Hybrid Parallel Programming on SMP Clusters using XPFortran and OpenMP

Y. Zhang, H. Iwashita, K. Ishii,
M. Kaneko, T. Nakamura, and K. Hotta

FUJITSU LIMITED

shaping tomorrow with you

Outline

- Introduction
- Hybrid Programming using XPFortran and OpenMP
  - Parallelize a Single Loop
  - Parallelize Nested Loops
- Performance Evaluation
- Conclusion

Outline

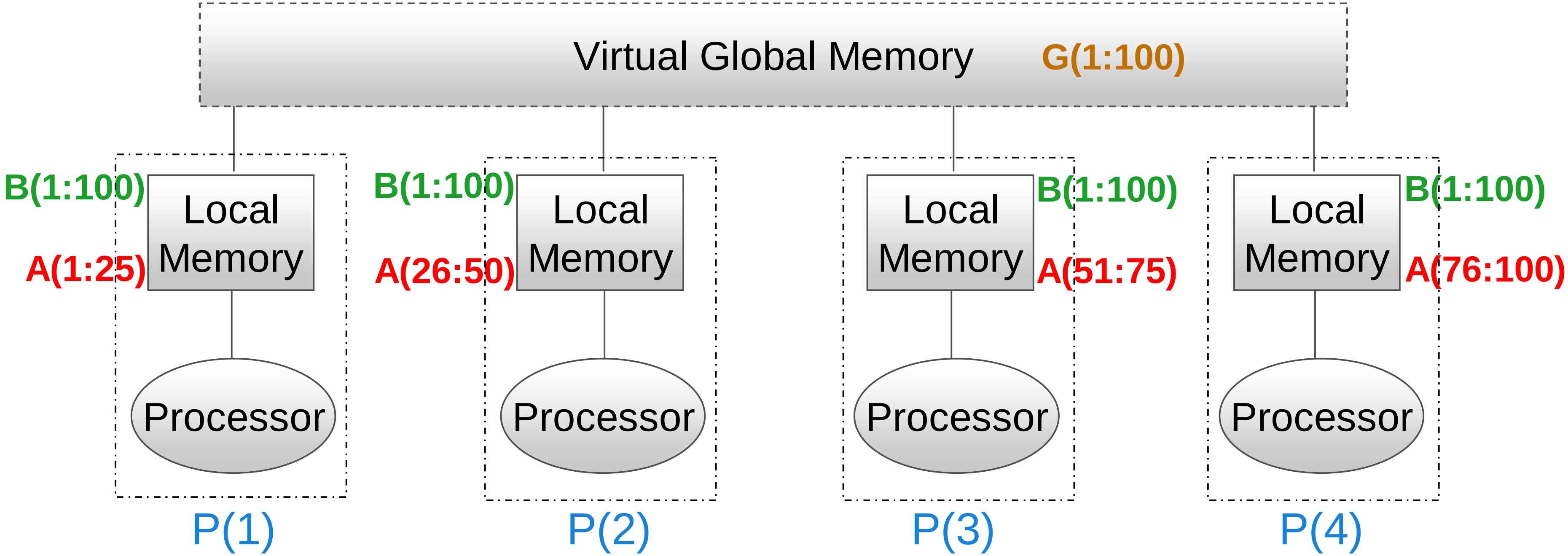
- Introduction
- Hybrid Programming using XPFortran and OpenMP
  - Parallelize a Single Loop
  - Parallelize Nested Loops
- Performance Evaluation
- Conclusion

Introduction

FUJITSU

- Process-thread hybrid programming is necessary
- MPI+OpenMP is widely used, however, MPI is difficult to program
- We implement hybrid programming with the data-parallel language XPFortran (XPF for short) and OpenMP
- This presentation discusses how to improve the performance of XPF-OpenMP hybrid programs
- What's XPF?
  - A data-parallel programming language for distributed memory, process-level parallelism
  - Fortran base, directive style
  - Hybrid execution with multi-threads, by OpenMP or automatic parallelization

