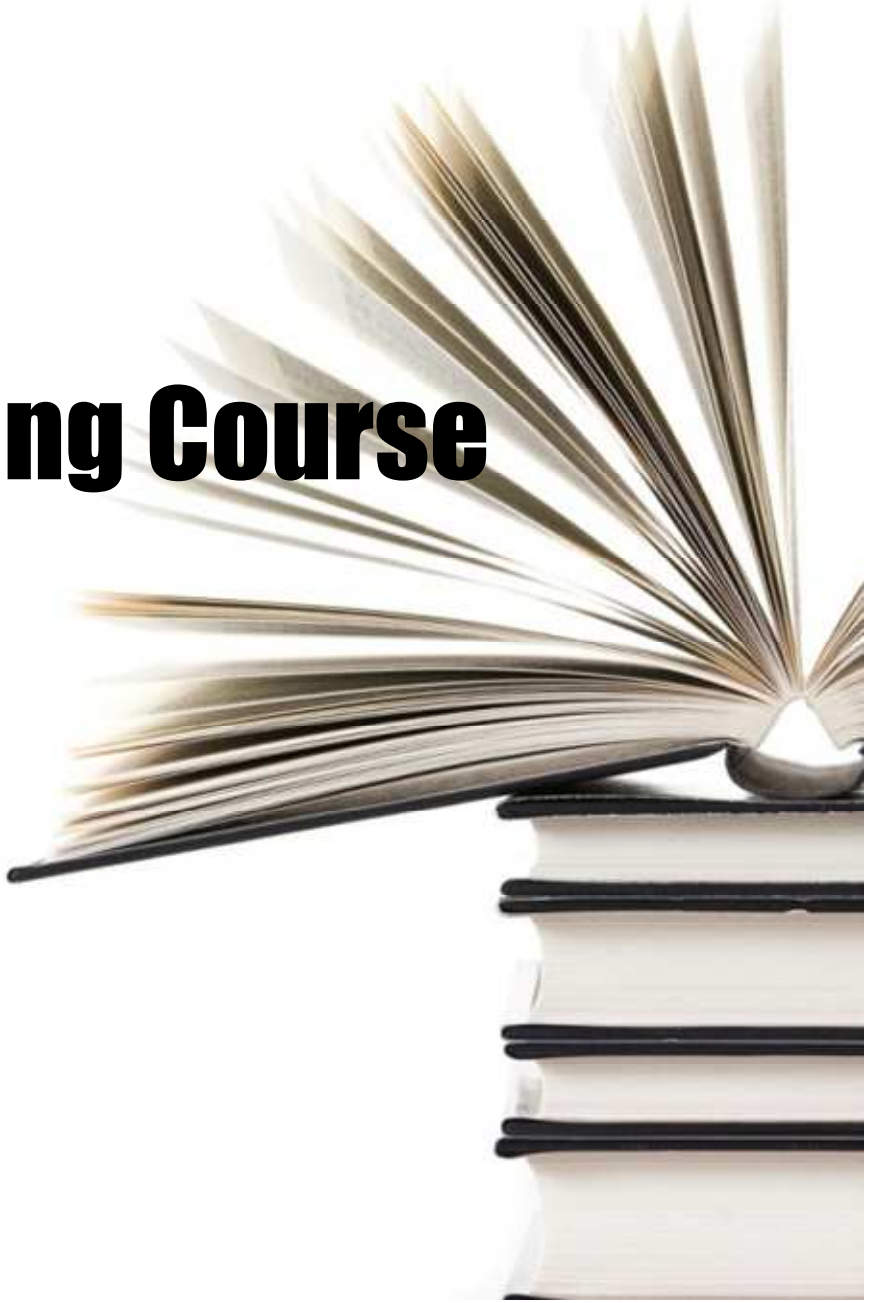
KiSTi SUPERCOMPUTING EDUCATION CENTER
www.kisti.re.kr

OpenMP Training Course

Part 2 Advanced

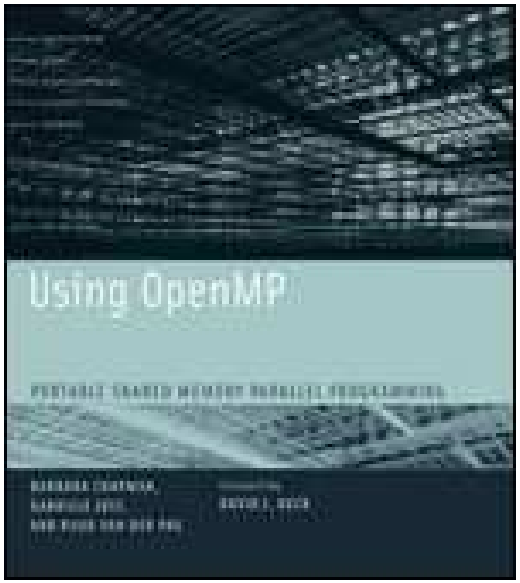


http://www.ksc.re.kr
http://edu.ksc.re.kr

National Institute of Supercomputing and Networking 1967

Resources

- http://edu.ksc.re.kr
- http://www.openmp.org
- http://www.compunity.org
- https://computing.llnl.gov/tutorials/openMP/
- http://www.citutor.org
- http://docs.oracle.com/cd/E19422-01/819-3694/



Agenda (Advanced Course)

- 09:30 – 10:00 Brief Review & Introduction to Education System
- 10:00 – 12:00 Work Sharing & Synchronization
- 12:00 – 13:30 Lunch
- 13:30 – 14:50 Schedule & Task
- 14:50 – 15:00 Break
- 15:00 – 16:30 OpenMP Performance & Hands-on
- 16:30 – 17:00 Summary

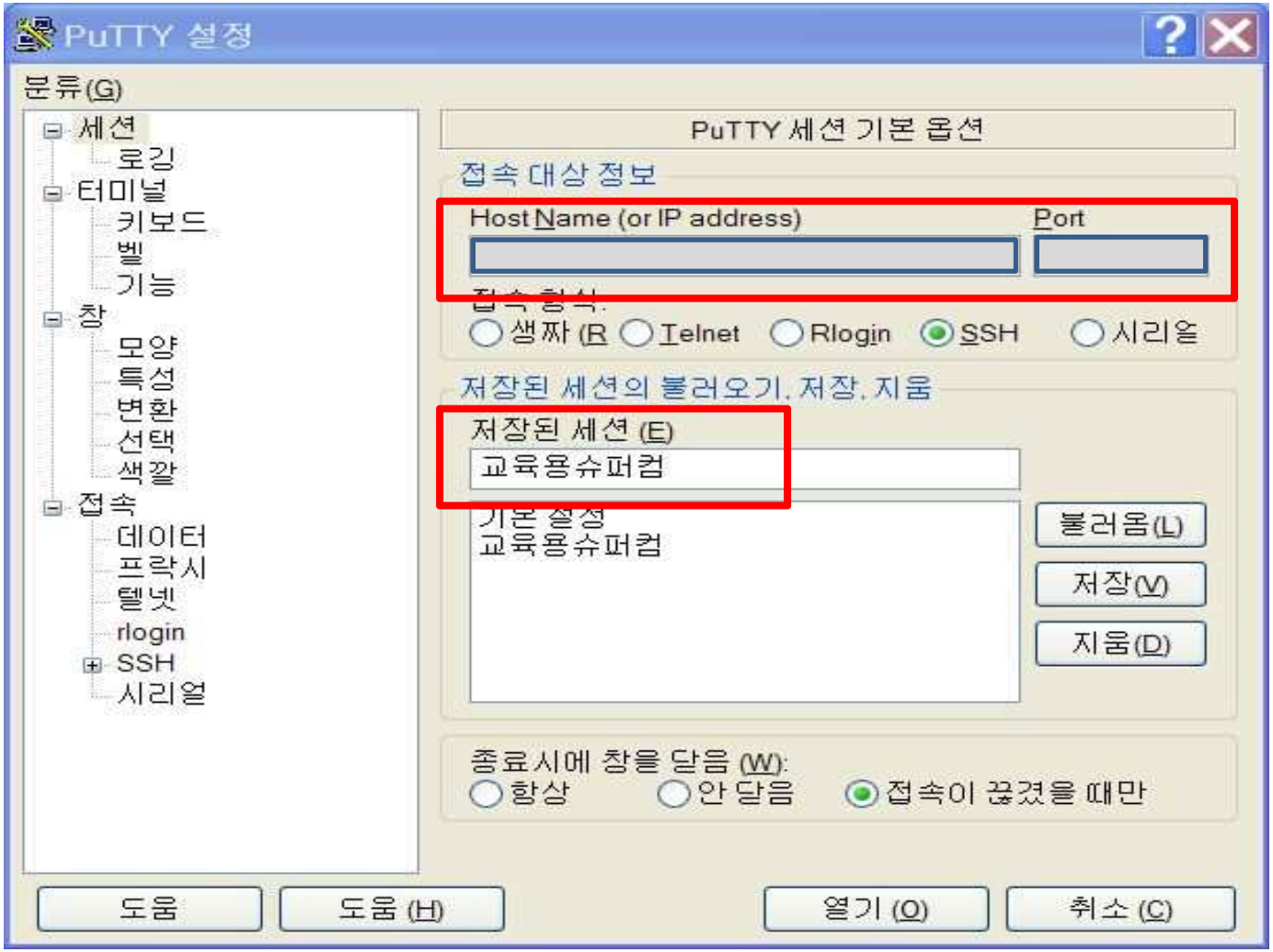
Agenda (Advanced Course)

- Brief Review
  - OpenMP Basics (Create Threads, Data Scope, Loop, Synchronization, Reduction)
- Introduction to Education System
  - Check Environment
- Create Threads [2.4, 3.2] (Nested Parallelism)
- Work Sharing
  - sections [2.5.2, 2.6.2], single [2.5.3], master [2.8.1]
- Synchronization
  - nowait, ordered [2.8.7], lock [3.3]
- Task
  - task [2.7.1], taskwait [2.8.4]
- Schedule [3.2]
- OpenMP Performance
  - Nested Parallel (Collapse)
  - Flush [2.8.6]
  - False Sharing
  - Data DepENDency
- Hands-on
  - Triangular Matrix Multiplication
  - LU Decomposition (Optional)
  - Fibonacci (Optional)
- Summary

Introduction to Education System

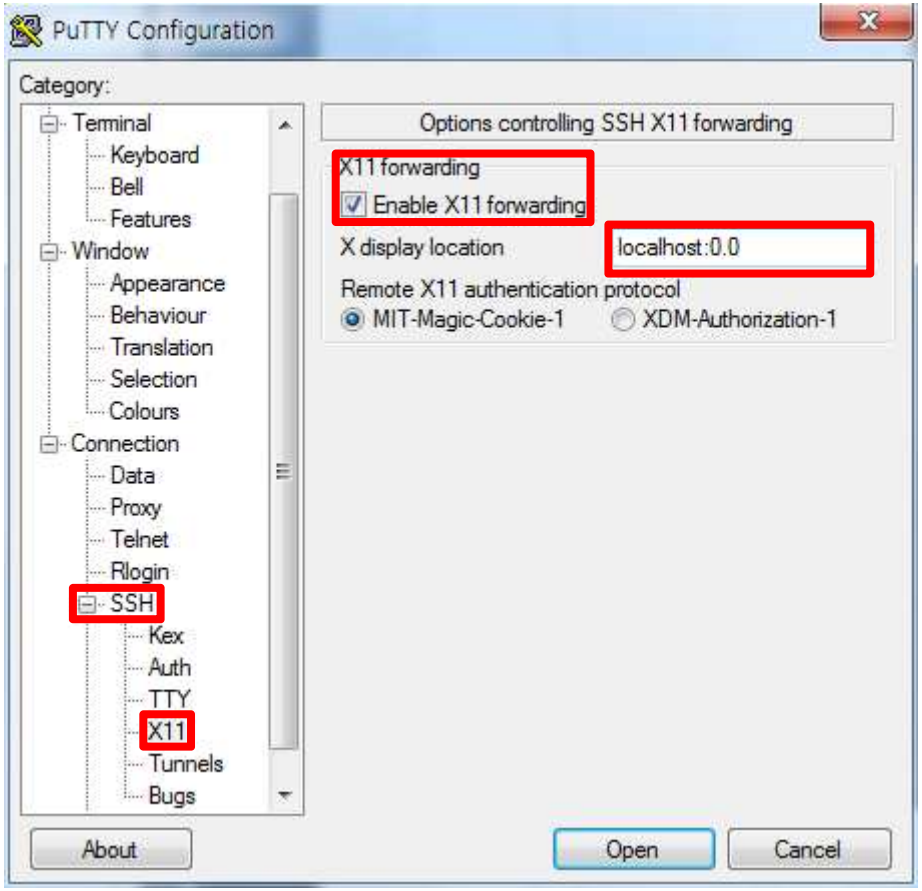
1. Education System
2. Check Environment

Putty 설정


PuTTY 설정
분류(G)
세션
로깅
터미널
키보드
벨
기능
창
모양
특성
변환
선택
색깔
접속
데이터
프락시
텔넷
rlogin
SSH
시리얼
PuTTY 세션 기본 옵션
접속 대상 정보
Host Name (or IP address)
Port
접속 형식:
생짜 (R
Telnet
Rlogin
SSH
시리얼
저장된 세션의 불러오기, 저장, 지움
저장된 세션 (E)
교육용슈퍼컴
기본 설정
교육용슈퍼컴
불러옴(L)
저장(V)
지움(D)
종료시에 창을 닫음 (W):
항상
안 닫음
접속이 끊겼을 때만
도움
도움 (H)
열기 (O)
취소 (C)

Putty 설정

- ssh => X11 [Enable X11 forwarding] 체크
- X display location : localhost : 0.0
- save



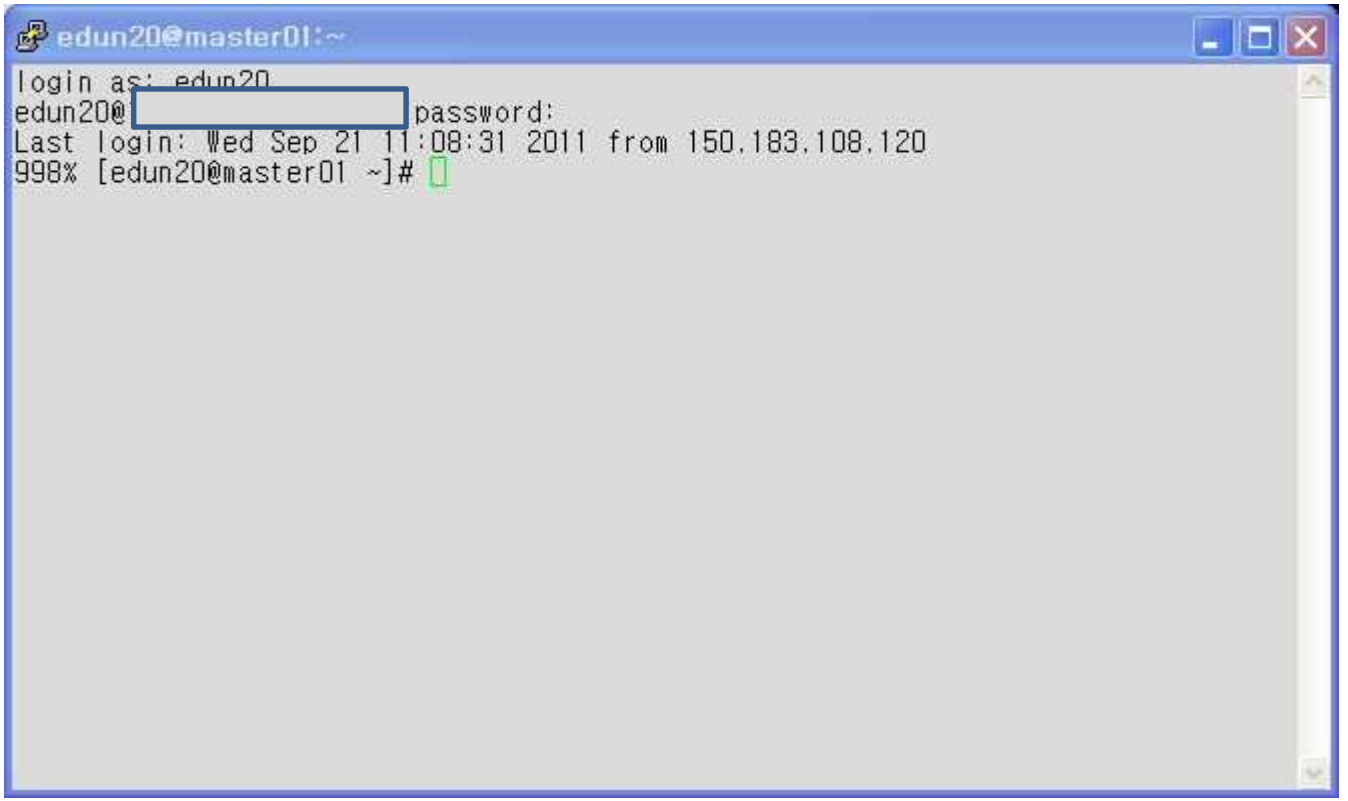
윈도우에서
Xming 실행

Putty 설정

- PuTTY Security Alert [예(y)] 선택

Putty 설정

- login as :
- edun##@xxx.xxx.xxx.xxx's password :



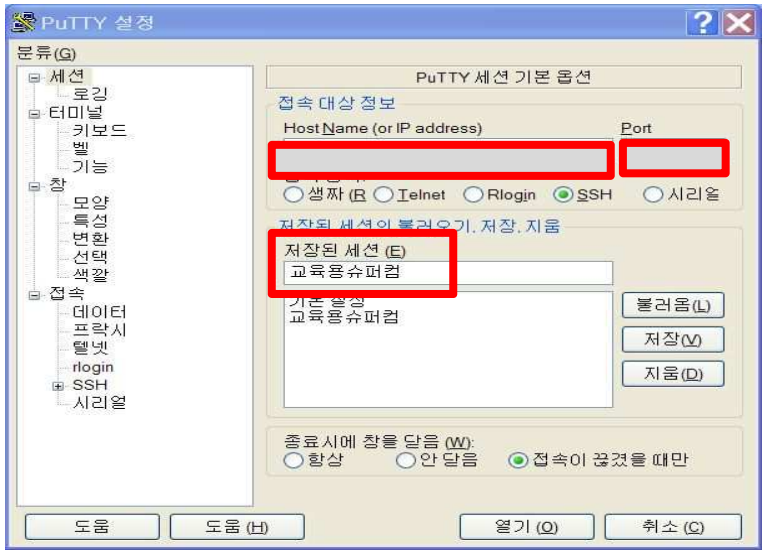
Xming

- Freeware
- 다운로드 웹싸이트
  - http://www.google.com 에서 Xming으로 검색
  - http://sourceforge.net/projects/xming/

- 설치 및 사용방법
  - 실행 파일 다운로드 후 설치
  - Xming 실행
  - Xwindows GUI 프로그램을 사용하기 위해서는 먼저 실행 필수

Education System

- In Linux
  - Open terminal (to compile and run PROGRAM)
  - Use familiar editor : vi, emacs, gedit, etc.
- In Windows
  - Use SSH client : SecureShellClient, PuTTY, etc.
    - IP : xxx.xxx.xxx.xxx, Port : xx
    - Login ID : password :
  - notepad++



Checking Environment

- Move to debugging and computing node
  - Use **ssh**
  - Usage : ssh s000#
  - ex) ssh s0001

```
[edun20@master01 ~]$ ssh s0001
Last login: Thu Aug 30 13:33:46 2012 from master01
[edun20@s0001 ~]$ hostname
s0001
[edun20@s0001 ~]$
```

- Compiler Environment
  - Use **module** command
  - Usage : module add compiler/gcc-4.4.6

```
[edun20@s0001 ~]$ module avilable
----- /applic/Modules/modulefiles -----
(module list) …
[edun20@s0001 ~]$ module add compiler/gcc-4.4.6
[edun20@s0001 ~]$ module list
Currently Loaded Modulefiles:
  1) compiler/gcc-4.4.6
```

Checking Environment

Fortran	C

PROGRAM hello_world
 	
!$omp parallel
PRINT *, 'Hello World'
!$omp end parallel
 	
END
	#include <stdio.h>
	int main() {
	#pragma omp parallel
	{
	printf ("Hello World\n");
	}
	return 0;
	}

$ gfortran –o hello.x hello.f90
$ gfortran –fopenmp –o hello_omp.x hello.f90
$ ./hello.x
$ ./hello_omp.x
	$ gcc –o hello.x hello.c
	$ gcc –fopenmp –o hello_omp.x hello.c
	$ ./hello.x
	$ ./hello_omp.x

Intel : ifort **–o** hello.x hello.f90 GCC : gfortran **–o** hello.x hello.f90 PGI : pgf90 **–o** hello.x hello.f90	Intel : icc **–o** hello.x hello.c GCC : gcc **–o** hello.x hello.c PGI : pgcc **–o** hello.x hello.c
Intel : ifort **-openmp** –o hello_omp.x hello.f90 GCC : gfortran **-fopenmp** –o hello_omp.x hello.f90 PGI : pgf90 **-mp** –o hello_omp.x hello.f90	Intel : icc **–openmp** –o hello_omp.x hello.c GCC : gcc **–fopenmp** –o hello_omp.x hello.c PGI : pgcc **–mp** –o hello_omp.x hello.c

for output filename

for output filename

for OpenMP

for OpenMP

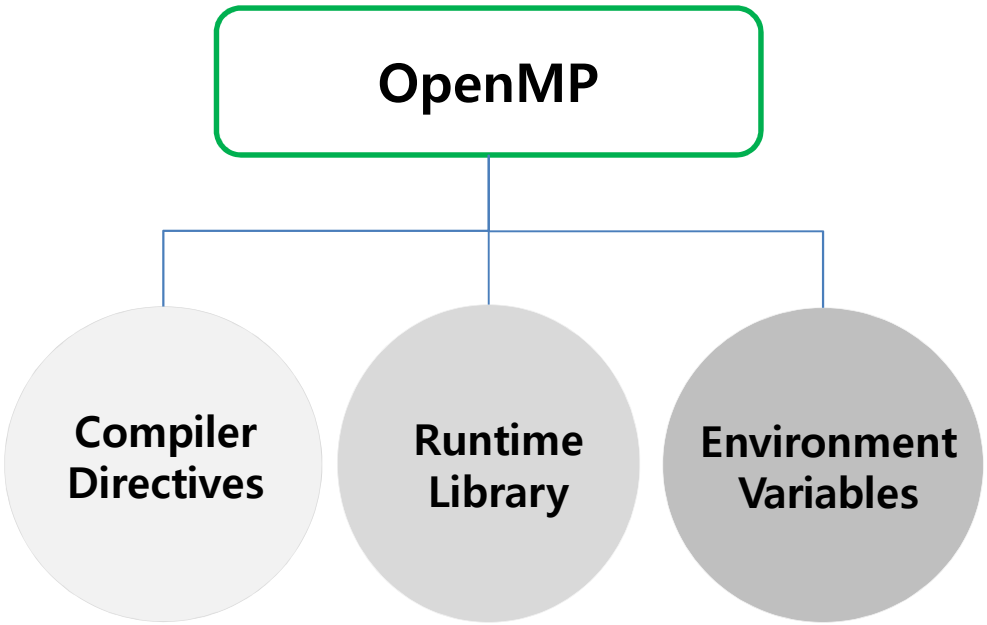
Brief Review

1. OpenMP Basics Review

What is OpenMP ?

