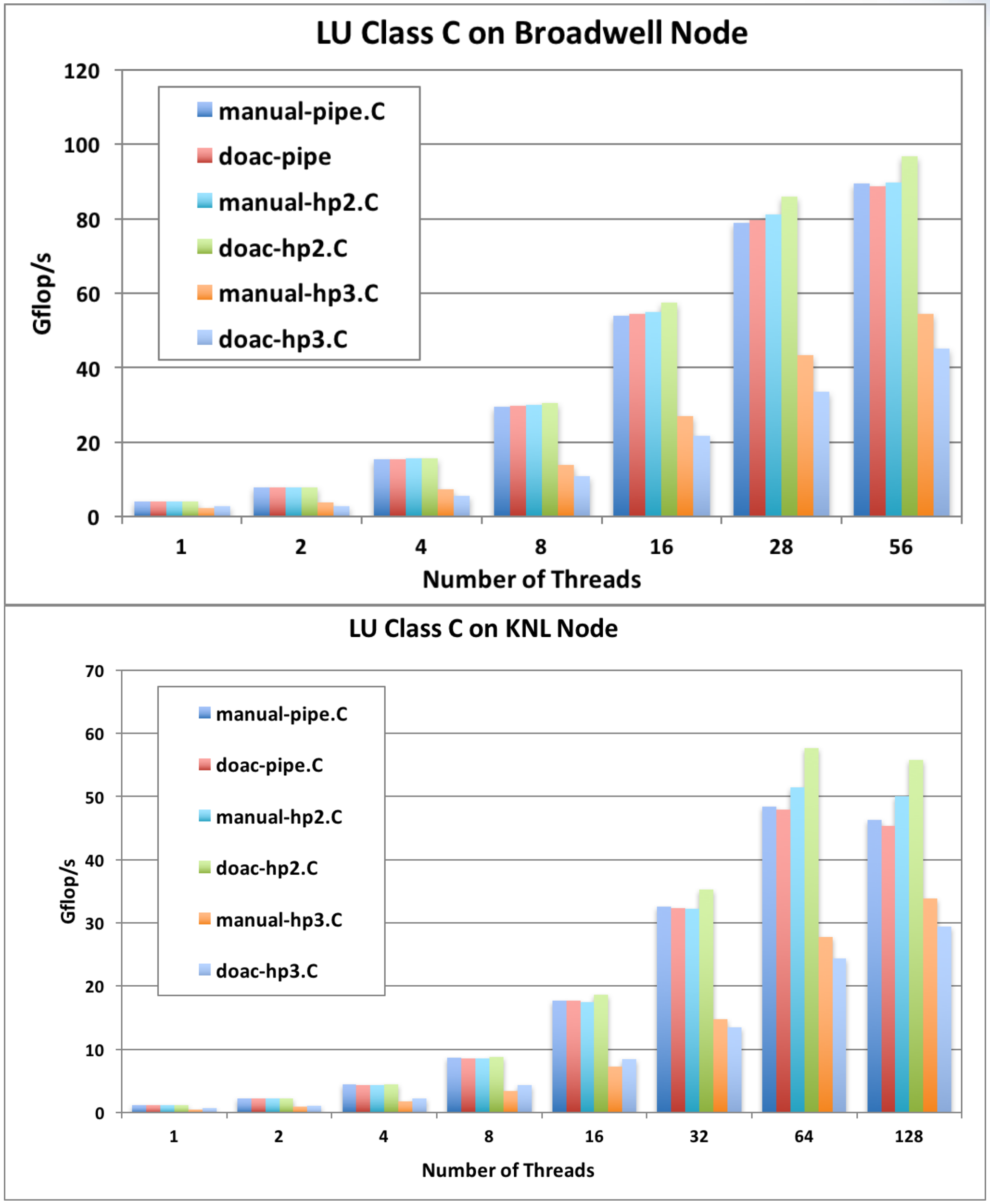
- Pleiades Super Computer and a KNL cluster at NASA Ames Research Center
- Intel Xeon Broadwell dual E5-2680v4 (2.4GHz) with 28 cores
- Intel Xeon Phi ™ with CPU 7230 (1.3 GHz) with 64 cores
- GNU gcc 7.1 : gfortran provides OpenMP 4.5 support
  - -O3 -fopenmp -mavx2 -g for Xeon Broadwell
  - -O3 -mavx512f –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