Memory Model

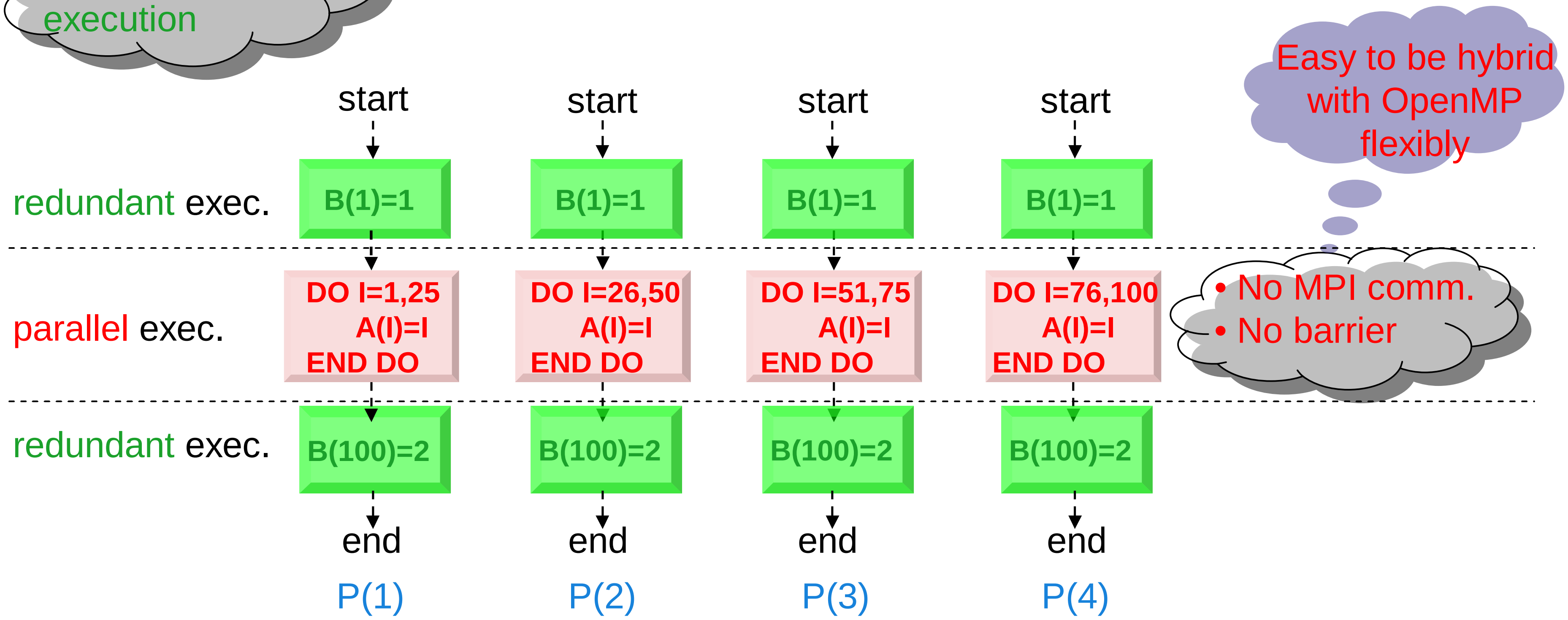



```
      INTEGER G(100), B(100), A(100)
!XOCL PROCESSOR P(4)        ! declare a 1-D process group P with 4 processes
!XOCL GLOBAL G              ! declare a global array
!XOCL LOCAL B               ! declare a duplicated local array
!XOCL LOCAL A(/(P))         ! declare a partitioned local array
```

Execution Model

```
B(1)=1
!XOCL SPREAD DO /(P)
DO I=1,100
   A(I)=I
END DO
!XOCL END SPREAD
B(100)=2
```

XPF program starts with redundant execution



One Example of SPMD Transformation for SPREAD DO

FUJITSU

XPF code

```
!XOCL PROCESSOR P(4)
      INTEGER A(1:100,1:100)
!XOCL LOCAL A(:,/(P))
      X=Y
!XOCL SPREAD DO
      DO J=1,100
         DO I=1,100
            A(I,J)=I+J
         END DO
      END DO
!XOCL END SPREAD
```

Transformed code

```
SUBROUTINE ORG___MAIN___()
INCLUDE 'mpif.h'
INTEGER(KIND=4):: a(1:100,1:25)
X=Y            ! redundant execution

! calculate start, initial value of j

! calculate end, final value of j

DO j=start,end,1
   DO i=1,100,1
      a(i,j) = i+j+25*ORG_RANK
   ENDDO
ENDDO

END SUBROUTINE
```

Outline

- Introduction
- Hybrid Programming using XPFortran and OpenMP
  - Parallelize a Single Loop
  - Parallelize Nested Loops
- Performance Evaluation
- Conclusion

Using OpenMP in XPF Programs

- OpenMP is supported in XPF
- Compilation of XPF-OpenMP hybrid program

XPF + OpenMP —compile→ Fortran + OpenMP + MPI call

- OpenMP directives can be used inside XPF constructs
- To improve performance, we also made it possible to use XPF SPREAD DO inside OpenMP constructs
- Here we only discuss hybrid parallelization of loops