- OpenMP is
  - De-facto standard API (Application PROGRAMming Interface) for multi-thread based shared memory parallel PROGRAMming
  - Not a new PROGRAMming language
  - Notation that can be added to a sequential PROGRAM in C, C++ and Fortran

- Consists of
  - Compiler Directives
  - Runtime Library
  - Environment Variables



Creating Threads

Fortran	C

PROGRAM HELLOW
IMPLICIT NONE
 	
INTEGER omp_get_thread_num
CALL **omp_set_num_threads(4)**
 	
!$omp parallel
PRINT *, 'Hello World', omp_get_thread_num()
!$omp end parallel
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	omp_set_num_threads(4);
	
	#pragma omp parallel **num_threads(4)**
	{
	printf ("Hello World %d\n", omp_get_thread_num());
	}
	
	return 0;
	}

$ **export OMP_NUM_THREADS=4** (ksh/bash) $ **setenv OMP_NUM_THREADS 4** (csh) Intel : ifort **-openmp** –o hello.x hello.f90 GCC : gfortran **-fopenmp** –o hello.x hello.f90 PGI : pgf90 **-mp** –o hello.x hello.f90	$ **export OMP_NUM_THREADS=4** (ksh/bash) $ **setenv OMP_NUM_THREADS 4** (csh) Intel : icc **–openmp** –o hello.x hello.c GCC : gcc **–fopenmp** –o hello.x hello.c PGI : pgcc **–mp** –o hello.x hello.c

Data Scope Attribute

Fortran	C

PROGRAM PRIVATE_KEYWORD
IMPLICIT NONE
 	
INTEGER tid, omp_get_thread_num
INTEGER :: i = 10
 	
CALL omp_set_num_threads(4)
 	
!$omp parallel **private(tid) firstprivate(i)**
tid = omp_get_thread_num()
PRINT *, 'Hello World ', 'tid=', tid, ' i=', i
!$omp end parallel
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	int tid, i=10;
	omp_set_num_threads(4);
	
	#pragma omp parallel **private(tid) firstprivate(i)**
	{
	tid = omp_get_thread_num();
	printf ("Hello World tid = %d i= %d\n", tid, i);
	}
	
	return 0;
	}

$ gfortran -fopenmp -o private_keyword.x private_keyword.f90 $ ./private_keyword.x	$ gcc -fopenmp -o private_keyword.x private_keyword.c $ ./private_keyword.x

Synchronization

Fortran	C

PROGRAM SYNC_REVIEW
IMPLICIT NONE
 	
INTEGER, PARAMETER :: N=2000000
INTEGER(8) :: i, sum=0
INTEGER, DIMENSION(0:N-1) :: A, B
 	
DO i=0, N-1
A(i) = i+1
B(i) = i+2
ENDDO
 	
CALL omp_set_num_threads(4)
 	
!$omp parallel
!$OMP DO
DO i=0, N-1
!$OMP ATOMIC
sum = sum + A(i) * B(i)
END DO
!$omp end parallel
 	
PRINT *, “sum =“, sum
 	
END
	#include <stdio.h>
	#include <omp.h>
	#define N 2000000
	
	int main()
	{
	long i, sum=0;
	int a[N], b[N];
	
	for (i=0; i<N; i++) {
	a[i] = i+1;
	b[i] = i+2;
	}
	omp_set_num_threads(4);
	#pragma omp parallel
	{
	#pragma omp for
	for (i=0; i<N; i++)
	#pragma omp atomic
	sum += a[i] * b[i];
	}
	printf(“sum = %ld\n”, sum);
	
	return 0;
	}

$ gfortran –fopenmp –o sync_review.x sync_review.f90
$ ulimit –s 512000
$ ./sync_review.x
	$ gcc –fopenmp –o sync_review.x sync_review.c
	$ ulimit –s 512000
	$ ./sync_review.x

Reduction

Fortran

```
PROGRAM  REDUCTION_REVIEW
IMPLICIT NONE

      INTEGER, PARAMETER :: N=2000000
      INTEGER(8) :: i, sum=0
      INTEGER, DIMENSION(0:N-1) :: A, B

      DO i=0, N-1
            A(i) = i+1
            B(i) = i+2
      ENDDO

      CALL omp_set_num_threads(4)

!$omp parallel DO reduction(+:sum)
      DO i=0, N-1
            sum = sum + A(i) * B(i)
      END DO
!$omp end parallel DO

      PRINT *, “sum =“, sum

END
```

```
$ gfortran –fopenmp –o reduce_review.x reduce_review.f90
$ ulimit –s 512000
$ ./reduce_review.x
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 2000000

int main()
{
      long i, sum=0;
      int a[N], b[N];

      for(i=0; i<N; i++) {
            a[i] = i+1;
            b[i] = i+2;
      }

      omp_set_num_threads(4);
#pragma omp parallel for reduction (+:sum)
      for(i=0; i<N; i++)
            sum += a[i] * b[i];

      printf(“sum = %ld\n”, sum);

      return 0;
}
```

```
$ gcc –fopenmp –o reduce_review.x reduce_review.c
$ ulimit –s 512000
$ ./reduce_review.x
```

Create Threads

1. Nested Parallel

Create Threads : Nested Parallel (1/3)

Fortran

```
PROGRAM nested_parallel
      INTEGER tid, omp_get_thread_num

      CALL omp_set_nested(1)
      CALL omp_set_num_threads(2)

!$omp parallel private(tid)
      tid = omp_get_thread_num()
      PRINT 10, 'thread id =', tid
      IF( tid == 1 ) THEN
            !$omp parallel private(tid)
            tid = omp_get_thread_num()
            print 20, 'thread id =', tid
            !$omp end parallel
      END IF
!$omp end parallel

10 FORMAT(A,I4)
20 FORMAT(T8,A,I4)

END
```

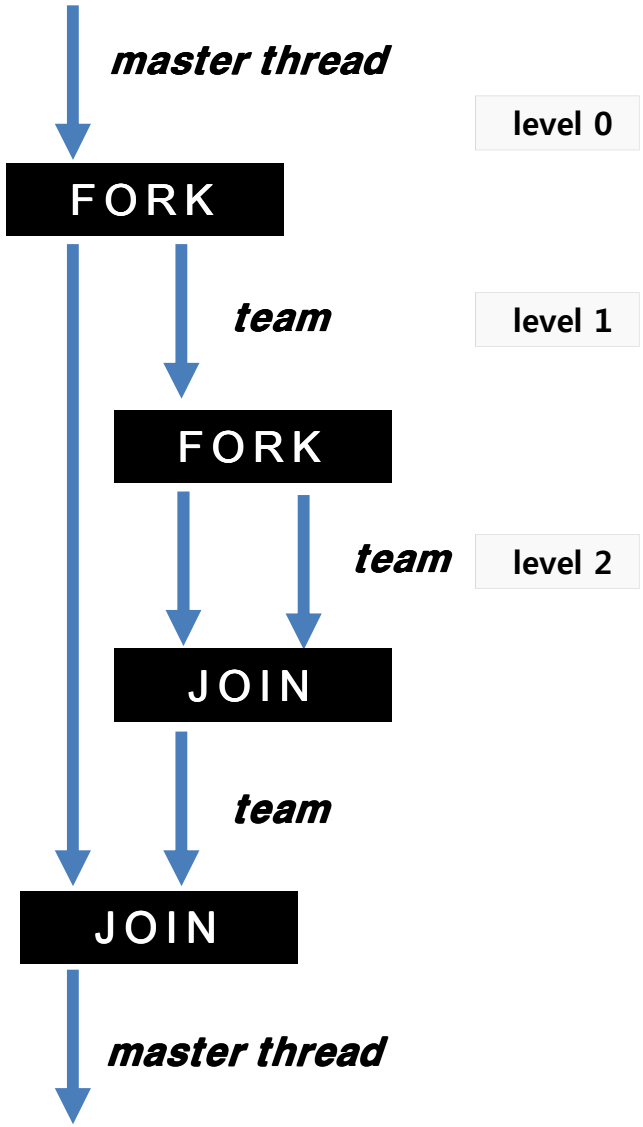
```
$ gfortran  -fopenmp  -o nested_parallel.x nested_parallel.f90
$ ./ nested_parallel.x
```

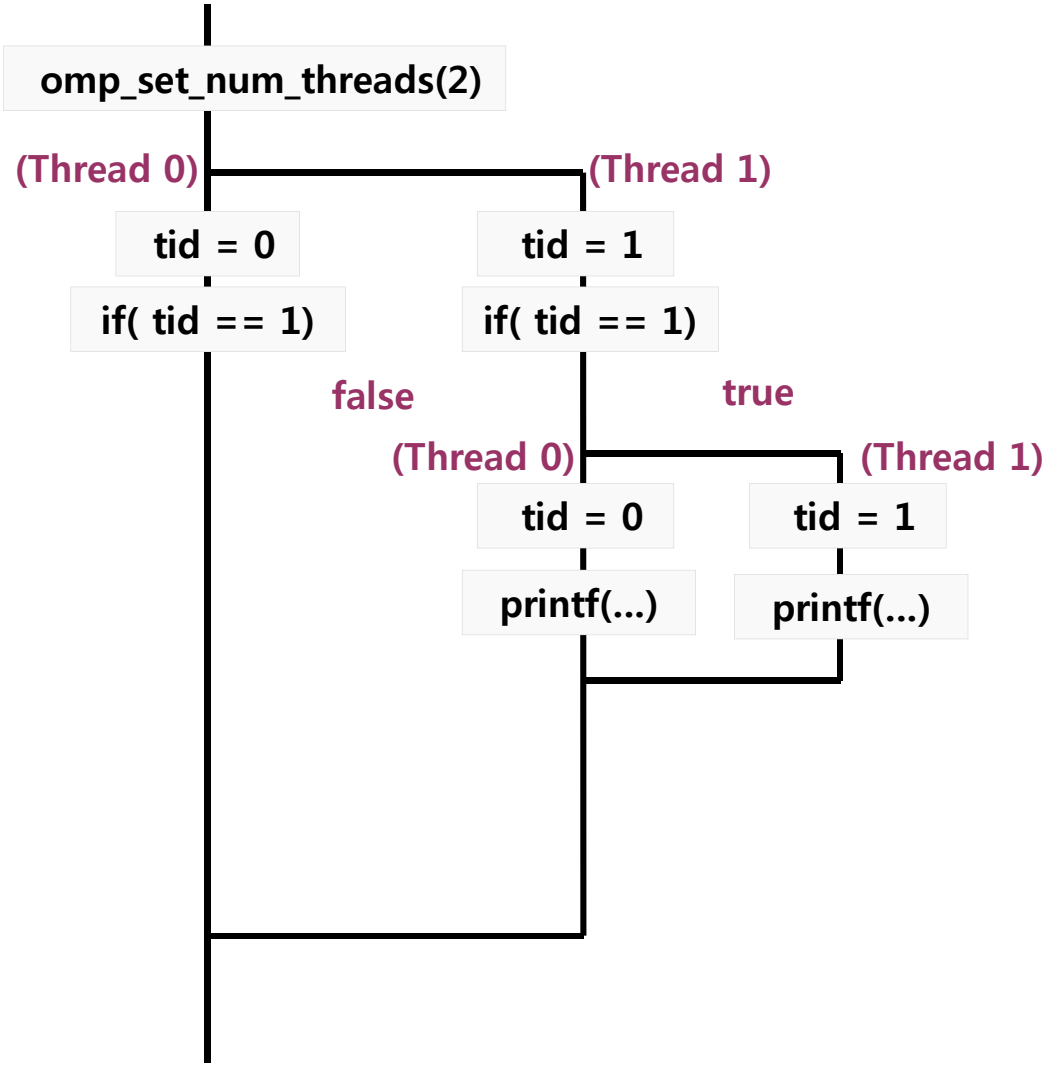
C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int tid;

      omp_set_nested(1);
      omp_set_num_threads(2);

#pragma omp parallel private(tid)
{
      tid = omp_get_thread_num();
      printf("thread id=%d\n", tid);
      if( tid == 1 ) {
            #pragma omp parallel private(tid)
            {
                  tid = omp_get_thread_num();
                  printf("\t thread id=%d\n", tid);
            }
      }
}
   return 0;
}
```

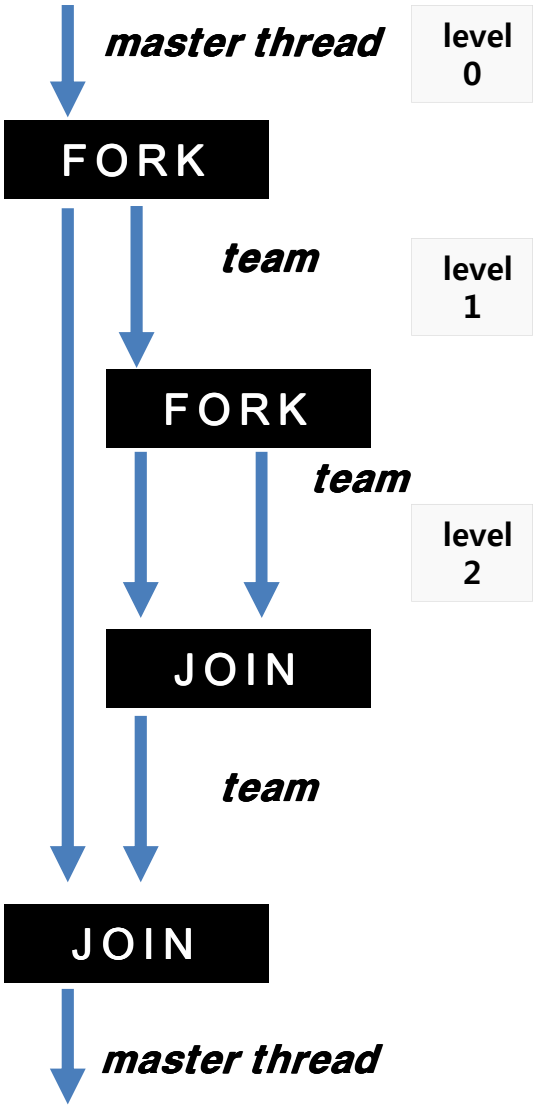
```
$ gcc -fopenmp -o nested_parallel.x nested_parallel.c
$ ./ nested_parallel.x
```

Create Threads : Nested Parallel (2/3)





Create Threads : Nested Parallel (2/3)



Fortran	C
omp_set_nested(nested) enables or disables nested parallelism	omp_set_nested(int nested) enables or disables nested parallelism
omp_get_nested() the value of the nes-var IVC	omp_get_nested() the value of the nes-var ICV
omp_get_level() the number of nested parallel regions	omp_get_level() the number of nested parallel regions
omp_get_active_level() the number of nested, active parallel regions	omp_get_active_level() the number of nested, active parallel regions
omp_get_ancestor_thread_num(level) the thread number of the ancestor or the current thread	omp_get_ancestor_thread_num(int level) the thread number of the ancestor or the current thread
omp_get_team_size(level) the size of the thread team	omp_get_team_size(int level) the size of the thread team

Create Threads : Nested Parallel (3/3)

Fortran

```
PROGRAM more_nested_para
      INTEGER tid, level
      INTEGER omp_get_thread_num
      INTEGER omp_get_level
      INTEGER omp_get_ancestor_thread_num

      CALL omp_set_nested(.true.)
      CALL omp_set_num_threads(4)
!$omp parallel private(tid, level)
      tid = omp_get_thread_num()
      level = omp_get_level()
      PRINT 10, 'thread id =', tid
      IF ( tid == 1 ) THEN
      !$omp parallel private(tid) num_threads(tid+2)
            tid = omp_get_thread_num()
            PRINT 20, 'thread id =', tid, ' ancestor_thread_num(', &
                  level, ')=', omp_get_ancestor_thread_num(level)
      !$omp end parallel
      END IF
!$omp end parallel

10 FORMAT(A,I4)
20 FORMAT(T8,A,I4,A,I4,A,I4)

END
```

```
$ gfortran -fopenmp -o more_nested_para.x  more_nested_para.f90
$ ./more_nested_para.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int tid, level;

      omp_set_nested(1);
      omp_set_num_threads(4);
#pragma omp parallel private(tid, level)
{
      tid = omp_get_thread_num();
      level = omp_get_level();
      printf("tid=%d level=%d\n", tid, level);
      if( tid == 1 ) {
            #pragma omp parallel private(tid) num_threads(tid+2)
            {
                  tid = omp_get_thread_num();
                  printf("\t tid=%d ancestor_thread_num(%d)=%d\n",
                  tid, level, omp_get_ancestor_thread_num(level));
            }
      }
}
      return 0;
}
```

```
$ gcc -fopenmp -o more_nested_para.x  more_nested_para.c
$ ./more_nested_para.x
```

Nested Parallel : Data Scope (1/3)

Fortran

```
PROGRAM nested_parallel_data_scope
      INTEGER x, y, z

      CALL omp_set_nested(1)
      CALL set_num_threads(4)

!$omp parallel private(y)
   x, y, z private? Shared?
      !$omp parallel num_threads(2) private(x)
            x, y, z private? shared?
      !$omp end parallel
!$omp end parallel

END PROGRAM
```

C

```
int main()
{
  int x, y, z;

  omp_set_nested(1);
  omp_set_num_threads(4);

#pragma omp parallel private(y)
{
      x,y,z private? shared?
      #pragma omp parallel num_threads(2) private(x)
      {
            x,y,z private? shared?
      }
}
      return 0;
}
```

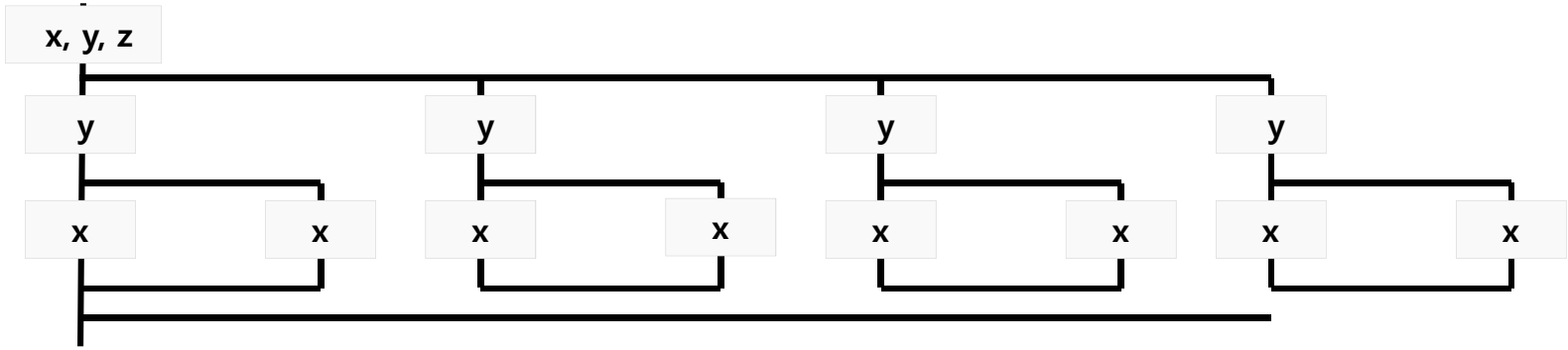
Nested Parallel : Data Scope (2/3)

Fortran

