OpenMP Training Course

Basic

Resources

- http://www.openmp.org
- http://openmpcon.org
- https://hpc-tutorials.llnl.gov/openmp/
- https://events.prace-ri.eu/event/176/contributions/64/attachments/128/209/OpenMP.pdf
- https://github.com/vishalk2/Computational-Fluid-Dynamics-CFD/tree/main/FDM/Elliptic/Stream_Function_Examples/Example_1

차시 별 세부 내용

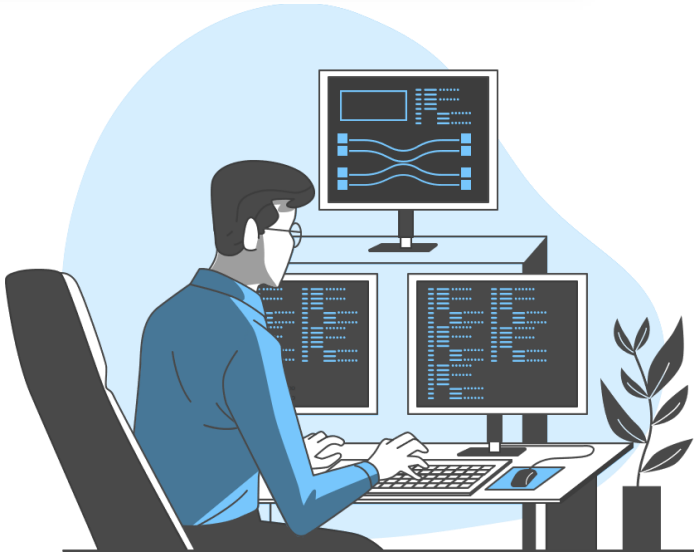
차시	구분	세부 내용
1	교육을 위한 기본 환경 이해	시스템 소개 및 접속
		module을 이용한 실습 환경 설정 및 OpenMP 컴파일 방법 설명
		vi 편집기 소개 및 pbs job scheduler 사용법 설명
2	OpenMP Basic I	OpenMP 소개
		OpenMP의 구성, 조건부 컴파일
		스레드 생성
3	OpenMP Basic II	데이터 유효범위 소개(private, firstprivate, shared)
		코드 설명(데이터 유효범위)
		스레드와 프로세스
4	OpenMP Basic III	루프 병렬화 소개
		루프 병렬화
		코드 설명(inner product 계산)
5	OpenMP Basic IV	동기화 소개
		critical, atomic, barrier
		코드 설명(동기화)
6	OpenMP Basic V	reduction 소개
		Reduction과 동기화 사용의 성능 차이 비교
		코드 설명(factorial)

차시 별 세부 내용

차시	구분	세부 내용
7	Nested parallel	Nested parallel 소개
		코드 설명(nested parallel)
		데이터 유효 범위
8	작업 분할 / 동기화	작업 분할 지시어 소개
		Sections, single, master
		동기화(nowait, ordered)
9	Shcedule / task	OpenMP scheduling
		코드 설명(schedule 사용)
		Task, taskwait
10	OpenMP performance issue	Nested parallel(collapsed)
		Flush, false sharing
		데이터 의존성
11	Hands-On	Pi 계산
		여러 파일 동시에 읽기
		2D FDM
12	Summary	Parallel block, fork-join model
		Parallel loop
		Review

01 교육을 위한 기본환경 이해

1. 시스템 소개 및 접속
2. module을 이용한 실습 환경 설정 및 OpenMP 컴파일 방법
3. vi 편집기 소개 및 pbs job scheduler 사용법 설명



학습 목표

- 윈도우 PC에서 리눅스 서버에 접속하는 방법을 익힌다.
- module 명령을 이용해서 실습 환경을 설정할 줄 안다.
- OpenMP 코드를 컴파일하고 실행시킬 줄 안다.
- vi 편집기의 기본 사용법을 익힌다.
- PBS job scheduler 사용법을 익힌다.

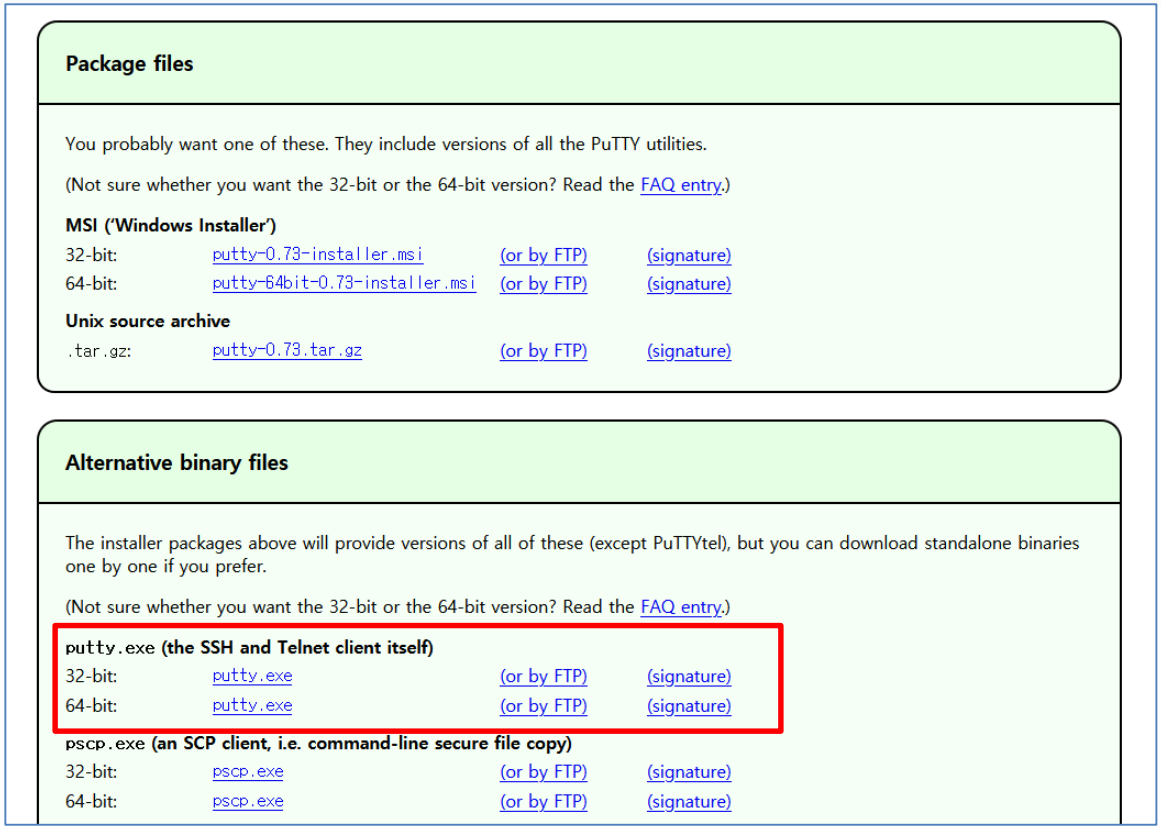
누리온 시스템

- KISTI(한국과학기술정보연구원)에서 운영중인 국가 슈퍼컴퓨터 5호기
- 노드 구성

		호스트 명	CPU Limit	비고
로그인 노드		nurion.ksc.re.kr	20분	▪ ssh/scp 접속 가능 ▪ 컴파일 및 batch 작업제출용 ▪ ftp 접속 불가
Datamover 노드		nurion-dm .ksc.re.kr	-	▪ ssh/scp/sftp 접속 가능 ▪ ftp 접속 가능 ▪ 컴파일 및 작업 제출 불가
계산 노드	KNL	node[0001-8305]	-	▪ PBS 스케줄러를 통해 작업 실행 가능 ▪ 일반사용자 직접 접근 불가
	CPU-Only	cpu[0001-0132]	-	

접속 프로그램 다운로드

- PUTTY :
  - https://putty.org
  - Google ➔ putty 검색



Package files

You probably want one of these. They include versions of all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the FAQ entry.)

MSI ('Windows Installer')

32-bit:	putty-0.73-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.73-installer.msi	(or by FTP)	(signature)

Unix source archive

.tar.gz:	putty-0.73.tar.gz	(or by FTP)	(signature)

Alternative binary files

The installer packages above will provide versions of all of these (except PuTTYtel), but you can download standalone binaries one by one if you prefer.

(Not sure whether you want the 32-bit or the 64-bit version? Read the FAQ entry.)

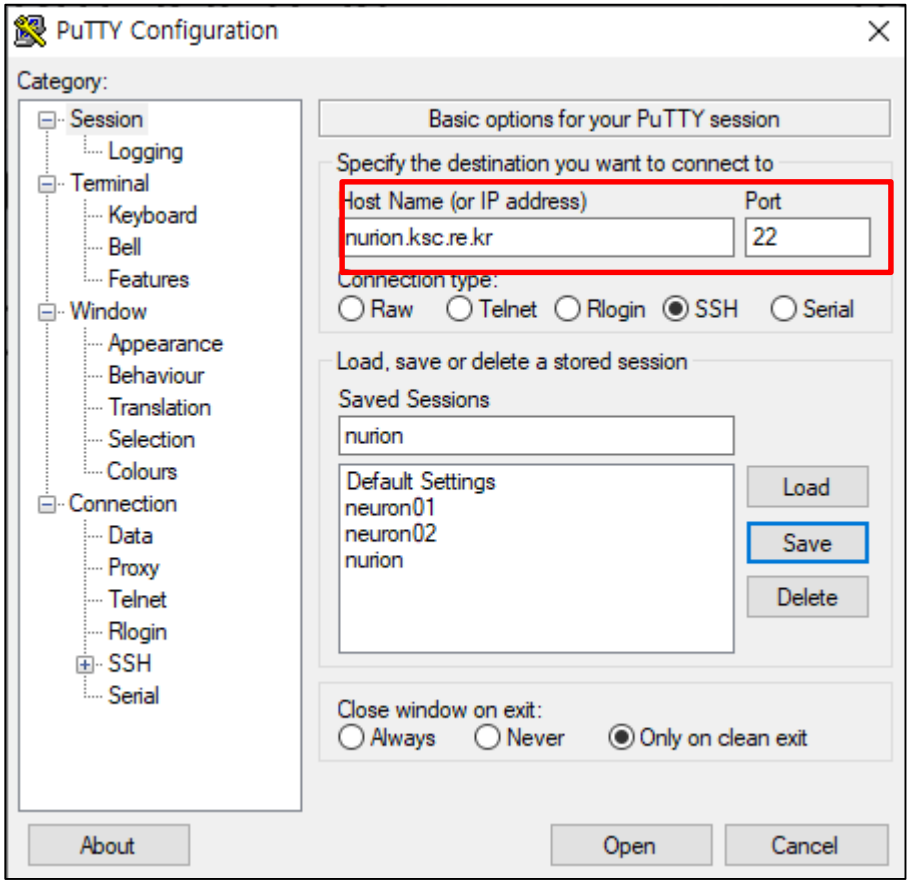
putty.exe (the SSH and Telnet client itself)

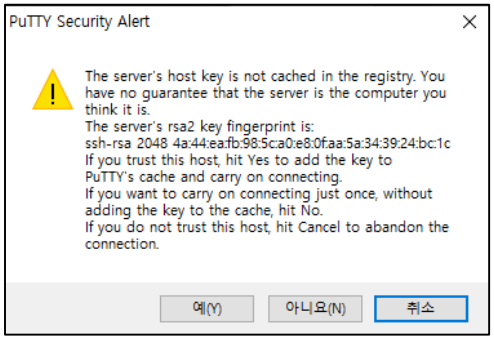
32-bit:	putty.exe	(or by FTP)	(signature)
64-bit:	putty.exe	(or by FTP)	(signature)

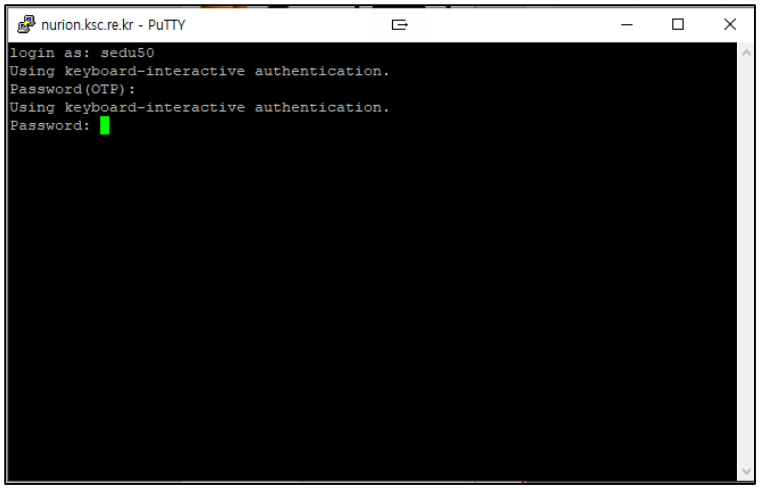
pscp.exe (an SCP client, i.e. command-line secure file copy)

32-bit:	pscp.exe	(or by FTP)	(signature)
64-bit:	pscp.exe	(or by FTP)	(signature)

Window: SSH Client(Putty)를 이용한 접속







환경 설정 : Environment Module

사용자가 셸 환경(shell environment)을 관리하도록 도와주는 도구

‘module’ 명령

- 부명령 (subcommand)
  - avail(av)
    - 사용 가능한 모듈파일들(modulefiles)을 보여줌
  - add(load)
    - 셸 환경으로 모듈파일들을 적재함(load)
  - rm(unload)
    - 셸 환경에서 적재된 모듈파일들을 제거함
  - li(list)
    - 적재된 모듈파일들을 나열함
  - purge
    - 적재된 모든 모듈파일들을 제거함

순차 프로그램 컴파일

프로그램	벤더	컴파일러	소스 확장자	사용 모듈
C / C++	Intel	icc / icpc	.C, .cc, .cpp, .cxx, .c++	intel/17.0.5 | intel/18.0.1 | intel/18.0.3
	GNU	gcc / g++		gcc/6.1.0 | gcc/7.2.0
	Cray	cc / CC		PrgEnv-cray/1.0.2 & cce/8.6.3
F77/F90	Intel	ifort	.f, .for, .ftn, .f90, .fpp, .F, .FOR, .FTN, .FPP, .F90	intel/17.0.5 | intel/18.0.1 | intel/18.0.3
	GNU	gfortran		gcc/6.1.0 | gcc/7.2.0
	Cray	ftn		PrgEnv-cray/1.0.2 & cce/8.6.3

```
$ {gcc|gfortran} [-O3] [-march=knl] test.{c|f90} -o test.exe
$ {icc|ifort} [-O3] [-xMIC-AVX512] test.{c|90} -o test.exe
$ {cc|ftn} -o test.exe [-hcpu=mic-knl] test.{c|f90}
```

‘OpenMP 프로그램 컴파일

```
$ {gcc|gfortran} -o test.exe -fopenmp [-O3] [-march=knl] test.{c|f90}
$ {icc|ifort} -o test.exe -qopenmp [-O3] [-xMIC-AVX512] test.{c|90}
$ {cc|ftn} -o test.exe [-h omp] [-hcpu=mic-knl] test.{c|f90}
```

VI Editor

vim(vi)

- 가장 기본적인 텍스트 에디터, OS에 기본적으로 포함됨
- VIsual display editor를 의미

파일 개방

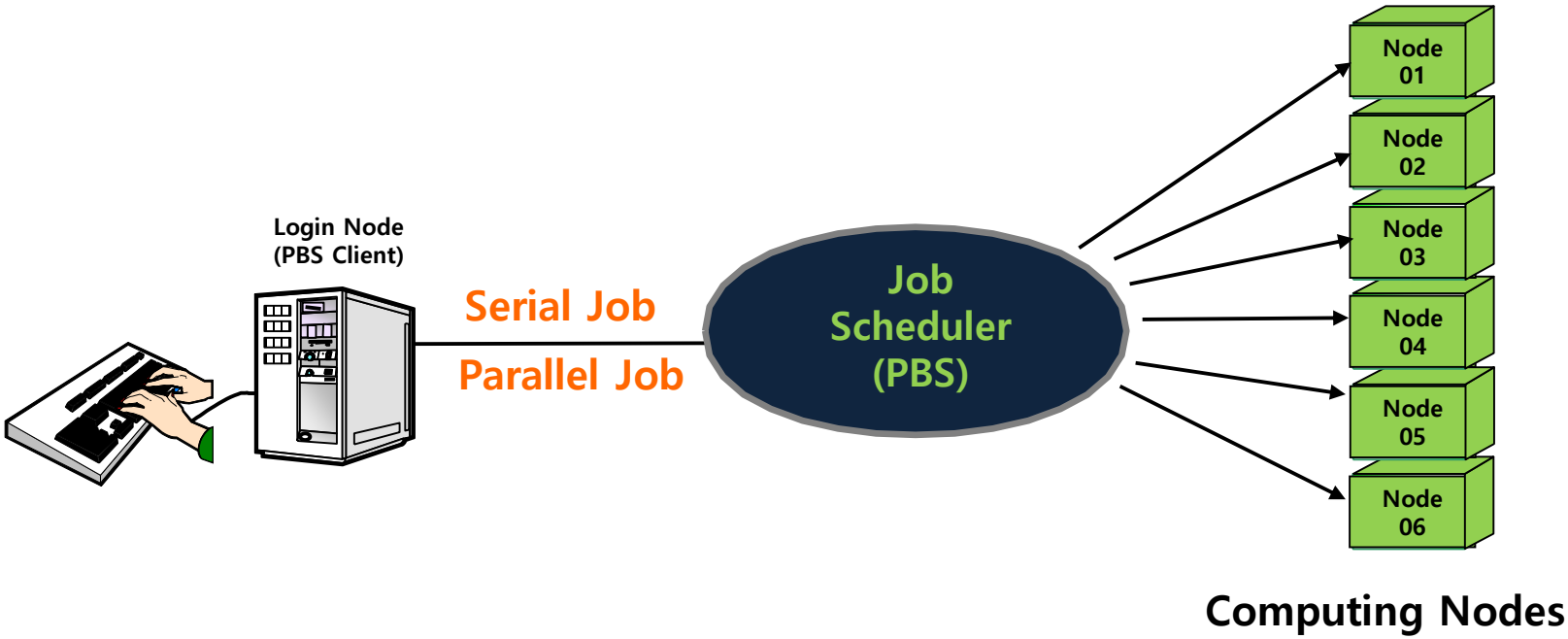
- $ vi file(편집 모드)
- $ view file(읽기 모드)

modes

- 입력 모드
  - 입력모드로 전환 : i (,I, a, A, o, O, R)
  - 입력하는 모든 것이 편집 버퍼에 입력됨
  - 입력 모드에서 빠져 나올 때(명령 행 모드로 변경 시) : “ESC” key
- 명령 행 모드
  - 입력하는 모든 것이 명령어 해석됨

파일 저장/종료 명령 : w, q

PBS, Slurm, OGE, LSF, LoadLeveler, …



1. 작업 스크립트 작성

2. 작업 제출: qsub

```
[sedu50@login01 MPI_Basic]$ qsub job.sh
4997834.pbs
```

주) SCRATCH DIR에서 제출 해야 함(/scratch/*user_id*)

3. 작업 상태 조회: qstat [-u User_ID]

```
[sedu50@login01 MPI_Basic]$ qstat -u sedu50

pbs:
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
4997797.pbs     sedu50   debug    STDIN       53617   2  20   25gb 02:00 R 00:11
4997834.pbs     sedu50   normal   MPI_job     54580   1  68     -- 00:02 R 00:00
```

4. 작업 취소: qdel ⟨JOBID⟩

작업 스크립트 파일 예

Serial Program

```
#!/bin/bash
#PBS -V
#PBS -N Serial_job
#PBS -q normal
#PBS -l walltime=00:05:00
#PBS -l select=1:ncpus=1
#PBS -A etc
cd $PBS_O_WORKDIR
./a.out
```

OpenMP Program

```
#!/bin/bash
#PBS -V
#PBS -N OMP_job
#PBS -q normal
#PBS -l walltime=00:20:00
#PBS -l select=1:ncpus=64:ompthreads=64
#PBS -A etc
cd $PBS_O_WORKDIR
./a.out
```

PBS Interactive 작업 제출

- 교육 중 실습은 debug 큐 이용
  - debug 큐에 작업 제출 또는 인터랙티브 작업 이용
- 인터랙티브 노드 작업: qsub –I (in the “scratch” directory)

```
$ cd /scratch/sedu##   or ( $ cds )
$ qsub -I -V -A etc -l select=1:ncpus=16:ompthreads=16 -l walltime=04:00:00 -q
debug
qsub: waiting for job 4997862.pbs to start
qsub: job 4997862.pbs ready

[sedu50@node8281 C]$ export OMP_NUM_THREADS=4
[sedu50@node8281 C]$ ./hello.x
Hello World
Hello World
Hello World
Hello World
----------------------------------------
[sedu01@node8281 C]$ exit
```

학습 요약

putty를 이용하여 누리온 시스템에 접속

- 사용자 계정, OTP passwd, passwd필요

실습 환경을 설정

- Module 명령을 이용하여 gcc 컴파일러 모듈환경을 로드

실습을 위한 최소한의 vi 편집기 사용법

- 입력 모드, 명령행 모드

PBS job scheduler 사용법

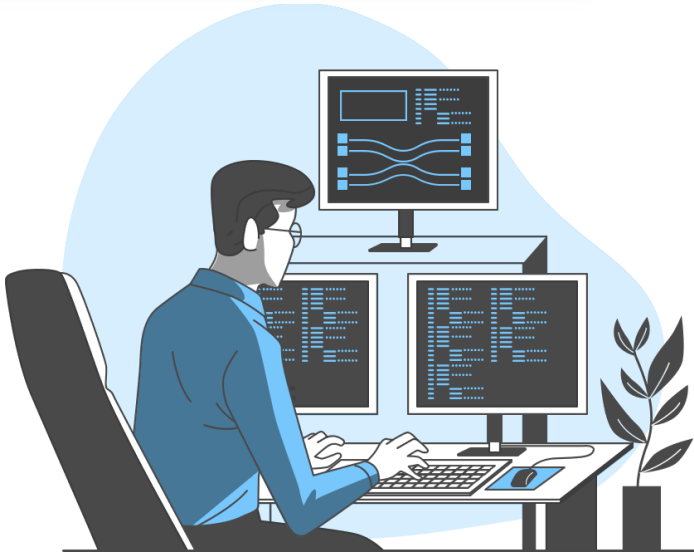
- 작업 스크립트 작성법
- Interactive mode로 작업 제출법

Outro

- putty를 이용하여 누리온 시스템에 접속
- 실습 환경을 설정
- vi 편집기 사용법
- PBS job scheduler 사용법

02 OpenMP Basic I

1. OpenMP 소개
2. OpenMP의 구성, 조건부 컴파일
3. 스레드 생성



학습 목표

- OpenMP의 개념 이해한다.
- OpenMP 구성 요소 이해한다.
- 조건부 컴파일 이해한다.
- 스레드 생성 법 이해한다.

OpenMP란?

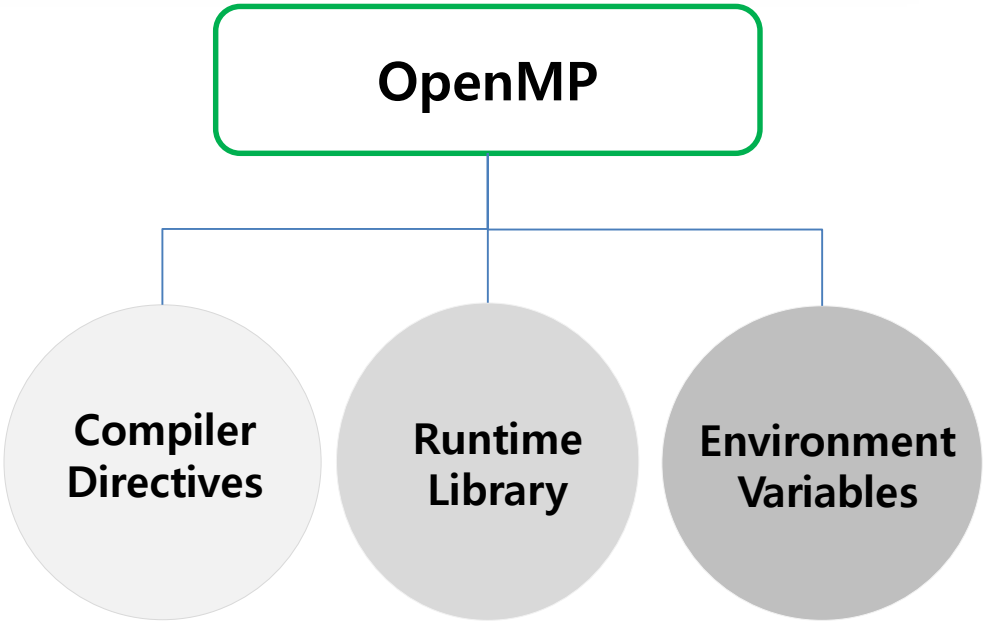
Open specifications for Multi Processing

(공유메모리 환경에서) 다중 스레드 병렬 프로그램 작성을 위한 응용프로그램 인터페이스(API)

- Language(Fortran, C/C++) Extensions
- 공유 메모리 프로그래밍 모델의 (사실상) 표준

구성

- 컴파일러 지시어(omp parallel)
- 런타임 라이브러리(함수)
  (omp_set_num_threads(n))
- 환경 변수
  (OMP_NUM_THREADS=n)



OpenMP란?

www.openmp.org

OpenMP란?