Outline

FUJITSU

- Introduction
- Hybrid Programming using XPFortran and OpenMP
  - Parallelize a Single Loop
  - Parallelize Nested Loops
- Performance Evaluation
- Conclusion

Parallelize a Single Loop

```
!XOCL SPREAD DO
!$OMP PARALLEL DO
          DO I=1,800
               A(I)=I
          END DO
!XOCL END SPREAD
```

or

```
!$OMP PARALLEL DO
!XOCL SPREAD DO
          DO I=1,800
               A(I)=I
          END DO
!XOCL END SPREAD
```

Transformed code

```
          start=...
          end=...
!$OMP PARALLEL DO
          DO I=start,end,1
               A(I)=...
          END DO
```

Distribution of loop iterations to processes happens before that to threads no matter the order of the directives

Outline

FUJITSU

- Introduction
- Hybrid Programming using XPFortran and OpenMP
  - Parallelize a Single Loop
  - Parallelize Nested Loops
- Performance Evaluation
- Conclusion

Parallelize Nested Loops (1/4)

- For nested loops, if no. of processes is small (compared to no. of iterations in outermost loop), we can parallelize the outermost loop using both processes and threads

```
DO J=1,800
      DO I=1,100
            A(I,J)=I
      END DO
END DO
```

4 processes
4 threads
⇒

```
!XOCL SPREAD DO
!$OMP PARALLEL DO
      DO J=1,800
            DO I=1,100
                  A(I,J)=I
            END DO
      END DO
!XOCL END SPREAD
```

If NP*NT <= NJ

⇩

both processes and threads can parallelize loop J

NP: no. of processes   NT: no. of threads
NJ: no. of iterations in J

Parallelize Nested Loops (2/4)

FUJITSU

What shall we do when no. of processes is large?

If NP*NT > NJ ⇨ NT > NJ/NP ⇨ not all threads work

say NP=400
NT=4

NJ/NP: no. of iterations in J in each process

```
!XOCL SPREAD DO
!$OMP PARALLEL DO
      DO J=1,800
         DO I=1,100
            A(I,J)=I
         END DO
      END DO
!XOCL END SPREAD
```

Simple solution:

```
!XOCL SPREAD DO
      DO J=1,800
!$OMP PARALLEL DO
         DO I=1,100
            A(I,J)=I
         END DO
      END DO
!XOCL END SPREAD
```

To reduce overhead of thread fork:

```
!XOCL SPREAD DO
!$OMP PARALLEL
      DO J=1, 800
!$OMP DO
         DO I=1,100
            A(I,J)=I
         END DO
      END DO
!$OMP END PARALLEL
!XOCL END SPREAD
```

To reduce overhead of thread sync.:

```
!XOCL SPREAD DO
!$OMP PARALLEL
      DO J=1, 800
!$OMP DO
         DO I=1,100
            A(I,J)=I
         END DO
!$OMP END DO NOWAIT
      END DO
!$OMP END PARALLEL
!XOCL END SPREAD
```

However, problem of data reference locality still exists

Parallelize Nested Loops (3/4)