```
PROGRAM nested_parallel_data_scope
      INTEGER x, y, z

      CALL omp_set_nested(1)
      CALL set_num_threads(4)

!$omp parallel private(y)
      x : shared y : private z : shared
      !$omp parallel num_threads(2) private(x)
            x : private y : shared z : shared
      !$omp end parallel
!$omp end parallel

END PROGRAM
```

C

```
int main()
{
      int x, y, z;

      omp_set_nested(1);
      omp_set_num_threads(4);
#pragma omp parallel private(y)
{
      x : shared y : private z : shared
      #pragma omp parallel num_threads(2) private(x)
      {
            x : private y : shared z : shared
      }
}
      return 0;
}
```



Nested Parallel : Data Scope (3/3)

Fortran

```
PROGRAM nested_parallel_data_scope
      INTEGER :: x=1, y=10, z=20, tid
      INTEGER omp_get_thread_num

      CALL omp_set_nested(.true.)
      CALL omp_set_num_threads(4)

!$omp parallel private(y, tid)
      tid = omp_get_thread_num()
      x = x + 1;    y = 12;    z = 22
      !$omp parallel num_threads(2) private(x,tid)
      tid = omp_get_thread_num()
      x = 10;    y = y + 1;    z = z + 1
      PRINT 30, 'tid =', tid, ' x =', x, ' y=', y, ' z=', z
      !$omp end parallel
      PRINT 20, 'tid =', tid, ' x =', x, ' y=', y, ' z=', z
!$omp end parallel
PRINT 10, ' x =', x, ' y=', y, ' z=', z

10 FORMAT(A,I4,A,I4,A,I4)
20 FORMAT(A,I4,A,I4,A,I4,A,I4)
30 FORMAT(T8,A,I4,A,I4,A,I4,A,I4)

END
```

```
$ gfortran -fopenmp -o nested_parallel_ds.x nested_parallel_ds.f90
$ ./nested_parallel_ds.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int x=1, y=10, z=20, tid;
      omp_set_nested(1);
      omp_set_num_threads(4);

#pragma omp parallel private(y, tid)
{
      tid = omp_get_thread_num();
      x++;   y = 12;    z = 22;
      #pragma omp parallel num_threads(2) private(x, tid)
      {
            tid = omp_get_thread_num();
            x = 10;
            y++;
            z++;
            printf("\ttid=%d x=%d y=%d z=%d\n", tid, x, y, z);
      }
      printf("tid=%d x=%d y=%d z=%d\n", tid, x, y, z);
}
      printf("x=%d y=%d z=%d\n", x, y, z);

      return 0;
}
```

```
$ gcc -fopenmp -o nested_parallel_ds.x nested_parallel_ds.c
$ ./nested_parallel_ds.x
```

Work Sharing

1. Sections
2. Single
3. master

Work-Share Constructs

- Types of Work-Sharing Constructs

  - **do/for**
  - **sections**
  - **single**
  - **workshare**

  – Work-Sharing construct divides the execution of the enclosed code region among members of the team encounter it

  – Work-Sharing constructs do not launch new threads

  – There is no implied barrier upon entry to a work-sharing construct, however there is an implied barrier at the END of a work sharing construct.

  – workshare construct is only The Fortran

Work-Share Constructs : Sections

Fortran

```
PROGRAM  SECTIONS
IMPLICIT NONE

      INTEGER i, a(0:9), b(0:19)
      CALL omp_set_num_threads(2)

!$omp parallel
      !$omp sections
      !$omp section
      do i=0, 9
            a(i) = i * 10 + 5
      END DO
      !$omp section
      do i=0, 19
            b(i) = i * 5 + 10
      END DO
      !$omp end sections
!$omp end parallel

      WRITE(*,10) a
      WRITE(*,20) b
10 FORMAT(10i4)
20 FORMAT(20i4)

END
```

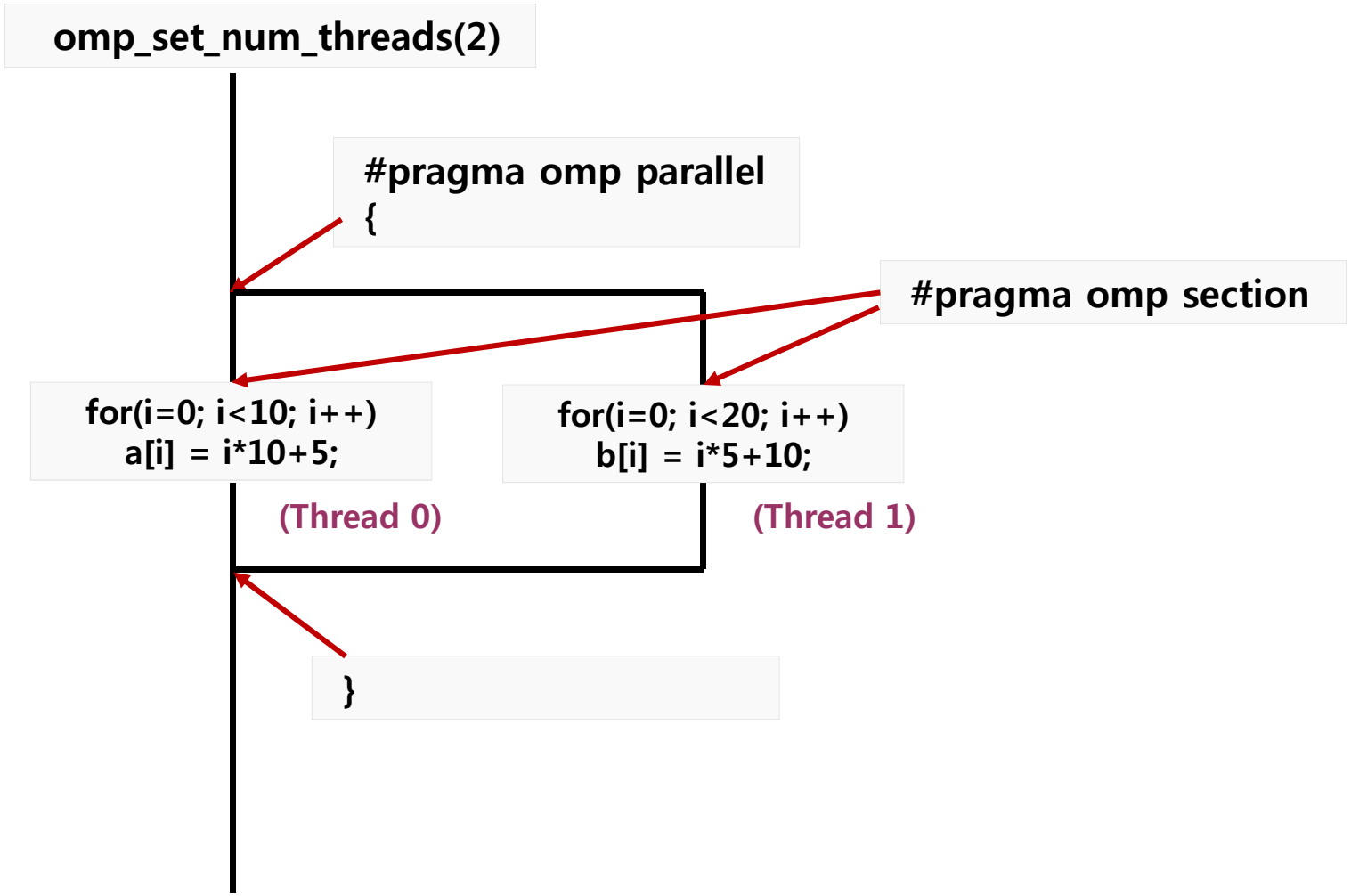
```
$ gfortran  -fopenmp  -o  sections.x  sections.f90
$ ./sections.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int i, a[10], b[20];

      omp_set_num_threads(2);
#pragma omp parallel private(i)
{
      #pragma omp sections
      {
      #pragma omp section
      for (i=0; i<10; i++)
            a[i] = i * 10 + 5;
      #pragma omp section
      for (i=0; i<20; i++)
            b[i] = i * 5 + 10;
      }
}
      for(i=0; i<10; i++)
            printf("%d ", a[i]);
      printf("\n");

      for (i=0; i<20; i++)
            printf("%d ", b[i]);
      printf("\n");

      return 0;
}
```

```
$ gcc  -fopenmp  -o  sections.x  sections.c
$ ./sections.x
```

Work-Share Constructs : Sections



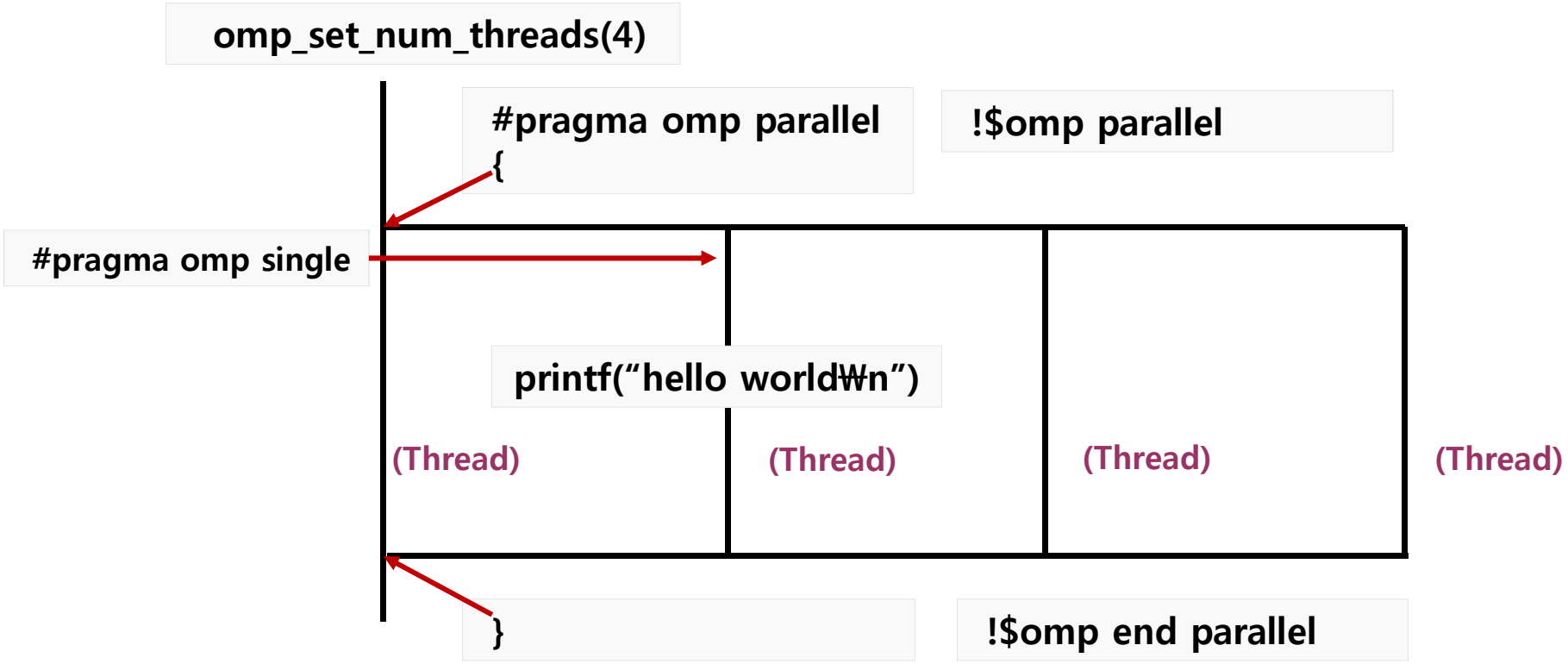
Work-Share Constructs : Single

Fortran	C

PROGRAM SINGLE
IMPLICIT NONE
 	
CALL omp_set_num_threads(4)
 	
!$omp parallel
!$omp single
PRINT *, 'Hello World'
!$omp end single
!$omp end parallel
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	omp_set_num_threads(4);
	
	#pragma omp parallel
	{
	#pragma omp single
	{
	printf("Hello world\n");
	}
	}
	return 0;
	}

$ gfortran -fopenmp -o single.x single.f90 $ ./single.x	$ gcc -fopenmp -o single.x single.c $ ./single.x

Work-Share Constructs : Single



Work-Share Constructs : master

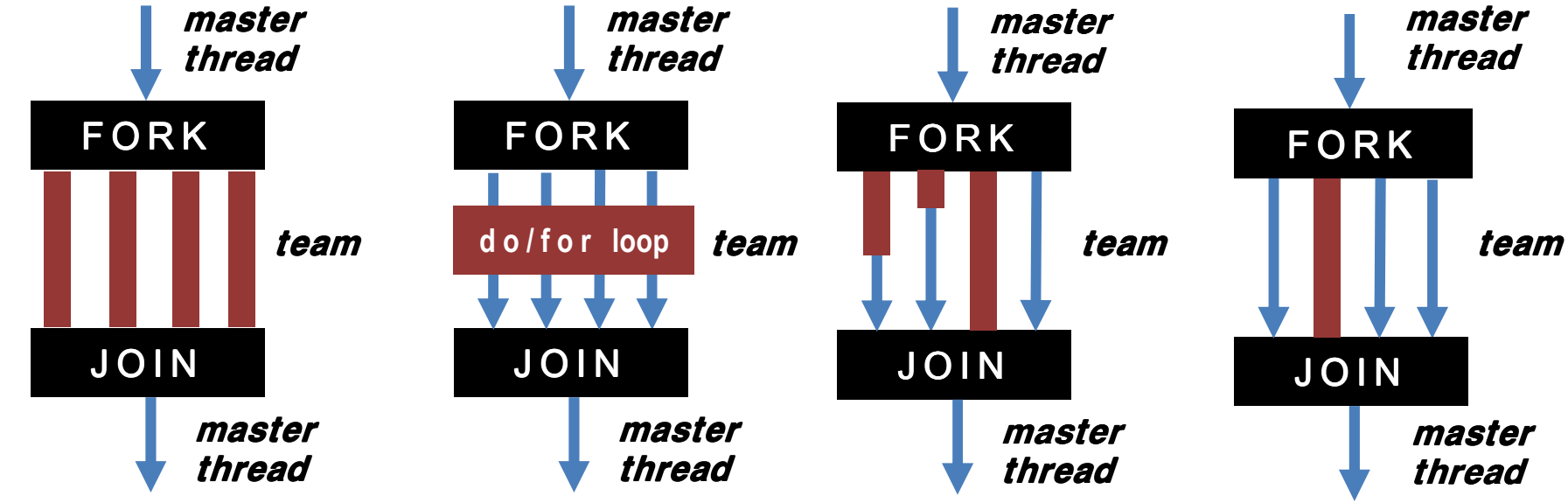
Fortran	C

PROGRAM MASTER
IMPLICIT NONE
 	
INTEGER omp_get_thread_num
CALL omp_set_num_threads(4)
 	
!$omp parallel
!$omp master
CALL sleep(1)
PRINT *, 'Hello World'
!$omp end master
PRINT *, 'tid=',omp_get_thread_num()
!$omp end parallel
 	
END
	#include <stdio.h>
	#include <unistd.h>
	#include <omp.h>
	
	int main()
	{
	omp_set_num_threads(4);
	#pragma omp parallel
	{
	#pragma omp master
	{
	sleep(1);
	printf("hello world\n");
	}
	printf("tid = %d\n", omp_get_thread_num());
	}
	return 0;
	}

$ gfortran -fopenmp -o master.x master.f90 $ ./master.x	$ gcc -fopenmp -o master.x master.c $ ./master.x

- MASTER
  - The MASTER directive specifies a region that is to be executed only by the master thread of the threads
  - All other threads on the threads skip this section of code
  - SINGLE on master thread
  - There is no implied barrier associated with this directive

Work-Share Constructs



parallel

```
#pragma omp parallel
{
}
```

do/for

```
#pragma omp parallel
{
  #pragma omp for
}
```

sections

```
#pragma omp parallel
{
  #pragma sections
  {
  #pragma omp section
  #pragma omp section
  #pragma omp section
  }
}
```

single

```
#pragma omp parallel
{
  #pragma omp single
}
```

Break!!

Synchronization

1. nowait
2. ordered/lock/flush

Synchronization : nowait

Fortran

```
PROGRAM  NOWAIT
IMPLICIT NONE

      INTEGER :: x=1

      CALL omp_set_num_threads(16)
!$omp parallel
      !$OMP SINGLE
            x = x+1
            PRINT *, 'x =', x
      !$omp end SINGLE

      !$omp SINGLE
            x = x+1
            PRINT *, 'x =', x
      !$omp end SINGLE
!$omp end parallel

END
```

```
$ ./a.out
x=2
x=3
```

```
$ gfortran -fopenmp -o nowait.x nowait.f90
$ ./nowait.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int x = 1;
      omp_set_num_threads(16);
#pragma omp parallel
{
      #pragma omp single
      {
            x++;
            printf("x=%d\n", x);
      }
      #pragma omp single
      {
            x++;
            printf("x=%d\n", x);
      }
}
      return 0;
}
```

```
$ ./a.out
x=2
x=3
```

```
$ gcc -fopenmp -o nowait.x nowait.c
$ ./nowait.x
```

Synchronization : nowait

Fortran

```
PROGRAM  NOWAIT1
IMPLICIT NONE

      INTEGER :: x=1

      CALL omp_set_num_threads(16)
!$omp parallel
      !$OMP SINGLE
            x=x+1
            print*, 'x =', x
      !$omp end SINGLE nowait

      !$OMP SINGLE
            x=x+1
            print*, 'x =', x
      !$omp end SINGLE
!$omp end parallel

END
```

What happen?