History

- 1997.10: OpenMP for Fortran 1.0
- 1998.10: OpenMP for C/C++ 1.0
- 2000.11: OpenMP for Fortran 2.0
- 2002.3: OpenMP for C/C++ 2.0
- 2005.5: OpenMP 2.5 (combined C/C++ & Fortran)
- 2008.5: OpenMP 3.0 (Task)
- 2011.7: OpenMP 3.1
- 2013.7: OpenMP 4.0 (Accelerator)
- 2015.11: OpenMP 4.5
- 2018.11: OpenMP 5.0
- 2020.11: OpenMP 5.1
- 2021.11: OpenMP 5.2

OpenMP의 장단점

장점	단점
• MPI에 비해 코딩, 디버깅 비교적 용이 • MPI보다 DATA 분할의 부담이 적다 • 각 Loop을 하나씩 병렬화하여 점진적인 병렬화가 가능 • 하나의 code를 병렬 code와 순차 code로 compile 가능 • 상대적으로 code의 크기가 작다	• 공유 메모리 환경의 다중 프로세서 아키텍처에서만 실행 가능 • 아키텍처(프로세서 수,메모리)의 한계로 원하는 성능을 얻기 힘들다 • OpenMP를 지원하는 compiler가 반드시 필요

OpenMP의 구성(1/2)

컴파일러 지시어

- 스레드 사이의 작업분담, 통신, 동기화를 담당
- 좁은 의미의 OpenMP

  ex) !$OMP PARALLEL DO

실행시간 라이브러리

- 병렬 매개변수(참여 스레드의 개수, 스레드 ID)의 설정과 조회

ex) CALL omp_set_num_threads(n) : 생성될 스레드 개수 지정

CALL omp_get_num_threads() : 생성된 스레드 개수 리턴

CALL omp_get_thread_num() : 스레드 ID 리턴

CALL omp_get_max_threads() : 사용 가능한 최대 스레드 개수 리턴

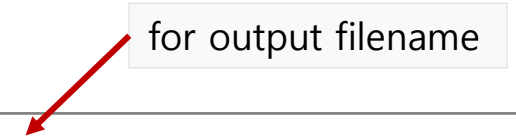
환경 변수(Environmental Variables)

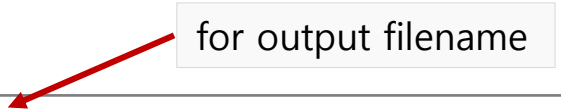
- 실행 시스템의 병렬 매개변수(스레드 개수 등) 정의

ex) export OMP_NUM_THREADS=8

Compiling & Execution Check

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"); } }
Intel : ifort -o hello.x hello.f90 GCC : gfortran -o hello.x hello.f90 Cray : ftn -o hello.x hello.f90	Intel : icc -o hello.x hello.c GCC : gcc -o hello.x hello.c Cray : cc -o hello.x hello.c
Intel : ifort -qopenmp –o hello_omp.x hello.f GCC : gfortran -fopenmp –o hello_omp.x hello.f Cray : ftn –o hello_omp.x hello.f	Intel : icc -qopenmp –o hello_omp.x hello.c GCC : gcc -fopenmp –o hello_omp.x hello.c Cray : cc -mp –o hello_omp.x hello.c
$ 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





OpenMP의 구성(2/2)

Fortran

```
PROGRAM hello_world

  INTEGER omp_get_thread_num
!$OMP PARALLEL
  PRINT *, 'Hello World', omp_get_thread_num()
!$OMP END PARALLEL

END
```

Compiler Directive

Runtime Library

```
$ gfortran -fopenmp -o omp_comp.x omp_comp.f90
$ export OMP_NUM_THREADS = 4
$ ./omp_component.x
```

Environment Variable

C

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

int main()
{
#pragma omp parallel
{
  printf ("Hello World %d\n",
  omp_get_thread_num());
}
}
```

Compiler Directive

Runtime Library

```
$ gcc -fopenmp -o omp_comp.x omp_comp.c
$ export OMP_NUM_THREADS = 4
$ ./omp_component.x
```

Environment Variable

	Fortran	C
Compiler Directive	!$OMP 〈directive〉	#pragma omp 〈directive〉
Runtime Library	omp_	omp_
Environmental Variable	OMP_	OMP_

조건부 컴파일

Fortran	C
program main use omp_lib #ifdef _OPENMP print*, '_OPENMP : ', _OPENMP !$OMP PARALLEL print*,'Hello World', omp_get_thread_num() !$OMP END PARALLEL #else print*, 'Hello World', 0 #endif end program main	#include <stdio.h> #include <omp.h> int main(void) { #ifdef _OPENMP #pragma omp parallel { printf("Hello World %d\n",omp_get_thread_num()); } #else printf("Hello World 0\n"); #endif }
$ gfortran -cpp conditional.f90 $ gfortran -cpp -fopenmp conditional.f90 $ **export OMP_NUM_THREADS = 4** $ ./a.out	$ gcc conditional.c $ gcc -fopenmp conditional.c $ **export OMP_NUM_THREADS = 4** $ ./a.out

스레드 생성(1/4)

Fortran	C
PROGRAM hello_world INTEGER omp_get_thread_num or (Include “omp_lib.h”) or (Use omp_lib) !$OMP PARALLEL PRINT *, ‘Hello World’, omp_get_thread_num() !$OMP END PARALLEL print *, ‘’ call omp_set_num_threads(4) !$OMP PARALLEL PRINT *, ‘Hello World’, omp_get_thread_num() !$OMP END PARALLEL print*, ‘’ !$OMP PARALLEL num_threads(2) PRINT *, ‘Hello World’, omp_get_thread_num() !$OMP END PARALLEL END	#include <stdio.h> #include <omp.h> int main() { #pragma omp parallel { printf (“Hello World %d\n”, omp_get_thread_num()); } printf(“\n”); **omp_set_num_threads(4);** #pragma omp parallel { printf (“Hello World %d\n”, omp_get_thread_num()); } printf(“\n”); #pragma omp parallel **num_threads(2)** { printf (“Hello World %d\n”, omp_get_thread_num()); } }
$ gfortran **-fopenmp** –o create_thread.x create_thread.f90 $ export OMP_NUM_THREADS = 8 $ ./create_thread.x	$ gcc **-fopenmp** –o create_thread.x create_thread.c $ export OMP_NUM_THREADS = 8 $ ./create_thread.x

스레드 생성(2/4)

병렬 영역 지정

Fortran	C
!$OMP PARALLEL !$OMP END PARALLEL	#pragma omp parallel { }

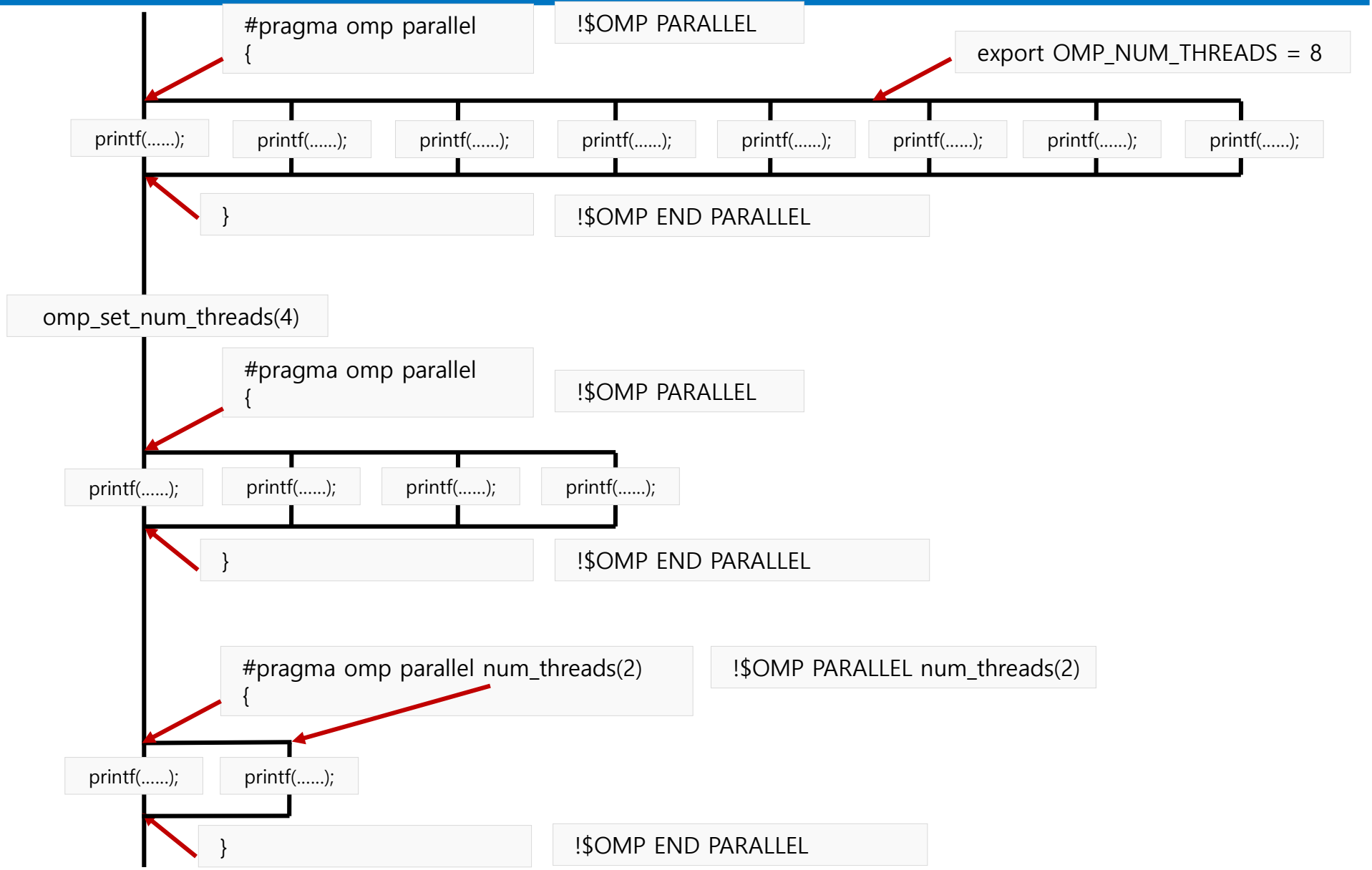
스레드 개수 설정

- 환경 변수 : export OMP_NUM_THREADS = xxx
- 실행시간 라이브러리 : omp_set_num_threads(xxx)
- 지시어 : #pragma omp parallel num_threads(xxx)

스레드와 관련된 Runtime libraries

- omp_set_num_threads(integer): 스레드 개수 설정
- omp_get_num_threads(): 현재 스레드 팀에서 스레드 개수 반환
- omp_get_thread_num(): 스레드 ID 반환

스레드 생성(3/4)


#pragma omp parallel
{
!$OMP PARALLEL
export OMP_NUM_THREADS = 8
printf(……);
printf(……);
printf(……);
printf(……);
printf(……);
printf(……);
printf(……);
printf(……);
}
!$OMP END PARALLEL
omp_set_num_threads(4)
#pragma omp parallel
{
!$OMP PARALLEL
printf(……);
printf(……);
printf(……);
printf(……);
}
!$OMP END PARALLEL
#pragma omp parallel num_threads(2)
{
!$OMP PARALLEL num_threads(2)
printf(……);
printf(……);
}
!$OMP END PARALLEL

스레드 생성(4/4)

OpenMP Programming Model

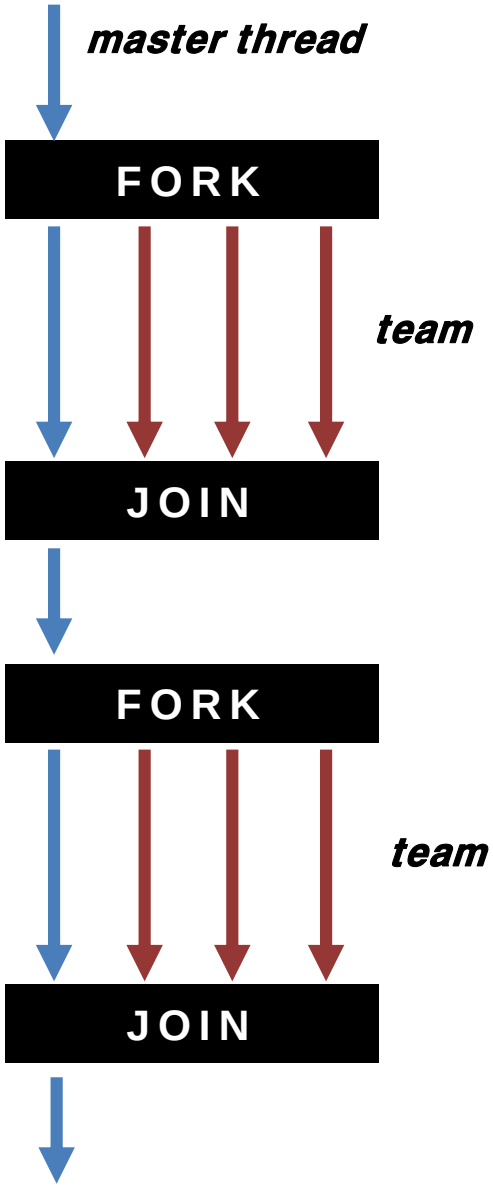
- Thread-Based
- Fork-Join Model

Fork-Join Model

- Master thread는 병렬 영역의 끝에서 합쳐지는 스레드 팀 생성
- 동일 팀에 속한 스레드들이 공동 작업

❖ 컴파일러가 지시어를 참고하여 다중 스레드 코드 생성

- OpenMP를 지원하는 컴파일러 필요



학습 요약

OpenMP 소개

- 공유 메모리환경에서 사용가능
- OpenMP 장/단점 소개

OpenMP 구성 요소

- 컴파일러 지시어, 런타임 라이브러리 함수, 환경 변수

조건부 컴파일

- _OPENMP 매크로를 사용

스레드 생성

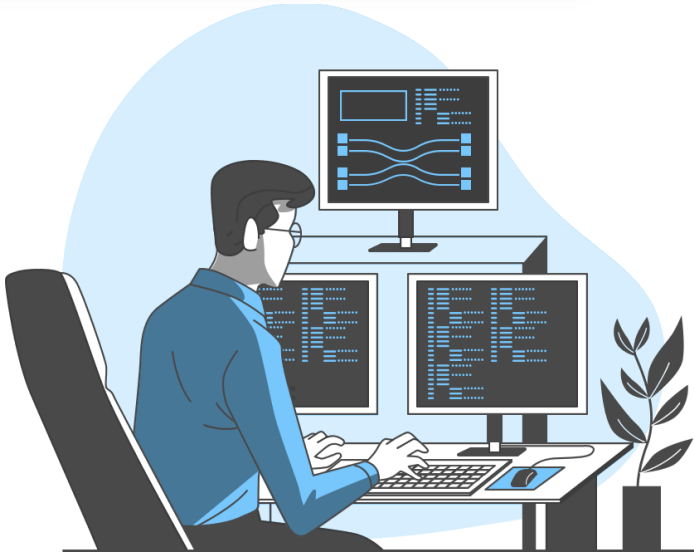
- 스레드 생성, 스레드 개수 지정, 스레드 번호 조회
- Fork-join model

Outro

- OpenMP 소개
- OpenMP 구성 요소
- 조건부 컴파일
- 스레드 생성

03 OpenMP Basic II

1. 데이터 유효 범위
2. 코드 설명(데이터 유효 범위)
3. 스레드와 프로세스



학습목표

- 데이터의 유효 범위를 이해한다.
- 데이터 유효 범위의 사용법을 익힌다
- 스레드와 프로세스의 차이를 이해한다.

데이터 유효범위

private(var1,var2,…..)

- 지정된 변수를 스레드끼리 공유 방지
- Private변수는 병렬영역 내에서만 정의
- 병렬영역 밖에서 초기화 불가능
- 병렬영역이 끝나면 private 변수도 소멸

private 특징

- 단순히 스레드로 메모리 영역이 할당되는 것이기 때문에 순차 영역의 오리지널 변수가 가지는 데이터 값은 전달되지 않는다
- 스레드 영역에서 갱신된 값은 메인 영역에 있는 변수로 전달되지 않는다
- 스레드 소멸과 함께 private변수도 소멸

private 선언 시 고려사항

- 병렬영역 내에서 값을 할당 받는 변수

데이터 유효범위

firstprivate(var1,var2,…..)

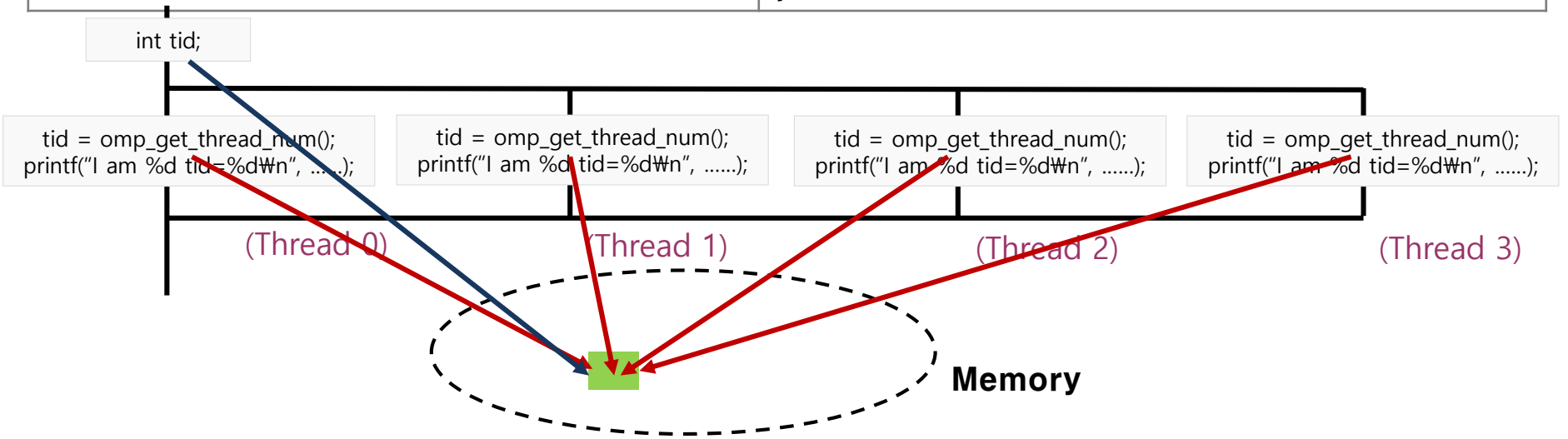
- private변수처럼 각 스레드에 개별적으로 변수 생성
- 각 스레드마다 순차영역에서 가져온 값으로 초기화
- 스레드가 소멸될 때 로컬변수의 값이 메인 영역으로 전달되지 않고 스레드와 함께 소멸
- 사용방법과 동작은 private와 동일

shared(var1,var2,….)

- 지시어를 사용하지 않으면 적용되는 기본값
- 변수 명에 지정된 변수를 스레드 팀의 모든 스레드가 공유

데이터 유효범위(1/8)

Fortran	C
PROGRAM hello_wrong INTEGER a, tid, omp_get_thread_num call **omp_set_num_threads(4)** !$OMP PARALLEL tid = omp_get_thread_num() PRINT *, ‘ I am ’, omp_get_thread_num(), ‘ tid = ’, tid !$OMP END PARALLEL END	#include <stdio.h> #include <omp.h> int main() { int tid; omp_set_num_threads(4); #pragma omp parallel { tid = omp_get_thread_num(); printf (“I am %d tid = %d\n”, omp_get_thread_num(), tid); } }



데이터 유효범위(2/8)

omp_set_num_threads(4)

int tid;

#pragma omp parallel
{

!$OMP PARALLEL

store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);

store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);

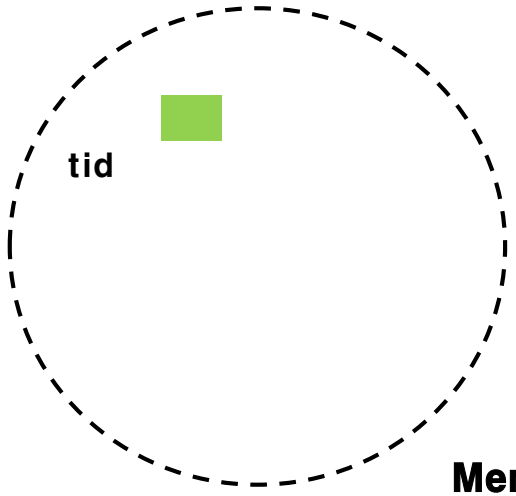
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);

store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);

(Thread 0) (Thread 1) (Thread 2) (Thread 3)

}

!$OMP END PARALLEL



scenario

store tid 0	(Thread 0)
store tid 1	(Thread 1)
store tid 2	(Thread 2)
load tid printf("Hello World %d₩n", tid);	(Thread 0)
load tid printf("Hello World %d₩n", tid);	(Thread 1)
.....	(.....)

데이터 유효범위(2/8)

omp_set_num_threads(4)

int tid;

#pragma omp parallel
{

!$OMP PARALLEL

store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);

store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);

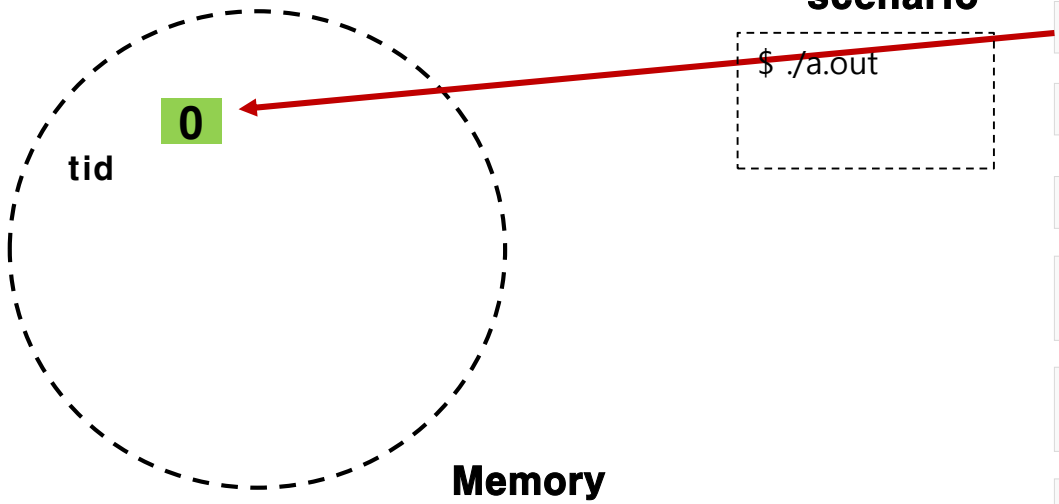
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);

store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);

(Thread 0) (Thread 1) (Thread 2) (Thread 3)

}

!$OMP END PARALLEL



scenario

$ ./a.out

store tid 0 (Thread 0)

store tid 1 (Thread 1)

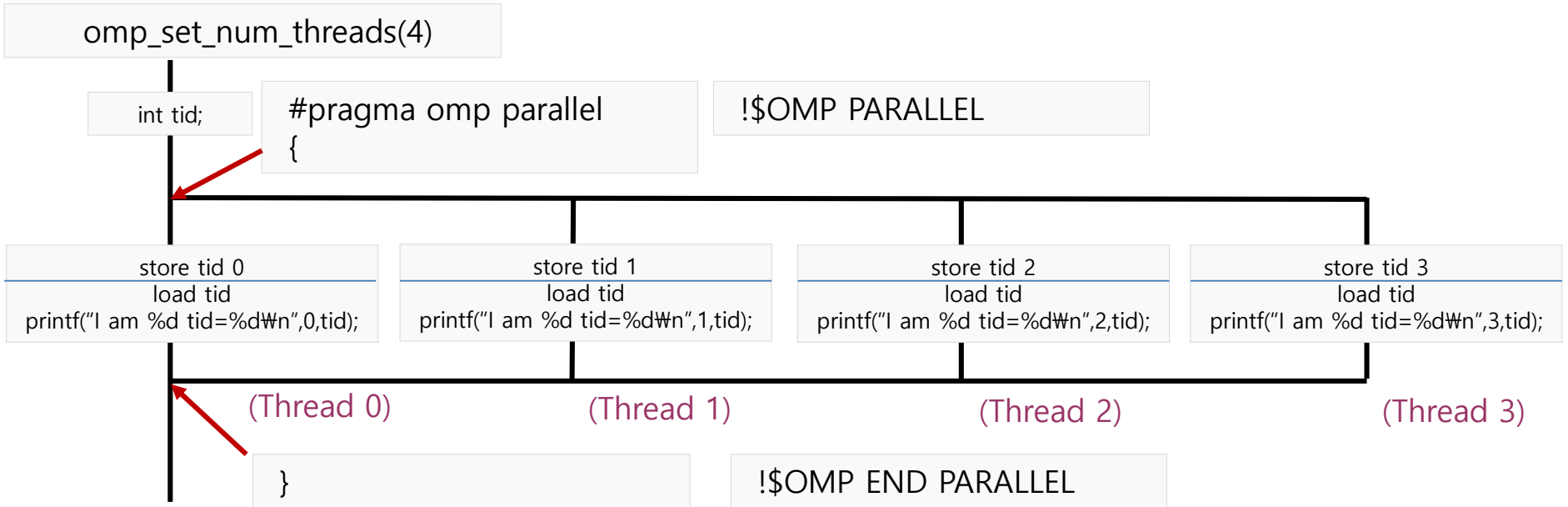
store tid 2 (Thread 2)

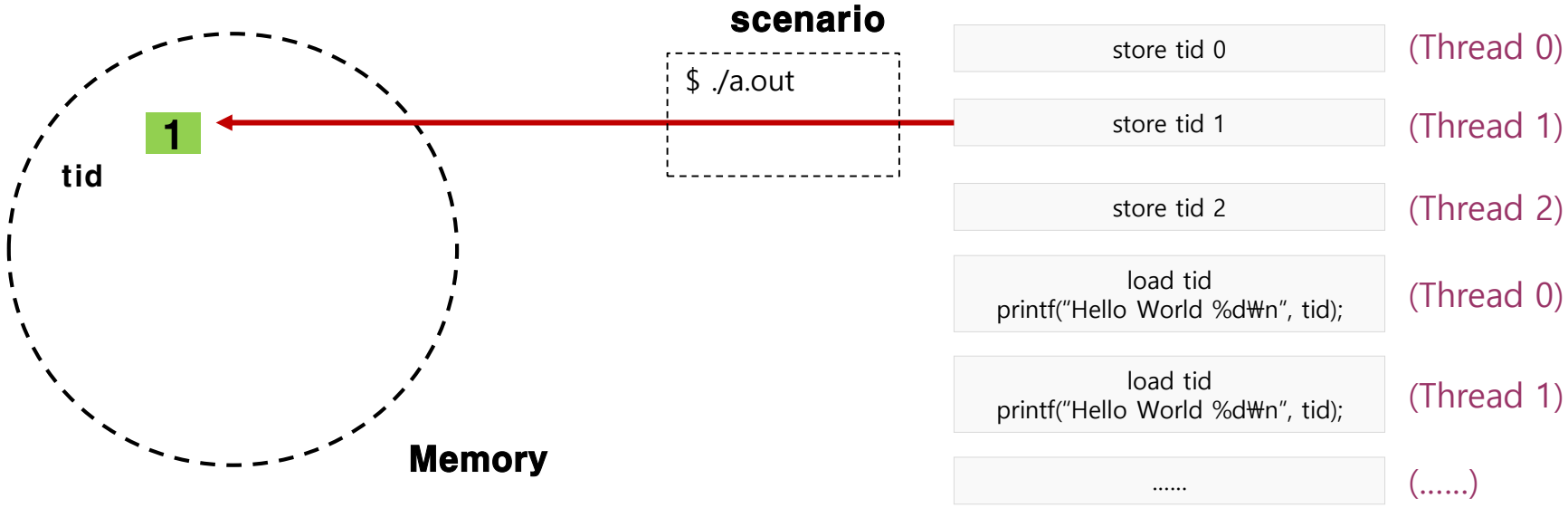
load tid
printf("Hello World %d₩n", tid); (Thread 0)

load tid
printf("Hello World %d₩n", tid); (Thread 1)

..... (.....)

데이터 유효범위(2/8)


omp_set_num_threads(4)
int tid;
#pragma omp parallel
{
!$OMP PARALLEL
store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);
store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);
store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL


scenario
$ ./a.out
1
tid
Memory
store tid 0
(Thread 0)
store tid 1
(Thread 1)
store tid 2
(Thread 2)
load tid
printf("Hello World %d₩n", tid);
(Thread 0)
load tid
printf("Hello World %d₩n", tid);
(Thread 1)
.....
(.....)

데이터 유효범위(2/8)

omp_set_num_threads(4)

int tid;

#pragma omp parallel
{

!$OMP PARALLEL

store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);

store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);

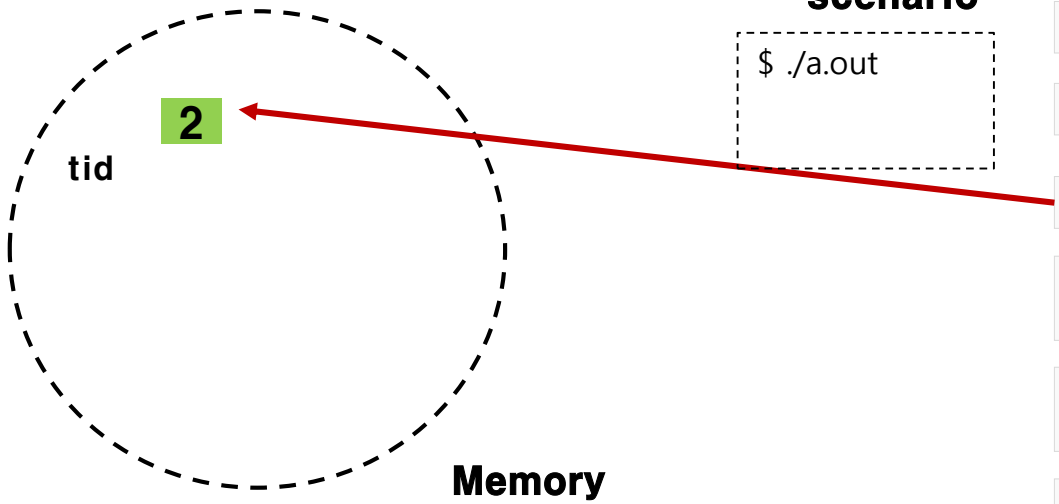
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);

store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);

(Thread 0) (Thread 1) (Thread 2) (Thread 3)

}

!$OMP END PARALLEL



scenario

$ ./a.out

store tid 0 (Thread 0)

store tid 1 (Thread 1)

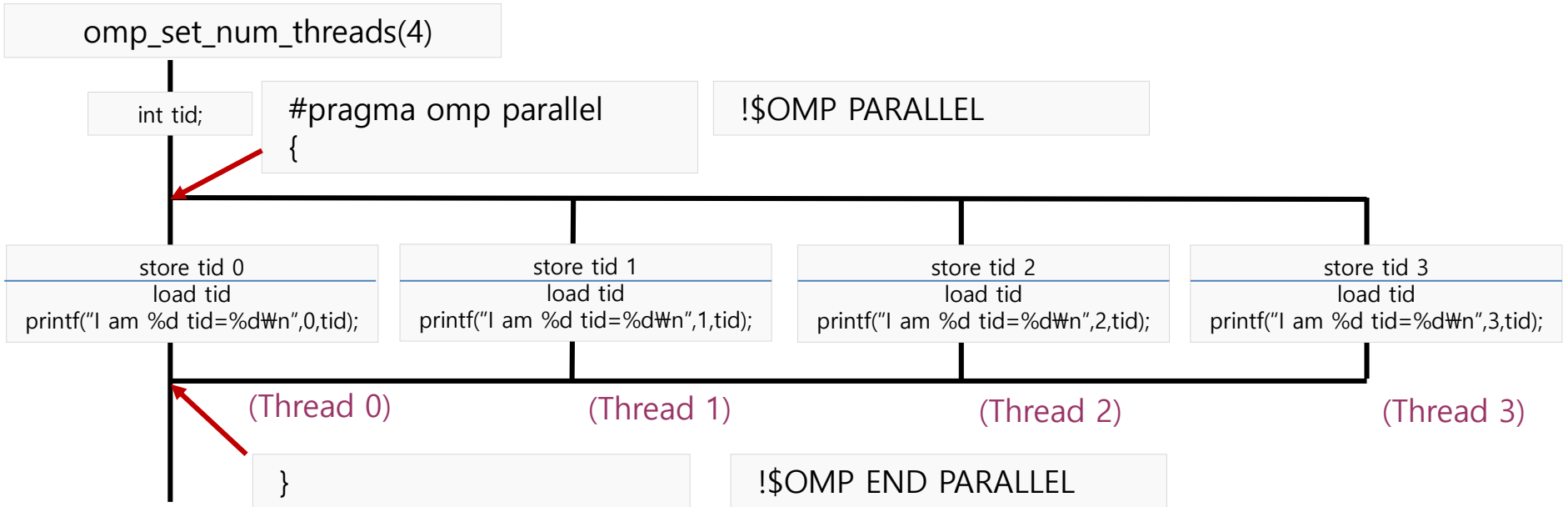
store tid 2 (Thread 2)

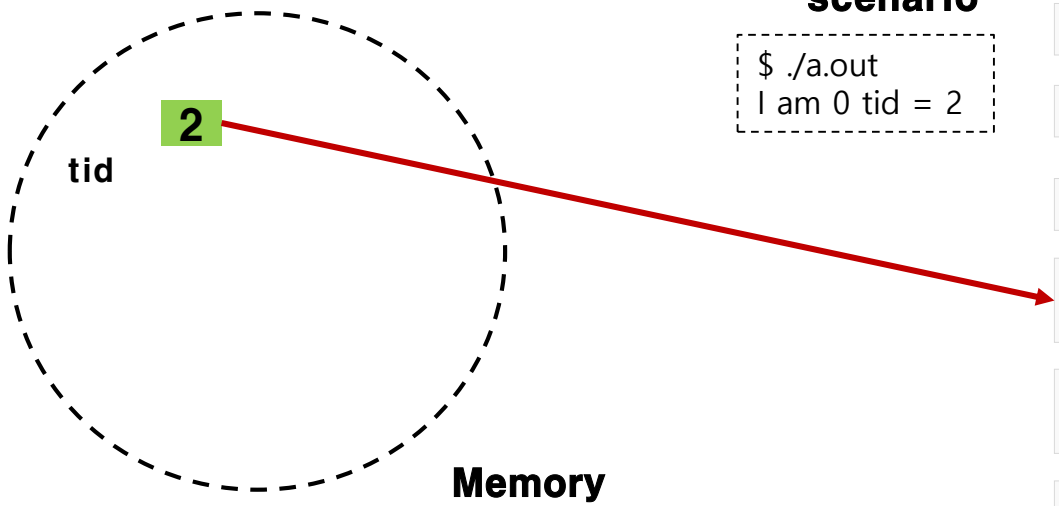
load tid
printf("Hello World %d₩n", tid); (Thread 0)

load tid
printf("Hello World %d₩n", tid); (Thread 1)

..... (.....)

데이터 유효범위(2/8)





scenario