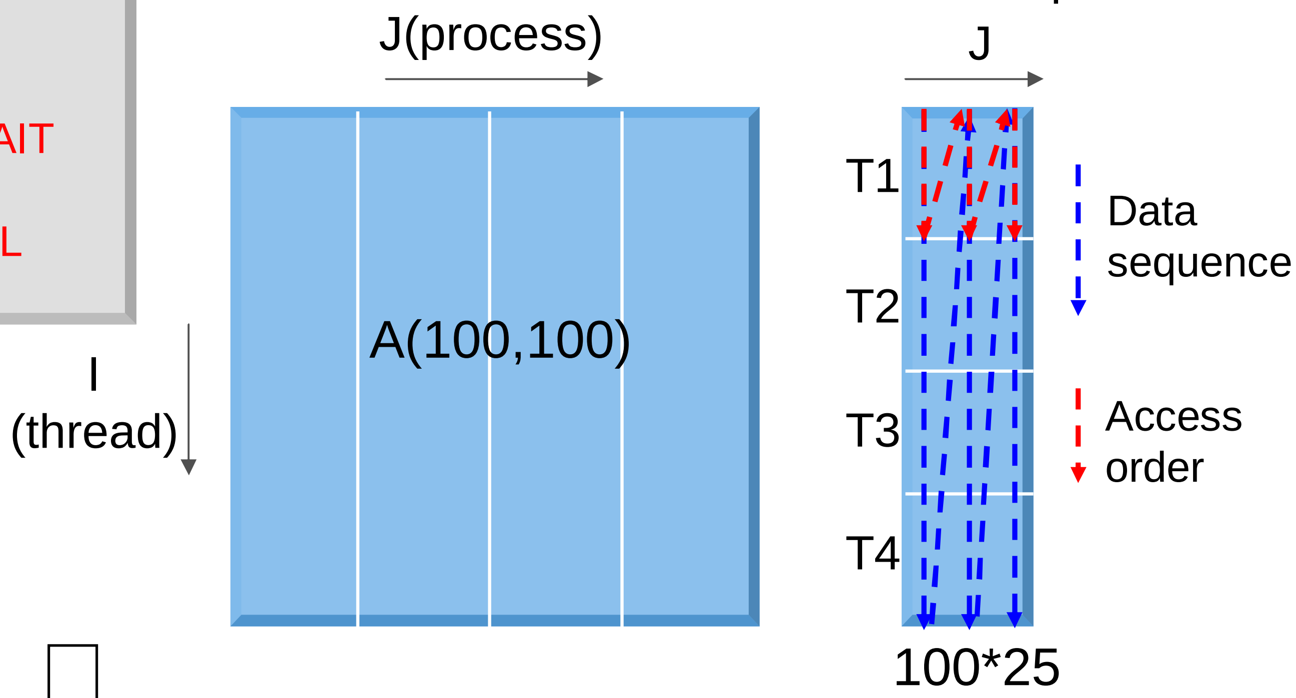
FUJITSU

```
        INTEGER A(100,100)
!XOCL SPREAD DO
!$OMP PARALLEL
        DO J=1,100
!$OMP DO
            DO I=1,100
              A(I,J)=I
            END DO
!$OMP END DO NOWAIT
        END DO
!$OMP END PARALLEL
!XOCL END SPREAD
```

In case of 4 processes, 4 threads:

```
!XOCL PROCESSOR P(4)
!XOCL LOCAL A(:,/(P))
```

A on each process:

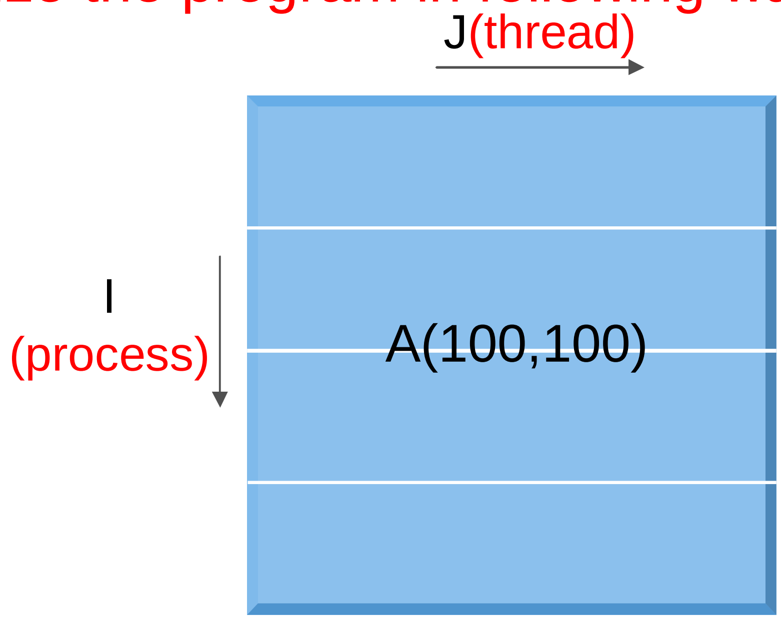


Data reference in each thread is discontinuous

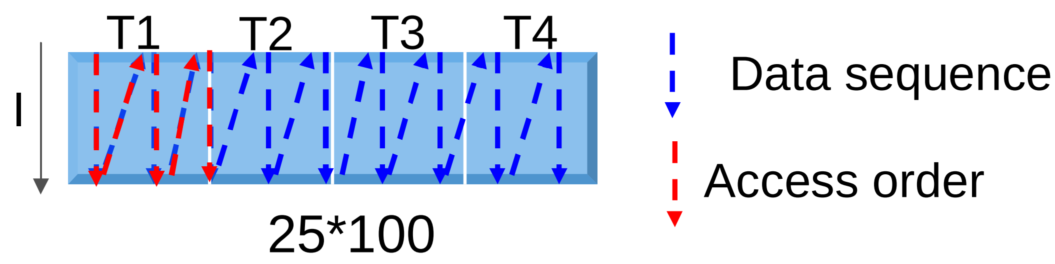
Parallelize Nested Loops (4/4)