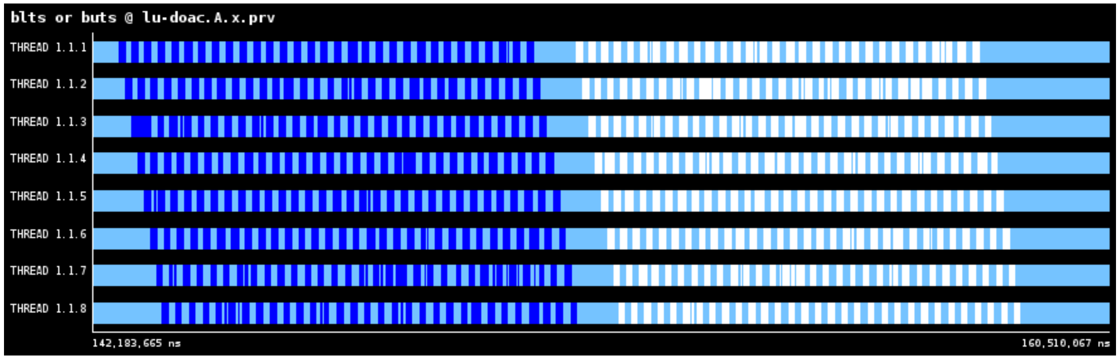
pipe-manual.C.x secs	**doac-pipe.C.x** secs
Elap: 30.71	Elap: 30.75
User: 767.40	User: 768.78
Sys: 67.48	Sys: 67.93
Thread: 19	Thread: 19
2.72 <-PARTIAL SUM	**2.90 <-PARTIAL SUM**
Total: 33.95 Symbol	Total: 33.88 Symbol
9.88 rhs_._omp_fn.0	9.83 rhs_._omp_fn
5.33 blts_	5.33 blts_
5.05 buts_	5.24 buts_
3.83 jacu_	3.67 jacld_
3.79 jacld_	3.48 jacu_
2.70 +sync_left_	**2.89 +libgomp.so.1.0.0::GOMP_doacross_wait**
1.88 ssor_._omp_fn.0	1.89 ssor_._omp_fn.0
1.35 libgomp.so.1.0.0::gomp_team_barrier_wait_end	1.3 libgomp.so.1.0.0::gomp_team_barrier_wait_end
0.07 exact_	0.07 exact_
0.03 erhs_._omp_fn.0	0.03 erhs_._omp_fn.0
0.02 +sync_right_	**0.01 +libgomp.so.1.0.0::GOMP_doacross_post**
0.01 libgomp.so.1.0.0::gomp_barrier_wait_end	0.01 libgomp.so.1.0.0::gomp_barrier_wait_end

PARTIAL SUM indicates the sum of the routines marked with +

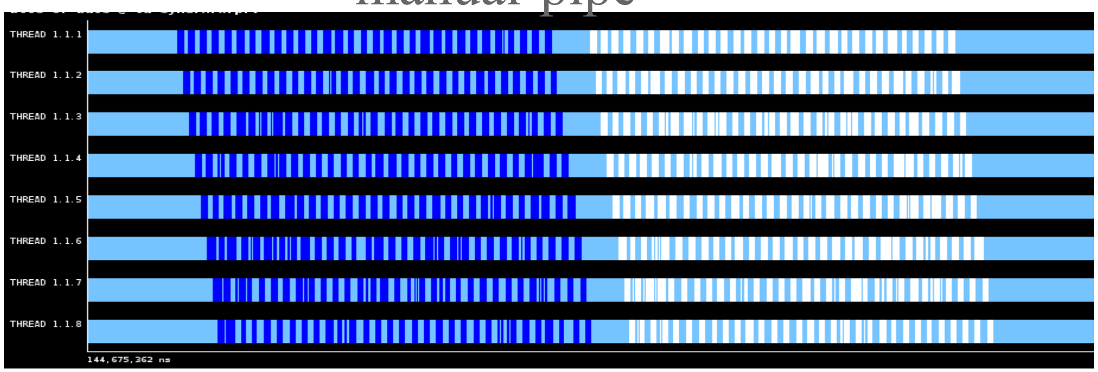
- *op_scope* profile for Class C; GNU GCC 7.1
- Time in seconds based on CPU_CLK_UNHALTED
- Displayed is the most time consuming thread
- Observation:
  - GOMP doacross synchronization time very close time in synchronization routines