```
$ ./a.out
x=2
x=3
$ ./a.out
x=3
x=2
$ ./a.out
x=3
x=3
```

$ gfortran -fopenmp -o nowait1.x nowait1.f90
$ ./nowait1.x

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int x = 1;
      omp_set_num_threads(16);
#pragma omp parallel
{
      #pragma omp single nowait
      {
            x++;
            printf("x=%d\n", x);
      }
      #pragma omp single
      {
            x++;
            printf("x=%d\n", x);
      }
}
      return 0;
}
```

What happen?

```
$ ./a.out
x=2
x=3
$ ./a.out
x=3
x=2
$ ./a.out
x=3
x=3
```

$ gcc -fopenmp -o nowait1.x nowait1.c
$ ./nowait1.x

Synchronization : nowait

Fortran

```
PROGRAM  NOWAIT_FOR
IMPLICIT NONE

      INTEGER, PARAMETER :: N = 9999
      INTEGER i, j, a(0:N, 0:N)
      CALL omp_set_num_threads(4)
!$omp parallel private(i,j)
      !$omp do
      DO j=0, N
            DO i=0, j
                  a(i,j) = i+j
            END DO
      END DO

      !$omp do
      DO j=0, N
            DO i=j+1, N
                  a(i,j) = i-j
            END DO
      END DO
!$omp end parallel

END
```

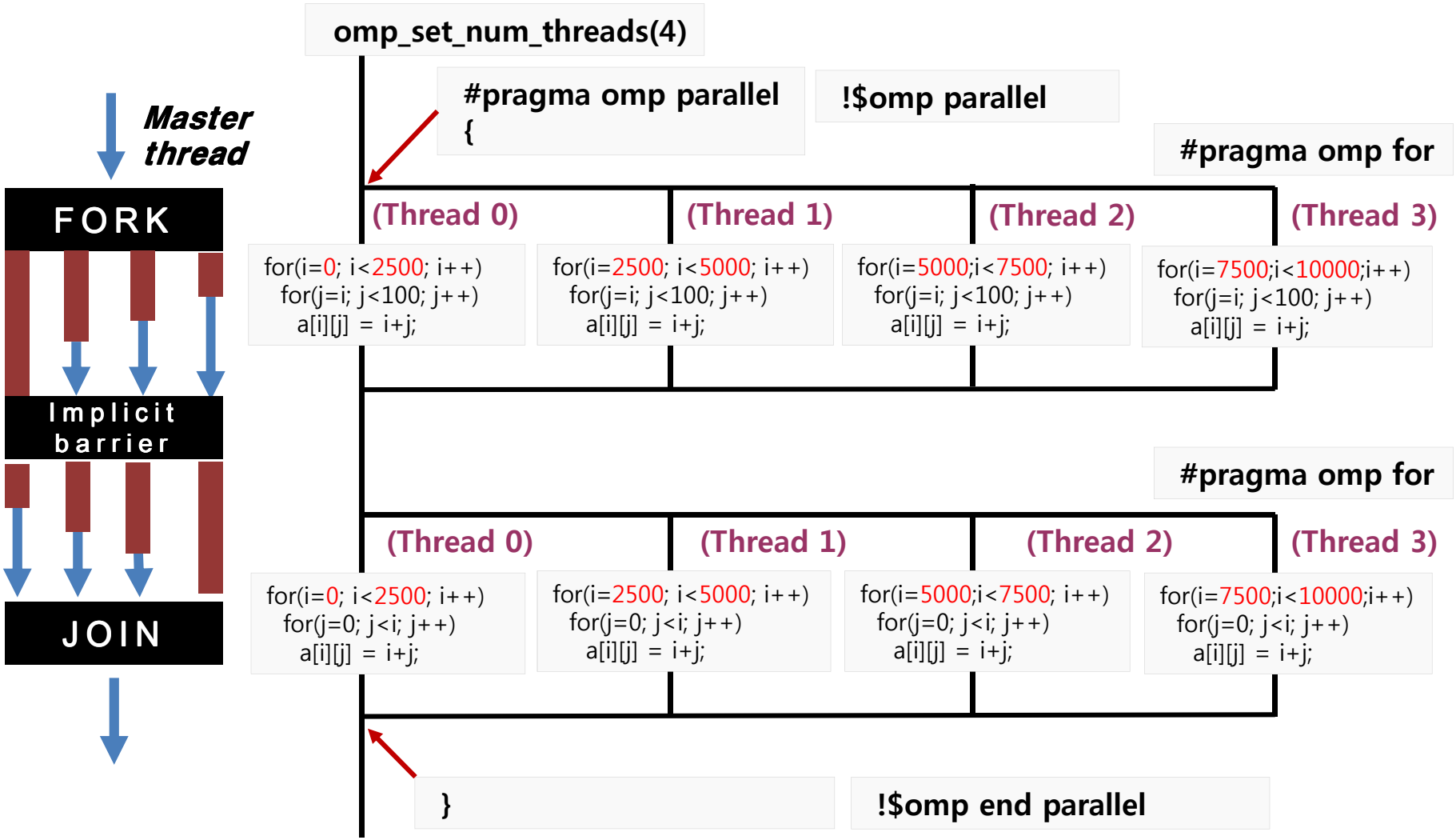
```
$ gfortran –fopenmp –o for_nowait_for.x nowait_for.f90
$ ulimit –s 512000
$ time ./nowait_for.x
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 10000

int main()
{
      int i, j, a[N][N];
      omp_set_num_threads(4);
#pragma omp parallel private(i, j)
{
      #pragma omp for
      for(i=0; i<N; i++)
            for(j=i; j<N; j++)
                  a[i][j] = i+j;

      #pragma omp for
      for(i=0; i<N; i++)
            for(j=0; j<i; j++)
                  a[i][j] = i-j;
}
      return 0;
}
```

```
$ gcc –fopenmp –o nowait_for.x nowait_for.c
$ ulimit –s 512000
$ time ./nowait_for.x
```

Synchronization : nowait



Synchronization : nowait

Fortran

```
PROGRAM nowait_for
      INTEGER, PARAMETER :: N = 9999
      INTEGER i, j, a(0:N, 0:N)
      CALL omp_set_num_threads(4)

!$omp parallel private(i,j)
      !$omp do
      DO j=0, N
            DO i=0, j
                  a(i,j) = i+j
            END DO
      END DO
      !$omp end DO nowait
      !$omp do
      DO j=0, N
            DO i=j+1, N
                  a(i,j) = i-j
            END DO
      END DO
!$omp end parallel

END
```

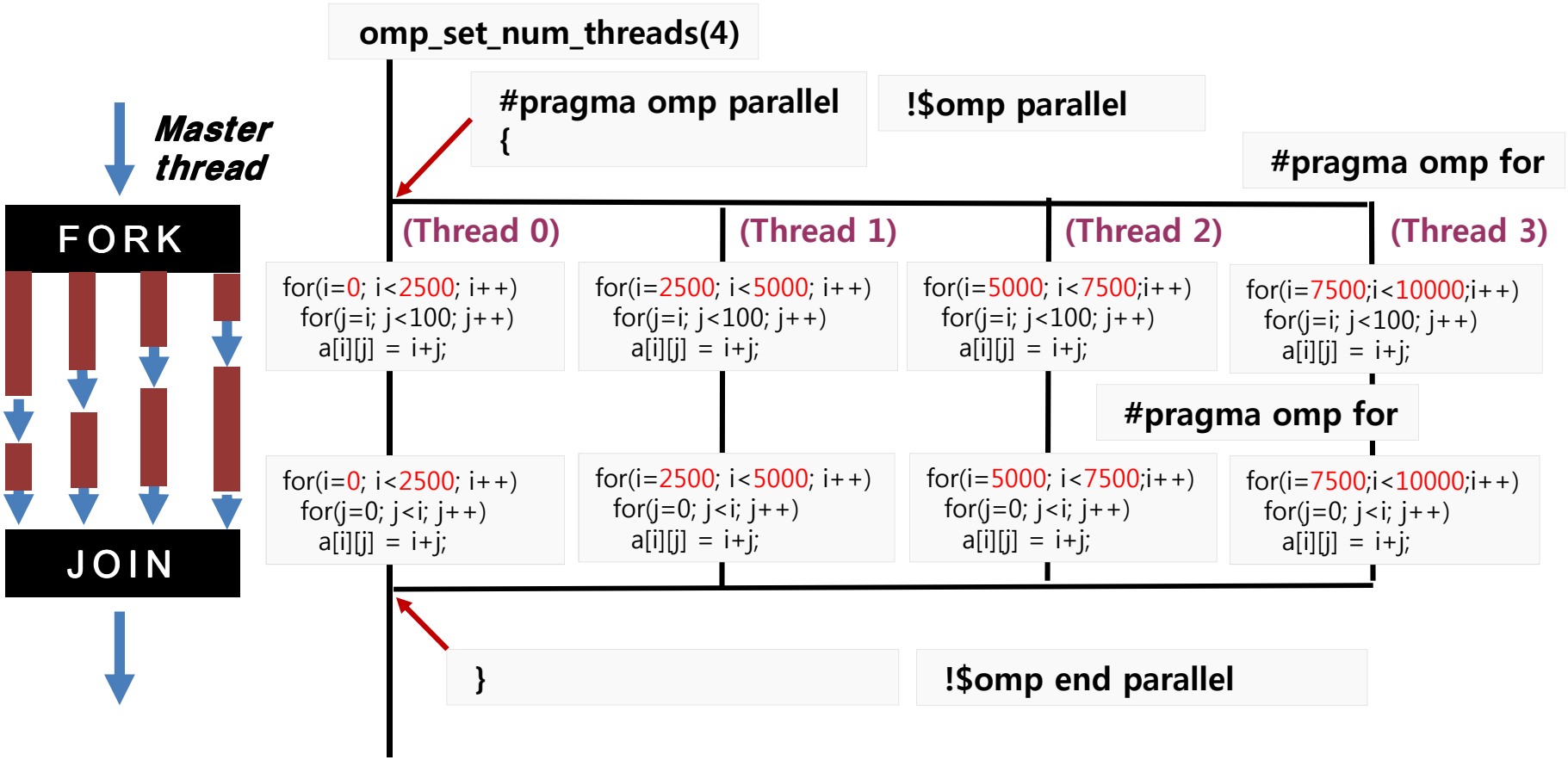
```
$ gfortran –fopenmp –o for_nowait2.x nowait.f90
$ ulimit –s 512000
$ time ./for_nowait2.x
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 10000

int main()
{
  int i, j, a[N][N];
  omp_set_num_threads(4);

#pragma omp parallel private(i, j)
{
  #pragma omp for nowait
      for(i=0; i<N; i++)
            for(j=i; j<N; j++)
                  a[i][j] = i+j;

  #pragma omp for
      for(i=0; i<N; i++)
            for(j=0; j<i; j++)
                  a[i][j] = i-j;
}
      return 0;
}
```

```
$ gcc –fopenmp –o nowait_for2.x nowait_for2.c
$ ulimit –s 512000
$ time ./nowait_for2.x
```

Synchronization : nowait



Synchronization : Simple Test (1/5)

Fortran

```
PROGRAM simple_test1
IMPLICIT NONE

      INTEGER, PARAMETER :: N = 20
      INTEGER :: i, tid, a(N), omp_get_thread_num

      CALL omp_set_num_threads(4)
!$omp parallel private(tid)
      tid = omp_get_thread_num()

      IF (tid /= 2) THEN
            CALL sleep(2)
      END IF

      !$OMP DO
      DO i=0, N-1
            a(i) = i
            PRINT *, 'a(',i,') =',a(i),'tid =',tid
      END DO
      !$omp end DO
      PRINT *, 'END', tid, ' thread'
!$omp end parallel

END
```

```
$ gfortran -fopenmp -o simple_test1.x simple_test1.f90
$ ./simple_test1.x
```

C

```
#include <stdio.h>
#include <unistd.h>
#include <omp.h>
#define N 20

int main()
{
      int i, tid, a[N];
      omp_set_num_threads(4);
#pragma omp parallel private(tid)
{
      tid = omp_get_thread_num();

      if ( tid != 2 )
            sleep(2);

      #pragma omp for
      for (i=0; i<N; i++) {
            a[i] = i;
            printf("a[%d]=%d tid=%d\n", i, a[i], tid );
      }
      printf("END %d thread\n", tid );
}
      return 0;
}
```

```
$ gcc -fopenmp -o simple_test1.x simple_test1.c
$ ./simple_test1.x
```

Synchronization : Simple Test (2/5)

Fortran

```
PROGRAM simple_test2
IMPLICIT NONE

      INTEGER, PARAMETER :: N = 20
      INTEGER :: i, tid, a(N), omp_get_thread_num

      CALL omp_set_num_threads(4)
!$omp parallel private(tid)
      tid = omp_get_thread_num()

      IF (tid /= 2) THEN
            CALL sleep(2)
      END IF

      !$OMP DO
      DO i=0, N-1
            a(i) = i
            PRINT *, 'a(',i,') =',a(i),'tid =',tid
      END DO
      !$omp end DO nowait
      PRINT *, 'END', tid, ' thread'
!$omp end parallel

END
```

```
$ gfortran -fopenmp -o simple_test2.x simple_test2.f90
$ ./simple_test2.x
```

C

```
#include <stdio.h>
#include <unistd.h>
#include <omp.h>
#define N 20

int main()
{
      int i, tid, a[N];
      omp_set_num_threads(4);

#pragma omp parallel private(tid)
{
      tid = omp_get_thread_num();

      if ( tid != 2 )
            sleep(2);

      #pragma omp for nowait
      for (i=0; i<N; i++) {
            a[i] = i;
            printf("a[%d]=%d tid=%d\n", i, a[i], tid );
      }
      printf("END %d thread\n", tid );
}
      return 0;
}
```

```
$ gcc -fopenmp -o simple_test2.x simple_test2.c
$ ./simple_test2.x
```

Synchronization : Simple Test (3/5)

Fortran

```
PROGRAM simple_test3
IMPLICIT NONE

      INTEGER, PARAMETER :: N = 4
      INTEGER :: i, tid, omp_get_thread_num

      CALL omp_set_num_threads(4)
!$omp parallel private(i,tid)
      tid = omp_get_thread_num()

      IF ( tid /= 2 ) THEN
            CALL sleep(2)
      END IF

      !$OMP SECTIONS
      !$OMP SECTION
      DO i=0, N-1
            PRINT *, 'L1 tid =', tid
      END DO

      !$OMP SECTION
      DO i=0, N-1
            PRINT *, 'L2 tid =', tid
      END DO
      !$omp end SECTIONS
!$omp end parallel

END
```

```
$ gfortran -fopenmp -o simple_test3.x simple_test3.f90
$ ./simple_test3.x
```

C

```
#include <stdio.h>
#incldue <unistd.h>
#include <omp.h>
#define N 4

int main()
{
      int i, tid;
      omp_set_num_threads(4);
#pragma omp parallel private(i,tid)
{
      tid = omp_get_thread_num();

      if( tid != 2 )
            sleep(2);
      #pragma omp sections
      {
            #pragma omp section
            for (i=0; i<N; i++)
                  printf("L1 tid=%d\n", tid );
            #pragma omp section
            for (i=0; i<N; i++)
                  printf("L2 tid=%d\n", tid );
      }
}
      return 0;
}
```

```
$ gcc -fopenmp -o simple_test3.x simple_test3.c
$ ./simple_test3.x
```

Synchronization : Simple Test (4/5)

Fortran

```
PROGRAM simple_test4
IMPLICIT NONE

      INTEGER, PARAMETER :: N = 4
      INTEGER :: i, tid, omp_get_thread_num

      CALL omp_set_num_threads(4)
!$omp parallel private(i,tid)
      tid = omp_get_thread_num()

      !$OMP SECTIONS
      !$OMP SECTION
      DO i=0, N-1
            PRINT *, 'L1 tid =', tid
      END DO
     !$OMP SECTION
      DO i=0, N-1
            PRINT *, 'L2 tid =', tid
      END DO
      CALL sleep(2)
      !$omp end SECTIONS
      PRINT *,'tid=', tid
!$omp end parallel

END
```

```
$ gfortran -fopenmp -o simple_test4.x simple_test4.f90
$ ./simple_test4.x
```

C

```
#include <stdio.h>
#incldue <unistd.h>
#include <omp.h>
#define N 4

int main()
{
      int i, tid;
      omp_set_num_threads(4);
#pragma omp parallel private(i,tid)
{
      tid = omp_get_thread_num();
      #pragma omp sections
      {
            #pragma omp section
            for(i=0; i<N; i++)
                  printf("L1 tid=%d\n", tid );
            #pragma omp section
            {
                  for(i=0; i<N; i++)
                        printf("L2 tid=%d\n", tid );
                  sleep(2);
            }
      }
      printf("tid = %d\n", tid );
}
      return 0;
}
```

```
$ gcc -fopenmp -o simple_test4.x simple_test4.c
$ ./simple_test4.x
```

Synchronization : Simple Test (5/5)

Fortran

```
PROGRAM simple_test4
IMPLICIT NONE

      INTEGER, PARAMETER :: N = 4
      INTEGER :: i, tid, omp_get_thread_num

      CALL omp_set_num_threads(4)
!$omp parallel private(i,tid)
      tid = omp_get_thread_num()

      !$OMP SECTIONS
      !$OMP SECTION
      DO i=0, N-1
            PRINT *, 'L1 tid =', tid
      END DO
     !$OMP SECTION
      DO i=0, N-1
            PRINT *, 'L2 tid =', tid
      END DO
      CALL sleep(2)
      !$omp end SECTIONS  nowait
      PRINT *,'tid=', tid
!$omp end parallel

END
```

```
$ gfortran -fopenmp -o simple_test5.x simple_test5.f90
$ ./simple_test5.x
```

C

```
#include <stdio.h>
#incldue <unistd.h>
#include <omp.h>
#define N 4

int main()
{
      int i, tid;
      omp_set_num_threads(4);
#pragma omp parallel private(i,tid)
{
      tid = omp_get_thread_num();
      #pragma omp sections  nowait
      {
            #pragma omp section
            for(i=0; i<N; i++)
                  printf("L1 tid=%d\n", tid );
            #pragma omp section
            {
                  for(i=0; i<N; i++)
                        printf("L2 tid=%d\n", tid );
                  sleep(2);
            }
      }
      printf("tid = %d\n", tid );
}
      return 0;
}
```

```
$ gcc -fopenmp -o simple_test5.x simple_test5.c
$ ./simple_test5.x
```

Synchronization : ordered

Fortran	C

PROGRAM ordered
IMPLICIT NONE
 	
INTEGER i, a(0:9)
 	
CALL omp_set_num_threads(4)
!$omp parallel private(i)
!$OMP DO **ordered**
DO i=0, 9
a(i) = i * 2
!$OMP ORDERED
PRINT *, 'a(', i, ') =', a(i)
!$omp end ORDERED
END DO
!$omp end parallel
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	int i, a[10];
	omp_set_num_threads(4);
	#pragma omp parallel private(i)
	{
	#pragma omp for **ordered**
	for(i=0; i<10; i++) {
	a[i] = i * 2;
	#pragma omp ordered
	printf("a[%d] = %d\n", i, a[i]);
	}
	}
	return 0;
	}

$ gfortran -fopenmp -o ordered.x ordered.f90 $ ./ordered.x	$ gcc -fopenmp -o ordered.x ordered.c $ ./ordered.x

- ORDERED
  - For pipelining loop iterations
  - Can exist only in the dynamic extent of a DO or PARALLEL DO directive
  - The DO directive to which it binds must have the ORDERED clause specified
  - Only one thread can enter at a time

Synchronization : lock

Fortran

```
PROGRAM lock
IMPLICIT NONE

      INTEGER, PARAMETER :: OMP_LOCK_KIND=8
      INTEGER :: x=1
      INTEGER(kind=OMP_LOCK_KIND) LCK

      CALL OMP_INIT_LOCK (LCK)

      CALL omp_set_num_threads(4)
!$omp parallel
      CALL omp_set_lock(LCK)
      x = x+1
      PRINT *, 'x =', x
      CALL omp_unset_lock(LCK)
!$omp end parallel
      CALL omp_destroy_lock(LCK)

END
```

mutually exclusive region

```
$ gfortran -fopenmp -o lock.x lock.f90
$ ./lock.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int x = 1;
      omp_lock_t lock;
      omp_init_lock(&lock);
      omp_set_num_threads(4);

#pragma omp parallel
{
      omp_set_lock(&lock);
      x++;
      printf("x=%d\n", x);
      omp_unset_lock(&lock);
}
      omp_destroy_lock(&lock);

      return 0;
}
```

mutually exclusive region

```
$ gcc -fopenmp -o lock.x lock.c
$ ./lock.x
```

Task

1. task
2. taskwait

Task (1/2)

Fortran	C

PROGRAM task1
IMPLICIT NONE
 	
!$omp parallel num_threads(32)
WRITE(*, '(A)', advance="no") 'A '
WRITE(*, '(A)', advance="no") 'B '
WRITE(*, '(A)', advance="no") 'C '
WRITE(*, '(A)', advance="no") 'D '
WRITE(*, *) ' ‘
!$omp end parallel
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	#pragma omp parallel num_threads(32)
	{
	printf(“A “);
	printf(“B “);
	printf(“C “);
	printf(“D “);
	printf(“\n”);
	}
	return 0;
	}

$ gfortran -fopenmp -o task1.x task1.f90 $ ./task1.x	$ gcc -fopenmp -o task1.x task1.c $ ./task1.x

Task (1/2)

Fortran	C

PROGRAM task2
IMPLICIT NONE
 	
INTEGER omp_get_thread_num
 	
!$omp parallel num_threads(32)
!$OMP SINGLE
PRINT *, 'A tid =', omp_get_thread_num()
PRINT *, 'B tid =', omp_get_thread_num()
PRINT *, 'C tid =', omp_get_thread_num()
PRINT *, 'D tid =', omp_get_thread_num()
!$omp end SINGLE
!$omp end parallel
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	#pragma omp parallel num_threads(32)
	{
	#pragma omp single
	{
	printf("A tid=%d\n", omp_get_thread_num());
	printf("B tid=%d\n", omp_get_thread_num());
	printf("C tid=%d\n", omp_get_thread_num());
	printf("D tid=%d\n", omp_get_thread_num());
	}
	}
	return 0;
	}

$ gfortran -fopenmp -o task2.x task2.f90 $ ./task2.x	$ gcc -fopenmp -o task2.x task2.c $ ./task2.x

Task (1/2)

Fortran

```
PROGRAM task3
IMPLICIT NONE

      INTEGER omp_get_thread_num

!$omp parallel num_threads(32)
      !$OMP SINGLE
      PRINT *, 'A tid =', omp_get_thread_num()

            !$OMP TASK
            PRINT *, 'B tid =', omp_get_thread_num()
            !$omp end TASK

            !$OMP TASK
            PRINT *, 'C tid =', omp_get_thread_num()
            !$omp end TASK

      PRINT *, 'D tid =', omp_get_thread_num()
      !$omp end SINGLE
!$omp end parallel

END
```

```
$ gfortran -fopenmp -o task3.x task3.f90
$ ./task3.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
#pragma omp parallel num_threads(32)
{
      #pragma omp single
      {
            printf("A tid=%d\n", omp_get_thread_num() );
            #pragma omp task
            {
                  printf("B tid=%d\n", omp_get_thread_num() );
            }
            #pragma omp task
            {
                  printf("C tid=%d\n", omp_get_thread_num() );
            }
            printf("D tid=%d\n", omp_get_thread_num() );
      }
}
      return 0;
}
```

```
$ gcc -fopenmp -o task3.x task3.c
$ ./task3.x
```

Task (1/2)

Fortran

```
PROGRAM task3
IMPLICIT NONE

      INTEGER, PARAMETER :: N = 20
      INTEGER :: i, omp_get_thread_num

      !$omp parallel num_threads(8)
      !$OMP SINGLE firstprivate(i)
            do while (i < N)
                !$OMP TASK
                PRINT *, 'index :', i, 'tid : ', omp_get_thread_num()
                !$omp end TASK
            end do
      !$omp end SINGLE
!$omp end parallel

END
```

```
$ gfortran -fopenmp -o task4.x task4.f90
$ ./task3.x
```