To solve the problem, we parallelize the program in following way:

```
      INTEGER A(100,100)
!XOCL PROCESSOR P(4)
!XOCL LOCAL A(/(P),:)
!$OMP PARALLEL DO
      DO J=1,JUB
!XOCL SPREAD DO
         DO I=1,IUB
            A(I,J)=I
         END DO
!XOCL END SPREAD
      END DO
```



A on each process(in continuous memory space):



⇨ Spatial locality of data reference is improved

How XPF Makes This Feasible

FUJITSU

- ✓ No process fork happens
- ✓ No MPI comm. in SPMD output of SPREAD DO construct
- ✓ No barrier sync.

Transformed code

```
        INTEGER A(100,100)
!XOCL PROCESSOR P(4)
!XOCL LOCAL A(/(P),:)
!$OMP PARALLEL DO
        DO J=1,100
!XOCL SPREAD DO
            DO I=1,100
               A(I,J)=I
            END DO
!XOCL END SPREAD
        END DO
```

```
        INTEGER A(25,100)
!$OMP PARALLEL DO
        DO J=1,100
            start=...
            end=...
            DO I=start,end,1
               A(I,J)=...
            END DO
        END DO
```

Optimize

```
        INTEGER A(25,100)
        start=...
        end=...
!$OMP PARALLEL DO
        DO J=1,100
            DO I=start,end,1
               A(I,J)=...
            END DO
        END DO
```

Outline

- Introduction
- Hybrid Programming using XPFortran and OpenMP
  - Parallelize a Single Loop
  - Parallelize Nested Loops
- Performance Evaluation
- Conclusion

Evaluation Environment

- Himeno Benchmark
  - http://accc.riken.jp/HPC_e/HimenoBMT_e.html
  - Measures speed of main loops in Poisson equation solver in MFLOPS
- Fujitsu FX1
  - SPARC64^{TM} VII CPU, 4 cores
    - Peak performance: 40 GFLOPS/CPU
- Compilers and MPI library
  - XPFortran compiler (under development)
  - Fujitsu Fortran compiler 8.1
  - Fujitsu MPI-2 7.2 library

Core Part of Himeno

```
!$OMP PARALLEL DO PRIVATE(S0, SS) REDUCTION(+:GOSA)
        DO K=2,KMAX-1
!$OMP PARALLEL DO PRIVATE(S0, SS) REDUCTION(+:GOSA)
          DO J=2,JMAX-1
!$OMP PARALLEL DO PRIVATE(S0, SS) REDUCTION(+:GOSA)
            DO I=2,IMAX-1
              S0=a_l(I,J,K,1)*p_l(I+1,J,K)+a_l(I,J,K,2)*p_l(I,J+1,K)+a_l(I,J,K,3)
                   *p_l(I,J,K+1)+b_l(I,J,K,1)*(p_l(I+1,J+1,K)-p_l(I+1,J-1,K)
                   -p_l(I-1,J+1,K)+p_l(I-1,J-1,K))+b_l(I,J,K,2)*(p_l(I,J+1,K+1)-
                   p_l(I,J-1,K+1)-p_l(I,J+1,K-1)+p_l(I,J-1,K-1)) + b_l(I,J,K,3)
                   *(p_l(I+1,J,K+1)-p_l(I-1,J,K+1)-p_l(I+1,J,K-1)+ p_l(I-1,J,K-1))
                   +c_l(I,J,K,1)*p_l(I-1,J,K)+c_l(I,J,K,2)*p_l(I,J-1,K)+c_l(I,J,K,3)
                   *p_l(I,J,K-1)+wrk1_l(I,J,K)
              SS=(S0*a_l(I,J,K,4)-p_l(I,J,K))*bnd_l(I,J,K)
              GOSA=GOSA+SS*SS
           END DO
          END DO
        END DO
```

a_l, b_l, c_l, p_l, bnd_l, wrk1_l are all partitioned local arrays

Reduction of SPREAD DO happens later

Variation of XPF-OpenMP Hybrid for Himeno

FUJITSU

- We parallelize Himeno in 4 ways with process group 4*4, and 4 threads in each process :

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP DO
            DO I=2,IMAX-1
```

(1)

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP DO NOWAIT
            DO I=2,IMAX-1
```

(2)