Paraver Timelines for piplelined LU

doac-pipe

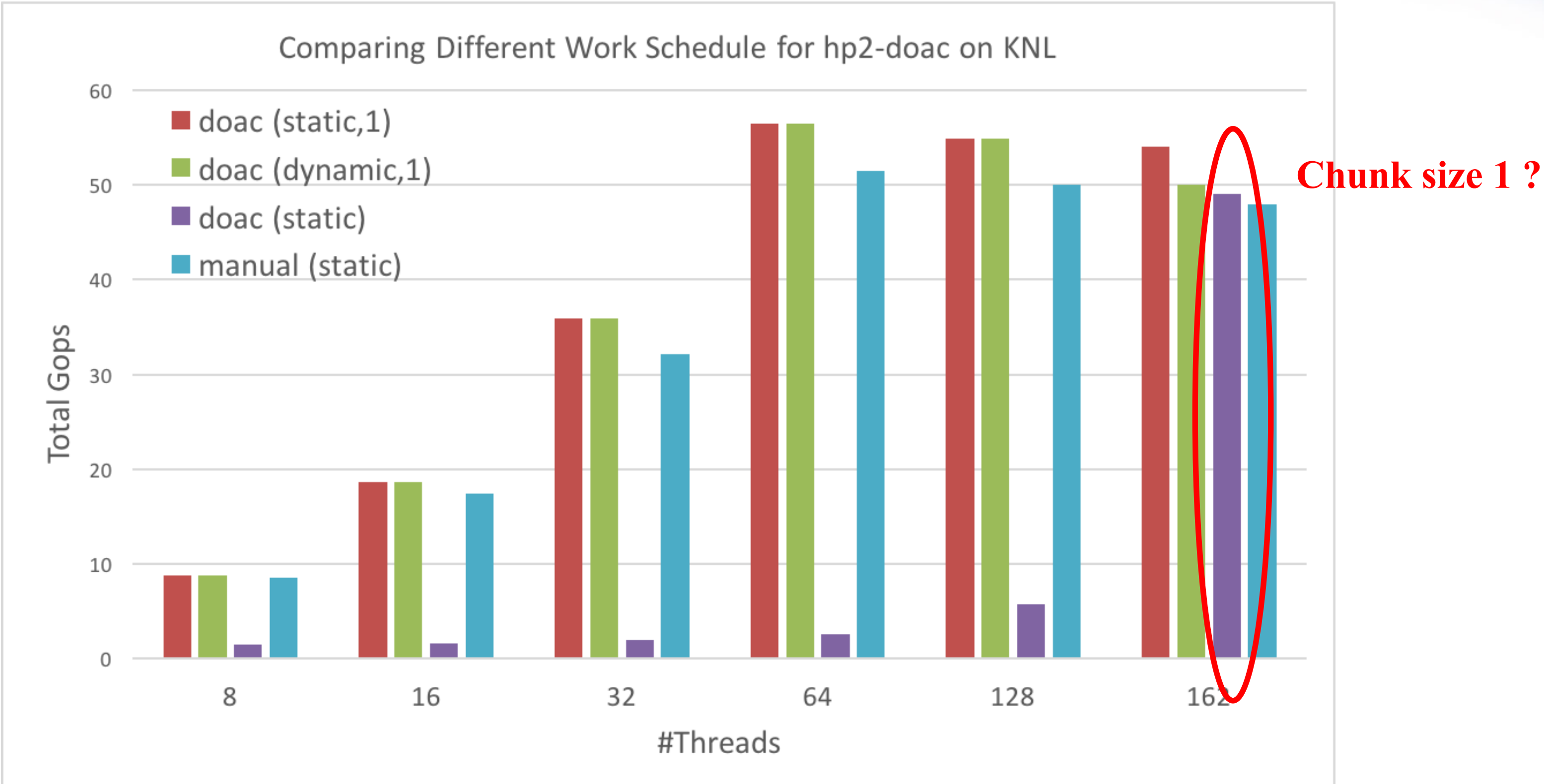


manual-pipe



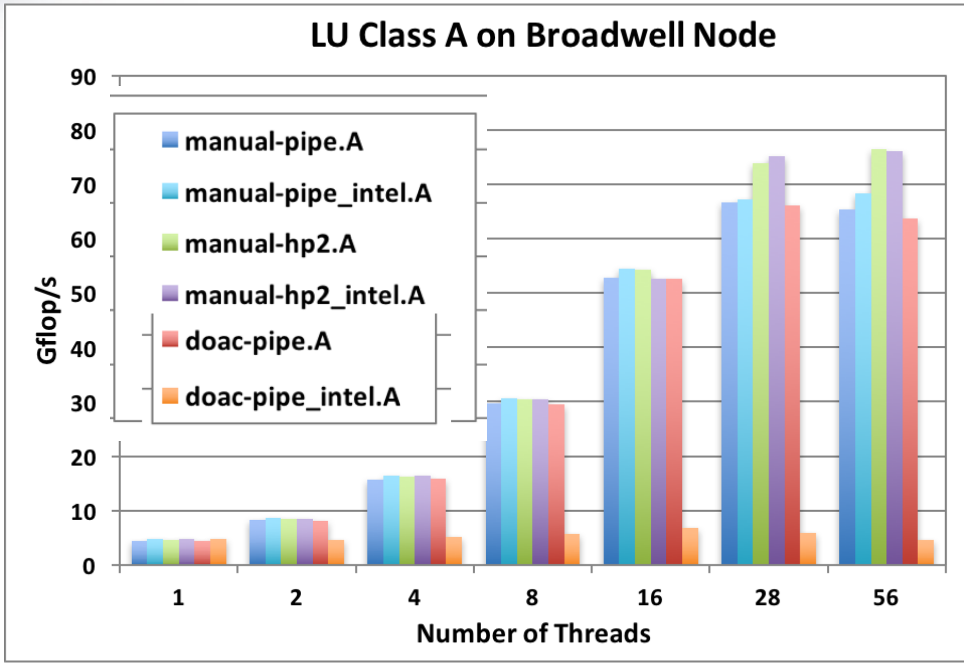
- Class A on 8 threads
- Paraver performance analysis tool time line
  - Vertical axis indicates time
  - Horizontal axis indicates thread ID
- Time spent in **`blts`** (dark blue), **`buts`** (white), tracing disabled (light blue)
- Traced are all even iterations
- Clearly detectable pipeline for both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