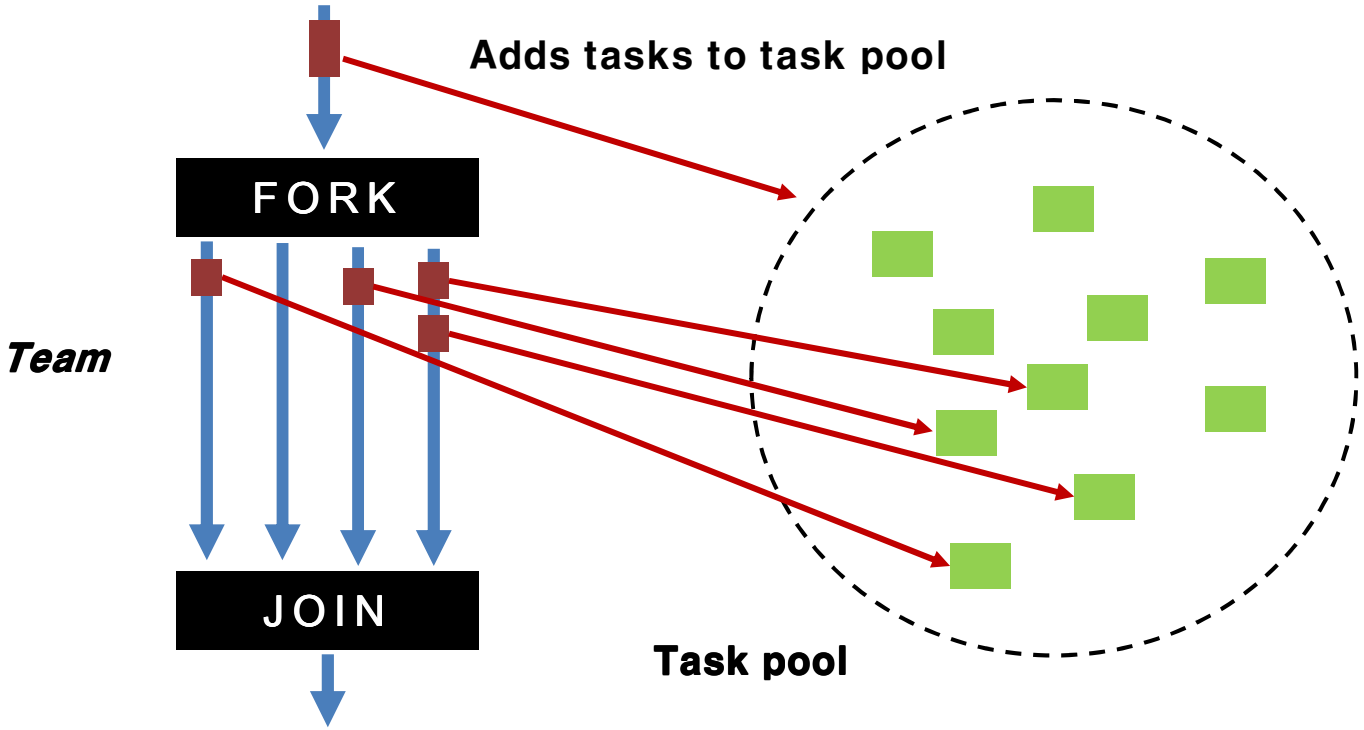
C

```
#include <stdio.h>
#include <omp.h>
#define N 20
int main()
{
#pragma omp parallel num_threads(8)
{
    #pragma omp single
    {
        int i = 0;
        while (i++ < N)
        #pragma omp task
        {
            printf("index : %d, tid : %d \n", i, omp_get_thread_num());
         }
     }
}
      return 0;
}
```

```
$ gcc -fopenmp -o task4.x task4.c
$ ./task3.x
```

Task (2/2)

- TASK Construct
  - New construct with OpenMP 3.0
  - The TASK construct defines an explicit task, which may be executed by the encountering thread or deferred for execution by any other thread
  - The data environment of the task is determined by the data sharing attribute clauses



Task : taskwait

Fortran

```
PROGRAM taskwait
IMPLICIT NONE

      INTEGER omp_get_thread_num

!$omp parallel num_threads(32)
!$OMP SINGLE
      PRINT *, 'A tid =', omp_get_thread_num()

      !$OMP TASK
            PRINT *, 'B tid =', omp_get_thread_num()
      !$omp end TASK

      !$OMP TASK
            PRINT *, 'C tid =', omp_get_thread_num()
      !$omp end TASK
!$OMP TASKWAIT
      PRINT *, 'D tid =', omp_get_thread_num()
!$omp end SINGLE
!$omp end parallel

END
```

```
$ gfortran -fopenmp -o taskwait.x taskwait.f90
$ ./taskwait.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
#pragma omp parallel num_threads(32)
{
      #pragma omp single
      {
            printf("A tid=%d\n", omp_get_thread_num() );

            #pragma omp task
            {
                  printf("B tid=%d\n", omp_get_thread_num() );
            }
            #pragma omp task
            {
                  printf("C tid=%d\n", omp_get_thread_num() );
            }
      #pragma omp taskwait
      printf("D tid=%d\n", omp_get_thread_num() );
      }
}
      return 0;
}
```

```
$ gcc -fopenmp -o taskwait.x taskwait.c
$ ./taskwait.x
```

Task : Data Scoping (1/2)

Fortran	C

PROGRAM task_data_scope
IMPLICIT NONE
 	
INTEGER :: a, b, c, d, e
 	
!$omp parallel private(b,d,e)
!$OMP TASK private(e)
a, b, c, d, e : private? shared?
!$omp end TASK
!$omp end parallel
 	
END
	int foo()
	{
	int a, b, c,d,e;
	#pragma omp parallel private(b,d,e)
	{
	#pragma omp task private(e)
	{ a, b, c, d, e : private? shared? }
	}
	}

PROGRAM task
USE OMP_LIB or include “omp_lib.h”
IMPLICIT NONE
 	
INTEGER :: a, b, c, d, e
!$omp parallel private(b,d,e)
!$OMP TASK private(e)
a : shared
b : firstprivate
c : shared
d : firstprivate
e : private
!$omp end TASK
!$omp end parallel
 	
END
	int foo()
	{
	int a, b, c,d,e;
	#pragma omp parallel private(b,d,e)
	{
	#pragma omp task
	{
	a : shared
	b : firstprivate
	c : shared
	d : firstprivate
	e : private
	}
	}
	}

Task : Data Scoping (2/2)

Fortran

```
PROGRAM task_data_scope
IMPLICIT NONE

      INTEGER :: a=1, b=2, c=3, d=4, e=5
      INTEGER tid, omp_get_thread_num
      CALL omp_set_num_threads(4)

!$omp parallel private(b,d,e,tid)
      tid = omp_get_thread_num()
      PRINT 10, 'tid=',tid, ' a=',a, ' b=',b, ' c=',c, ' d=',d, ' e=',e
      !$omp single
      a=2; b=3; c=4; d=5;e=6
      !$omp end single
      !$omp task private(e)
      PRINT 10, 'task tid=',omp_get_thread_num(), &
            & ' a=',a, ' b=',b, ' c=',c, ' d=',d, ' e=',e
      !$omp end task
!$omp end parallel

10 FORMAT(A,I4,A,I4,A,I4,A,I4,A,I4,A,I4)

END
```

```
$ gfortran -fopenmp -o task_data_scope.x task_data_scope.f90
$ ./task_data_scope.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int a=1, b=2, c=3, d=4, e=5, tid;

      omp_set_num_threads(4);
#pragma omp parallel private(b,d,e, tid)
{
      tid = omp_get_thread_num();
      printf("tid=%d a=%d b=%d c=%d d=%d e=%d\n",
            tid, a, b, c, d, e);
      #pragma omp single
      { a=2, b=3, c=4, d=5, e=6; }
      #pragma omp task private(e)
      {
            printf("task tid=%d a=%d b=%d c=%d d=%d e=%d\n",
                  omp_get_thread_num(), a, b, c, d, e);
      }
}
      return 0;
}
```

```
$ gcc -fopenmp -o task_data_scope.x
task_data_scope.c$ ./task_data_scope.x
```

Break!!

Schedule

1. Schedule

SCHEDULE

- Scheduling clauses
  - Specifies how iterations of the for loop are divided among the threads of the team.
  - Useful if the work sharing construct is a do-loop of for-loop. The iteration in the work sharing construct are assigned to threads according to the scheduling method defined by this clause.

  - Static [, chunk]
    - Each thread is assigned a fixed number of chunks to work on
    - Chunks are assigned to the threads in Round-Robin fashion

  - Dynamic [, chunk]
    - The assignment of iterations to threads is determined at runtime

  - Guided [, chunk]
    - a variant of dynamic scheduling.
    - the first chunk of iterations is of some implementation-depENDent size, and the size of each successive chunk is a fixed fraction of the preceding chunk until a minimum size of chunk_size is reached.

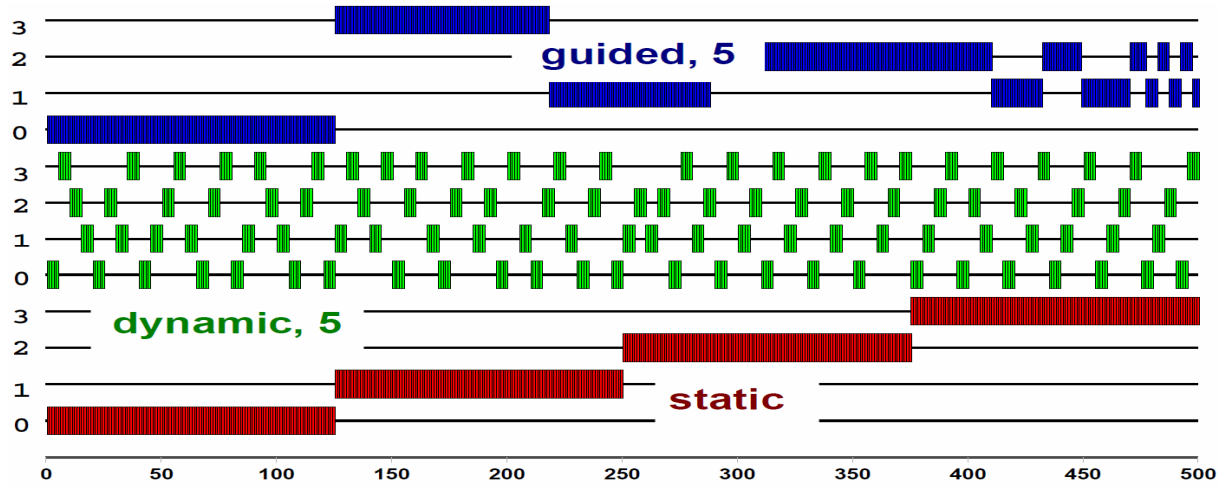
  - runtime
    - allows the scheduling to be determined at runtime. The chunk_size PARAMETER must not appear.
    - The type is chosen at runtime based on the value of the environment variable OMP_SCHEDULE.

SCHEDULE

Fortran	C

PROGRAM static
IMPLICIT NONE
 	
INTEGER :: a(0:499), I
 	
!$omp parallel DO schedule(static) num_threads(4)
DO i=0, 499
a(i) = i
END DO
 	
END
	#include <stdio.h>
	#include <omp.h>
	
	int main()
	{
	int i, a[500];
	
	#pragma omp parallel for schedule(static) num_threads(4)
	for(i=0; i<500; i++)
	a[i] = i;
	
	return 0;
	}

$ gfortran -fopenmp -o static.x static.f90 $ time ./static.x	$ gcc -fopenmp -o static.x static.c $ time ./static.x



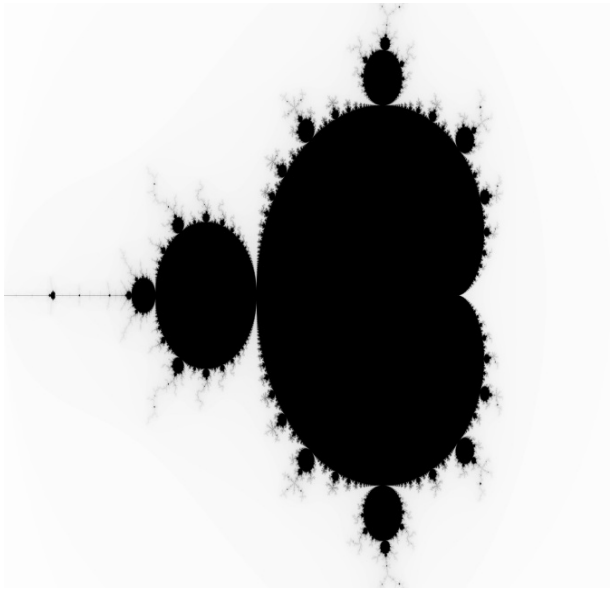
- Mandelbrot
  - Mandelbrot is an image computing and display computation
  - Pixels of an image are stored in a 2D array
  - Each pixel is computed by iterating the COMPLEX function

$$Z_{k+1} = {Z_k}^2 + C\,, \qquad Z_k = a_k + b_k i, \qquad \mathrm{C} = C_{real} + C_{imag} i$$

$$Z_{k+1} = (a_k + b_k i)^2 + (C_{real} + C_{imag} i\,)$$

$$= \left({a_k}^2 - {b_k}^2 + C_{real} \right) + \left(2 a_k b_k + C_{imag}\right) i$$

  - The magnitude of z is given by $Z_{length} = \sqrt{a^2 + b^2}$



SCHEDULE (Exercise)

C (Serial)

```
#include <stdio.h>

#define       X_RESN  4000      /* x resolution */
#define       Y_RESN  4000      /* y resolution */
#define       X_MIN   -2.0
#define       X_MAX    2.0
#define       Y_MIN   -2.0
#define       Y_MAX    2.0

typedef struct COMPLEXtype {
   float REAL, imag;
} Compl;

int main ( int argc, char* argv[])
{
   int i, j, k, maxIterations = 1000;
   Compl   z, c;
   float   lengthsq, temp;
   int res[X_RESN][Y_RESN];
```

```
   for (i=0; i < X_RESN; i++) {
      for (j=0; j < Y_RESN; j++) {
         z.REAL = z.imag = 0.0;
         c.REAL =  X_MIN + j * (X_MAX - X_MIN) / X_RESN;
         c.imag = Y_MAX - i * (Y_MAX - Y_MIN) / Y_RESN;
         k = 0;

         do {
            temp = z.REAL*z.REAL - z.imag*z.imag + c.REAL;
            z.imag = 2.0*z.REAL*z.imag + c.imag;
            z.REAL = temp;
            lengthsq = z.REAL*z.REAL+z.imag*z.imag;
            k++;
         } while (lengthsq < 4.0 && k < maxIterations);

         if (k >= maxIterations) res[i][j] = 0;
         else res[i][j] = 1;
      }
   }
   return 0;
}
```

```
$ gcc -fopenmp -o schedule.x schedule.c
$ ulimit -s 512000
$ time ./schedule.x
```

SCHEDULE (Exercise)

Fortran (Serial)

```
PROGRAM mandel_serial
IMPLICIT NONE

  INTEGER, PARAMETER :: X_RESN=4000, Y_RESN=4000
  REAL, PARAMETER  :: X_MIN=-2.0, X_MAX=2.0, &
                      Y_MIN=-2.0, Y_MAX=2.0

  COMPLEX :: z, c
  INTEGER  :: i, j, k, maxIterations=1000
  REAL  :: lengthsq, temp
  INTEGER, dimension(0:X_RESN-1, 0:Y_RESN-1)  :: res

  DO j=0, Y_RESN-1
    DO i=0, X_RESN-1
      z = (0.0, 0.0)
      c = cmplx( X_MIN+REAL(j) * (X_MAX-X_MIN) &
                      / REAL(X_RESN), &
                Y_MAX - REAL(i) * (Y_MAX - Y_MIN) &
                      / REAL(Y_RESN) )
      k=0
      loop: DO
        temp = REAL(z) * REAL(z) - AIMAG(z) * AIMAG(z) &
                + REAL(c)
        z=cmplx( temp, 2.0*REAL(z)*AIMAG(z)+AIMAG(c) )
        lengthsq = REAL(z)*REAL(z)+AIMAG(z)*AIMAG(z)
        k=k+1
        IF (lengthsq >= 4.0 .or. k >= maxIterations) exit loop
      END DO loop

      IF ( k >= maxIterations ) THEN
        res(i, j) = 0
      ELSE
        res(i, j) = 1
      END IF

    END DO
  END DO

END
```

```
$ gfortran  -fopenmp  -o  schedule.x  schedule.f90
$ ulimit -s  512000
$ time  ./schedule.x
```

SCHEDULE (Solution)

C (Parallel)

```
#include <stdio.h>
#include <omp.h>

#define       X_RESN  4000       /* x resolution */
#define       Y_RESN  4000       /* y resolution */
#define       X_MIN   -2.0
#define       X_MAX    2.0
#define       Y_MIN   -2.0
#define       Y_MAX    2.0

typedef struct COMPLEXtype {
   float REAL, imag;
} Compl;

int main ( int argc, char* argv[])
{
   int i, j, k, maxIterations = 1000;
   Compl   z, c;
   float   lengthsq, temp;
   int res[X_RESN][Y_RESN];
```

```
#pragma omp parallel for shared(res, maxIterations)
    private(j,z,c,k,temp,lengthsq)
   for (i=0; i < X_RESN; i++) {
      for (j=0; j < Y_RESN; j++) {
         z.REAL = z.imag = 0.0;
         c.REAL = X_MIN + j * (X_MAX - X_MIN)/X_RESN;
         c.imag = Y_MAX - i * (Y_MAX - Y_MIN)/Y_RESN;
         k = 0;

         do  {
            temp = z.REAL*z.REAL - z.imag*z.imag + c.REAL;
            z.imag = 2.0*z.REAL*z.imag + c.imag;
            z.REAL = temp;
            lengthsq = z.REAL*z.REAL+z.imag*z.imag;
            k++;
         } while (lengthsq < 4.0 && k < maxIterations);

         if (k >= maxIterations) res[i][j] = 0;
         else res[i][j] = 1;
      }
   }
}
```

```
$ gcc -fopenmp -o schedule.x schedule.c
$ export OMP_NUM_THREADS=8
$ time ./schedule.x
```

SCHEDULE (Solution)

Fortran (Parallel)

```
PROGRAM solution
IMPLICIT NONE

   INTEGER, PARAMETER :: X_RESN=4000, Y_RESN=4000
   REAL, PARAMETER  :: X_MIN=-2.0, X_MAX=2.0,
                       Y_MIN=-2.0, Y_MAX=2.0

   COMPLEX :: z, c
   INTEGER  :: i, j, k, maxIterations=1000
   REAL  :: lengthsq, temp
   INTEGER, demension(0:X_RESN-1, 0:Y_RESN-1)  :: res

!$omp parallel DO shared(res, maxIterations) private(z, c, k, temp,
    lengthsq)
   DO j=0, Y_RESN-1
      DO i=0, X_RESN-1
         z = (0.0, 0.0)
         c = cmplx( X_MIN + REAL(j) * (X_MAX - Y_MIN)
                          / REAL(X_RESN),
                    Y_MAX - REAL(i) * (Y_MAX - Y_MIN)
                          / REAL(Y_RESN) )
         k=0
         loop: DO
            temp = REAL(z) * REAL(z) - AIMAG(z) * AIMAG(z)
                   + REAL(c)
            z=cmplx( temp, 2.0*REAL(z)*AIMAG(z)+AIMAG(c) )
            lengthsq = REAL(z)*REAL(z)+AIMAG(z)*AIMAG(z)
            k=k+1
            IF (lengthsq >= 4.0 .or. k >= maxIterations) exit loop
         END DO loop

         IF ( k >= maxIterations ) THEN
            res(i, j) = 0
         ELSE
            res(i, j) = 1
         END IF

      END DO
   ENDDO

END
```

```
$ gfortran -fopenmp -o schedule.x schedule.f90
$ export OMP_NUM_THREADS=8
$ time ./schedule.x
```

SCHEDULE (Solution Cont.)