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
        DO K=2,KMAX-1
!$OMP DO
          DO J=2,JMAX-1
!XOCL SPREAD DO
            DO I=2,IMAX-1
```

(3)

```
!$OMP PARALLEL REDUCTION
!$OMP DO
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!XOCL SPREAD DO
            DO I=2,IMAX-1
```

(4)

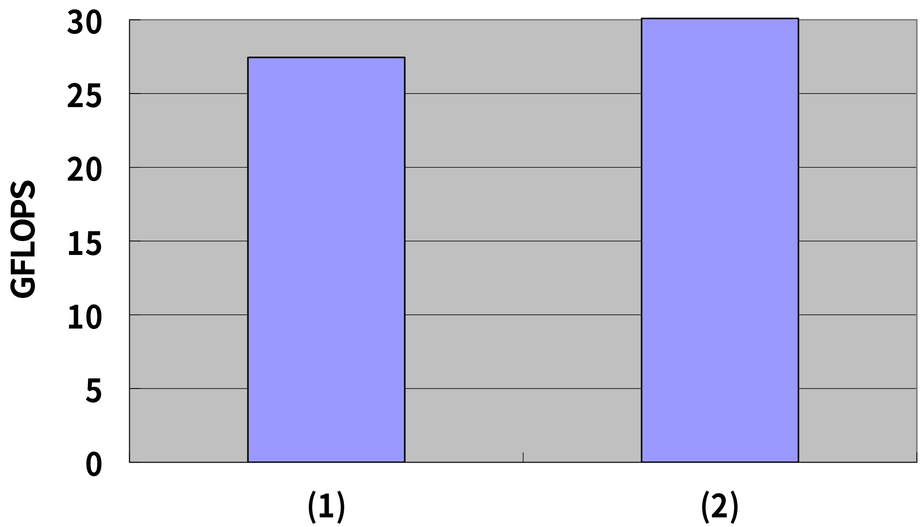
Evaluation Results (1/2) -- Thread Synchronization

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP DO
            DO I=2,IMAX-1
```

(1)

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP DO NOWAIT
            DO I=2,IMAX-1
```

(2)



⇨ Influence of thread sync. is not so significant

Evaluation Results (2/2)
-- Thread-Process Combination

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP DO
            DO I=2,IMAX-1
```

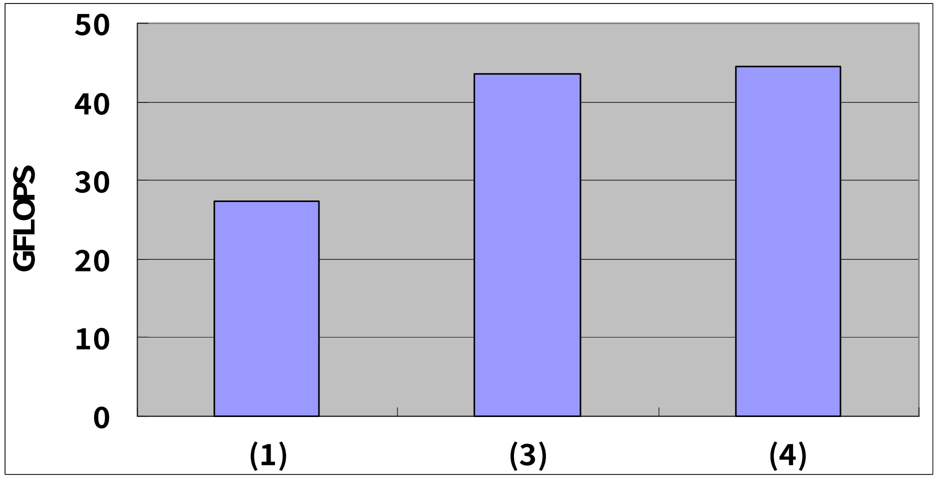
(1)

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
        DO K=2,KMAX-1
!$OMP DO
          DO J=2,JMAX-1
!XOCL SPREAD DO
            DO I=2,IMAX-1
```

(3)

```
!$OMP PARALLEL
!$OMP DO REDUCTION
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!XOCL SPREAD DO
            DO I=2,IMAX-1
```

(4)



⇩

Performance to parallelize outermost loop with threads & inner ones with processes is the best

Large Scale Evaluation

FUJITSU

We evaluated following 5 cases in larger scale:

No. of processes varies from 8,16, until 256. No. of threads is 4.