```
$ ./a.out
I am 0 tid = 2
```

store tid 0	(Thread 0)
store tid 1	(Thread 1)
store tid 2	(Thread 2)
load tid ← 2 printf("I am %d tid=%d₩n",0,tid);	(Thread 0)
load tid printf("Hello World %d₩n", tid);	(Thread 1)
.....	(.....)

데이터 유효범위(2/8)

omp_set_num_threads(4)

int tid;

#pragma omp parallel
{

!$OMP PARALLEL

store tid 0
load tid
printf("I am %d tid=%d₩n",0,tid);

store tid 1
load tid
printf("I am %d tid=%d₩n",1,tid);

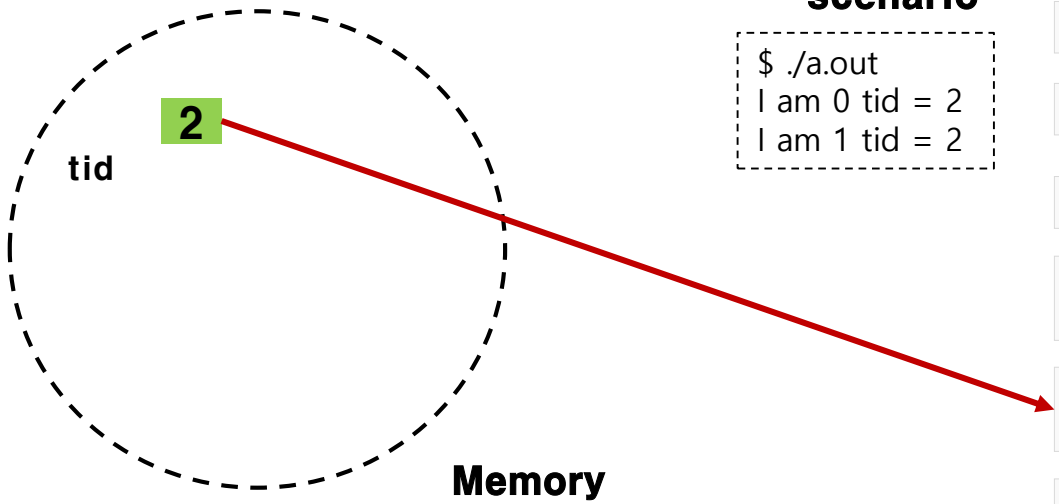
store tid 2
load tid
printf("I am %d tid=%d₩n",2,tid);

store tid 3
load tid
printf("I am %d tid=%d₩n",3,tid);

(Thread 0) (Thread 1) (Thread 2) (Thread 3)

}

!$OMP END PARALLEL



scenario

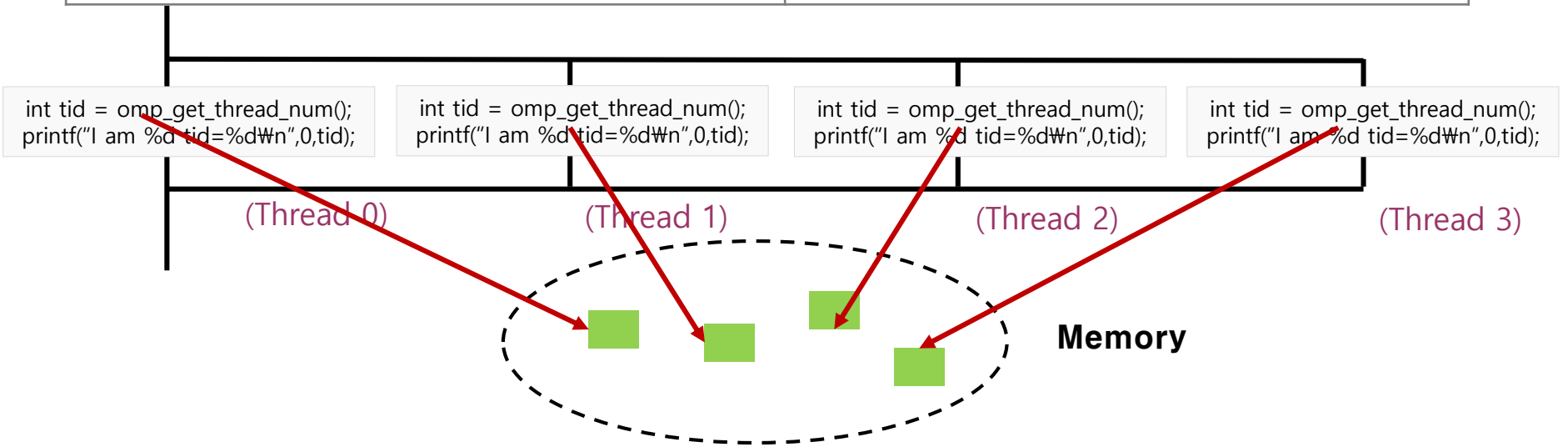
```
$ ./a.out
I am 0 tid = 2
I am 1 tid = 2
```

store tid 0	(Thread 0)
store tid 1	(Thread 1)
store tid 2	(Thread 2)
load tid ← 2 printf("Hello World %d₩n", tid);	(Thread 0)
load tid ← 2 printf("Hello World %d₩n", tid);	(Thread 1)
.....	(.....)

데이터 유효범위(3/8)

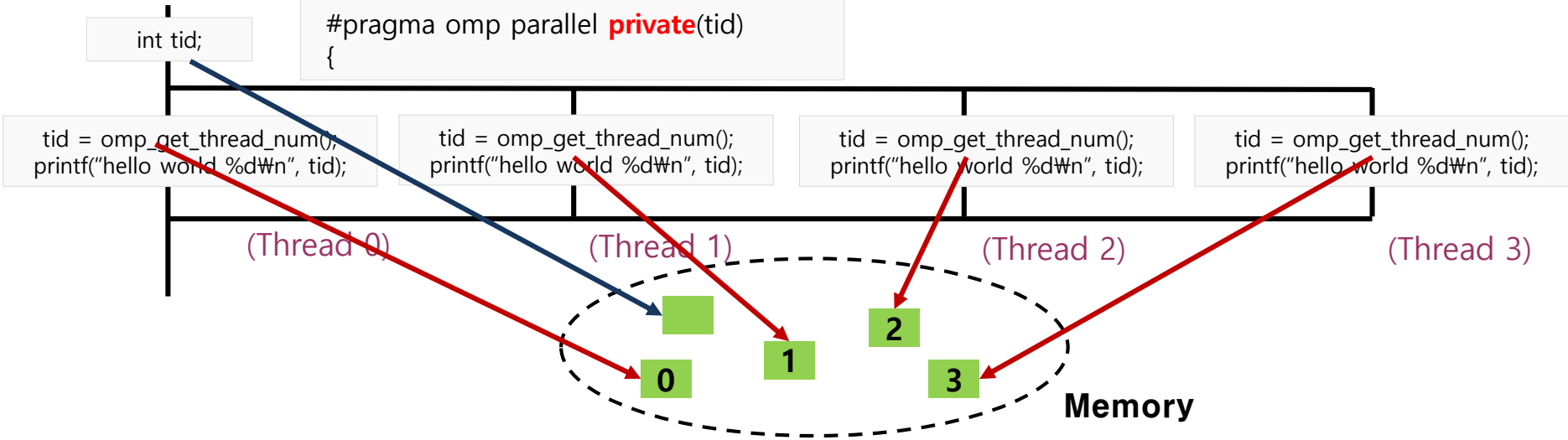
Fortran	C
PROGRAM hello_wrong INTEGER omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL INTEGER tid tid = OMP_GET_THREAD_NUM() PRINT *, 'I am', & OMP_GET_THREAD_NUM(), & 'TID =' tid !$OMP END PARALLEL END	#include <stdio.h> #include <omp.h> int main() { omp_set_num_threads(4); #pragma omp parallel { int tid = omp_get_thread_num(); printf ("I am %d tid = %d\n", omp_get_thread_num(), tid); } }

Not allowed



데이터 유효범위(4/8)

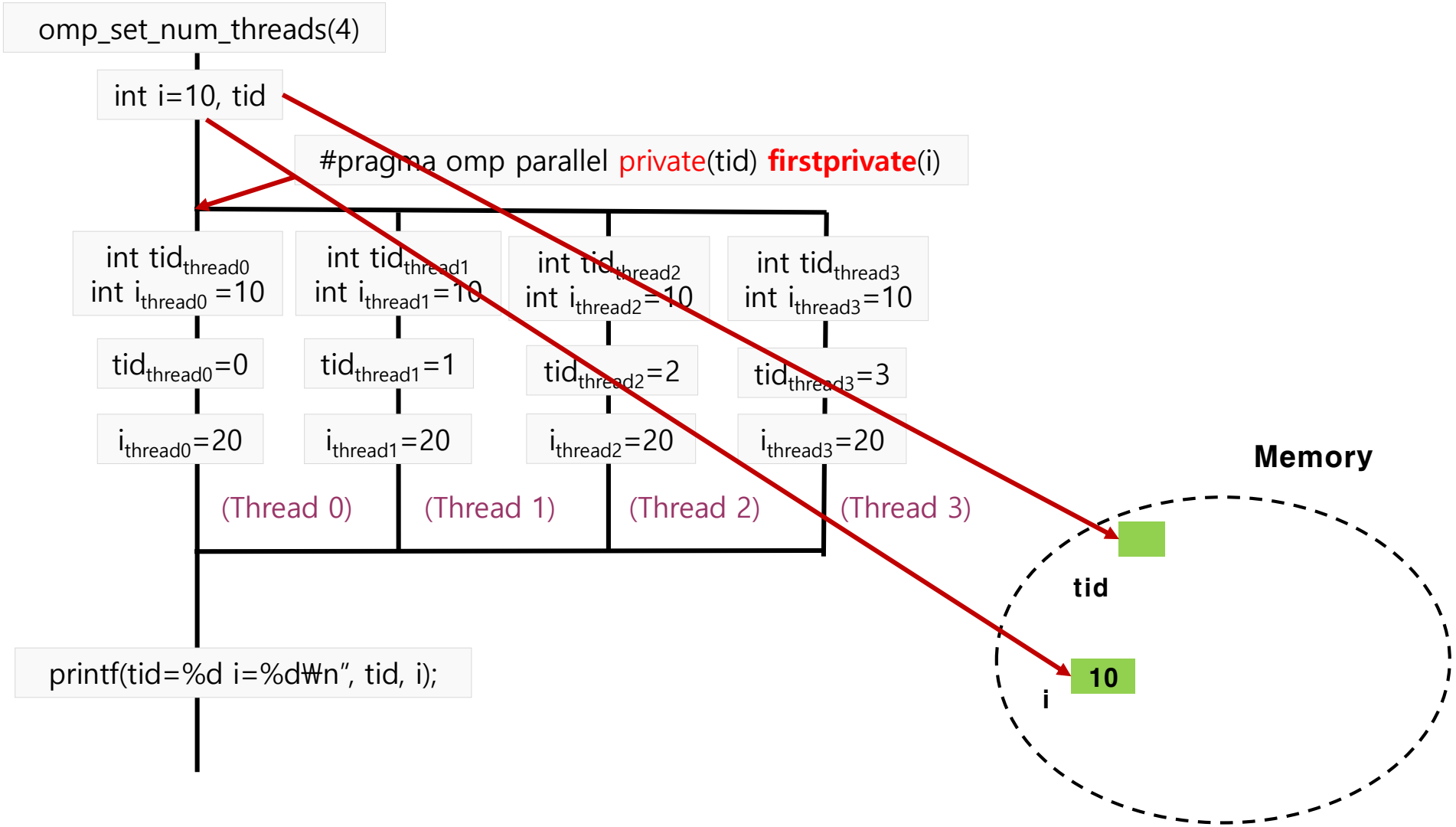
Fortran	C
PROGRAM hello_right INTEGER tid, omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL **PRIVATE(tid)** tid = OMP_GET_THREAD_NUM() PRINT *, 'I am ', OMP_GET_THREAD_NUM(), ' tid = ' tid !$OMP END PARALLEL END	#include <stdio.h> #include <omp.h> int main() { int tid; omp_set_num_threads(4); #pragma omp parallel **private(tid)** { tid = omp_get_thread_num(); printf ("I am %d tid = %d\n", omp_get_thread_num(), tid); } }



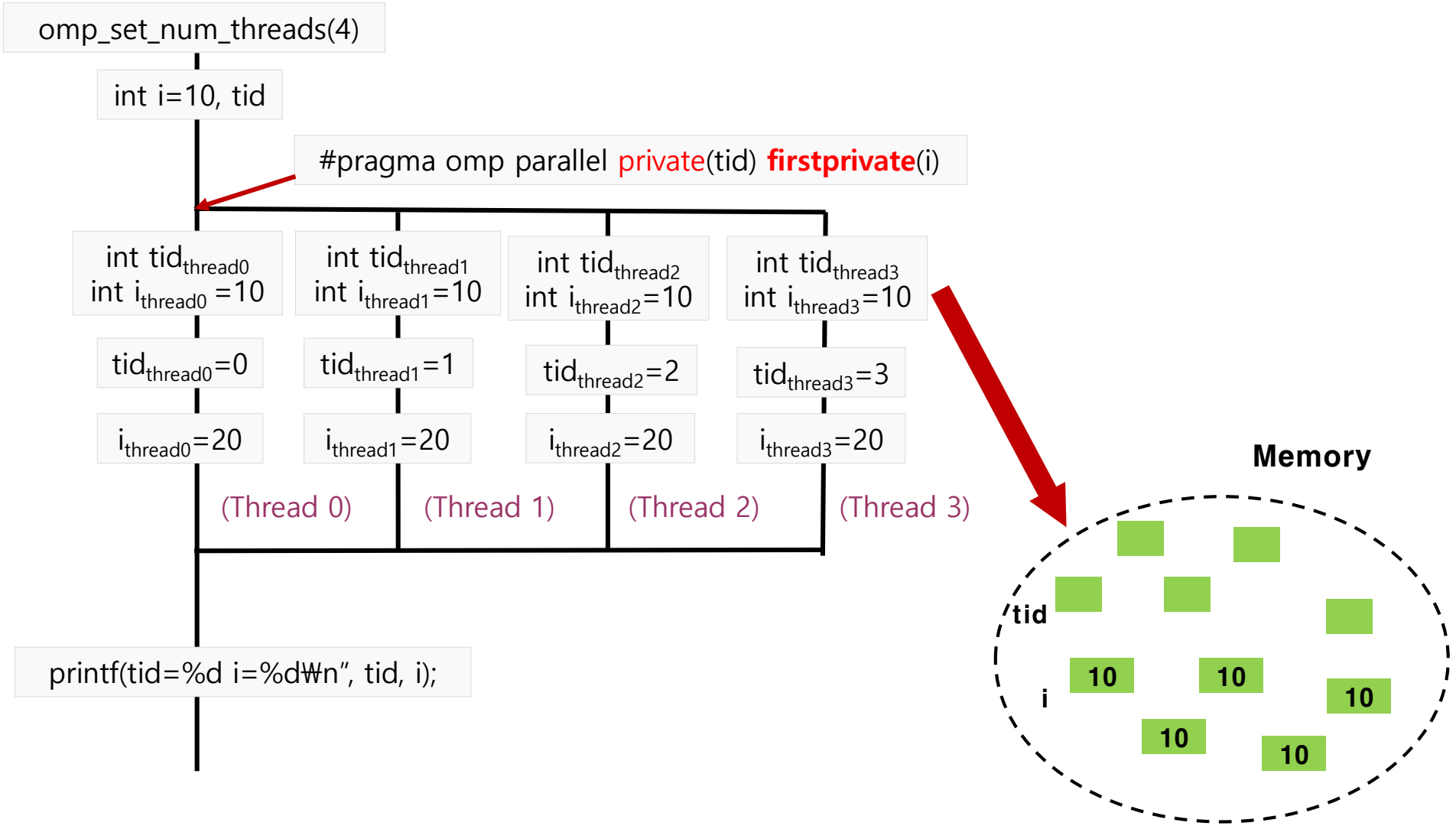
데이터 유효범위(5/8)

Fortran	C
PROGRAM data_scope_firstprivate INTEGER :: i=10, tid, omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL PRIVATE(tid) **FIRSTPRIVATE(i)** tid = OMP_GET_THREAD_NUM() PRINT *, 'tid =' , tid, 'i =', i i = 20 !$OMP END PARALLEL print *, 'tid =', tid, 'i =', i END	#include <stdio.h> #include <omp.h> int main() { int i = 10, tid; omp_set_num_threads(4); #pragma omp parallel private(tid) **firstprivate(i)** { tid = omp_get_thread_num(); printf ("tid = %d i = %d\n", tid, i); i = 20; } printf("tid = %d i=%d\n", tid, i); }

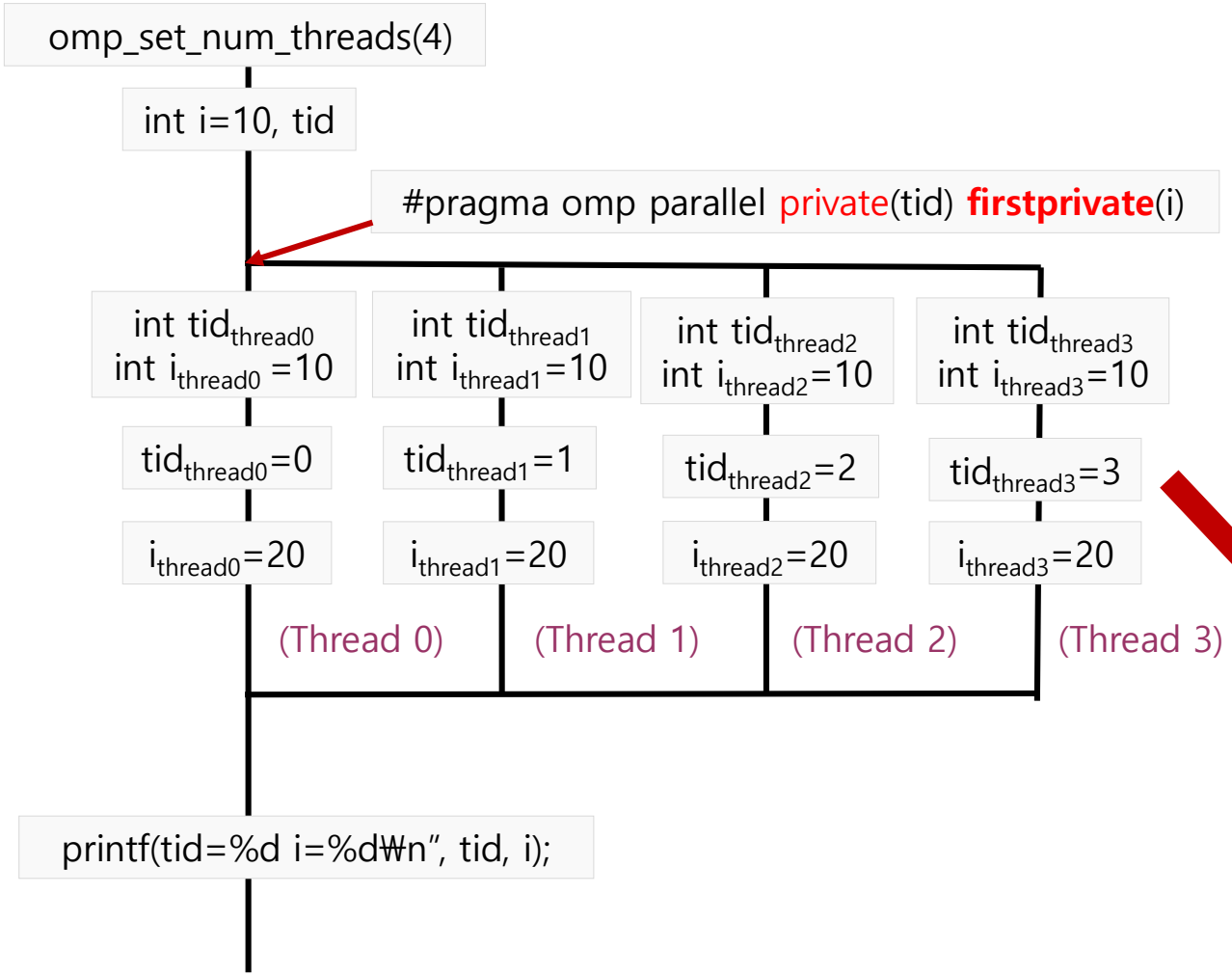
데이터 유효범위(6/8)

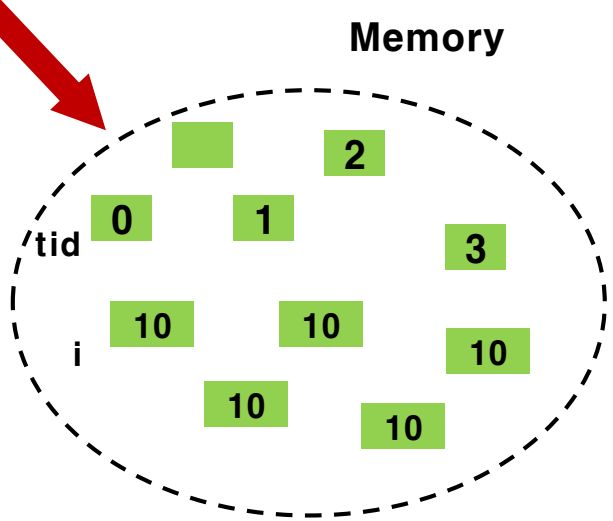

omp_set_num_threads(4)
int i=10, tid
#pragma omp parallel private(tid) firstprivate(i)
int tid$_{thread0}$
int i$_{thread0}$ =10
int tid$_{thread1}$
int i$_{thread1}$=10
int tid$_{thread2}$
int i$_{thread2}$=10
int tid$_{thread3}$
int i$_{thread3}$=10
tid$_{thread0}$=0
tid$_{thread1}$=1
tid$_{thread2}$=2
tid$_{thread3}$=3
i$_{thread0}$=20
i$_{thread1}$=20
i$_{thread2}$=20
i$_{thread3}$=20
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
printf(tid=%d i=%d₩n", tid, i);
Memory
tid
10
i