C (parallel with dynamic schedule)

```
#include <stdio.h>
#include <omp.h>

#define       X_RESN  4000      /* x resolution */
#define       Y_RESN  4000      /* y resolution */
#define       X_MIN   -2.0
#define       X_MAX    2.0
#define       Y_MIN   -2.0
#define       Y_MAX    2.0

typedef struct COMPLEXtype {
   float REAL, imag;
} Compl;

int main ( int argc, char* argv[])
{
   int i, j, k, maxIterations = 1000;
   Compl   z, c;
   float   lengthsq, temp;
   int res[X_RESN][Y_RESN];
```

```
#pragma omp parallel for shared(res, maxIterations)
    private(j,z,c,k,temp,lengthsq) schedule(dynamic, 5)
   for(i=0; i < X_RESN; i++) {
      for(j=0; j < Y_RESN; j++) {
         z.REAL = z.imag = 0.0;
         c.REAL = X_MIN + j * (X_MAX - X_MIN)/X_RESN;
         c.imag = Y_MAX - i * (Y_MAX - Y_MIN)/Y_RESN;
         k = 0;

         do {
            temp = z.REAL * z.REAL - z.imag * z.imag + c.REAL;
            z.imag = 2.0 * z.REAL * z.imag + c.imag;
            z.REAL = temp;
            lengthsq = z.REAL * z.REAL + z.imag * z.imag;
            k++;
         } while (lengthsq < 4.0 && k < maxIterations);

         IF (k >= maxIterations) res[i][j] = 0;
         ELSE res[i][j] = 1;
      }
   }
}
```

```
$ gcc -fopenmp -o schedule.x schedule.c
$ export OMP_NUM_THREADS=8
$ time ./schedule.x
```

SCHEDULE (Exercise)

Fortran (Parallel)

```
PROGRAM ex
USE OMP_LIB or include “omp_lib.h”
IMPLICIT NONE

  INTEGER, PARAMETER :: X_RESN=4000, Y_RESN=4000
  REAL, PARAMETER  :: X_MIN=-2.0, X_MAX=2.0, Y_MIN=-
    2.0, Y_MAX=2.0

  COMPLEX :: z, c
  INTEGER  :: i, j, k, maxIterations=1000
  REAL  :: lengthsq, temp
  INTEGER, demension(0:X_RESN-1, 0:Y_RESN-1)  :: res

!$omp parallel DO shared(res, maxIterations) private(z, c, k, temp,
    lengthsq) schedule(dynamic, 5)
  DO j=0, Y_RESN-1
    DO i=0, X_RESN-1
      z = (0.0, 0.0)
      c = cmplx( X_MIN+REAL(j)*(X_MAX-X_MIN) /
   REAL(X_RESN), Y_MAX-REAL(i) * (Y_MAX-Y_MIN) /
   REAL(Y_RESN) )
      k=0
loop :  DO
        temp = REAL(z)*REAL(z)-AIMAG(z)*AIMAG(z) +
   REAL(c)
        z=cmplx( temp, 2.0*REAL(z)*AIMAG(z)+AIMAG(c) )
        lengthsq = REAL(z)*REAL(z)+AIMAG(z)*AIMAG(z)
        k=k+1
        if (lengthsq >= 4.0 && k >= maxIterations) THEN
          exit loop
      END DO loop

      if ( k >= maxIterations ) THEN
        res(i, j) = 0
      else
        res(i, j) = 1
      END IF

    END DO
  ENDDO

END
```

```
$ gfortran -fopenmp -o schedule.x schedule.f90
$ export OMP_NUM_THREADS=8
$ time ./schedule.x
```

OpenMP Performance Issue

1. Nested Parallel (Collapse)
2. Flush
3. False Sharing
4. DepENDency

Collapse (1/4)

Fortran

```
PROGRAM collapse
USE OMP_LIB or include “omp_lib.h”
IMPLICIT NONE

      INTEGER, PARAMETER  :: N=10000
      INTEGER  :: a(0:N-1, 0:N-1), i, j

      CALL omp_set_nested(1)
      CALL omp_set_num_threads(4)
!$omp parallel do private(i, j)
      DO i=0, N-1
      !$omp parallel do
            DO j=0, N-1
                  a(i, j) = a(i, j) * 2
            END DO
      END DO

END
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 10000

int main()
{
      int a[N][N], i, j;

      omp_set_nested(1);
      omp_set_num_threads(4);
#pragma omp parallel for private(i,j)
      for(i=0; i<N; i++) {
            #pragma omp parallel for
            for(j=0; j<N; j++)
                  a[i][j] = a[i][j] * 2;
      }

      return 0;
}
```

Better performance?

Collapse (2/4)

Fortran

```
PROGRAM collapse2
USE OMP_LIB or include “omp_lib.h”
IMPLICIT NONE

      INTEGER, PARAMETER  :: N=10000
      INTEGER  :: a(0:N-1, 0:N-1), i, j

      CALL omp_set_nested(1)
      CALL omp_set_num_threads(4)
!$omp parallel private(i, j)

!$OMP DO
      DO i=0, N-1
      !$OMP DO
            DO j=0, N-1
                  a(i, j) = a(i, j) * 2
            END DO
      END DO

END
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 10000

int main()
{
      int a[N][N], i, j;

      omp_set_nested(1);
      omp_set_num_threads(4);
#pragma omp parallel private(i,j)
{
      #pragma omp for
      for(i=0; i<N; i++) {
            #pragma omp for
            for(j=0; j<N; j++)
                  a[i][j] = a[i][j] * 2;
      }
}
      return 0;
}
```

Possible ???

Collapse (3/4)

Fortran	C

PROGRAM collapse3
IMPLICIT NONE
 	
INTEGER, PARAMETER :: N=10000
INTEGER :: a(0:N-1, 0:N-1), i
 	
CALL omp_set_num_threads(4)
 	
!$omp parallel DO
DO i=0, N*N-1
a(i/N, MOD(i, N)) = a(i/N, MOD(i, N)) * 2
END DO
 	
END
	#include <stdio.h>
	#include <omp.h>
	#define N 10000
	
	int main()
	{
	static int a[N][N], i, j;
	
	omp_set_num_threads(4);
	
	#pragma omp parallel for
	for(i=0; i<N*N; i++)
	a[i/N][i%N] = a[i/N][j%N] * 2;
	
	return 0;
	}

$ gfortran -fopenmp -o collapse.x collapse.f90
$ ulimit -s 512000
$ time ./collapse.x
	$ gcc -fopenmp -o collapse.x collapse.c
	$ time ./collapse.x

Collapse (4/4)

Fortran

```
PROGRAM collapse4
IMPLICIT NONE

      INTEGER, PARAMETER  :: N=9999
      INTEGER  :: a(0:N, 0:N), i, j

      CALL omp_set_num_threads(4)

!$omp parallel do private(i, j) collapse(2)
      DO j=0, N
            DO i=0, N
                  a(i, j) = a(i, j) * 2
            END DO
      END DO
!$omp end parallel do

END
```

```
$ gfortran -fopenmp -o collapse4.x collapse4.f90
$ ulimit -s 512000
$ time ./collapse4.x
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 10000

int main()
{
      int a[N][N], i, j;

      omp_set_num_threads(4);

#pragma omp parallel for private(i,j) collapse(2)
      for(i=0; i<N; i++)
            for(j=0; j<N; j++)
                  a[i][j] = a[i][j] * 2;

      return 0;
}
```

```
$ gcc -fopenmp -o collapse4.x collapse4.c
$ time ./collapse4.x
```

Flush (1/3)

Fortran

```
PROGRAM flush1
IMPLICIT NONE

      INTEGER  :: x=0, tid, omp_get_thread_num
      CALL omp_set_num_threads(4)

!$omp parallel private(tid)
      tid = omp_get_thread_num()

      IF ( tid == 0 ) THEN
            CALL sleep(3)
            x = 1
            CALL sleep(3)
      END IF

      DO WHILE ( x == 0 )
      END DO

      PRINT *, "x =", x, "in", tid, "-th thread"
!$omp end parallel

END
```

```
$ gfortran –fopenmp –o ok.x  flush.f90
$ ./ok.x
$ gfortran –O3 –fopenmp –o no.x  flush.f90
$ ./no.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int x=0, tid;
      omp_set_num_threads(4);

#pragma omp parallel private(tid)
{
      tid = omp_get_thread_num();

      if ( tid == 0 ) {
            sleep(3);
            x = 1;
            sleep(3);
      }

      while( x == 0 ) { }

      printf("x=%d in %d-th thread\n", x, tid);
}
       return 0;
}
```

```
$ gcc –fopenmp –o ok.x  flush.c
$ ./ok.x
$ gcc –O3 –fopenmp –o no.x  flush.c
$ ./no.x
```

Flush (2/3)

- FLUSH Directive
  - Acts as a memory barrier/fence
  - The compiler and hardware do not reorder your reads and WRITEs
  - cf. volatile, WRITE-through, WRITE-back

```
while( x == 0 ) {
}
x = x + 1;
```

Pseudo code

Without optimization option

```
L1:
   mov eax, 0x7c385400
   cmp eax, 0
   je L1
   mov eax, 0x7c385400
   inc eax
   mov 0x7c385400, eax
```

With optimization option (-O3)

```
   mov eax, 0x7c385400
L1:
   cmp eax, 0
   je L1
   inc eax
   mov 0x7c385400, eax
```

Flush (3/3)

Fortran

```
PROGRAM flush2
IMPLICIT NONE

      INTEGER  :: x=0, tid
      CALL omp_set_num_threads(4)
!$omp parallel private(tid)
      tid = omp_get_thread_num()
      IF ( tid == 0 ) THEN
            CALL sleep(3)
            x = 1
            CALL sleep(3)
      END IF

      DO WHILE ( x == 0 )
            !$OMP FLUSH(x)
      END DO
      PRINT *, "x =", x, "in", tid, "-th thread"
!$omp end parallel

END
```

```
$ gfortran –O3 –fopenmp –o flush.x flush.c
$ ./flush.x
```

C

```
#include <stdio.h>
#include <omp.h>

int main()
{
      int x=0, tid;
      omp_set_num_threads(4);
#pragma omp parallel private(tid)
{
      tid = omp_get_thread_num();
      if ( tid == 0 ) {
            sleep(3);
            x = 1;
            sleep(3);
      }

      while ( x == 0 ) {
            #pragma omp flush(x)
      }
      printf("x=%d in %d-th thread\n", x, tid);
}
      return 0;
}
```

```
$ gcc –O3 –fopenmp –o flush.x flush.c
$ ./flush.x
```

False Sharing (1/4)

Fortran

```
PROGRAM false_sharing
IMPLICIT NONE

      INTEGER, PARAMETER  :: N=1000000000
      INTEGER(8) :: total_sum=0, a(0:3)
      INTEGER i, tid, omp_get_thread_num
      a=0
      CALL omp_set_num_threads(4)

!$omp parallel private(tid)
      tid = omp_get_thread_num()
      !$OMP DO
      DO i=1, N
            a(tid) = a(tid) + i
      END DO
      !$OMP ATOMIC
      total_sum = total_sum + a(tid)
!$omp end parallel
      PRINT *, “sum =“, total_sum

END
```

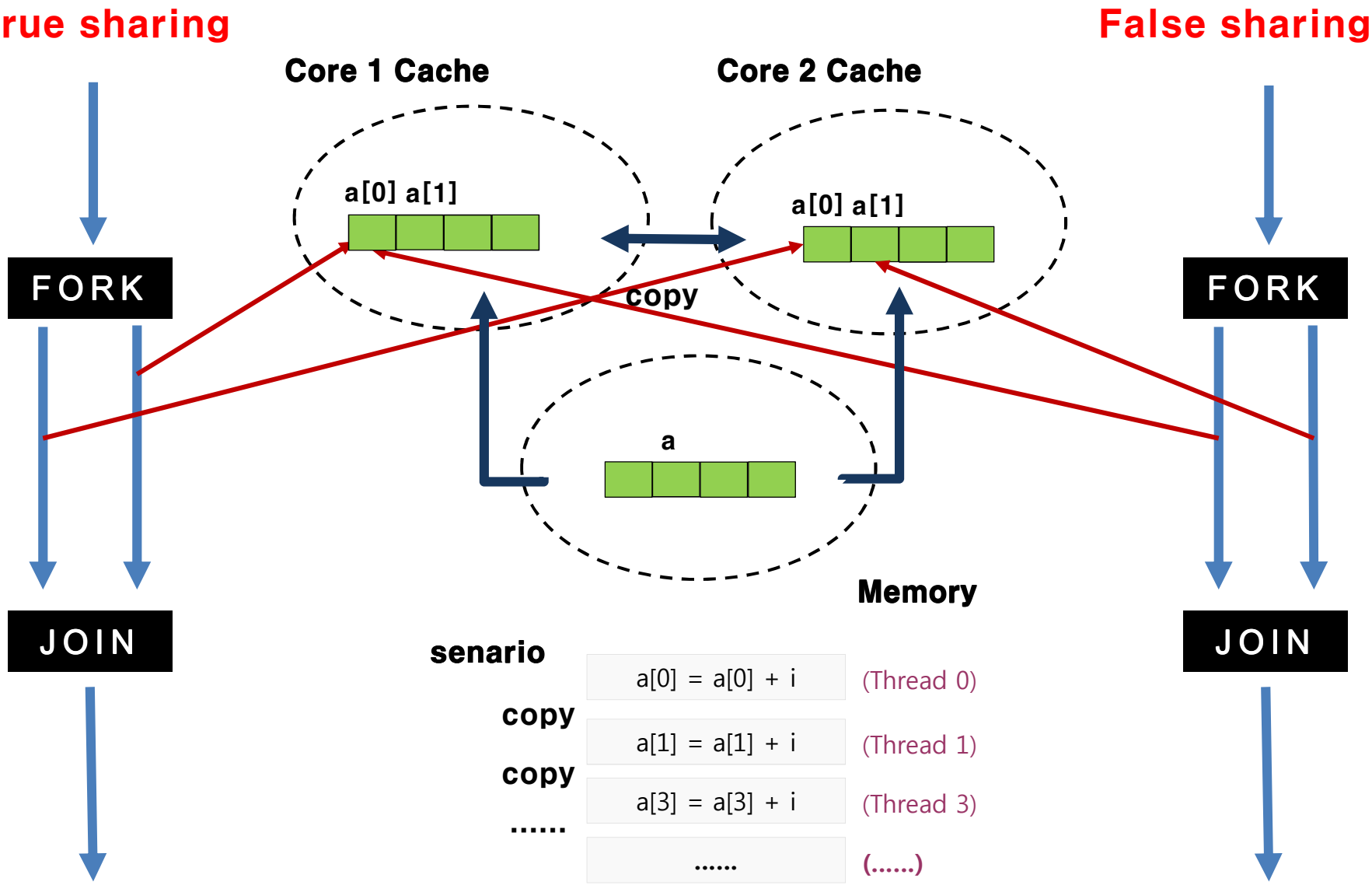
```
$ gfortran -fopenmp -o false_sharing.x false_sharing.f90
$ time ./false_sharing.x
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 1000000000

int main()
{
      long sum=0, a[4] = { 0 }, i, tid;

      omp_set_num_threads(4);
#pragma omp parallel private(tid)
{
      tid = omp_get_thread_num();
      #pragma omp for
      for(i=1; i<=N; i++)
            a[tid] += i;
      #pragma omp atomic
      sum += a[tid];
}
      printf(“sum = %ld\n”, sum);
      return 0;
}
```

```
$ gcc -fopenmp -o false_sharing.x false_sharing.c
$ time ./false_sharing.x
```

False Sharing (2/4)



False Sharing (3/4)

Fortran

```
PROGRAM false_sharing_solved
IMPLICIT NONE

      INTEGER, PARAMETER  :: N=1000000000
      INTEGER(8) :: total_sum=0, a(0:3*8)
      INTEGER i, tid, omp_get_thread_num
      a=0
      CALL omp_set_num_threads(4)

!$omp parallel private(tid)
      tid = omp_get_thread_num()
      !$OMP DO
      DO i=1, N
            a(tid*8) = a(tid*8) + i
      END DO
      !$OMP ATOMIC
      total_sum = total_sum + a(tid*8)
!$omp end parallel
      PRINT *, “sum =“, total_sum

END
```

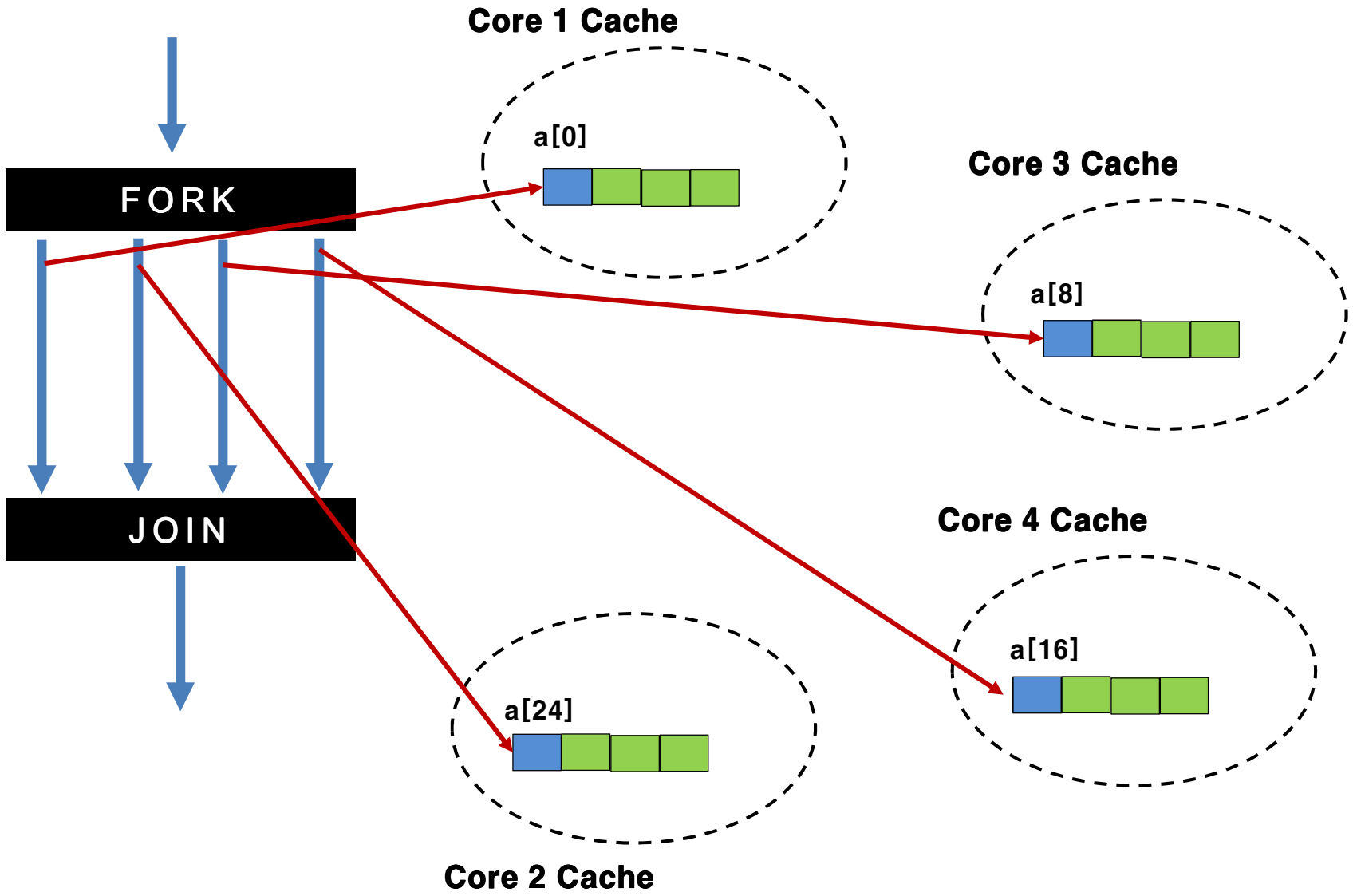
```
$ gfortran -fopenmp -o false_sharing_sol.x false_sharing_sol.f90
$ time ./false_sharing_sol.x
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 1000000000

int main()
{
      long sum=0, a[4*8] = { 0 }, i, tid;

      omp_set_num_threads(4);
#pragma omp parallel private(tid)
{
      tid = omp_get_thread_num();
      #pragma omp for
      for(i=1; i<=N; i++)
            a[tid*8] += i;
      #pragma omp atomic
      sum += a[tid*8];
}
      printf(“sum = %ld\n”, sum);
      return 0;
}
```

```
$ gcc -fopenmp -o false_sharing_sol.x false_sharing_sol.c
$ time ./false_sharing_sol.x
```

False Sharing (4/4)



False Sharing (4/4)

Fortran

```
PROGRAM use_local_sum
IMPLICIT NONE

      INTEGER, PARAMETER  :: N=1000000000
      INTEGER(8) :: total_sum=0, local_sum
      INTEGER i, tid, omp_get_thread_num

      CALL omp_set_num_threads(4)
!$omp parallel private(tid, local_sum)
      local_sum = 0
      tid = omp_get_thread_num()
      !$OMP DO
      DO i=1, N
            local_sum = local_sum + i
      END DO
      !$OMP ATOMIC
      total_sum = total_sum + local_sum
!$omp end parallel
      PRINT *, "sum =", total_sum

END
```

```
$ gfortran -fopenmp -o use_local_sum.x use_local_sum.f90
$ time ./use_local_sum.x
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 1000000000

int main()
{
      long sum=0, local_sum, i, tid;
      omp_set_num_threads(4);

#pragma omp parallel private(tid, local_sum)
{
      local_sum = 0;
      tid = omp_get_thread_num();

      #pragma omp for
      for(i=1; i<=N; i++)
            local_sum += i;
      #pragma omp atomic
      sum +=  local_sum;
}
      printf("sum = %ld\n", sum);
      return 0;
}
```

```
$ gcc -fopenmp -o use_local_sum.x use_local_sum.c
$ time ./use_local_sum.x
```

Data Dependency (1/2)