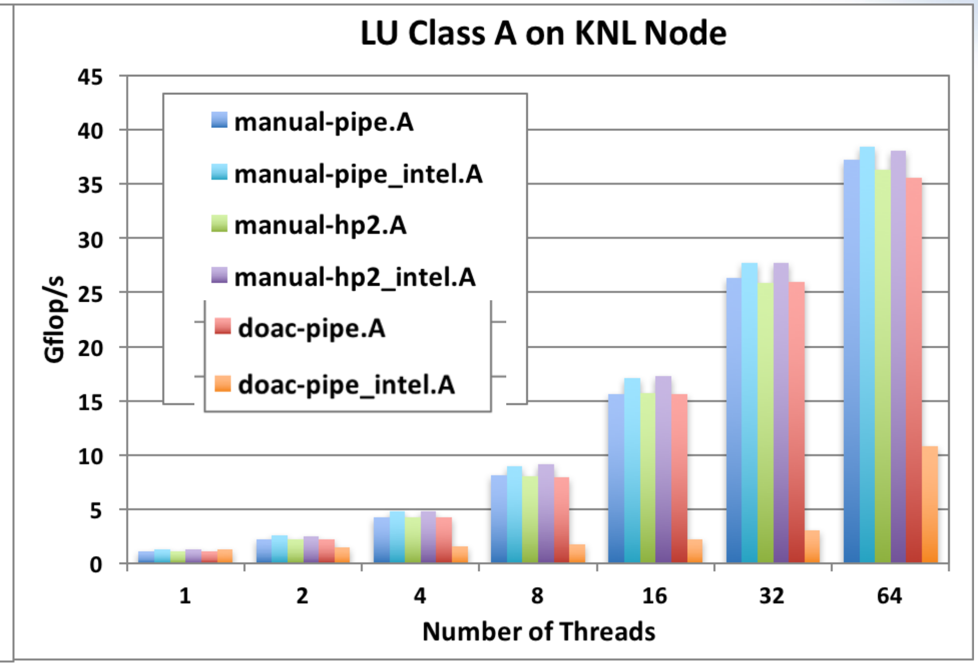
2D Hyperplane Work Schedule Comparison on KNL



- 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 for162 threads !
- Similar observations on Xeon Broadwell

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
    - 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:
  - Shirako J., Unnikrishnan P., Chatterjee S., Li K., Sarkar V. (2013) Expressing DOACROSS Loop Dependences in OpenMP. In: Rendell A.P., Chapman B.M., Müller M.S. (eds) OpenMP in the Era of Low Power Devices and Accelerators. IWOMP 2013. Lecture Notes in Computer Science, vol 8122. Springer, Berlin, Heidelberg
- OpenMP 4.5 Specifications
  - http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf
- OpenMP 4.5 Examples
  - http://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf
- Paraver
  - https://tools.bsc.es/paraver
- op_scope
  - http://www.supersmith.com/site/op_scope_files/op_scope_3.0_draft.pdf
- NAS Parallel Benchmarks:
  - H. Jin, M. Frumkin, J. Yan, NAS Technical Report NAS-99-011 October 1999
  - https://www.nas.nasa.gov/publications/npb.html

References

- Overflow LU-SGS:
  - R.H.Nichols, R.W.Trmel. P.G.Buning, “Solver and Turbulence Model Upgrades to OVEFLOW 2 for Unstaedy and High-Speed Applications”, 25th Applied Aerodynamics Conferencem 5-6 June 2006, San Francisco, California
- Pleiades Super Computer
  - https://www.nas.nasa.gov/hecc/resources/pleiades.html