```
!XOCL SPREAD DO
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP PARALLEL
!$OMP DO REDUCTION
            DO I=2,IMAX-1
```

(A)

```
!XOCL SPREAD DO
!$OMP PARALLEL
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP DO REDUCTION
            DO I=2,IMAX-1
```

(B)

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP DO
            DO I=2,IMAX-1
```

(C)

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
        DO K=2,KMAX-1
!XOCL SPREAD DO
!$OMP DO
          DO J=2,JMAX-1
            DO I=2,IMAX-1
```

(D)

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
!$OMP DO
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
            DO I=2,IMAX-1
```

(E)

Evaluation Results (1/3) -- Thread Fork

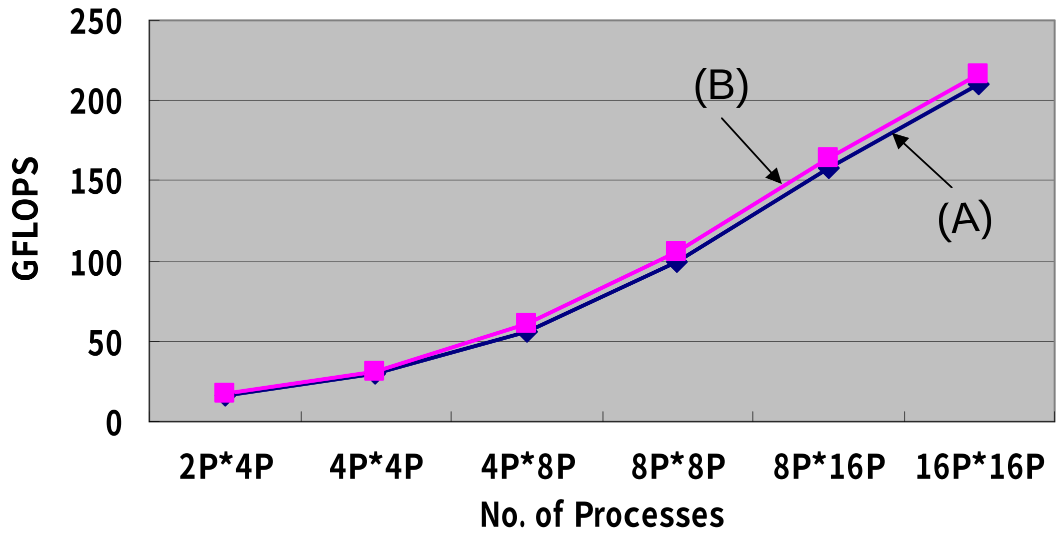
FUJITSU

```
!XOCL SPREAD DO
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP PARALLEL
!$OMP DO REDUCTION
            DO I=2,IMAX-1
```

(A)

```
!XOCL SPREAD DO
!$OMP PARALLEL
        DO K=2,KMAX-1
!XOCL SPREAD DO
          DO J=2,JMAX-1
!$OMP DO REDUCTION
            DO I=2,IMAX-1
```

(B)



⇒ Overhead of thread fork is low

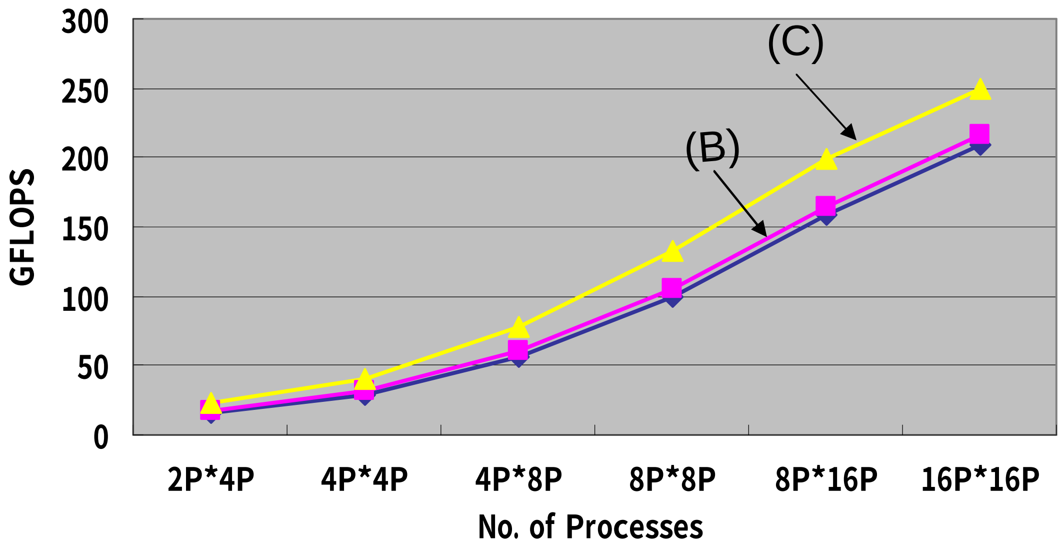
Evaluation Results (2/3)
-- Thread Reduction

```
!XOCL SPREAD DO
!$OMP PARALLEL
      DO K=2,KMAX-1
!XOCL SPREAD DO
        DO J=2,JMAX-1
!$OMP DO REDUCTION
          DO I=2,IMAX-1
```