Fortran	C

DO time=0, max_time-1
  DO j=1, n-2
  DO i=1, n-2
  h(i, j) = 0.25 * (h(i-1, j) + h(i+1, j) + h(i, j-1) + h(i, j+1)
  END DO
  END DO
END DO
	for (time=0; time < max_time; time++) {
	  for (i=1; i<n-1; i++)
	  for (j=1; j<n-1; j++)
	  h[i][j] = 0.25 * (h[i-1][j] + h[i+1][j] +
	  h[i][j-1] + h[i][j+1]);
	}

DO time=0, max_time-1
!$omp parallel DO
  DO j=1, n-2
  DO i=1, n-2
  newh(i, j) =0.25*(h(i-1, j) + h(i+1, j) + h(i, j-1) + h(i, j+1)
  END DO
  END DO
!$omp parallel DO
  DO j=1, n-2
  DO i=1, n-2
  h(i, j) = newh(i, j)
  END DO
  END DO
END DO
	for(time=0; time < max_time; time++) {
	  #pragma omp parallel for private(j)
	  for(i=1; i<n-1; i++)
	  for(j=1; j<n-1; j++)
	  newh[i][j] = 0.25 * (h[i-1][j] + h[i+1][j] +
	  h[i][j-1] + h[i][j+1]);
	  #pragma omp parallel for private(j)
	  for(i=1; i<n-1; i++)
	  for(j=1; j<n-1; j++)
	  h[i][j] = newh[i][j];
	}

- Heat Distribution Problem

$$h_{i,j} = \frac{h_{i-1,j} + h_{i+1,j} + h_{i,j-1} + h_{i,j+1}}{4}$$

Data Dependency (2/2)

Fortran	C

PROGRAM fibonacci
  IMPLICIT NONE
 	
  INTEGER, PARAMETER :: N=10
  INTEGER :: i
  DO i=0, N
  PRINT *, “fibonacci F(“, i, “) =“, fibon(i)
  END DO
 	
  CONTAINS
  recursive INTEGER function fibon(n) RESULT(fib)
  INTEGER :: n
  IF (n < 2) THEN
  fib = n
  ELSE
  fib = fibon(n-1) + fibon(n-2)
  END IF
  END function
END PROGRAM
	#include <stdio.h>
	#define N 10
	
	int fibon(int n)
	{
	  if (n < 2) return n;
	  return (fibon(n-1) + fibon(n-2));
	}
	
	int main()
	{
	  int i;
	  for (i=0; i<=N; i++)
	  printf(“fibonacci F(%d) = %d\n”, i, fibon(i));
	
	  return 0;
	}

$ gfortran -fopenmp -o fibonacci.x fibonacci.f90 $ ./fibonacci.x	$ gcc -fopenmp -o fibonacci.x fibonacci.c $ ./fibonacci.x

- Fibonacci
  - F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1, n = 2,3,4,…

Data Dependency (2/2)

Fortran	C

PROGRAM fibonacci
IMPLICIT NONE
 	
INTEGER, PARAMETER :: N=10
INTEGER :: i
INTEGER, dimension(0:N) :: fibon
 	
fibon(0) = 0
fibon(1) = 1
DO i=2, N
fibon(i) = fibon(i-1) + fibon(i-2)
END DO
 	
DO i=0, N
PRINT *, “fibonacci F(“, i, “) =“, fibon(i)
END DO
 	
END
	#include <stdio.h>
	#define N 10
	
	int main()
	{
	int i, fibon[N+1];
	fibon[0] = 0;
	fibon[1] = 1;
	
	for(i=2; i<=N; i++) {
	fibon[i] = fibon[i-1] + fibon[i-2];
	}
	
	for(i=0; i<=N; i++)
	printf(“fibonacci F(%d) = %d\n”, i, fibon[i]);
	
	return 0;
	}

$ gfortran -fopenmp -o fibonacci2.x fibonacci2.f90 $ ./fibonacci2.x	$ gcc -fopenmp -o fibonacci2.x fibonacci.c $ ./fibonacci2.x

- Fibonacci
  - F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1, n = 2,3,4,…

Data Dependency (2/2)

Fortran	C

PROGRAM fibonacci
IMPLICIT NONE
 	
INTEGER, PARAMETER :: N=10
INTEGER i, fibon(0:N)
DOUBLE PRECISION :: alpha, beta, c
 	
alpha=0.5*(1.0+sqrt(REAL(5)))
beta=0.5*(1.0-sqrt(REAL(5)))
c=1/sqrt(REAL(5))
 	
!$omp parallel DO num_threads(2)
DO i=0, N
fibon(i) = c * (alpha**i-beta**i)
END DO
 	
DO i=0, N
PRINT *, “fibonacci F(“, i, “) =“, fibon(i)
END DO
 	
END
	#include <stdio.h>
	#include <math.h>
	#include <omp.h>
	#define N 10
	
	int main()
	{
	int i, fibon[N+1];
	double alpha = 0.5 * (1.0+sqrt(5));
	double beta = 0.5 * (1.0-sqrt(5));
	double c = 1 / sqrt(5);
	
	#pragma omp parallel for num_threads(2)
	for(i=0; i<=10; i++)
	fibon[i] = c * (pow(alpha,i)-pow(beta,i));
	
	for(i=0; i<=N; i++)
	printf(“fibonacci F(%d) = %d\n”, i, fibon[i]);
	
	return 0;
	}

$ gfortran -fopenmp -o fibonacci3.x fibonacci3.f90 $ ./fibonacci3.x	$ gcc -fopenmp -o fibonacci3.x fibonacci3.c -lm $ ./fibonacci3.x

- Fibonacci
  - F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1, n = 2,3,… $F(n) = \frac{1}{\sqrt{5}}\left\{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right\}1$

Hands-on

1. Matrix Multiplication
2. LU Decomposition (Optional)
3. Fibonacci (Optional)

Ex1. Matrix Multiplication

```
// KISTI educational Matrix
// Multiplication problem
// Copyright (c) 2012 KISTI
// Supercomputing Center

#include <stdio.h>
#define N 10

int main()
{
  int i,j,k;
  int a[N][N], b[N][N], c[N][N];

  for(i=0; i<N; i++) {
    for(j=0; j<i; j++) {
      a[i][j] = b[i][j] = c[i][j] = 0;
    }
    for(j=i; j<N; j++) {
      a[i][j] = i+j+1;
      b[i][j] = i+j+2;
      c[i][j] = 0;
    }
  }
  for(i=0; i<N; i++)
    for(j=0; j<N; j++)
      for(k=0; k<N; k++)
        c[i][j] += a[i][k] * b[k][j];
```

```
  printf(''matrix a\n'');
  for(i=0; i<N; i++) {
    for(j=0; j<N; j++)
      printf(''%4d '', a[i][j]);
    printf(''\n'');
  }
  printf(''matrix b\n'');
  for(i=0; i<N; i++) {
    for(j=0; j<N; j++)
      printf(''%4d '', b[i][j]);
    printf(''\n'');
  }
  printf(''matrix c\n'');
  for(i=0; i<N; i++) {
    for(j=0; j<N; j++)
      printf(''%4d '', c[i][j]);
    printf(''\n'');
  }
}
```

a

```
1   2   3   4   5   6   7   8   9  10
0   3   4   5   6   7   8   9  10  11
0   0   5   6   7   8   9  10  11  12
0   0   0   7   8   9  10  11  12  13
0   0   0   0   9  10  11  12  13  14
0   0   0   0   0  11  12  13  14  15
0   0   0   0   0   0  13  14  15  16
0   0   0   0   0   0   0  15  16  17
0   0   0   0   0   0   0   0  17  18
0   0   0   0   0   0   0   0   0  19
```

b

```
2   3   4   5   6   7   8   9  10  11
0   4   5   6   7   8   9  10  11  12
0   0   6   7   8   9  10  11  12  13
0   0   0   8   9  10  11  12  13  14
0   0   0   0  10  11  12  13  14  15
0   0   0   0   0  12  13  14  15  16
0   0   0   0   0   0  14  15  16  17
0   0   0   0   0   0   0  16  17  18
0   0   0   0   0   0   0   0  18  19
0   0   0   0   0   0   0   0   0  20
```

c

```
2   11   32   70  130  217  336  492  690  935
0   12   39   86  158  260  397  574  796 1068
0    0   30   83  164  278  430  625  868 1164
0    0    0   56  143  266  430  640  901 1218
0    0    0    0   90  219  392  614  890 1225
0    0    0    0    0  132  311  542  830 1180
0    0    0    0    0    0  182  419  716 1078
0    0    0    0    0    0    0  240  543  914
0    0    0    0    0    0    0    0  306  683
0    0    0    0    0    0    0    0    0  380
```

Ex1. Matrix Multiplication

```
// KISTI educational Matrix
// Multiplication problem
// Copyright (c) 2012 KISTI
// Supercomputing Center

PROGRAM ex1
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=9
  INTEGER :: i, j, k, a(0:N,0:N), b(0:N,0:N),
       c(0:N,0:N)
  a = 0
  b = 0
  c =0
  DO j=0, N
    DO i=0, j
      a(i, j) = i + j + 1
      b(i, j) = i + j + 2
    END DO
  END DO

  DO j=0, N
    DO i=0, N
      DO k=0, N
        c(i, j) = c(i, j) + a(i, k) * b(k, j)
      END DO
    END DO
  END DO
```

```
  PRINT *, "matrix a"
  DO i=0, N
    WRITE(*, 100) (a(i, j), j=0, N)
  END DO

  PRINT *, "matrix b"
  DO i=0, N
    WRITE(*, 100) (b(i, j), j=0, N)
  END DO

  PRINT *, "matrix c"
  DO i=0, N
    WRITE(*, 100) (c(i, j), j=0, N)
  END DO

100 FORMAT(10(i4))

END
```

a

```
1   2   3   4   5   6   7   8   9  10
0   3   4   5   6   7   8   9  10  11
0   0   5   6   7   8   9  10  11  12
0   0   0   7   8   9  10  11  12  13
0   0   0   0   9  10  11  12  13  14
0   0   0   0   0  11  12  13  14  15
0   0   0   0   0   0  13  14  15  16
0   0   0   0   0   0   0  15  16  17
0   0   0   0   0   0   0   0  17  18
0   0   0   0   0   0   0   0   0  19
```

b

```
2   3   4   5   6   7   8   9  10  11
0   4   5   6   7   8   9  10  11  12
0   0   6   7   8   9  10  11  12  13
0   0   0   8   9  10  11  12  13  14
0   0   0   0  10  11  12  13  14  15
0   0   0   0   0  12  13  14  15  16
0   0   0   0   0   0  14  15  16  17
0   0   0   0   0   0   0  16  17  18
0   0   0   0   0   0   0   0  18  19
0   0   0   0   0   0   0   0   0  20
```

c

```
2  11  32  70 130 217 336 492 690 935
0  12  39  86 158 260 397 574 796 1068
0   0  30  83 164 278 430 625 868 1164
0   0   0  56 143 266 430 640 901 1218
0   0   0   0  90 219 392 614 890 1225
0   0   0   0   0 132 311 542 830 1180
0   0   0   0   0   0 182 419 716 1078
0   0   0   0   0   0   0 240 543 914
0   0   0   0   0   0   0   0 306 683
0   0   0   0   0   0   0   0   0 380
```

Ex1. Matrix Multiplication

```
// KISTI educational Matrix
// Multiplication problem
// Copyright (c) 2012 KISTI
// Supercomputing Center

#include <stdio.h>
#define N 10

int main()
{
  int i,j,k;
  int a[N][N], b[N][N], c[N][N];

  for(i=0; i<N; i++) {
    for(j=0; j<i; j++) {
      a[i][j] = b[i][j] = c[i][j] = 0;
    }
    for(j=i; j<N; j++) {
      a[i][j] = i+j+1;
      b[i][j] = i+j+2;
      c[i][j] = 0;
    }
  }
  for(i=0; i<N; i++)
    for(j=i; j<N; j++)
      for(k=i; k<j+1; k++)
        c[i][j] += a[i][k] * b[k][j];
```

a

1	2	3	4	5	6	7	8	9	10
0	3	4	5	6	7	8	9	10	11
0	0	5	6	7	8	9	10	11	12
0	0	0	7	8	9	10	11	12	13
0	0	0	0	9	10	11	12	13	14
0	0	0	0	0	11	12	13	14	15
0	0	0	0	0	0	13	14	15	16
0	0	0	0	0	0	0	15	16	17
0	0	0	0	0	0	0	0	17	18
0	0	0	0	0	0	0	0	0	19

b

2	3	4	5	6	7	8	9	10	11
0	4	5	6	7	8	9	10	11	12
0	0	6	7	8	9	10	11	12	13
0	0	0	8	9	10	11	12	13	14
0	0	0	0	10	11	12	13	14	15
0	0	0	0	0	12	13	14	15	16
0	0	0	0	0	0	14	15	16	17
0	0	0	0	0	0	0	16	17	18
0	0	0	0	0	0	0	0	18	19
0	0	0	0	0	0	0	0	0	20

i=4, j=4

```
for(k=4; k<5; k++)
   c[4][4] = a[4][k] * b[k][4];
```

c

2	11	32	70	130	217	336	492	690	935
0	12	39	86	158	260	397	574	796	1068
0	0	30	83	164	278	430	625	868	1164
0	0	0	56	143	266	430	640	901	1218
0	0	0	0	90	219	392	614	890	1225
0	0	0	0	0	132	311	542	830	1180
0	0	0	0	0	0	182	419	716	1078
0	0	0	0	0	0	0	240	543	914
0	0	0	0	0	0	0	0	306	683
0	0	0	0	0	0	0	0	0	380

Ex1. Matrix Multiplication

```
// KISTI educational Matrix
// Multiplication problem
// Copyright (c) 2012 KISTI
// Supercomputing Center

PROGRAM ex1
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=9
  INTEGER :: i, j, k, a(0:N,0:N), b(0:N,0:N),
      c(0:N,0:N)
  a = 0
  b = 0
  c =0
  DO j=0, N
    DO i=0, j
      a(i, j) = i + j + 1
      b(i, j) = i + j + 2
    END DO
  END DO

  DO j=0, N
    DO i=0, j
      DO k=i, j
        c(i, j) = c(i, j) + a(i, k) * b(k, j)
      END DO
    END DO
  END DO
```

a

1	2	3	4	5	6	7	8	9	10
0	3	4	5	6	7	8	9	10	11
0	0	5	6	7	8	9	10	11	12
0	0	0	7	8	9	10	11	12	13
0	0	0	0	9	10	11	12	13	14
0	0	0	0	0	11	12	13	14	15
0	0	0	0	0	0	13	14	15	16
0	0	0	0	0	0	0	15	16	17
0	0	0	0	0	0	0	0	17	18
0	0	0	0	0	0	0	0	0	19

b

2	3	4	5	6	7	8	9	10	11
0	4	5	6	7	8	9	10	11	12
0	0	6	7	8	9	10	11	12	13
0	0	0	8	9	10	11	12	13	14
0	0	0	0	10	11	12	13	14	15
0	0	0	0	0	12	13	14	15	16
0	0	0	0	0	0	14	15	16	17
0	0	0	0	0	0	0	16	17	18
0	0	0	0	0	0	0	0	18	19
0	0	0	0	0	0	0	0	0	20

```
i=4, j=4
for(k=4; k<5; k++)
   c[4][4] = a[4][k] * b[k][4];
```

c

2	11	32	70	130	217	336	492	690	935
0	12	39	86	158	260	397	574	796	1068
0	0	30	83	164	278	430	625	868	1164
0	0	0	56	143	266	430	640	901	1218
0	0	0	0	90	219	392	614	890	1225
0	0	0	0	0	132	311	542	830	1180
0	0	0	0	0	0	182	419	716	1078
0	0	0	0	0	0	0	240	543	914
0	0	0	0	0	0	0	0	306	683
0	0	0	0	0	0	0	0	0	380

Ex1. Matrix Multiplication

```
// KISTI educational Matrix
// Multiplication problem
// Copyright (c) 2012 KISTI
// Supercomputing Center

#include <stdio.h>
#define N 10

int main()
{
  int i,j,k;
  int a[N][N], b[N][N], c[N][N];

  for(i=0; i<N; i++) {
    for(j=0; j<i; j++) {
      a[i][j] = b[i][j] = c[i][j] = 0;
    }
    for(j=i; j<N; j++) {
      a[i][j] = i+j+1;
      b[i][j] = i+j+2;
      c[i][j] = 0;
    }
  }
  for(i=0; i<N; i++)
    for(j=i; j<N; j++)
      for(k=i; k<j+1; k++)
        c[i][j] += a[i][k] * b[k][j];
```

a

1	2	3	4	5	6	7	8	9	10
0	3	4	5	6	7	8	9	10	11
0	0	5	6	7	8	9	10	11	12
0	0	0	7	8	9	10	11	12	13
0	0	0	0	9	10	11	12	13	14
0	0	0	0	0	11	12	13	14	15
0	0	0	0	0	0	13	14	15	16
0	0	0	0	0	0	0	15	16	17
0	0	0	0	0	0	0	0	17	18
0	0	0	0	0	0	0	0	0	19

b

2	3	4	5	6	7	8	9	10	11
0	4	5	6	7	8	9	10	11	12
0	0	6	7	8	9	10	11	12	13
0	0	0	8	9	10	11	12	13	14
0	0	0	0	10	11	12	13	14	15
0	0	0	0	0	12	13	14	15	16
0	0	0	0	0	0	14	15	16	17
0	0	0	0	0	0	0	16	17	18
0	0	0	0	0	0	0	0	18	19
0	0	0	0	0	0	0	0	0	20

i=4, j=5

```
for(k=4; k<6; k++)
   c[4][5] = a[4][k] * b[k][5];
```

c

2	11	32	70	130	217	336	492	690	935
0	12	39	86	158	260	397	574	796	1068
0	0	30	83	164	278	430	625	868	1164
0	0	0	56	143	266	430	640	901	1218
0	0	0	0	90	219	392	614	890	1225
0	0	0	0	0	132	311	542	830	1180
0	0	0	0	0	0	182	419	716	1078
0	0	0	0	0	0	0	240	543	914
0	0	0	0	0	0	0	0	306	683
0	0	0	0	0	0	0	0	0	380

Ex1. Matrix Multiplication

```
// KISTI educational Matrix
// Multiplication problem
// Copyright (c) 2012 KISTI
// Supercomputing Center

PROGRAM ex1
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=9
  INTEGER :: i, j, k, a(0:N,0:N), b(0:N,0:N),
      c(0:N,0:N)
  a = 0
  b = 0
  c =0
  DO j=0, N
    DO i=0, j
      a(i, j) = i + j + 1
      b(i, j) = i + j + 2
    END DO
  END DO

  DO j=0, N
    DO i=0, j
      DO k=i, j
        c(i, j) = c(i, j) + a(i, k) * b(k, j)
      END DO
    END DO
  END DO
```

a

1	2	3	4	5	6	7	8	9	10
0	3	4	5	6	7	8	9	10	11
0	0	5	6	7	8	9	10	11	12
0	0	0	7	8	9	10	11	12	13
0	0	0	0	9	10	11	12	13	14
0	0	0	0	0	11	12	13	14	15
0	0	0	0	0	0	13	14	15	16
0	0	0	0	0	0	0	15	16	17
0	0	0	0	0	0	0	0	17	18
0	0	0	0	0	0	0	0	0	19

b

2	3	4	5	6	7	8	9	10	11
0	4	5	6	7	8	9	10	11	12
0	0	6	7	8	9	10	11	12	13
0	0	0	8	9	10	11	12	13	14
0	0	0	0	10	11	12	13	14	15
0	0	0	0	0	12	13	14	15	16
0	0	0	0	0	0	14	15	16	17
0	0	0	0	0	0	0	16	17	18
0	0	0	0	0	0	0	0	18	19
0	0	0	0	0	0	0	0	0	20

```
i=4, j=5
for(k=4; k<6; k++)
   c[4][5] = a[4][k] * b[k][5];
```

c

2	11	32	70	130	217	336	492	690	935
0	12	39	86	158	260	397	574	796	1068
0	0	30	83	164	278	430	625	868	1164
0	0	0	56	143	266	430	640	901	1218
0	0	0	0	90	219	392	614	890	1225
0	0	0	0	0	132	311	542	830	1180
0	0	0	0	0	0	182	419	716	1078
0	0	0	0	0	0	0	240	543	914
0	0	0	0	0	0	0	0	306	683
0	0	0	0	0	0	0	0	0	380

Ex1. Matrix Multiplication

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

#include <stdio.h>
#define N 2000

int main()
{
  int i,j,k;
  int a[N][N], b[N][N], c[N][N];

  for(i=0; i<N; i++) {
    for(j=0; j<i; j++) {
      a[i][j] = b[i][j] = c[i][j] = 0;
    }
    for(j=i; j<N; j++) {
      a[i][j] = i+j+1;
      b[i][j] = i+j+2;
      c[i][j] = 0;
    }
  }
```

```
 for(i=0; i<N; i++)
     for(j=0; j<N; j++)
       for(k=0; k<N; k++)
         c[i][j] += a[i][k] * b[k][j];
}
```