데이터 유효범위(6/8)

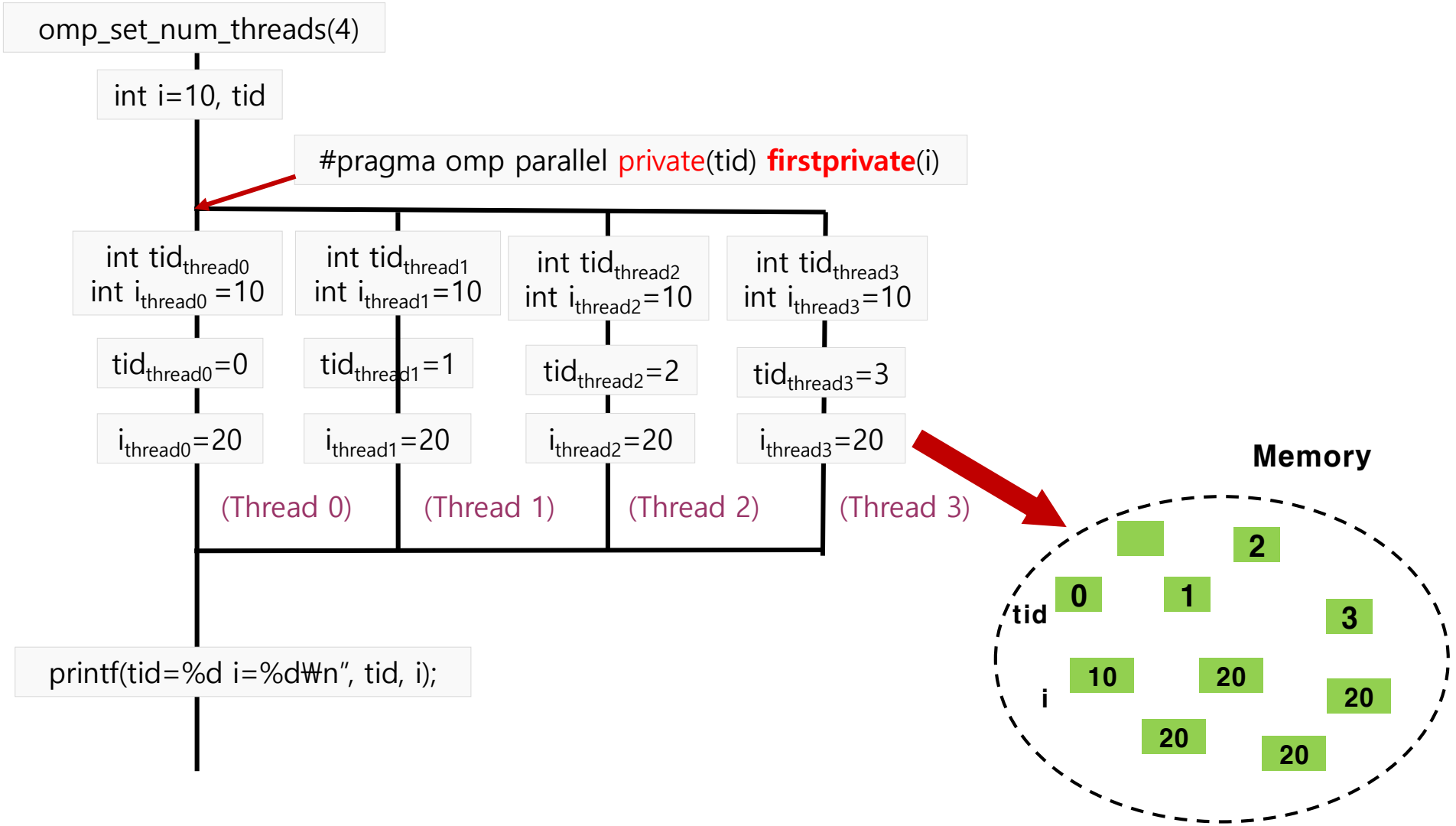

omp_set_num_threads(4)
int i=10, tid
#pragma omp parallel private(tid) firstprivate(i)
int tid_thread0
int i_thread0 =10
int tid_thread1
int i_thread1=10
int tid_thread2
int i_thread2=10
int tid_thread3
int i_thread3=10
tid_thread0=0
tid_thread1=1
tid_thread2=2
tid_thread3=3
i_thread0=20
i_thread1=20
i_thread2=20
i_thread3=20
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
printf(tid=%d i=%d₩n", tid, i);
Memory
tid
i
10
10
10
10
10

데이터 유효범위(6/8)

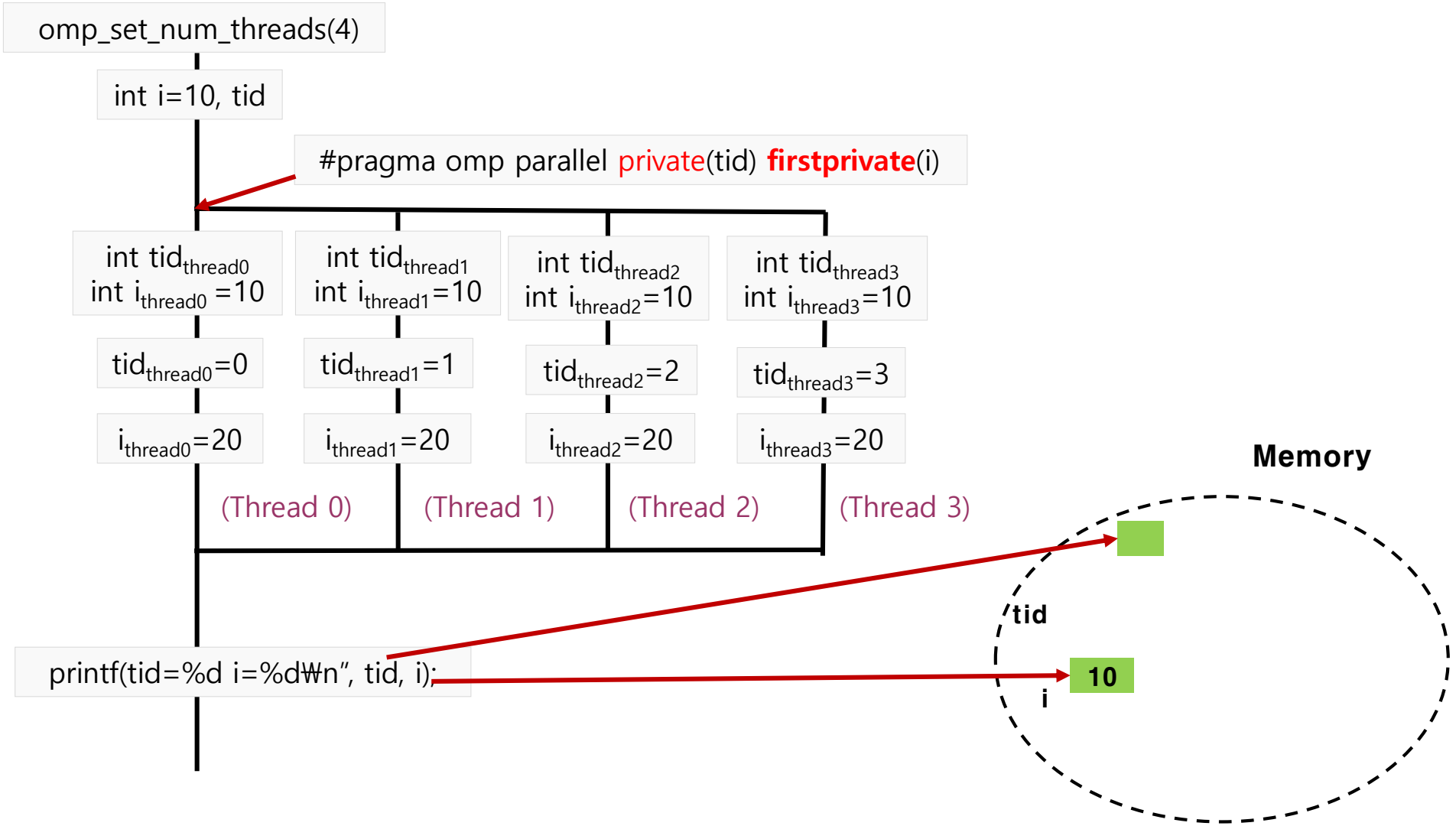

omp_set_num_threads(4)
int i=10, tid
#pragma omp parallel private(tid) firstprivate(i)
int tid_thread0
int i_thread0 =10
int tid_thread1
int i_thread1=10
int tid_thread2
int i_thread2=10
int tid_thread3
int i_thread3=10
tid_thread0=0
tid_thread1=1
tid_thread2=2
tid_thread3=3
i_thread0=20
i_thread1=20
i_thread2=20
i_thread3=20
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
printf(tid=%d i=%d₩n", tid, i);


Memory
tid
0
1
2
3
i
10
10
10
10
10

데이터 유효범위(6/8)

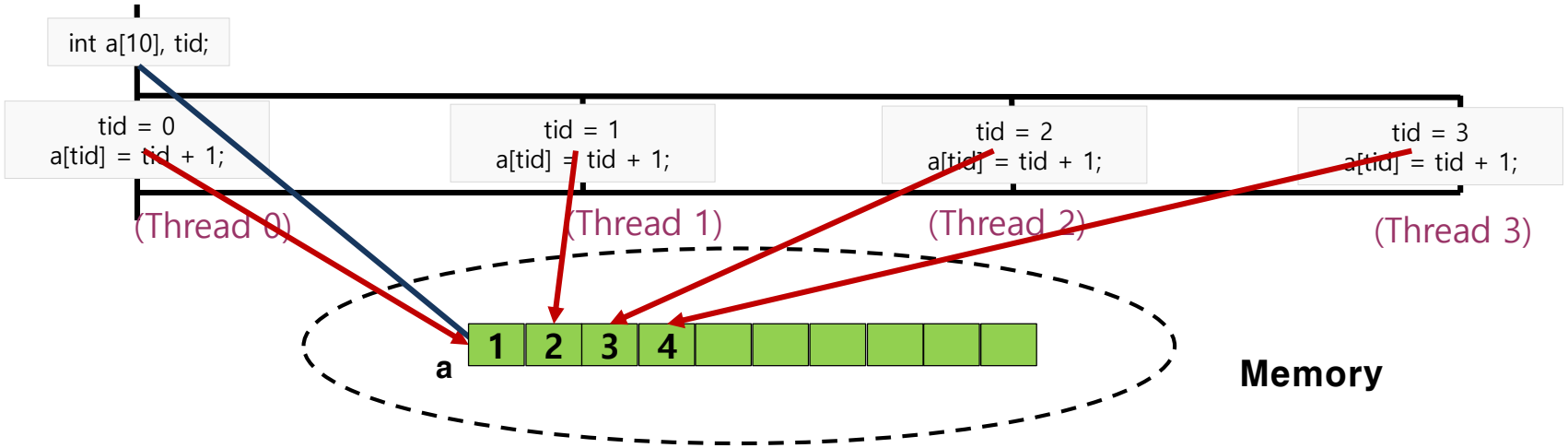

omp_set_num_threads(4)
int i=10, tid
#pragma omp parallel private(tid) firstprivate(i)
int tid_thread0
int i_thread0 =10
int tid_thread1
int i_thread1=10
int tid_thread2
int i_thread2=10
int tid_thread3
int i_thread3=10
tid_thread0=0
tid_thread1=1
tid_thread2=2
tid_thread3=3
i_thread0=20
i_thread1=20
i_thread2=20
i_thread3=20
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
printf(tid=%d i=%d₩n", tid, i);
Memory
tid
0
1
2
3
i
10
20
20
20
20

데이터 유효범위(6/8)


omp_set_num_threads(4)
int i=10, tid
#pragma omp parallel private(tid) firstprivate(i)
int tid_thread0
int i_thread0 =10
int tid_thread1
int i_thread1=10
int tid_thread2
int i_thread2=10
int tid_thread3
int i_thread3=10
tid_thread0=0
tid_thread1=1
tid_thread2=2
tid_thread3=3
i_thread0=20
i_thread1=20
i_thread2=20
i_thread3=20
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
printf(tid=%d i=%d₩n", tid, i);
Memory
tid
10
i

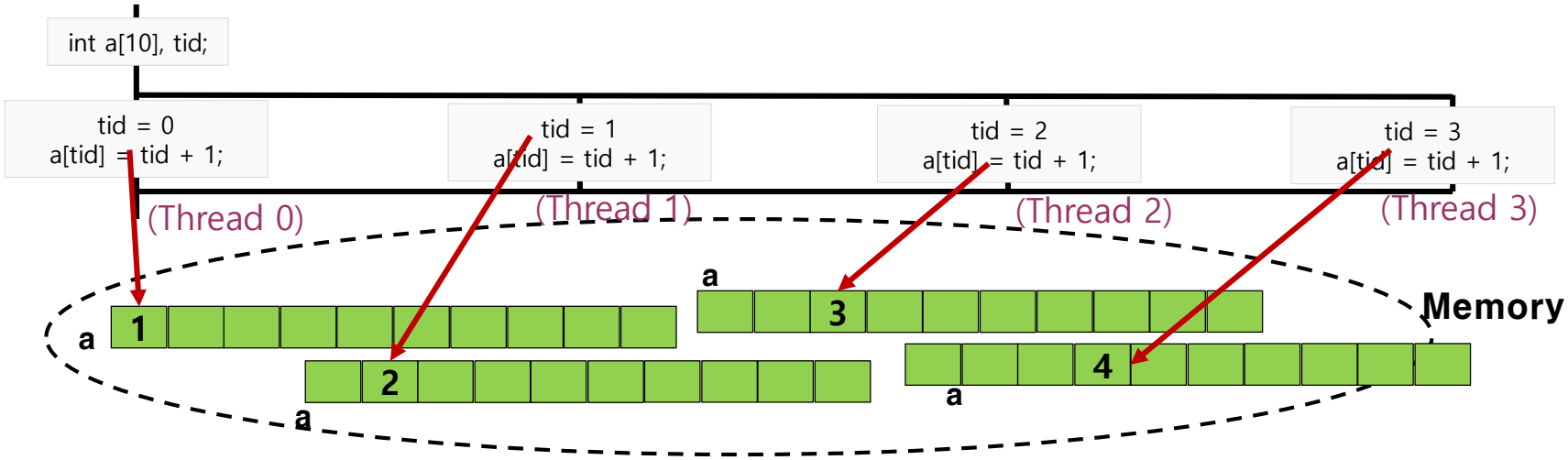
데이터 유효범위(7/8)

Fortran	C
PROGRAM data_scope_shared INTEGER a(0:9), tid, i, omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL shared(a) PRIVATE(tid) tid = OMP_GET_THREAD_NUM() a(tid) = tid + 1 !$OMP END PARALLEL do i=0, 3 print *, 'a(', i, ') =', a(i) end do END	#include <stdio.h> #include <omp.h> int main() { int a[10], tid, i; omp_set_num_threads(4); #pragma omp parallel **shared**(a) private(tid) { tid = omp_get_thread_num(); a[tid] = tid + 1; } for(i=0; i<4; i++) printf("a[%d] = %d\n", i, a[i]); }



데이터 유효범위(8/8)

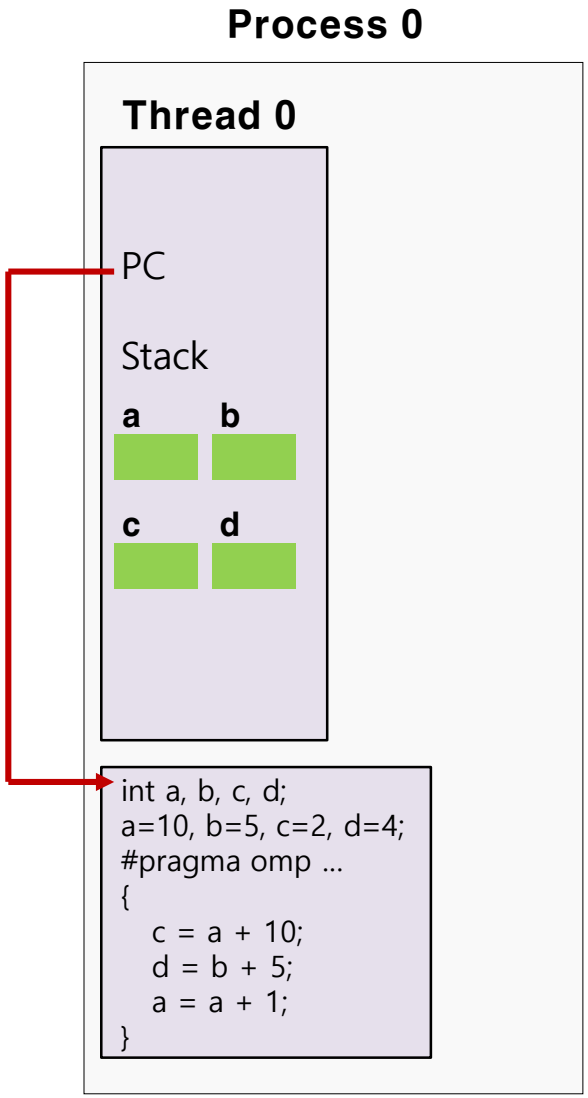
Fortran	C
PROGRAM data_scope_private_array INTEGER a(0:9), tid, i, omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL **private**(a) PRIVATE(tid) tid = OMP_GET_THREAD_NUM() a(tid) = tid + 1 !$OMP END PARALLEL do i=0, 3 print *, 'a(', i, ') =', a(i) end do END	#include <stdio.h> #include <omp.h> int main() { int a[10], tid, i; omp_set_num_threads(4); #pragma omp parallel **private**(a) private(tid) { tid = omp_get_thread_num(); a[tid] = tid + 1; } for(i=0; i<4; i++) printf("a[%d] = %d\n", i, a[i]); }



스레드와 프로세스