(B)

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
      DO K=2,KMAX-1
!XOCL SPREAD DO
        DO J=2,JMAX-1
!$OMP DO
          DO I=2,IMAX-1
```

(C)



⇨ To decrease thread reduction is important

Evaluation Results (3/3)
-- Thread-Process Combination

FUJITSU

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
      DO K=2,KMAX-1
!XOCL SPREAD DO
        DO J=2,JMAX-1
!$OMP DO
          DO I=2,IMAX-1
```

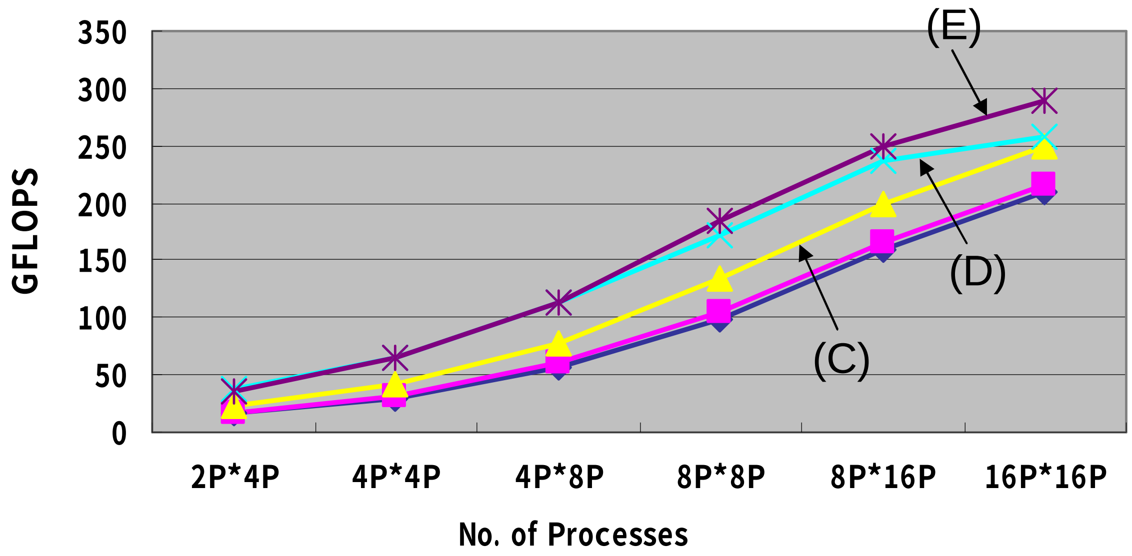
(C)

```
!XOCL SPREAD DO
!$OMP PARALLEL REDUCTION
      DO K=2,KMAX-1
!XOCL SPREAD DO
!$OMP DO
        DO J=2,JMAX-1
          DO I=2,IMAX-1
```

(D)

```
!XOCL SPREAD DO
!$OMP PARALLEL
!$OMP DO REDUCTION
      DO K=2,KMAX-1
!XOCL SPREAD DO
        DO J=2,JMAX-1
          DO I=2,IMAX-1
```

(E)



Performance to use threads for the outermost loop is the best because of improvement of data locality

Outline

- Introduction
- Hybrid Programming using XPFortran and OpenMP
  - Parallelize a Single Loop
  - Parallelize Nested Loops
- Performance Evaluation
- Conclusion

Conclusion

- Improvement of the performance of an XPF-OpenMP hybrid program by spreading the outer loop of nested loops to threads & the inner loop(s) to processes:
  - Data reference locality is improved
  - No MPI comm. & no sync. in the implementation of SPREAD DO
  - Overhead of SPREAD DO is low because
    - Its SPMD output only contains some computational code
    - By optimization the SPREAD DO code parallelizing inner loop(s) often can be moved out before the outer loop
- A data-parallel language, such as XPF, makes this possible
  - Comm. and sync. are separated from computational loops

Thank you for your attention!

shaping tomorrow with you