```
$ gcc –o mm0.x mat_mul.c
$ time ./mm0.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm0.x
```

Ex1. Matrix Multiplication

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

PROGRAM ex1
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=1999
  INTEGER :: i, j, k, a(0:N,0:N), b(0:N,0:N), c(0:N,0:N)
  a = 0
  b = 0
  c =0
  DO j=0, N
    DO i=0, j
      a(i, j) = i + j + 1
      b(i, j) = i + j + 2
    END DO
  END DO

  DO j=0, N
    DO i=0, N
      DO k=0, N
        c(i, j) = c(i, j) + a(i, k) * b(k, j)
      END DO
    END DO
  END DO
```

```
$ gfortran –o mm0.x mat_mul.f90
$ time ./mm0.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm0.x
```

Ex1. Matrix Multiplication

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

#include <stdio.h>
#define N 2000

int main()
{
  int i,j,k;
  int a[N][N], b[N][N], c[N][N];

  for(i=0; i<N; i++) {
    for(j=0; j<i; j++) {
      a[i][j] = b[i][j] = c[i][j] = 0;
    }
    for(j=i; j<N; j++) {
      a[i][j] = i+j+1;
      b[i][j] = i+j+2;
      c[i][j] = 0;
    }
  }
```

```
 for(i=0; i<N; i++)
     for(j=i; j<N; j++)
       for(k=i; k<j+1; k++)
         c[i][j] += a[i][k] * b[k][j];
}
```

```
$ gcc –o mm1.x mat_mul.c
$ time ./mm1.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm1.x
```

Ex1. Matrix Multiplication

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

PROGRAM ex1
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=1999
  INTEGER :: i, j, k, a(0:N,0:N), b(0:N,0:N), c(0:N,0:N)
  a = 0
  b = 0
  c =0
  DO j=0, N
    DO i=0, j
      a(i, j) = i + j + 1
      b(i, j) = i + j + 2
    END DO
  END DO

  DO j=0, N
    DO i=0, j
      DO k=i, j
        c(i, j) = c(i, j) + a(i, k) * b(k, j)
      END DO
    END DO
  END DO
```

```
$ gcc –o mm1.x mat_mul.c
$ time ./mm1.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm1.x
```

Ex1. Matrix Multiplication (Apply OpenMP)

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center
#include <stdio.h>
#define N 2000
int main()
{
  int i,j,k;
  int a[N][N], b[N][N], c[N][N];

  #pragma omp parallel for ???
  for(i=0; i<N; i++) {
    #pragma omp sections ???
    for(j=0; j<i; j++) {
      a[i][j] = b[i][j] = c[i][j] = 0;
    }
    for(j=i; j<N; j++) {
      a[i][j] = i+j+1;
      b[i][j] = i+j+2;
      c[i][j] = 0;
    }
  }
```

```
#pragma omp parallel for ???
  for(i=0; i<N; i++)
    for(j=i; j<N; j++)
      for(k=i; k<j+1; k++)
        c[i][j] += a[i][k] * b[k][j];
}
```

```
$ export OMP_NUM_THREADS=4
$ gcc –o mm2.x mat_mul.c
$ time ./mm2.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm2.x
```

Ex1. Matrix Multiplication (Apply OpenMP)

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

PROGRAM ex1
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=1999
  INTEGER :: i, j, k, a(0:N,0:N), b(0:N,0:N), c(0:N,0:N)
  a = 0
  b = 0
  c =0

!$omp parallel do???
  DO j=0, N
    DO i=0, j
      a(i, j) = i + j + 1
      b(i, j) = i + j + 2
    END DO
  END DO
```

```
!$omp parallel do???
  DO j=0, N
    DO i=0, j
      DO k=i, j
        c(i, j) = c(i, j) + a(i, k) * b(k, j)
      END DO
    END DO
  END DO
```

```
$ export OMP_NUM_THREADS=4
$ gcc –o mm2.x mat_mul.c
$ time ./mm2.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm2.x
```

Ex1. Matrix Multiplication (Apply OpenMP)

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center
#include <stdio.h>
#define N 2000
int main()
{
  int i,j,k;
  int a[N][N], b[N][N], c[N][N];

  #pragma omp parallel for private(j)
  for(i=0; i<N; i++) {
    for(j=0;j j<i; j++) {
      a[i][j] = b[i][j] = c[i][j] = 0;
    }
    for(j=i; j<N; j++) {
      a[i][j] = i+j+1;
      b[i][j] = i+j+2;
      c[i][j] = 0;
    }
  }
```

```
#pragma omp parallel for private(j,k)
  for(i=0; i<N; i++)
    for(j=i; j<N; j++)
      for(k=i; k<j+1; k++)
        c[i][j] += a[i][k] * b[k][j];
}
```

```
$ export OMP_NUM_THREADS=4
$ gcc –o mm3.x mat_mul.c
$ time ./mm3.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm3.x
```

Ex1. Matrix Multiplication (Apply OpenMP)

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

PROGRAM ex1
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=1999
  INTEGER :: i, j, k, a(0:N,0:N), b(0:N,0:N), c(0:N,0:N)
  a = 0
  b = 0
  c =0
!$omp parallel do private(i)
  DO j=0, N
    DO i=0, j
      a(i, j) = i + j + 1
      b(i, j) = i + j + 2
    END DO
  END DO
```

```
!$omp parallel do private(i,k)
  DO j=0, N
    DO i=0, j
      DO k=i, j
        c(i, j) = c(i, j) + a(i, k) * b(k, j)
      END DO
    END DO
  END DO
```

```
$ export OMP_NUM_THREADS=4
$ gcc –o mm3.x mat_mul.c
$ time ./mm3.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm3.x
```

Ex1. Matrix Multiplication (Apply OpenMP)

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center
#include <stdio.h>
#define N 2000
int main()
{
  int i,j,k;
  int a[N][N], b[N][N], c[N][N];

  #pragma omp parallel for private(j)
  for(i=0; i<N; i++) {
    for(j=0;j j<i; j++) {
      a[i][j] = b[i][j] = c[i][j] = 0;
    }
    for(j=i; j<N; j++) {
      a[i][j] = i+j+1;
      b[i][j] = i+j+2;
      c[i][j] = 0;
    }
  }
```

```
#pragma omp parallel for private(j,k) schedule(dynamic,5)
  for(i=0; i<N; i++)
    for(j=i; j<N; j++)
      for(k=i; k<j+1; k++)
        c[i][j] += a[i][k] * b[k][j];
}
```

```
$ export OMP_NUM_THREADS=4
$ gcc –o mm4.x mat_mul.c
$ time ./mm4.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm4.x
```

Ex1. Matrix Multiplication (Apply OpenMP)

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

PROGRAM ex1
  IMPLICIT NONE
  INTEGER, PARAMETER :: N=1999
  INTEGER :: i, j, k, a(0:N,0:N), b(0:N,0:N), c(0:N,0:N)
  a = 0
  b = 0
  c =0
!$omp parallel do private(i)
  DO j=0, N
    DO i=0, j
      a(i, j) = i + j + 1
      b(i, j) = i + j + 2
    END DO
  END DO
```

```
!$omp parallel do private(i,k) schedule(dynamic,5)
  DO j=0, N
    DO i=0, j
      DO k=i, j
        c(i, j) = c(i, j) + a(i, k) * b(k, j)
      END DO
    END DO
  END DO
```

```
$ export OMP_NUM_THREADS=4
$ gcc –o mm4.x mat_mul.c
$ time ./mm4.x
Segmentation fault (core dumped)
$ ulimit –s 512000
$ time ./mm4.x
```

Ex2. LU Decomposition - C (Serial)

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

#include <stdio.h>
#define N 5

void LU_decomp(double A[N][N])
{
   int i, j, k;
   double ajj;
   for(j = 0; j < N-1; j++) {
      ajj = A[j][j];
      if( ajj != 0.0 ) {
         for(i=j+1; i<N; i++) {
            A[i][j] /= ajj;
            for(k=j+1; k<N; k++)
               A[i][k] -= (A[i][j] * A[j][k]);
         }
      }
   }
}
```

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

int main()
{
      double A[N][N];
      int i;
      for(i=0; i<N; i++)
            A[i][j] = rand() % 100 + 5.0;

      for(i=0; i<N; i++) {
         for(j=0; j<N; j++)
            printf("%lf ", A[i][j]);
         printf("\n");
      }
      printf("\n");
      LU_decomp(A);
      for(i=0; i<N; i++) {
         for(j=0; j<N; j++)
            printf("%lf ", A[i][j]);
         printf("\n");
      }
}
```

Ex2. LU Decomposition - Fortran(Serial)

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

PROGRAM LU
IMPLICIT NONE

  INTEGER,PARAMETER :: N=5
  INTEGER :: i, j
  DOUBLE PRECISION :: A(0:N-1, 0:N-1)

  DO i=0, N-1
    CALL random_number( A(i, j) )
    A(i, j) = nint(A(i, j) * 100) + 5.0
  END DO
  CALL LU_decomp(A, N);
  DO i=0, N-1
    WRITE(*, 100) (c(i, j), j=0, N-1)
  END DO

100 FORMAT(10(F8.3))
END
```

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

SUBROUTINE LU_decomp(A, N)
   INTEGER :: i, j, k, N;
   DOUBLE PRECISION :: aji, A(0:N-1, 0:N-1)
   DO j=0, N-2
     ajj = A(j, j);
     if( ajj != 0.0 ) THEN
        DO i=j+1, N-1
          A(i, j) = A(i, j) / ajj;
          DO k=j+1, N-1
            A(i, k) = A(i, k) - ( A(i, j) * A(j, k) );
          END DO
        END DO
     END IF
   END DO
END SUBROUTINE
```

Ex2. LU Decomposition - C(Parallel) Step 1

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

#include <stdio.h>
#define N 5

void LU_decomp(double A[N][N])
{
   int i, j, k;
   double ajj;
   #pragma omp parallel for private(i,j,k,ajj)
   for(j = 0; j < N-1; j++) {
      ajj = A[j][j];
      if( ajj != 0.0 ) {
         for(i=j+1; i<N; i++) {
            A[i][j] /= ajj;
            for(k=j+1; k<N; k++)
               A[i][k] -= (A[i][j] * A[j][k]);
         }
      }
   }
}
```

No problem ???

```
j=0

i=1; i<N
  k=1; k<N
    A[i][k] -= (A[i][0]*A[0][k])
```

```
j=2

i=3; i<N
  k=3; k<N
    A[i][k] -= (A[i][2]*A[2][k])
```

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

int main()
{
      double A[N][N];
      int i;
      for(i=0; i<N; i++)
            A[i][j] = rand() % 100 + 5.0;

      for(i=0; i<N; i++) {
         for(j=0; j<N; j++)
            printf("%lf ", A[i][j]);
         printf("\n");
      }
      printf("\n");
      LU_decomp(A);
      for(i=0; i<N; i++) {
         for(j=0; j<N; j++)
            printf("%lf ", A[i][j]);
         printf("\n");
      }
}
```

Ex2. LU Decomposition - Fortran(Parallel) Step1

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

PROGRAM LU
IMPLICIT NONE

  INTEGER,PARAMETER :: N=5
  INTEGER :: i, j
  DOUBLE PRECISION :: A(0:N-1, 0:N-1)

  DO i=0, N-1
    CALL random_number( A(i, j) )
    A(i, j) = nint(A(i, j) * 100) + 5.0
  END DO
  CALL LU_decomp(A, N);
  DO i=0, N-1
    WRITE(*, 100) (c(i, j), j=0, N-1)
  END DO

100 FORMAT(10(F8.3))
END
```

```
j=0

do i=1, N-1
  do k=1, N-1
    A(i,k) = A(i,k) – ( A(i,0)*A(0,k) )
```

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

SUBROUTINE LU_decomp(A, N)
   INTEGER :: i, j, k, N;
   DOUBLE PRECISION :: ajj, A(0:N-1, 0:N-1)
!$omp parallel for private(i,j,k,ajj)
   DO j=0, N-2
     ajj = A(j, j);
     if( ajj != 0.0 ) THEN
       DO i=j+1, N-1
         A(i, j) = A(i, j) / ajj;
         DO k=j+1, N-1
           A(i, k) = A(i, k) - ( A(i, j) * A(j, k) );
         END DO
       END DO
     END IF
   END DO
!$omp end parallel for
END SUBROUTINE
```

No problem ???

```
j=2

do i=3, N-1
  do k=3, N-1
    A(i,k) = A(i,k) – ( A(i,2)*A(2,k) )
```

Ex2. LU Decomposition - C(Parallel) Step 2

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

#include <stdio.h>
#define N 5

void LU_decomp(double A[N][N])
{
   int i, j, k;
   double ajj;
   #pragma omp parallel for private(i,j,k,ajj)
   for(j = 0; j < N-1; j++) {
      ajj = A[j][j];
      if( ajj != 0.0 ) {
         for(i=j+1; i<N; i++) {
            A[i][j] /= ajj;
            for(k=j+1; k<N; k++)
               #pragma omp atomic
               A[i][k] -= (A[i][j] * A[j][k]);
         }
      }
   }
}
```

No problem ???

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

int main()
{
      double A[N][N];
      int i;
      for(i=0; i<N; i++)
            A[i][j] = rand() % 100 + 5.0;

      for(i=0; i<N; i++) {
         for(j=0; j<N; j++)
            printf("%lf ", A[i][j]);
         printf("\n");
      }
      printf("\n");
      LU_decomp(A);
      for(i=0; i<N; i++) {
         for(j=0; j<N; j++)
            printf("%lf ", A[i][j]);
         printf("\n");
      }
}
```

Ex2. LU Decomposition - Fortran(Parallel) Step2

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

PROGRAM LU
IMPLICIT NONE

  INTEGER,PARAMETER :: N=5
  INTEGER :: i, j
  DOUBLE PRECISION :: A(0:N-1, 0:N-1)

  DO i=0, N-1
    CALL random_number( A(i, j) )
    A(i, j) = nint(A(i, j) * 100) + 5.0
  END DO
  CALL LU_decomp(A, N);
  DO i=0, N-1
    WRITE(*, 100) (c(i, j), j=0, N-1)
  END DO

100 FORMAT(10(F8.3))
END
```

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

SUBROUTINE LU_decomp(A, N)
   INTEGER :: i, j, k, N;
   DOUBLE PRECISION :: ajj, A(0:N-1, 0:N-1)
!$omp parallel for private(i,j,k,ajj)
   DO j=0, N-2
     ajj = A(j, j);
     if( ajj != 0.0 ) THEN
       DO i=j+1, N-1
         A(i, j) = A(i, j) / ajj;
         DO k=j+1, N-1
           !$omp atomic
           A(i, k) = A(i, k) - ( A(i, j) * A(j, k) );
         END DO
       END DO
     END IF
   END DO
!$omp end parallel for
END SUBROUTINE
```

No problem ???

Ex2. LU Decomposition - C(Parallel) Solution

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

#include <stdio.h>
#define N 5

void LU_decomp(double A[N][N])
{
   int i, j, k;
   double ajj;
   for(j = 0; j < N-1; j++) {
      ajj = A[j][j];
      if( ajj != 0.0 ) {
         #pragma omp parallel for private(i,k)
         for(i=j+1; i<N; i++) {
            A[i][j] /= ajj;
            for(k=j+1; k<N; k++)
               A[i][k] -= (A[i][j] * A[j][k]);
         }
      }
   }
}
```

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

int main()
{
      double A[N][N];
      int i;
      for(i=0; i<N; i++)
            A[i][j] = rand() % 100 + 5.0;

      for(i=0; i<N; i++) {
         for(j=0; j<N; j++)
            printf("%lf ", A[i][j]);
         printf("\n");
      }
      printf("\n");
      LU_decomp(A);
      for(i=0; i<N; i++) {
         for(j=0; j<N; j++)
            printf("%lf ", A[i][j]);
         printf("\n");
      }
}
```

Ex2. LU Decomposition - Fortran(Parallel) Solution

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

PROGRAM LU
IMPLICIT NONE

  INTEGER,PARAMETER :: N=5
  INTEGER :: i, j
  DOUBLE PRECISION :: A(0:N-1, 0:N-1)

  DO i=0, N-1
    CALL random_number( A(i, j) )
    A(i, j) = nint(A(i, j) * 100) + 5.0
  END DO
  CALL LU_decomp(A, N);
  DO i=0, N-1
    WRITE(*, 100) (c(i, j), j=0, N-1)
  END DO

100 FORMAT(10(F8.3))
END
```

```
// KISTI educational Matrix Multiplication problem
// Copyright (c) 2012 KISTI Supercomputing Center

SUBROUTINE LU_decomp(A, N)
   INTEGER :: i, j, k, N;
   DOUBLE PRECISION :: ajj, A(0:N-1, 0:N-1)
   DO j=0, N-2
     ajj = A(j, j);
     if( ajj != 0.0 ) THEN
       !$omp parallel for private(i,k)
       DO i=j+1, N-1
         A(i, j) = A(i, j) / ajj;
         DO k=j+1, N-1
           A(i, k) = A(i, k) - ( A(i, j) * A(j, k) );
         END DO
       END DO
       !$omp end parallel for
     END IF
   END DO
END SUBROUTINE
```

Ex3. Fibonacci

Fortran

```
PROGRAM fibonacci
IMPLICIT NONE
  INTEGER :: result
  result = fibon(MAX)
  PRINT *, “Fibonacci (“, MAX, ”) =“, result

  contains
recursive INTEGER function fibon(n) RESULT(fib)
IMPLICIT NONE
  INTEGER :: n, x, y
  if(n<2) THEN
    fib=n
  else
    x=fibon(n-1);
    y=fibon(n-2);
    fib = x + y
END function
END
```

```
$ gfortran –DMAX=40 –o fibon.x fibonacci.f90
$ ./fibon.x
```

C

```
#include <stdio.h>

int fibon(int n) // f(n) = f(n-1) + f(n-2)
{
  int x, y;
  if(n<2) return n;
  x=fibon(n-1);
  y=fibon(n-2);
  return (x+y);
}

int main()
{
  int result = fibon(MAX);
  printf(“Fibonacci (%d) = %d\n”, MAX, result);
  return 0;
}
```

```
$ gcc –DMAX=40 –o fibon.x fibonacci.c
$ ./fibon.x
```

Ex3. Fibonacci (Solution with Task)

Fortran

```
PROGRAM fibonacci
USE OMP_LIB or include "omp_lib.h"
  INTEGER :: result
!$omp parallel
!$OMP SINGLE
  result = fibon(MAX)
!$omp end SINGLE
!$omp end parallel
  PRINT *, "Fibonacci (", MAX, ") =", result
END

recursive INTEGER function fibon(n) RESULT(fib)
  INTEGER :: n, x, y
  if(n<2) THEN
     fib = n
  else
!$OMP TASK shared(x)
     x=fibon(n-1)
!$omp end TASK
!$OMP TASK shared(y)
     y=fibon(n-2)
!$omp end TASK
!$OMP TASKWAIT
     fib = x + y
  END IF
END function
```

```
$ gfortran –DMAX=40 –o fibon2.x fibonacci2.f90
$ ./fibon2.x
```

C

```
#include <stdio.h>
#include <omp.h>

int fibon(int n) // f(n) = f(n-1) + f(n-2)
{
  int x, y;
  if(n<2) return n;
  #pragma omp task shared(x)
  { x=fibon(n-1); }
  #pragma omp task shared(y)
  { y=fibon(n-2); }
  #pragma omp taskwait
  return (x+y);
}

int main()
{
  int result;
#pragma omp parallel
{
  #pragma omp single
  result = fibon(MAX);
}
  printf("Fibonacci (%d) = %d\n", MAX, result);
  return 0;
}
```

```
$ gcc –DMAX=40 –o fibon2.x fibonacci2.c
$ ./fibon2.x
```

Ex3. Fibonacci (Optimized Solution with Task)

Fortran

```
PROGRAM fibonacci
USE OMP_LIB or include "omp_lib.h"
  INTEGER :: result, fibon
!$omp parallel
!$OMP SINGLE
  result = fibon(MAX)
!$omp end SINGLE
!$omp end parallel
  PRINT *, "Fibonacci (", MAX, ") =", result

  contains
recursive INTEGER function fibon(n) RESULT(fib)
  INTEGER n, x , y
  if(n<2) THEN
    fib = n
  else if(n>30) THEN
    fib = fibon(n-1) + fibon(n-2)
  else
!$OMP TASK shared(x)
    x=fibon(n-1)
!$omp end TASK
!$OMP TASK shared(y)
    y=fibon(n-2)
!$omp end TASK
!$OMP TASKWAIT
     fib = x + y
END function
END
```

C

```
#include <stdio.h>
#include <omp.h>

int fibon(int n) // f(n) = f(n-1) + f(n-2)
{
  int x, y;
  if(n<2) return n;
  if(n<30) return fibon(n-1)+fibon(n-2);
  #pragma omp task shared(x)
  { x=fibon(n-1); }
  #pragma omp task shared(y)
  { y=fibon(n-2); }
  #pragma omp taskwait
  return (x+y);
}

int main()
{
  int result;
#pragma omp parallel
{
  #pragma omp single
  result = fibon(MAX);
}
  printf("Fibonacci (%d) = %d\n", MAX, result);
  return 0;
}
```


Q&A


Thank
you