```
void foo()
{
   int a, b, c, d;
   a = 10;
   b = 5;
   c = 2;
   d = 4;

   #pragma omp parallel
  \
     private(c,d) \
     num_threads(2)
   {
      c = a + 10;
      d = b + 5;
      a = a + 1;
   }
}
```

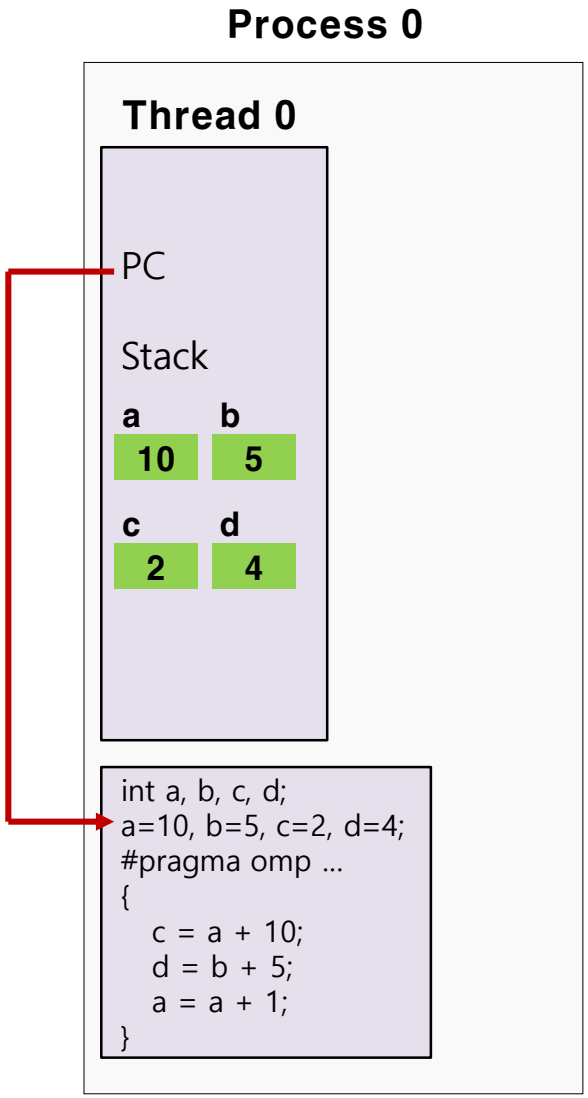
```
$ gcc  foo.c
$ ./a.out
```



스레드와 프로세스

```
void foo()
{
   int a, b, c, d;
   a = 10;
   b = 5;
   c = 2;
   d = 4;

   #pragma omp parallel
  \
     private(c,d) \
     num_threads(2)
   {
      c = a + 10;
      d = b + 5;
      a = a + 1;
   }
}
```

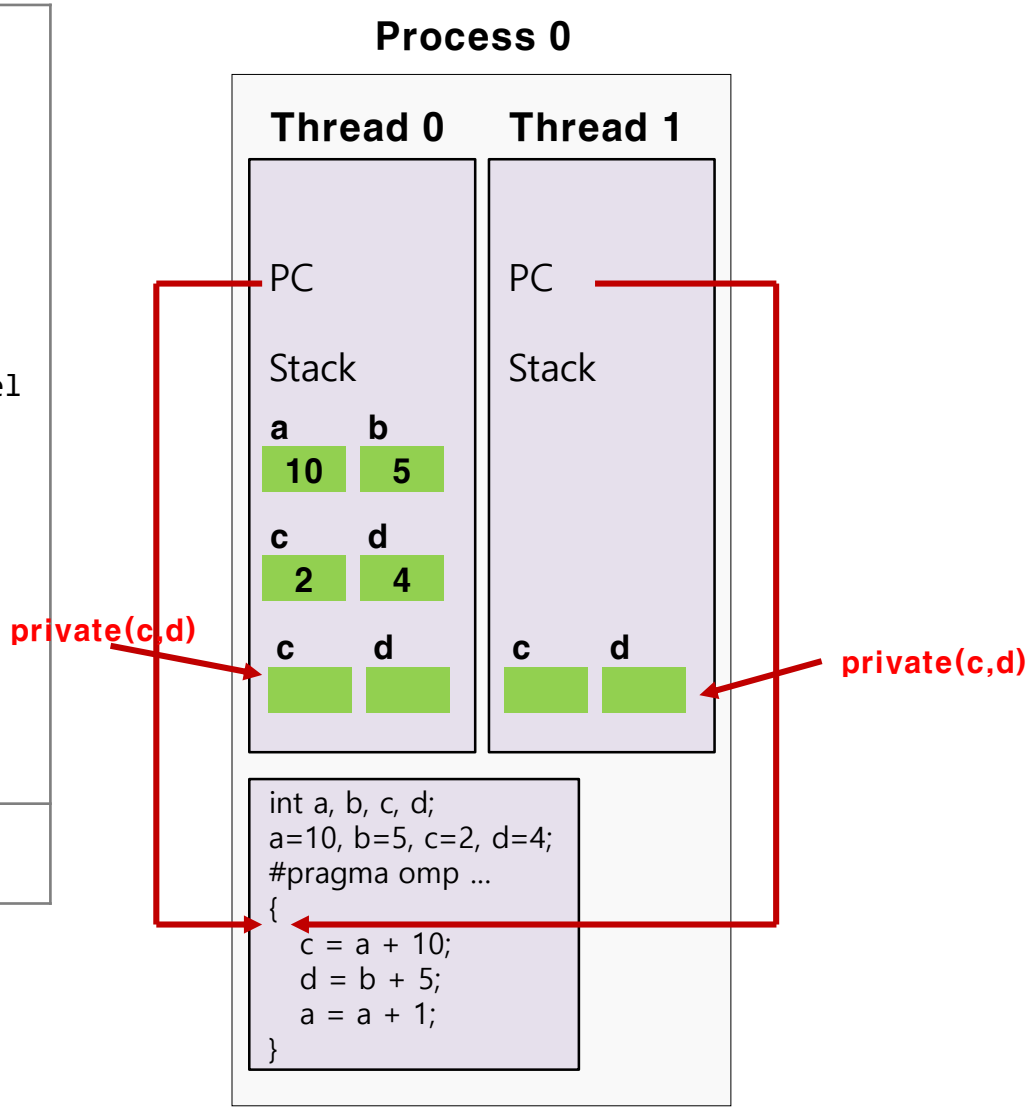
```
$ gcc  foo.c
$ ./a.out
```



스레드와 프로세스

```
void foo()
{
   int a, b, c, d;
   a = 10;
   b = 5;
   c = 2;
   d = 4;

   #pragma omp parallel
  \
     private(c,d) \
     num_threads(2)
   {
      c = a + 10;
      d = b + 5;
      a = a + 1;
   }
}
```

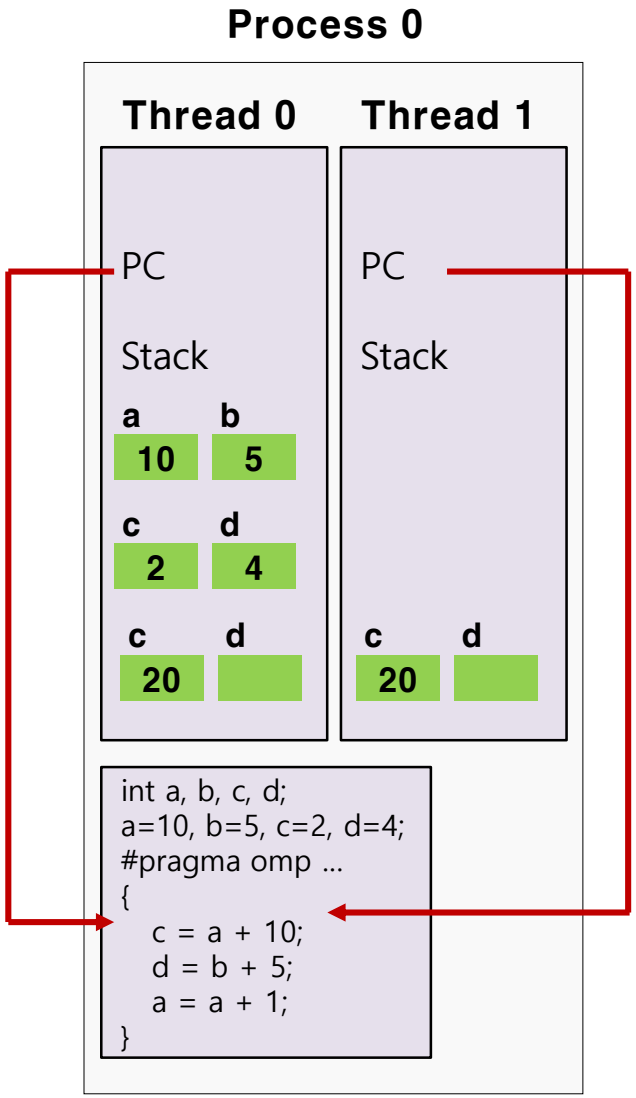
```
$ gcc  foo.c
$ ./a.out
```



스레드와 프로세스

```
void foo()
{
   int a, b, c, d;
   a = 10;
   b = 5;
   c = 2;
   d = 4;

   #pragma omp parallel
  \
     private(c,d) \
     num_threads(2)
   {
      c = a + 10;
      d = b + 5;
      a = a + 1;
   }
}
```

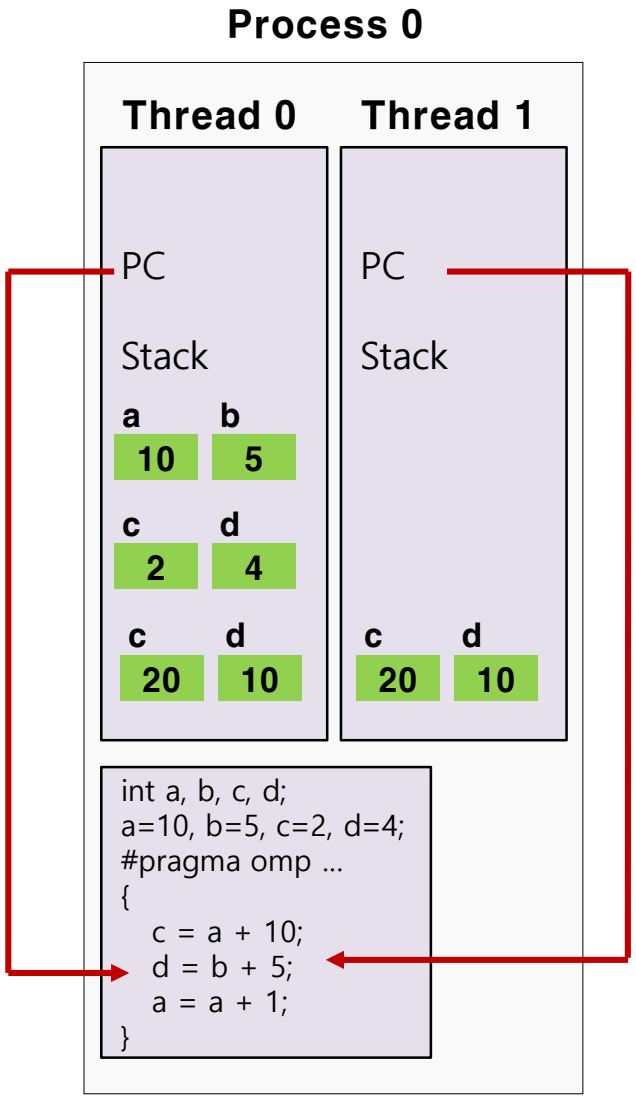
```
$ gcc  foo.c
$ ./a.out
```



스레드와 프로세스

```
void foo()
{
   int a, b, c, d;
   a = 10;
   b = 5;
   c = 2;
   d = 4;

   #pragma omp parallel
  \
     private(c,d) \
     num_threads(2)
   {
      c = a + 10;
      d = b + 5;
      a = a + 1;
   }
}
```

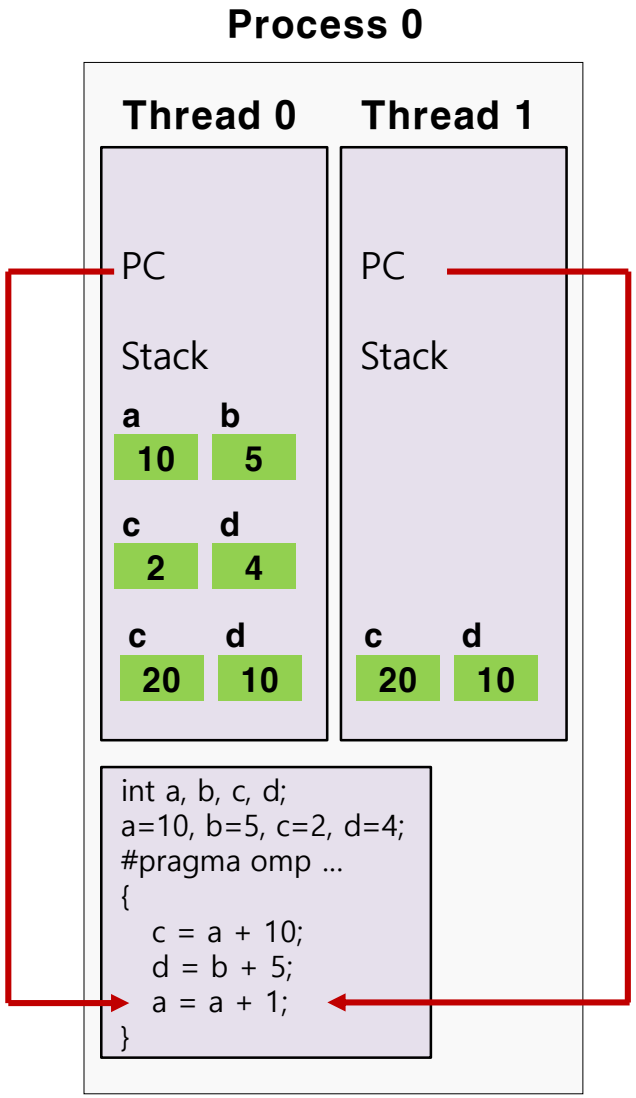
```
$ gcc  foo.c
$ ./a.out
```



스레드와 프로세스

```
void foo()
{
   int a, b, c, d;
   a = 10;
   b = 5;
   c = 2;
   d = 4;

   #pragma omp parallel
  \
     private(c,d) \
     num_threads(2)
   {
      c = a + 10;
      d = b + 5;
      a = a + 1;
   }
}
```

```
$ gcc  foo.c
$ ./a.out
```



Thread 0

```
MEMORY :
 a = 10
CPU :
 a = 10 + 1
 a = 11
MEMORY:
 a = 11
```

Thread 1

```
MEMORY :
 a = 10
CPU :
 a = 10 + 1
 a = 11
MEMORY :
 a = 11
```

Thread 0

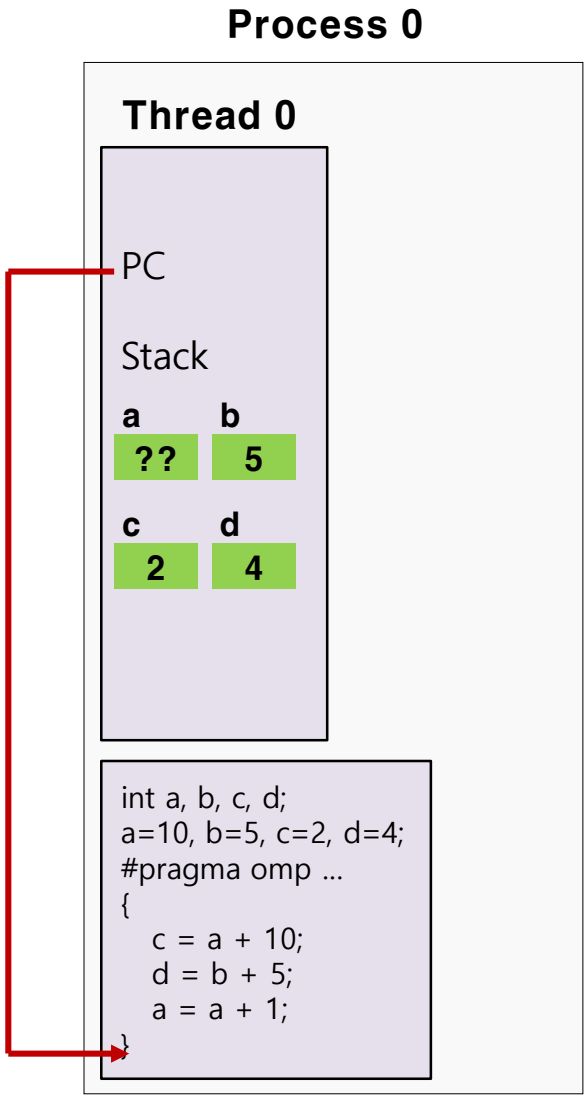
```
MEMORY:
 a = 10
CPU :
 a = 10 + 1
 a = 11
MEMORY:
 a = 11
```

Thread 1

```
MEMORY:
 a = 11
CPU :
 a = 11 + 1
 a = 12
MEMORY:
 a = 12
```

```
void foo()
{
   int a, b, c, d;
   a = 10;
   b = 5;
   c = 2;
   d = 4;

   #pragma omp parallel
  \
     private(c,d) \
     num_threads(2)
   {
      c = a + 10;
      d = b + 5;
      a = a + 1;
   }
}
```

```
$ gcc  foo.c
$ ./a.out
```



학습 요약

데이터 유효범위

- private, firstprivate, shared

데이터 유효범위 사용 예

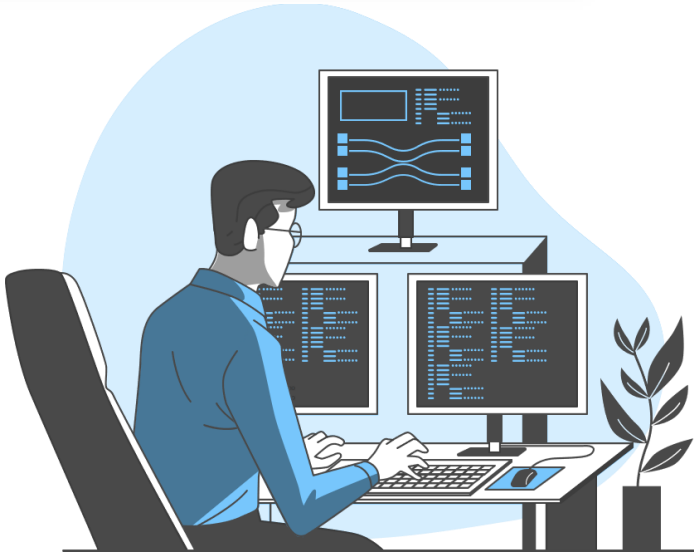
프로세스와 스레드

- 하나의 프로세스는 여러 개의 스레드를 가질 수 있음

- 데이터 유효범위
- 데이터 유효범위 사용 예
- 프로세스와 스레드

04 OpenMP Basic III

1. 루프 병렬화 소개
2. 루프 병렬화
3. 코드 설명(inner product)



학습목표

- 루프 병렬화를 이해한다.
- 루프 병렬화를 이용하여 병렬 코드를 작성할 줄 안다.
- Inner product를 OpenMP 병렬 코드로 작성할 줄 안다.
  - 미완성 코드
  - 동기화 필요 ➔ reduction 연산 필요

병렬 루프

Fortran	C
PROGRAM serial_loop INTEGER, PARAMETER :: N=20 INTEGER :: tid=0, i, istart=0, iend=N-1 DO i=istart, iend PRINT *, 'Hello World', tid, i END DO END	#include <stdio.h> #define N 20 int main() { int tid = 0; int i; int istart=0, iend=N; for(i=istart; i<iend; i++) printf("Hello World %d %d\n", tid, i); }

병렬 루프

Fortran

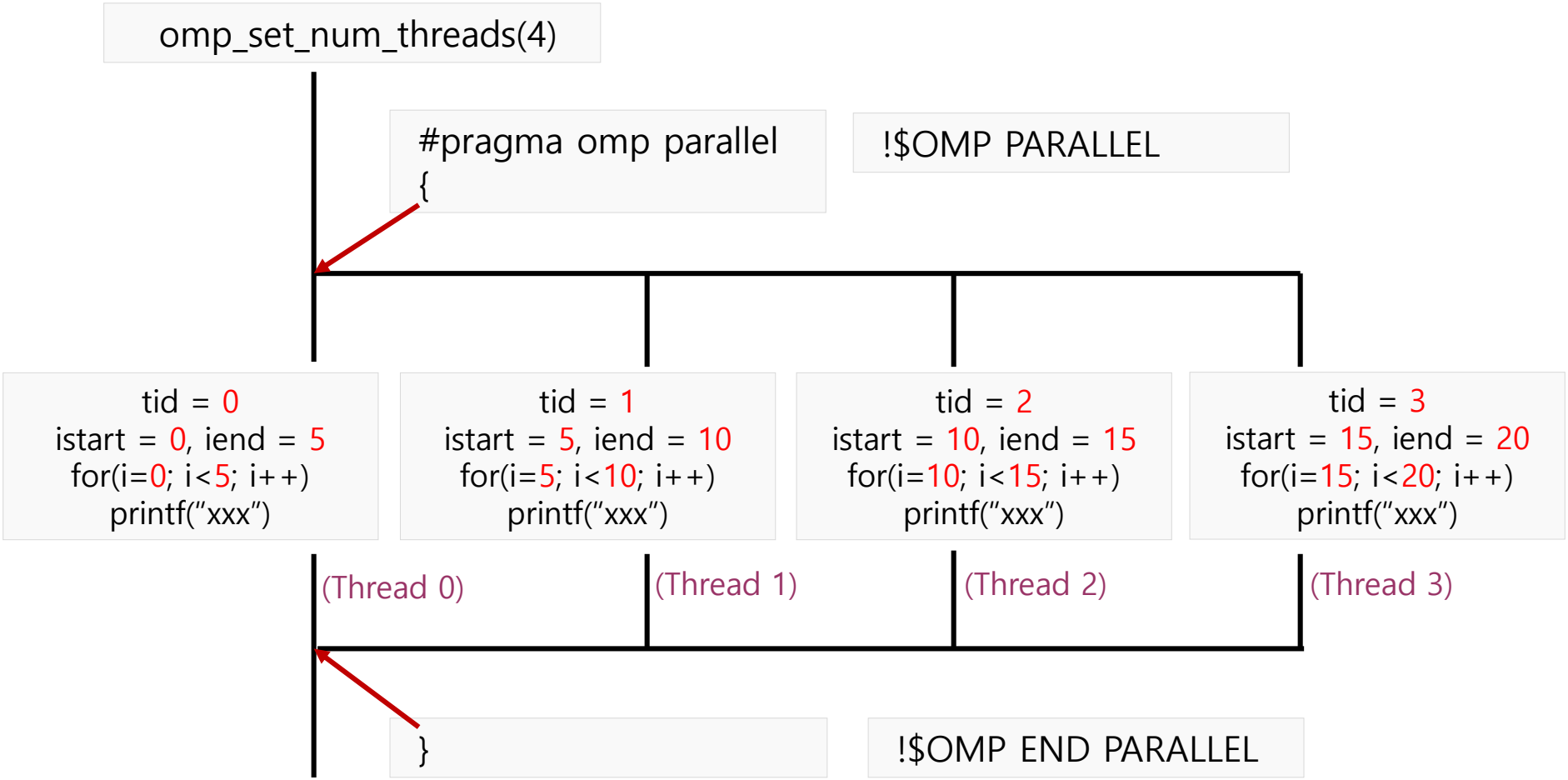
```
PROGRAM parallel_loop
   INTEGER, PARAMETER :: N=20

   INTEGER :: tid, i, istart, iend,
  omp_get_thread_num
   call omp_set_num_threads(4)
!$OMP PARALLEL private(tid, istart, iend)
   tid = omp_get_thread_num()
   istart=???; iend=???   !! hint : tid, N,
  4
   DO i=istart, iend
      PRINT *, 'Hello World', tid, i
   END DO
!$OMP END PARALLEL
END
```

C

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

int main()
{
  int tid, i, istart, iend
omp_set_num_threads(4);
#pragma omp parallel private(i, tid,
  istart, iend)
{
   tid = omp_get_thread_num();
   istart=???, iend=???;  // hint : tid, N,
  4
   for(i=istart; i<iend; i++)
     printf("Hello World %d %d\n", tid, i);
}
}
```

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

병렬 루프


omp_set_num_threads(4)
#pragma omp parallel
{
!$OMP PARALLEL
tid = 0
istart = 0, iend = 5
for(i=0; i<5; i++)
printf("xxx")
tid = 1
istart = 5, iend = 10
for(i=5; i<10; i++)
printf("xxx")
tid = 2
istart = 10, iend = 15
for(i=10; i<15; i++)
printf("xxx")
tid = 3
istart = 15, iend = 20
for(i=15; i<20; i++)
printf("xxx")
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL

병렬 루프

Fortran

```
PROGRAM parallel_loop
   INTEGER, PARAMETER  :: N=20

   INTEGER::tid, i, istart, iend,
  omp_get_thread_num

   call omp_set_num_threads(4)
!$OMP PARALLEL private(tid, istart, iend)
   tid = omp_get_thread_num()

   istart = tid * N / 4
   iend = (tid+1) * N / 4 - 1

   DO i=istart, iend
      PRINT *, 'Hello World', tid, i
   END DO
!$OMP END PARALLEL
END
```

C

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

int main()
{
int i, tid, istart, iend;
omp_set_num_threads(4);
#pragma omp parallel private(tid, istart,
  iend)
{
   tid = omp_get_thread_num();
   istart= tid * N / 4;
   iend= (tid+1) * N / 4;
   for(i=istart; i<iend; i++)
     printf("Hello World %d %d\n", tid,
  i);
}
}
```

병렬 루프

Fortran

```
PROGRAM parallel_loop
  INTEGER, PARAMETER  :: N=20

  INTEGER  :: tid, i, istart, iend,
  nthreads &
  omp_get_thread_num,
  omp_get_num_threads

  call omp_set_num_threads(4)
!$OMP PARALLEL private(tid, istart,
  iend)
  tid = omp_get_thread_num()
  nthreads = omp_get_num_threads()

  istart = tid * N / nthreads
  iend = (tid+1) * N / nthreads - 1

  DO i=istart, iend
     PRINT *, 'Hello World', tid, i
  END DO
!$OMP END PARALLEL
END
```

C

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

int main()
{
   int i, tid, istart, iend, nthreads;
   omp_set_num_threads(4);
#pragma omp parallel private(tid,
  istart, iend)
{
   tid = omp_get_thread_num();
   nthreads = omp_get_num_threads();
   istart= tid * N / nthreads;
   iend= (tid+1) * N / nthreads;
   for(i=istart; i<iend; i++)
     printf("Hello World %d %d\n",
  tid, i);
}
}
```

병렬 루프

Fortran

```
PROGRAM parallel_loop
  INTEGER, PARAMETER  :: N=20

  INTEGER  :: tid, i, istart, iend,
  nthreads &
  omp_get_thread_num,
  omp_get_num_threads

  call omp_set_num_threads(4)
!$OMP PARALLEL private(tid, istart,
  iend)
  tid = omp_get_thread_num()
  nthreads = omp_get_num_threads()

  istart = tid*N / nthreads
  iend = (tid+1)*N / nthreads - 1

  DO i=0istart, N-1iend
     PRINT *, 'Hello World', tid, i
  END DO
!$OMP END PARALLEL
END
```

C

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

int main()
{
   int i, tid, istart, iend, nthreads;
   omp_set_num_threads(4);
#pragma omp parallel private(tid,
  istart, iend)
{
   tid = omp_get_thread_num();
   nthreads = omp_get_num_threads();
   istart= tid * N / nthreads;
   iend= (tid+1) * N / nthreads;
   for(i=0istart; i<Niend; i++)
     printf("Hello World %d %d\n",
  tid, i);
}
}
```

for 지시어

- #pragma omp for 지시어 다음에 오는 for 루프문을 병렬화한다
- for 루프문의 반복 작업을 자동으로 배분하는 기능 제공
- 기본 schedule은 static
  - 반복 루프 수행 시 총 횟수를 스레드 개수로 나눠 스레드별로 동일 배분

❖ For 루프의 인덱스 변수는 스레드의 private 변수로 선언(기본값)

```
#pragma omp for [보조 지시어 [,보조 지시어 …]]
  for (초기값; 조건식; 증감값)
  {
          구조블록 // 코드 작성
  }
```

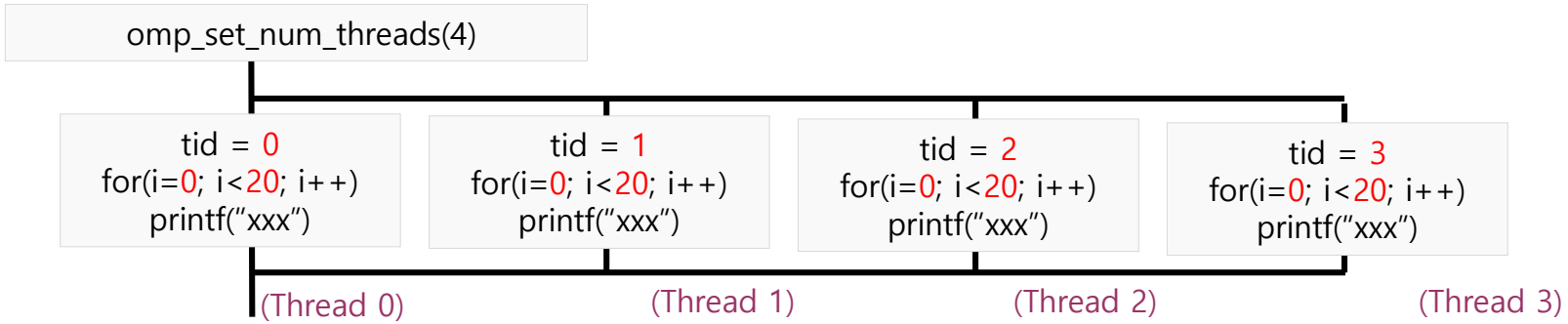
병렬 루프

Fortran	C
PROGRAM parallel_for INTEGER, PARAMETER :: N=20 INTEGER :: tid, i, omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL private(tid) tid = omp_get_thread_num() **!$OMP DO** DO i=**0, N-1** PRINT *, 'Hello World', tid, i END DO **!$OMP END DO !!![optional]** !$OMP END PARALLEL END	#include <stdio.h> #include <omp.h> #define N 20 int main() { int tid, i; omp_set_num_threads(4); #pragma omp parallel private(tid) { tid = omp_get_thread_num(); **#pragma omp for** for(i=**0**; i<**N**; i++) printf("Hello World %d %d\n", tid, i); } }

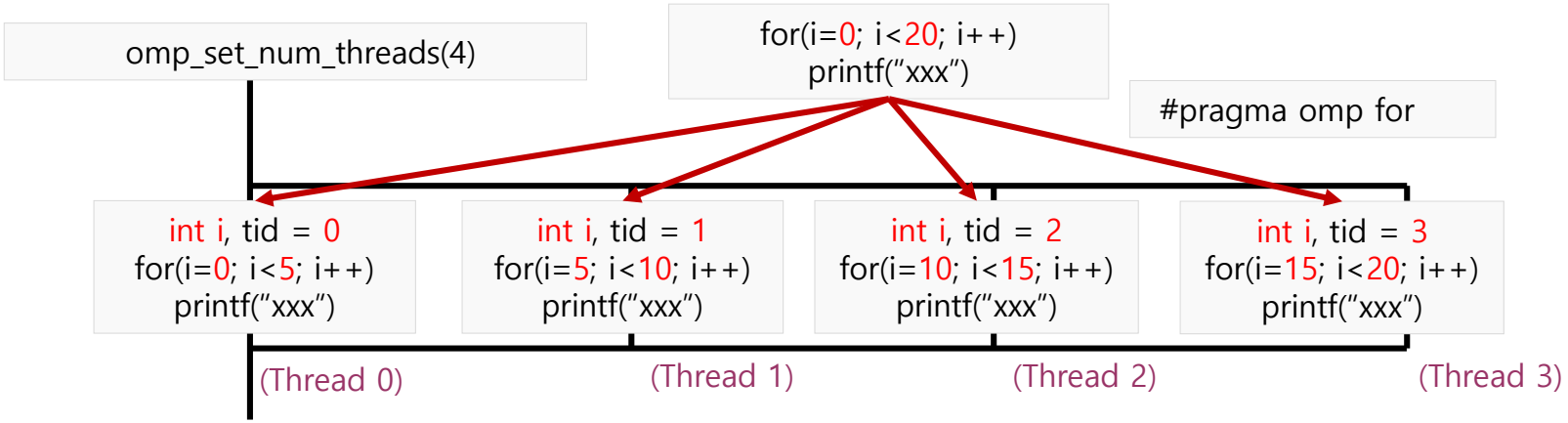
병렬 루프

Fortran	C
PROGRAM parallel_for INTEGER, PARAMETER :: N=20 INTEGER :: tid, i, omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL private(tid) !!! i : shared?? tid = omp_get_thread_num() !$OMP DO DO i=0, N-1 PRINT *, 'Hello World', tid, i END DO !$OMP END DO !!![optional] !$OMP END PARALLEL END	#include <stdio.h> #include <omp.h> #define N 20 int main() { int tid, i; omp_set_num_threads(4); #pragma omp parallel private(tid) // i : shared? { tid = omp_get_thread_num(); #pragma omp for for(i=0; i<N; i++) printf("Hello World %d %d\n", tid, i); } }

Without #pragma omp for



With #pragma omp for



작업 분배

- 스레드들 사이의 작업 분배
- Do/for, Sections/section, Single, Task construct, etc.
- cf) WORKSHARE clause in Fortran

Fortran	C
!$OMP PARALLEL private(tid) tid = omp_get_thread_num() DO i=0, N-1 print *, “Hello World”, tid, i END DO !$OMP END PARALLEL	#pragma omp parallel private(tid) { tid = omp_get_thread_num(); for(i=0; i<N; i++) printf(“Hello World %d %d\n”, tid, i); }
!$OMP PARALLEL private(tid) tid = omp_get_thread_num() !$OMP DO DO i=0, N-1 print *, “Hello World”, tid, i END DO !$OMP END DO [optional] !$OMP END PARALLEL	#pragma omp parallel private(tid) { tid = omp_get_thread_num(); #pragma omp for for(i=0; i<N; i++) printf(“Hello World %d %d\n”, tid, i); }

OpenMP shortcut

- 결합된 병렬 작업 분할 지시어

Fortran	C
!$OMP PARALLEL !$OMP DO DO i=0, N-1 print *, “Hello World”, i END DO !$OMP END PARALLEL	#pragma omp parallel { #pragma omp for for(i=0; i<N; i++) printf(“Hello World %d \n”, i); }
!$OMP PARALLEL DO DO i=0, N-1 print *, “Hello World”, i END DO !$OMP END PARALLEL DO	#pragma omp parallel for for(i=0; i<N; i++) printf(“Hello World %d %d\n”, i);

Implement Inner(dot) Product (Serial)

Fortran

```
PROGRAM inner_product

  INTEGER, PARAMETER :: N=20
  INTEGER :: i, sum=0
  INTEGER, DIMENSION(0:N-1) :: A, B

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

  DO i=0, N-1
     sum = sum + A(i) * B(i)
  END DO

  print *, "sum =", sum
END
```

C

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

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

   for(i=0; i<N; i++)
     sum += a[i] * b[i];

   printf("sum = %d\n", sum);
}
```

Implement Inner(dot) Product (Parallel)

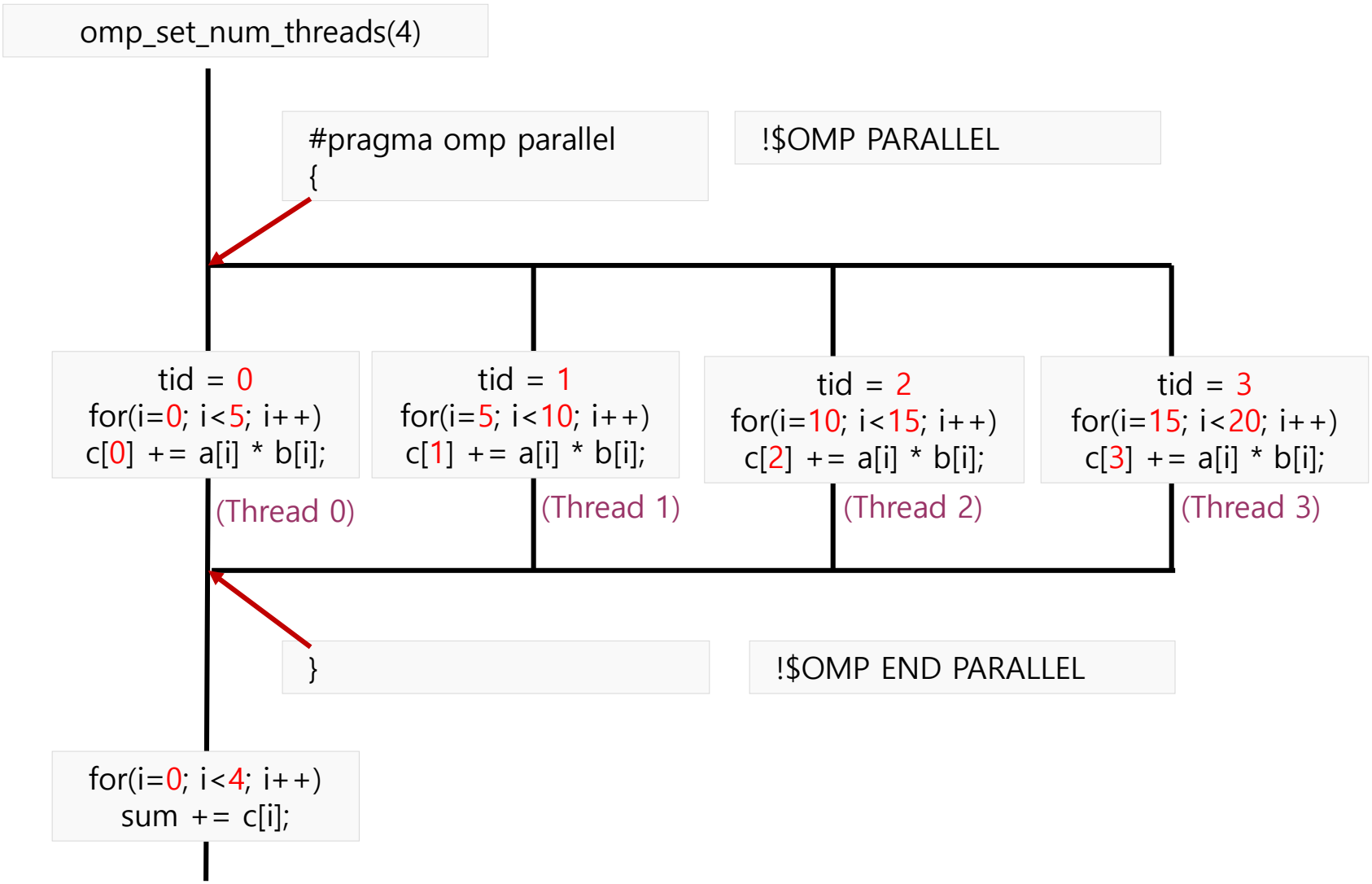
Fortran

```
PROGRAM inner_product

  INTEGER, PARAMETER :: N=20
  INTEGER :: i, tid, sum=0
  INTEGER, DIMENSION(0:N-1) :: A, B
  INTEGER, DIMENSION(0:3)  :: C
  DO i=0, N-1
      A(i) = i+1; b(i)= i+2
  ENDDO
!! set # of threads = 4
!! create threads
  tid = !! get thread index
  C(tid) = 0
!! worksharing
  DO i=0, N-1
     C(tid) = C(tid) + A(i) * B(i)
  END DO
!! join threads
   DO i=0, 3
        sum = sum+C(i)
   END DO
   print *, "sum =", sum
END
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 20
int main(){
   int i, tid, sum=0;
   int a[N], b[N], c[4];
   for(i=0; i<N; i++) {
      a[i] = i+1; b[i] = i+2;
   }
// set # of threads = 4
// create threads
{
   tid = // get thread index
   c[tid] = 0;
// worksharing
   for(i=0; i<N; i++)
     c[tid] += a[i] * b[i];
}
   for(i=0; i<4; i++)
      sum += c[i];

   printf("sum = %d\n", sum);
}
```

병렬 루프


omp_set_num_threads(4)
#pragma omp parallel
{
!$OMP PARALLEL
tid = 0
for(i=0; i<5; i++)
c[0] += a[i] * b[i];
tid = 1
for(i=5; i<10; i++)
c[1] += a[i] * b[i];
tid = 2
for(i=10; i<15; i++)
c[2] += a[i] * b[i];
tid = 3
for(i=15; i<20; i++)
c[3] += a[i] * b[i];
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL
for(i=0; i<4; i++)
sum += c[i];

Implement BLAS1 Inner(dot) Product

Fortran

```
PROGRAM inner_product
  INTEGER, PARAMETER :: N=20
  INTEGER :: i, tid, sum=0,  omp_get_thread_num
  INTEGER, DIMENSION(0:N-1) :: A, B
  INTEGER, DIMENSION(0:3)  :: C

    DO i=0, N-1
        A(i) = i+1; b(i)= i+2
    END DO

    call omp_set_num_threads(4)
!$OMP PARALLEL private(tid)
    tid = omp_get_thread_num()
    C(tid) = 0
    !$OMP DO
    DO i=0, N-1
        C(tid) = C(tid) + A(i) * B(i)
    END DO
!$OMP END PARALLEL

    DO i=0, 3
         sum = sum+C(i)
    END DO
    print *, "sum =", sum
END
```

C

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

int main(){
   int i, tid, sum=0;
   int a[N], b[N],c[4];
   for(i=0; i<N; i++) {
      a[i] = i+1; b[i] = i+2;
   }
omp_set_num_threads(4);
#pragma omp parallel private(tid)
{
   tid = omp_get_thread_num();
   c[tid] = 0;
#pragma omp for
   for(i=0; i<N; i++)
     c[tid] += a[i] * b[i];
}
   for(i=0; i<4; i++)
      sum += c[i];

   printf("sum = %d\n", sum);
}
```

학습 요약

루프 병렬화 소개

- 루프 병렬화가 필요한 이유

루프 병렬화

- #pragma omp for

Inner product 계산

- 결과는 나오지만 원본 코드가 많이 손상되었음

루프 병렬화가 필요한 이유

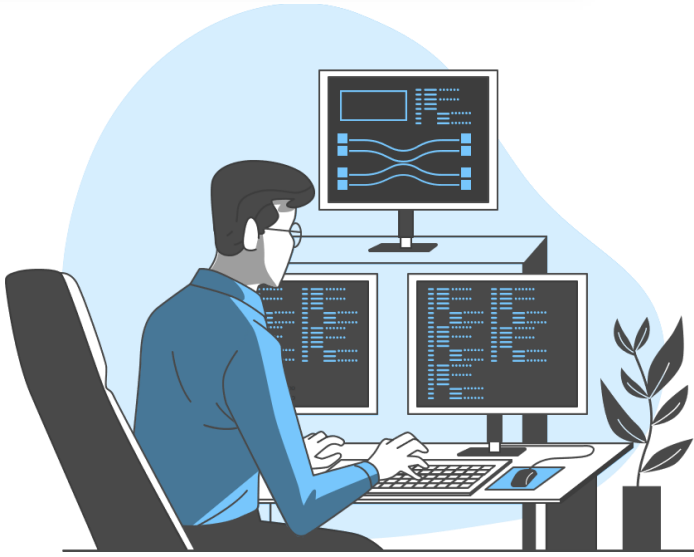
- 루프 병렬화가 필요한 이유

루프 병렬화

Inner product 계산

05 OpenMP Basic IV

1. 동기화 소개
2. Critical, atomic, barrier
3. 코드 설명(동기화)



학습목표

- 동기화의 필요성을 이해한다.
- critical, atomic, barrier을 적절히 사용할 줄 안다.
- Inner product 코드 개선

Implement BLAS1 Inner(dot) Product

Fortran

```
PROGRAM inner_product
  INTEGER, PARAMETER :: N=20

  INTEGER :: i, tid, sum, omp_get_thread_num
  INTEGER, DIMENSION(0:N-1) :: A, B
  INTEGER, DIMENSION(0:3)  :: C
  DO i=0, N-1
      A(i) = i+1; b(i)= i+2
  END DO

  call omp_set_num_threads(4)
!$OMP PARALLEL private(tid)
  tid = omp_get_thread_num()
  C[tid] = 0
!$OMP DO
  DO i=0, N-1
     C[tid] = C[tid] sum = sum + A(i) * B(i)
  END DO
!$OMP END PARALLEL

   DO i=0, 3
        sum = sum+C(i)
   END DO
   print *, "sum =", sum
END
```

C

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

int main()
{
   int i, tid, sum=0;
   int a[N], b[N],c[4];
   for(i=0; i<N; i++) {
      a[i] = i+1; b[i] = i+2;
   }
omp_set_num_threads(4);
#pragma omp parallel private(tid)
{
   tid = omp_get_thread_num();
   c[tid] = 0;
#pragma omp for
   for(i=0; i<N; i++)
     c[tid] sum += a[i] * b[i];
}
   for(i=0; i<4; i++)
      sum += c[i];

   printf("sum = %d\n", sum);
}
```

Implement BLAS1 Inner(dot) Product

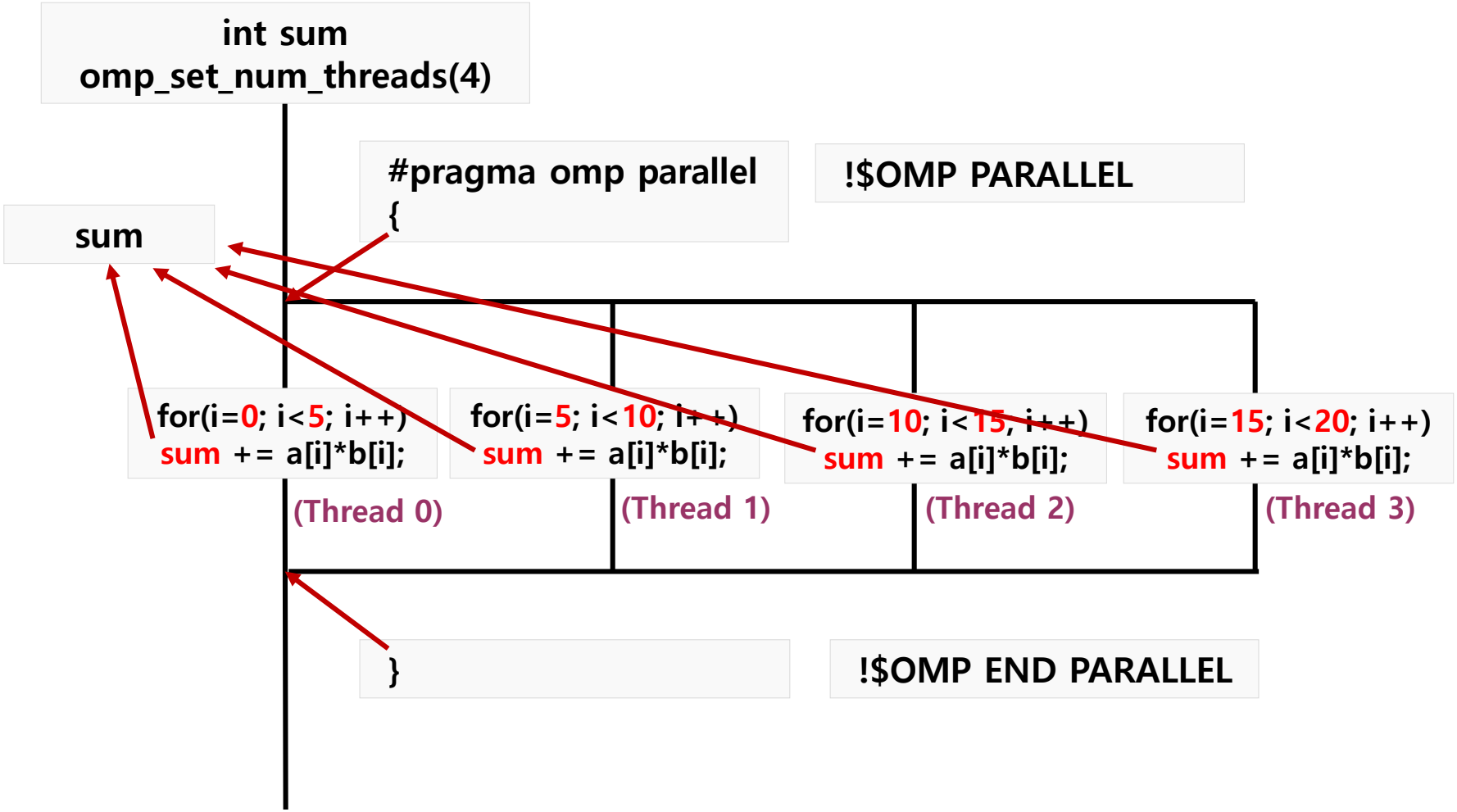
Fortran

```
PROGRAM wrong_inner_product
  INTEGER, PARAMETER :: N=20

  INTEGER :: i, tid, sum=0
  INTEGER, DIMENSION(0:N-1) :: A, B

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

    call omp_set_num_threads(4)
!$OMP PARALLEL
!$OMP DO
    DO i=0, N-1
        sum = sum + A(i) * B(i)
    END DO
!$OMP END PARALLEL
    print *, “sum =“, sum
END
```

C

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

int main()
{
   int 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++)
      sum += a[i] * b[i];
}
   printf(“sum = %d\n”, sum);
}
```

동기화


int sum
omp_set_num_threads(4)
#pragma omp parallel
{
!$OMP PARALLEL
sum
for(i=0; i<5; i++)
sum += a[i]*b[i];
for(i=5; i<10; i++)
sum += a[i]*b[i];
for(i=10; i<15; i++)
sum += a[i]*b[i];
for(i=15; i<20; i++)
sum += a[i]*b[i];
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
}
!$OMP END PARALLEL

Implement BLAS1 Inner(dot) Product

Fortran

```
PROGRAM sync_critical
  INTEGER, PARAMETER :: N=20

  INTEGER :: i, tid, sum=0
  INTEGER, DIMENSION(0:N-1) :: A, B

  DO i=0, N-1
      A(i) = i+1; b(i)= i+2
  END DO

    call omp_set_num_threads(4)
!$OMP PARALLEL
    !$OMP DO
    DO i=0, N-1
        !$OMP CRITICAL (name1)
        sum = sum + A(i) * B(i)
        !$OMP END CRITICAL
    END DO
!$OMP END PARALLEL

    print *, "sum =", sum
END
```

C

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

int main()
{
   int 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 critical (name1)
      sum += a[i] * b[i];
}
   printf("sum = %d\n", sum);
}
```

Implement BLAS1 Inner(dot) Product

Fortran

```
PROGRAM sync_atomic
  INTEGER, PARAMETER :: N=20

  INTEGER :: i, tid, sum=0
  INTEGER, DIMENSION(0:N-1) :: A, B

  DO i=0, N-1
      A(i) = i+1; b(i)= i+2
  END DO

    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
```

C

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

int main()
{
   int 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 = %d\n”, sum);
}
```

동기화

Synchronization
- 스레드에 할당된 작업 실행 순서에 제한을 둘 수 있음
- 공유 데이터에 대한 접근을 제어

고 수준 동기화
- critical, atomic, barrier, ordered

저 수준 동기화
- flush, locks

critical vs atomic
- critical : 한번에 하나의 스레드만이 이름으로 구분되는 critical 영역에 진입
- atomic : 한 메모리 위치에만 적용됨, mini-critical section 역할

Fortran

```
!$OMP PARALLEL
  DO i=0, 99
!$OMP CRITICAL(n1)
  call sub(a,b)
!$OMP END CRITICAL(n1)
  END DO
!$OMP END PARALLEL
```

```
!$OMP PARALLEL
   DO i=0, 99
!$OMP ATOMIC
        a=a+b
   END DO
!$OMP END PARALLEL
```

C

```
#pragma omp parallel
{
   for(i=0; i<100; i++)
   {
#pragma omp critical(n1)
      func1(a,b);
   }
}
```

```
#pragma omp parallel
{
    for(i=0; i<100; i++)
   {
#pragma omp atomic
       a += b;
   }
}
```

critical 지시어

- #pragma omp critical [(크리티컬 객체 이름)] 형식으로 지정 사용
- 같은 크리티컬 객체 명으로 지정된 경우 영역은 한번에 하나의 스레드만 실행
- 다른 스레드가 같은 이름의 크리티컬 객체 사용 시, 먼저 실행한 스레드 동작이 완료될 때까지 진입 지점에서 대기
- atomic 지시어로 지정할 수 없던 수식도 critical 영역으로 지정 가능

atomic 지시어

- #pragma omp atomic 형식으로 지정 사용
- 여러 개의 스레드가 하나의 공유 변수를 동시에 변경할 때 발생하는 경쟁상태 방지
- 공유변수에 대하여, 읽기/쓰기가 한번에 이루어지는 연산자에서만 사용
- +, -, *, /, &, |, ^, <<, >> 연산자일 경우 사용 가능
- atomic 구문에서는 {중괄호}를 사용할 수 없다

BARRIER

- 모든 스레드들이 barrier에 도달할 때까지 대기
- 스레드팀의 모든 스레드 동기화
- 작업분할 구문 내에서 사용할 수 없음(if문, while문, for문, switch문)

Fortran

```
PROGRAM barrier
   INTEGER :: x=1

   call omp_set_num_threads(4)

!$OMP PARALLEL shared(x)
    x=x+1
    !$OMP BARRIER
    print *, ‘x =‘, x
!$OMP END PARALLEL

END
```

C

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

int main()
{
   int x=1;
   omp_set_num_threads(4);
#pragma omp parallel shared(x)
{
   x++;
#pragma omp barrier
   printf (“x = %d\n”, x);
}
}
```

Exercise

```
PROGRAM sync_exercise
   implicit none
   integer, parameter :: N=100
   integer :: i, sum=0, local_sum

!! set number of threads
!! create threads
   local_sum = 0
   DO i=1, N
      local_sum = local_sum + i
   END DO
      sum = sum + local_sum

!! join threads
   print *, 'sum =', sum

END
```

```
#include <stdio.h>
#define N 100

int main()
{
   int i, sum = 0, local_sum;

   // set number of threads
   // create threads
{
   local_sum = 0;
   for(i=1; i<=N; i++)
      local_sum = local_sum + i;

   sum = sum + local_sum;
}
   printf("sum = %d\n", sum);
}
```

동기화 (Solution)

Solution

```
PROGRAM sync_solution
   implicit none
   integer, parameter :: N=100

   integer :: i, sum=0, local_sum

   call omp_set_num_threads(4)
!$OMP PARALLEL private(local_sum)
   local_sum = 0

   !$OMP DO
   DO i=1, N
      local_sum = local_sum + i
   END DO

   !$OMP ATOMIC
   sum = sum + local_sum
!$OMP END PARALLEL

   print *, 'sum =', sum
END
```

```
#include <stdio.h>
#include <omp.h>
#define N 100

int main()
{
   int i, sum = 0, local_sum;

   omp_set_num_threads(4);
#pragma omp parallel private(local_sum)
{
   local_sum = 0;
   #pragma omp for
   for(i=1; i<=N; i++)
      local_sum = local_sum + i;

   #pragma omp atomic
   sum = sum + local_sum;
}
   printf("sum = %d\n", sum);
}
```

학습 요약

동기화 소개

- 동기화가 필요한 이유

Critical, atomic, barrier

- Critical과 atomic을 구별해서 사용할 수 있어야 함

Inner product 계산

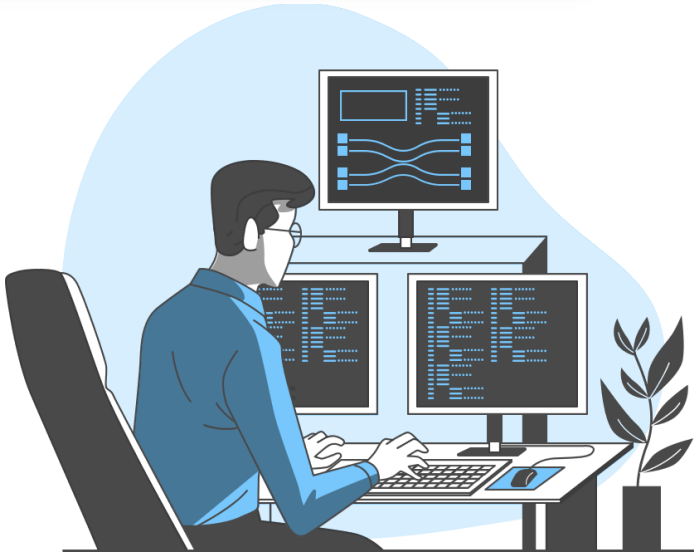
- 아직은 불완전한 코드

Outro

- 동기화 소개
- Critical, atomic, barrier
- Inner product 계산

06 OpenMP Basic V

1. reduction 소개
2. Reduction과 동기화 사용의 성능 차이 비교
3. 코드 설명(factorial)



학습목표

- Reduction의 필요성을 이해한다.
- Inner product 예제를 통해 critical, atomic, reduction 사용시 성능 차이를 이해한다.
- Reduction을 이용하여 factorial 계산을 할 줄 안다.

Implement BLAS1 Inner(dot) Product

Fortran

```
PROGRAM sync_atomic
  INTEGER, PARAMETER :: N=200000
  INTEGER :: i, tid, sum=0
  INTEGER, DIMENSION(0:N-1) :: A, B

  DO i=0, N-1
      A(i) = i+1; b(i)= i+2
  END DO
  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
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 200000
int main(){
   int 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 = %d\n", sum);
}
```

Implement BLAS1 Inner(dot) Product

Fortran

```
PROGRAM reduction
   INTEGER, PARAMETER :: N=200000
   INTEGER :: i, sum=0
   INTEGER, DIMENSION(0:N-1) :: A, B

   DO i=0, N-1
      A(i) = i+1; b(i)= i+2
   END DO

   call omp_set_num_threads(4)
!$OMP PARALLEL
   !$OMP DO reduction(+: sum)
   DO i=0, N-1
       sum = sum + A(i) * B(i)
   END DO
!$OMP END PARALLEL

   print *, "sum =", sum
END
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 200000

int main(){
   int 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 reduction (+:sum)
   for(i=0; i<N; i++)
      sum += a[i] * b[i];
}
   printf("sum = %d\n", sum);
}
```

Implement BLAS1 Inner(dot) Product

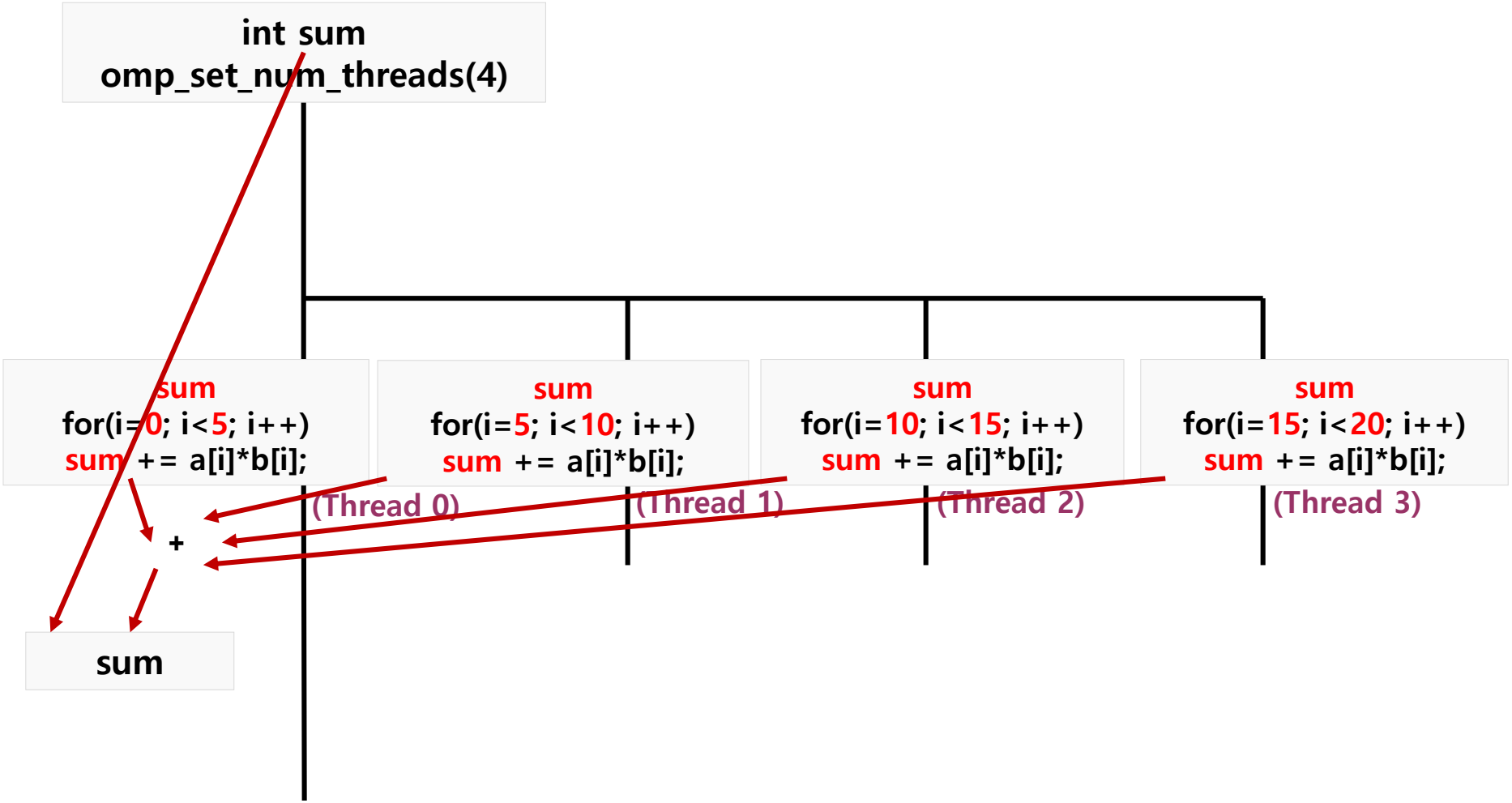
Fortran

```
PROGRAM reduction_shortcut
   INTEGER, PARAMETER :: N=200000

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

   DO i=0, N-1
      A(i) = i+1; b(i)= i+2
   END DO

   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
```

C

```
#include <stdio.h>
#include <omp.h>
#define N 200000

int main()
{
   int 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 = %d\n", sum);
}
```

Reduction


int sum
omp_set_num_threads(4)
sum
for(i=0; i<5; i++)
sum += a[i]*b[i];
(Thread 0)
sum
for(i=5; i<10; i++)
sum += a[i]*b[i];
(Thread 1)
sum
for(i=10; i<15; i++)
sum += a[i]*b[i];
(Thread 2)
sum
for(i=15; i<20; i++)
sum += a[i]*b[i];
(Thread 3)
+
sum

Reduction Operators : Fortran

Operator	Data Types	Initial Value
+	integer, floating point (complex or real)	0
*	integer, floating point (complex or real)	1
-	integer, floating point (complex or real)	0
.AND.	logical	.TRUE.
.OR.	logical	.FALSE.
.EQV.	logical	.TRUE.
.NEQV.	logical	.FALSE.
MAX	integer, floating point (real only)	가능한 최소값
MIN	integer, floating point (real only)	가능한 최대값
IAND	integer	all bits on
IOR	integer	0
IEOR	integer	0

Reduction Operators : C

Operator	Data Types	Initial Value
+	integer, floating point	0
*	integer, floating point	1
–	integer, floating point	0
&	integer	all bits on
|	integer	0
^	integer	0
&&	integer	1
||	integer	0

성능 비교

omp_get_wtime

- 특정 지점에서 경과된 wall clock time을 초 단위로 반환
  - C/C++
    double omp_get_wtime()
  - Fortran
    double precision function omp_get_wtime()

```
#include <omp.h>
…
double stime, etime;
…
stime=omp_get_wtime();
// Computing….
etime=omp_get_wtime()-stime;
printf("Elapsed Time : %lf sec. \n ",etime);
```

성능 비교

CASE 1 : 순차 코드(serial_dot_product.c)

```
#define N 100000000
…
stime=omp_get_wtime();
for(i=0; i<N; i++)
  sum += a[i] * b[i];
etime=omp_get_wtime()-stime;
```

CASE 2 : critical(synch_critical.c)

```
stime=omp_get_wtime();
#pragma omp parallel
{
  #pragma omp for
  for(i=0; i<N; i++)
    #pragma omp critical
    sum += a[i] * b[i];
}
etime=omp_get_wtime()-stime;
```

성능 비교

- CASE 3 : atomic(루프 내부) (synch_atomic.c)
- CASE 4 : atomic(루프 외부) (synch_solution.c)

```
stime=omp_get_wtime();
#pragma omp parallel
{
  #pragma omp for
  for(i=0; i<N; i++)
    #pragma omp atomic
    sum += a[i] * b[i];
}
  etime=omp_get_wtime()-stime;
```

```
 stime=omp_get_wtime();
#pragma omp parallel
private(local_sum)
{
  local_sum=0;
  #pragma omp for
  for(i=0; i<N; i++)
    local_sum += a[i] * b[i];

  #pragma omp atomic
  sum += local_sum;
}
  etime=omp_get_wtime()-stime;
```

성능 비교

CASE 5 : reduction(dot_product_reduction.c)

```
stime=omp_get_wtime();
#pragma omp parallel for reduction(+:sum)
for(i=0; i<N; i++)
  local_sum += a[i] * b[i];
etime=omp_get_wtime()-stime;
```

	Elapsed time(sec.)	Speed-Up	
CASE 1 : Serial	1.48	1x	
CASE 2	190.0	0.0078x	1x
CASE 3	8.0	0.168x	21.7x
CASE 4	0.058	25.5x	
CASE 5	0.058	25.5x	

Factorial

Fortran	C
PROGRAM reduction_exercise INTEGER, PARAMETER :: N=10 INTEGER :: i, fac=1 DO i=1, N fac = fac * i END DO print *, “factorial =“, fac END	#include <stdio.h> #define N 10 int main() { int i, fac = 1; for(i=1; i<=N; i++) fac *= i; printf(“factorial = %d\n”, fac); }

Reduction (Solution)

Fortran	C
PROGRAM reduction_solution INTEGER, PARAMETER :: N=10 INTEGER :: i, fac=1 call omp_set_num_threads(4) !$OMP PARALLEL DO reduction(*:fac) DO i=1, N fac = fac * i END DO !$OMP END PARALLEL DO print *, “factorial =“, fac END	#include <stdio.h> #include <omp.h> #define N 10 int main() { int i, fac = 1; omp_set_num_threads(4); #pragma omp parallel for reduction (*:fac) for(i=1; i<=N; i++) fac *= i; printf(“factorial = %d\n”, fac); }

#pragma omp reduction(op: var)

- reduction연산은 병렬 계산에서 많이 사용되는 연산 중 하나
- reduction을 사용하지 않으면 코드 수정이 많이 필요함
- reduction을 사용하면 코드 변경을 최소화할 수 있음

inner product 성능 비교

- Critical, atomic, reduction 사용시 성능 비교
- Reduction을 사용하는 경우 가장 성능이 좋음
- Critical과 atomic의 경우 atomic의 성능이 더 좋음

Reduction을 이용한 factorial 계산

Outro

Reduction 보조 지시어

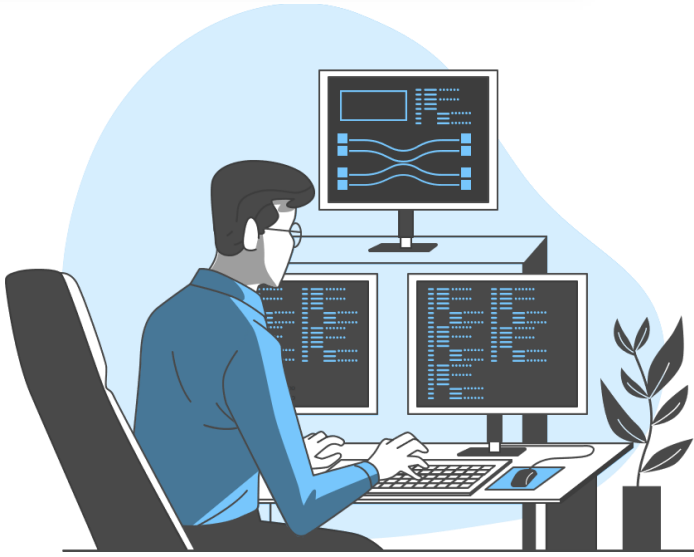
- #pragma omp reduction(op: var)

Dot product 성능 비교

Reduction을 이용한 factorial 계산

07 Nested parallel

1. Nested parallel 소개
2. 코드 설명(Nested parallel)
3. 데이터 유효 범위



학습목표

- Nested parallel이 무엇인지 이해한다.
- Nested parallel의 사용법을 이해한다.
- Nested parallel에서 데이터 유효범위를 이해한다.

Fortran	C
PROGRAM nested_parallel_do implicit none INTEGER, PARAMETER :: N=20 INTEGER :: tid, i, omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL private(tid) tid = omp_get_thread_num() **!$OMP PARALLEL DO !! ???** **!! netsted parallel (tomorrow)** DO i=0, N-1 PRINT *, 'Hello World', tid, i END DO !$OMP END PARALLEL END	#include <stdio.h> #include <omp.h> #define N 20 int main() { int tid, i; omp_set_num_threads(4); #pragma omp parallel private(tid) { tid = omp_get_thread_num(); **#pragma omp parallel for // ???** for(i=0; i<N; i++) printf("Hello World %d %d\n", tid, i); } }

스레드 생성 : Nested Parallel

Nested Parallel

- OpenMP 3.0부터 지원
- 병렬화 효율성 및 편의성의 증대
- omp_set_nested() 함수 : nested parallel 사용 여부 설정 기능
  - 인수 1 : 사용, 인수 0 : 사용 안함
- omp_get_nested() 함수 : nested parallel 지원 여부 결과 리턴
- OpenMP 5.2에서 deprecate됨

❖ HPC 환경에서는 사용하지 않는 것을 권장함

스레드 생성 : Nested Parallel (1/3)

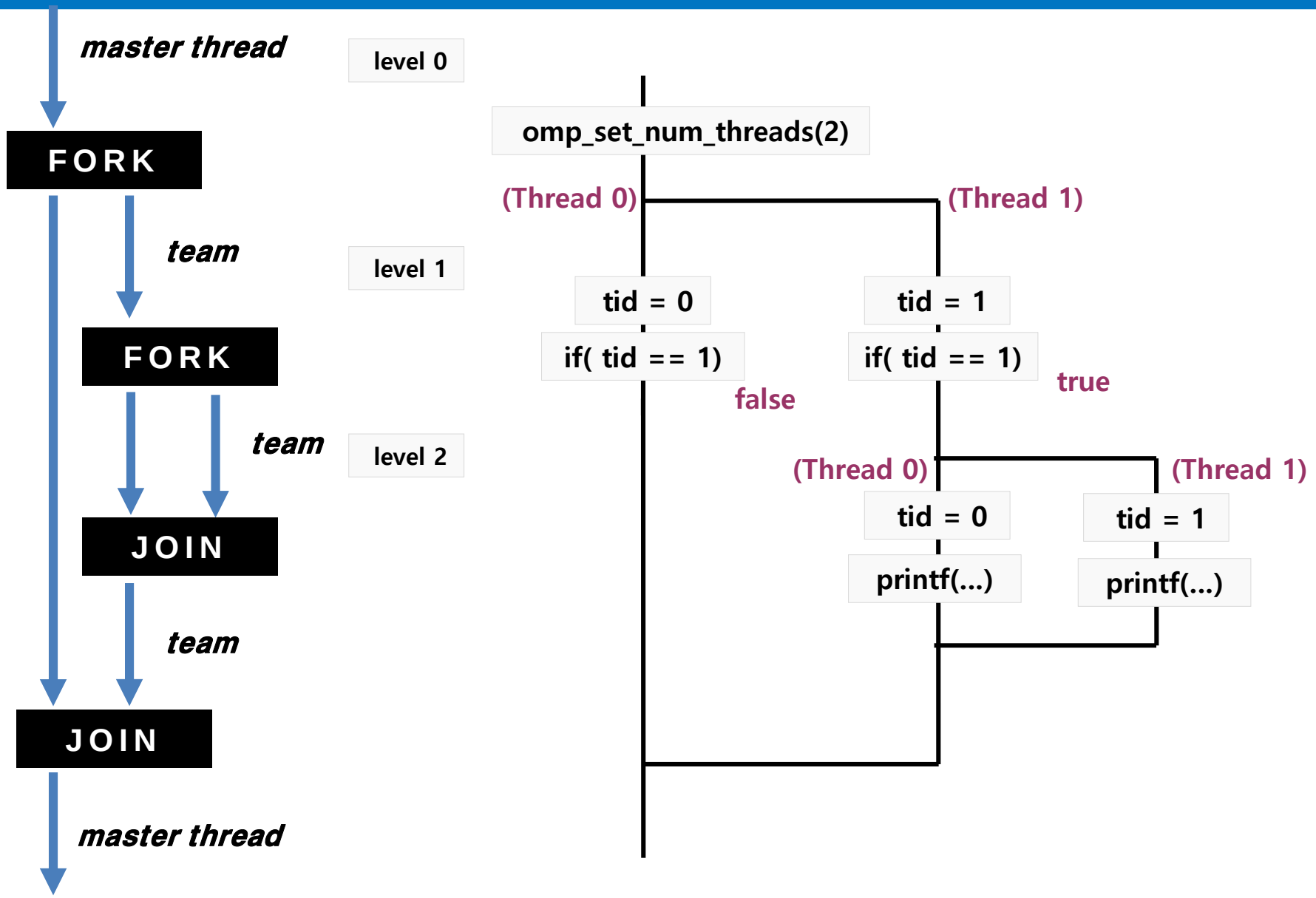
Fortran

```
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
```

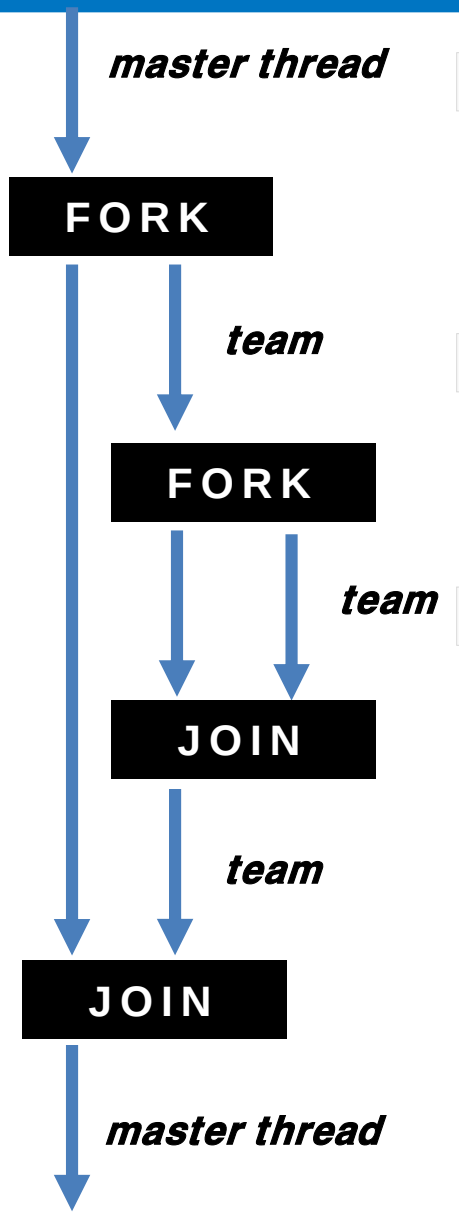
C

```
#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);
      }
   }
}
}
```

스레드 생성 : Nested Parallel (2/3)



스레드 생성 : Nested Parallel (2/3)



Fortran	C
omp_set_nested(nested) enables or disables nested parallelism	omp_set_nested(nested) enables or disables nested parallelism
omp_get_nested() the value of the nes-var ICV	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_ancestor_thread_num(level) the thread number of the ancestor of the current thread	omp_get_ancestor_thread_num(level) the thread number of the ancestor of the current thread
omp_get_team_size(level) the size of the thread team	omp_get_team_size(level) the size of the thread team

스레드 생성 : Nested Parallel (3/3)

Fortran

```
program more_nested_parallel
    integer x, 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
```

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));
      }
   }
}
}
```

Nested Parallel : 데이터 유효범위(1/3)

Fortran	C
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	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? } } }

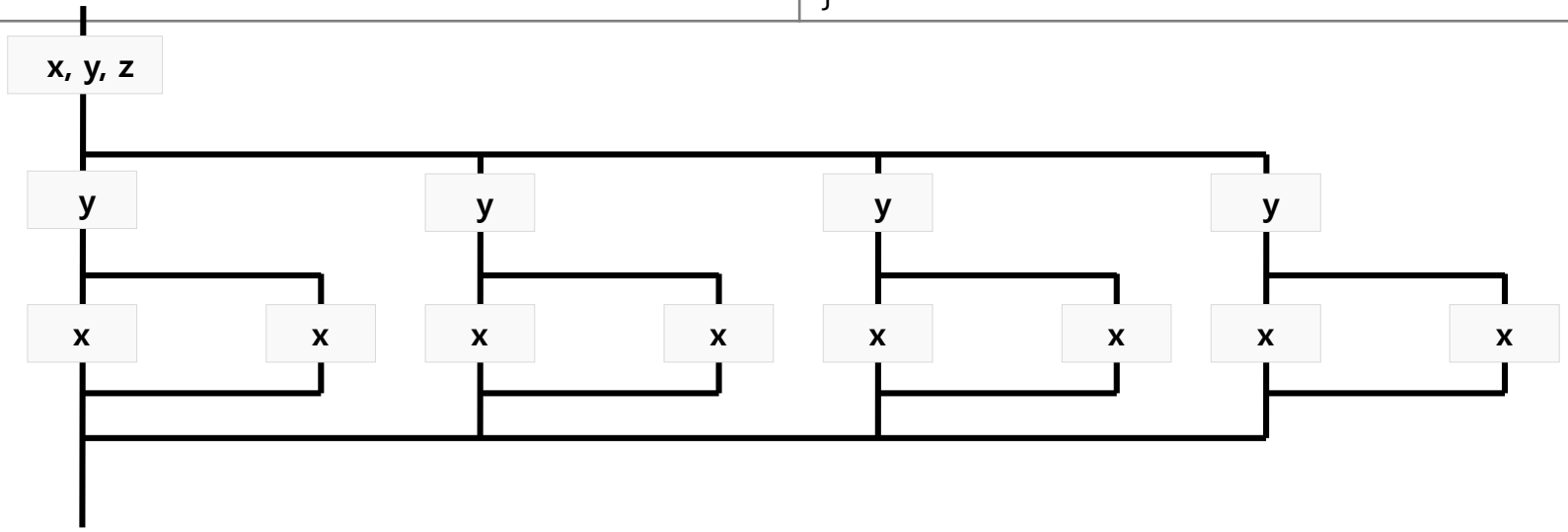
Nested Parallel : 데이터 유효범위(2/3)

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
   }
}
}
```



Nested Parallel : 데이터 유효범위(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
```

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("\t tid=%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("\n"); printf("x=%d y=%d z=%d\n", x,
  y, z);
}
```

학습 요약

Nested parallel에 대해서 설명

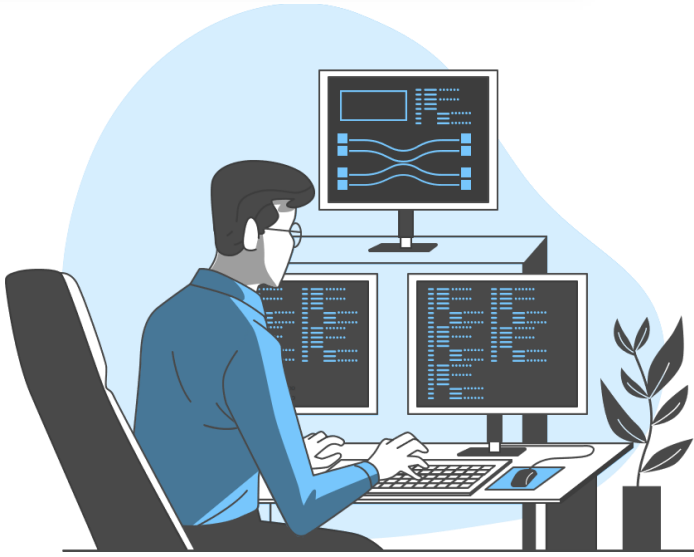
- 병렬 영역 안에서 스레드들은 또 다른 스레드들을 만들 수 있음
- omp_set_nested(1)을 사용해야 함
- 알고리즘의 병렬화를 쉽게 구현할 수 있도록 함
- HPC에서는 추천하지 않음

데이터 유효 범위

- Nested parallel
- 데이터 유효 범위

08 작업 분할/동기화

1. 작업 분할 지시어 소개
2. sections, single, master
3. 동기화(nowait, ordered)



학습목표

- 추가적인 작업 분할 지시어의 종류를 익힌다.
- sections, single, master의 의미와 사용법을 익힌다.
- nowait, ordered 지시어의 의미와 사용법을 익힌다.

작업분할 지시어

- do/for
- sections
- single
- workshare

- 병렬 영역 내부에 삽입하여 작업할당에 이용
- 새로운 스레드 생성 없이 기존 스레드에 관련작업 실행 분배
- 구문의 시작점: 스레드 사이의 동기화 없음
  - 먼저 접근하는 스레드에 우선적 작업 할당
- 구문의 끝점: 동기화(암시적 장벽)
  - 작업이 먼저 끝나도 다른 작업의 완료까지 대기
  - 대기 없이 다른 작업을 실행하려면 nowait clause 사용
- workshare 구문은 Fortran에서만 사용가능

작업분할 지시어

Sections

- #pragma omp sections 정의
- 여러 스레드에 작업단위로 분배, 병렬 처리 기능
- 스레드 팀에 있는 각각 스레드를 #pragma omp section로 작업을 나눔
  - 해당 작업은 스레드 팀 중에서 하나의 스레드에 의해 한번만 실행
  - 어떤 스레드가 어떤 section을 수행할 지 보장하지 않음
- 작업을 할당 받지 못한 스레드는 암시적 동기화 지점에서 대기

Single

- #pragma omp single 정의
- 스레드 팀 중 하나의 스레드만 해당 코드를 실행할 것을 지정
- 다른 스레드들은 single 지시어로 지정된 스레드가 실행을 완료할 때까지 암시적 동기화 지점에서 대기
- 가정 먼저 single 지시어에 도착한 스레드가 해당 작업 수행

작업분할 지시어 : Sections

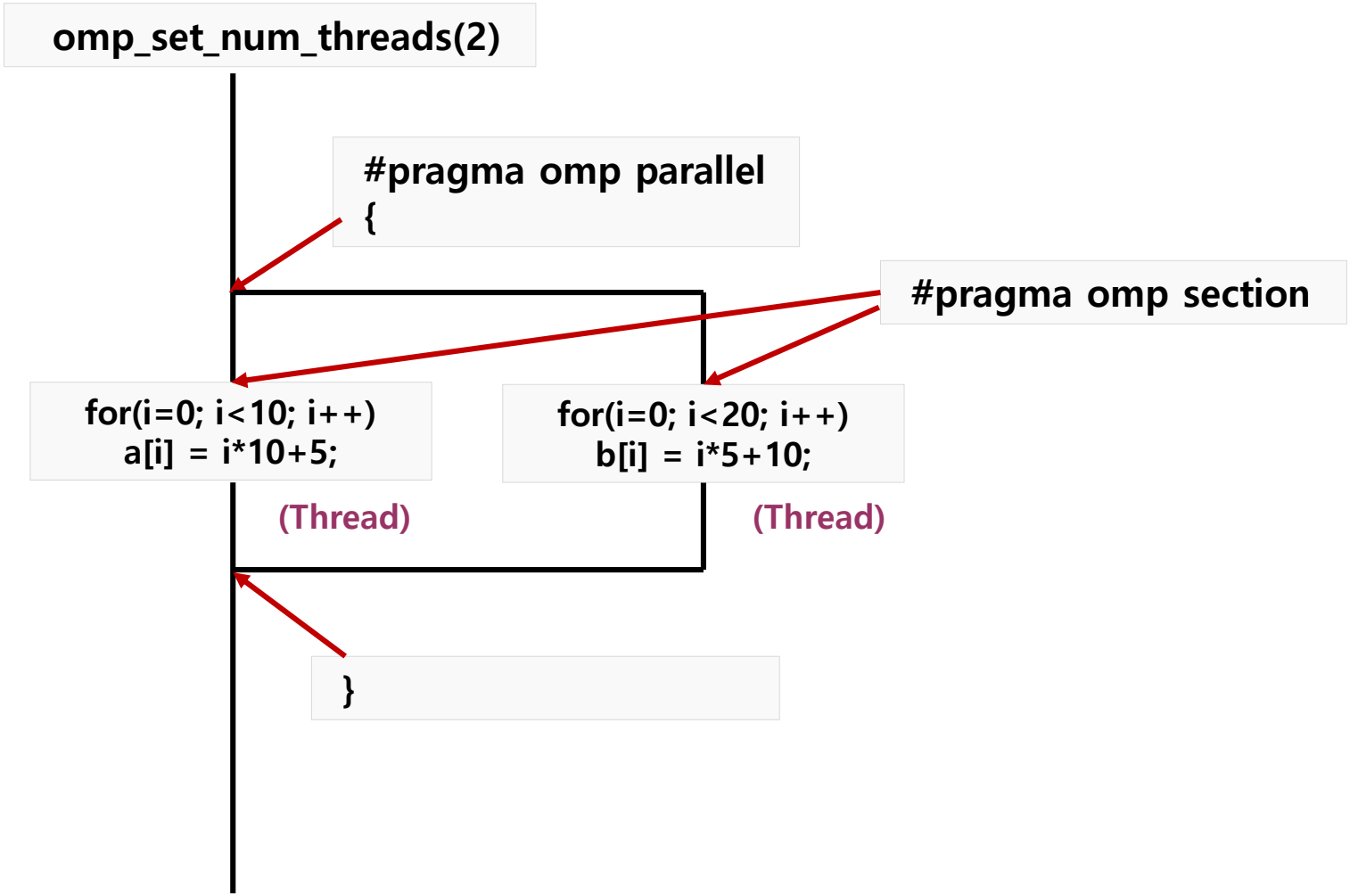
Fortran

```
program sections
    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
```

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");
}
```

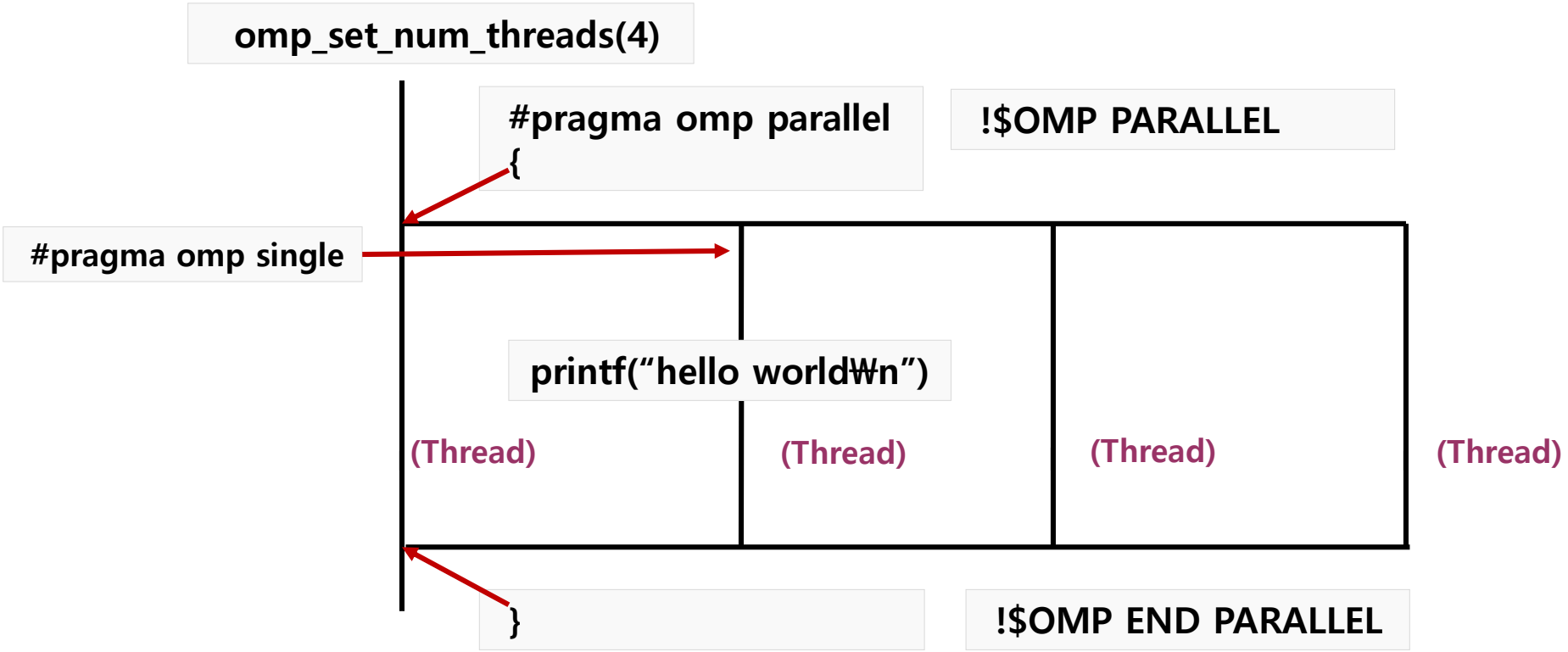
작업분할 지시어 : Sections



작업분할 지시어 : Single

Fortran	C
PROGRAM single 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"); } } }

작업분할 지시어 : Single



작업분할 지시어 : master

Fortran

```
program master
    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
```

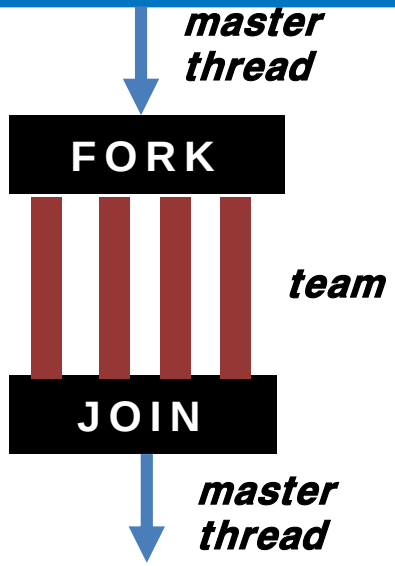
C

```
#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() );
}
}
```

MASTER

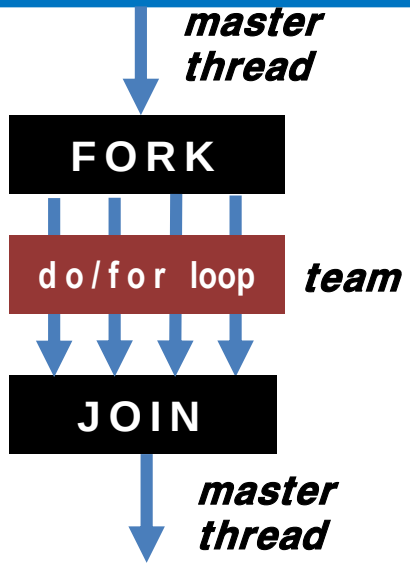
- 'master' section을 마스터 스레드에서만 실행시킴
- 다른 스레드들은 'master' section을 건너뛰게 되고 암시적 장벽은 없음

작업분할 지시어



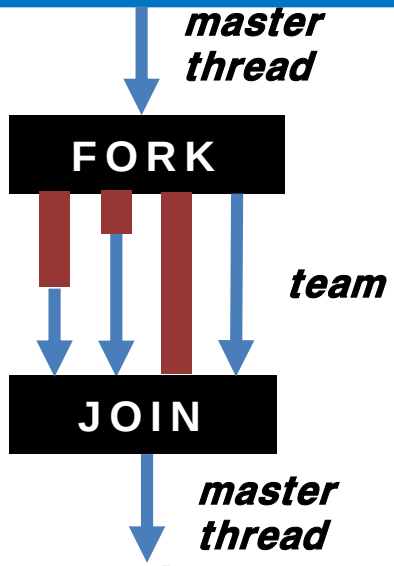
parallel

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



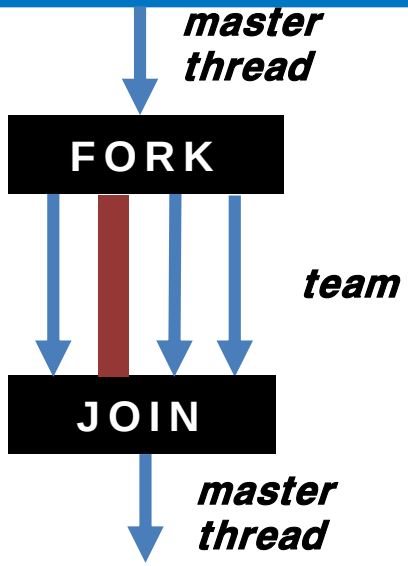
do/for

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



section

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



single

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

동기화 : nowait

Fortran

```
PROGRAM nowait

   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
```

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
```

동기화 : nowait

nowait

- nowait 보조지시어를 사용, 암시적인 barrier 제거
- nowait 지정 시, 먼저 작업 종료한 스레드는 barrier에 대기하지 않고 다음 작업 실행
- 병렬 영역 내에 독립된 복수의 분할 작업이 있는 경우, nowait 사용하여 성능 개선 가능

❖ Reduction과 nowait을 동시에 사용할 경우, 각 스레드의 결과값을 합산하는 과정 중 데이터 경합 발생 가능성이 있음

동기화 : nowait

Fortran

```
PROGRAM nowait

   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
```

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
```

동기화 : 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
    !$omp do
    do j=0, N
        do i=j+1, N
            a(i,j) = i-j
        end do
    end do
!$omp end parallel
End
```

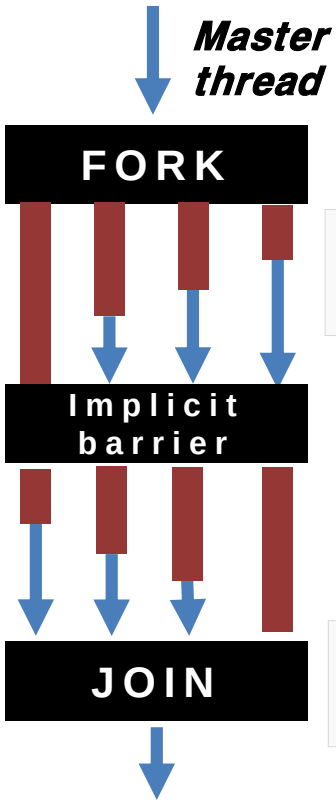
```
$ gfortran –fopenmp –o for_nowait1.x
   nowait.f90
$ ulimit –s 512000
$ time ./for_nowait1.x
```

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;
}
}
```

```
$ gcc –fopenmp –o for_nowait1.x nowait.c
$ ulimit –s 512000
$ time ./for_nowait1.x
```

동기화 : nowait




```
omp_set_num_threads(4)
#pragma omp parallel        !$OMP PARALLEL
{
                                          #pragma omp for
(Thread 0)                (Thread 1)                (Thread 2)                (Thread 3)
for(i=0; i<2500; i++)     for(i=2500; i<5000; i++)  for(i=5000;i<7500; i++)   for(i=7500;i<10000;i++)
  for(j=i; j<10000; j++)    for(j=i; j<10000; j++)    for(j=i; j<10000; j++)    for(j=i; j<10000; j++)
    a[i][j] = i+j;            a[i][j] = i+j;            a[i][j] = i+j;            a[i][j] = i+j;

                                          #pragma omp for
(Thread 0)                (Thread 1)                (Thread 2)                (Thread 3)
for(i=0; i<2500; i++)     for(i=2500; i<5000; i++)  for(i=5000;i<7500; i++)   for(i=7500;i<10000;i++)
  for(j=0; j<i; j++)        for(j=0; j<i; j++)        for(j=0; j<i; j++)        for(j=0; j<i; j++)
    a[i][j] = i-j;            a[i][j] = i-j;            a[i][j] = i-j;            a[i][j] = i-j;
}                                         !$OMP END PARALLEL
```

동기화 : 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
```

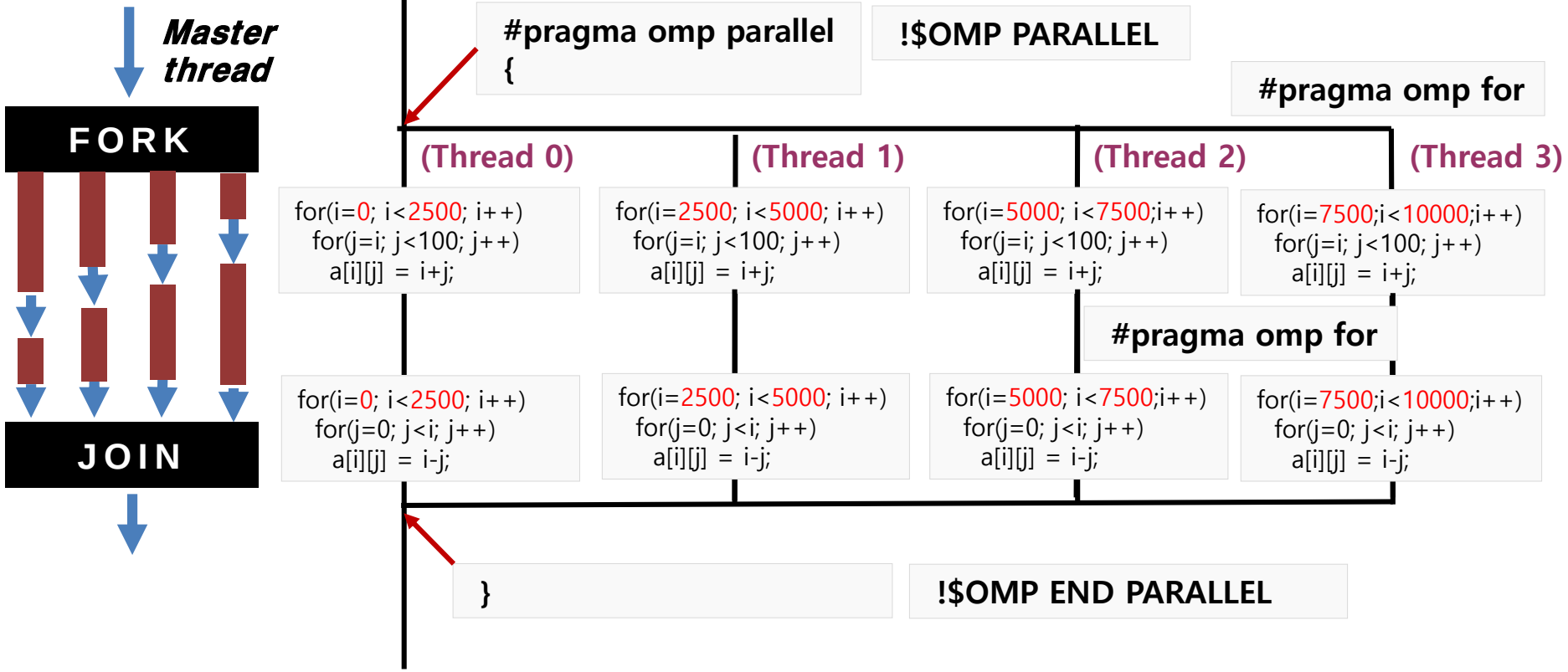
```
$ gfortran –fopenmp –o for_nowait2.x
  nowait.f90
$ ulimit –s 512000
$ time ./for_nowait2.x
```

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;
}
}
```

```
$ gcc –fopenmp –o for_nowait2.x nowait.c
$ ulimit –s 512000
$ time ./for_nowait2.x
```

동기화 : nowait


omp_set_num_threads(4)
Master thread
FORK
JOIN
#pragma omp parallel
{
!$OMP PARALLEL
#pragma omp for
(Thread 0)
(Thread 1)
(Thread 2)
(Thread 3)
for(i=0; i<2500; i++)
for(j=i; j<100; j++)
a[i][j] = i+j;
for(i=2500; i<5000; i++)
for(j=i; j<100; j++)
a[i][j] = i+j;
for(i=5000; i<7500;i++)
for(j=i; j<100; j++)
a[i][j] = i+j;
for(i=7500;i<10000;i++)
for(j=i; j<100; j++)
a[i][j] = i+j;
#pragma omp for
for(i=0; i<2500; i++)
for(j=0; j<i; j++)
a[i][j] = i-j;
for(i=2500; i<5000; i++)
for(j=0; j<i; j++)
a[i][j] = i-j;
for(i=5000; i<7500;i++)
for(j=0; j<i; j++)
a[i][j] = i-j;
for(i=7500;i<10000;i++)
for(j=0; j<i; j++)
a[i][j] = i-j;
}
!$OMP END PARALLEL

동기화 : ordered

Fortran

```
PROGRAM ordered
  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
```

C

```
#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]);
   }
}
}
```

ORDERED

- 'ordered' section 내부의 루프 실행을 순차적으로 진행
- 하나의 병렬루프에 하나만 허용
- 병렬루프 지시어는 'ordered' clause를 가져야 함
- 'ordered' section을 실행하는 스레드는 매 순간 한 개

학습 요약

추가적인 작업 분할 지시어

- sections, single, master

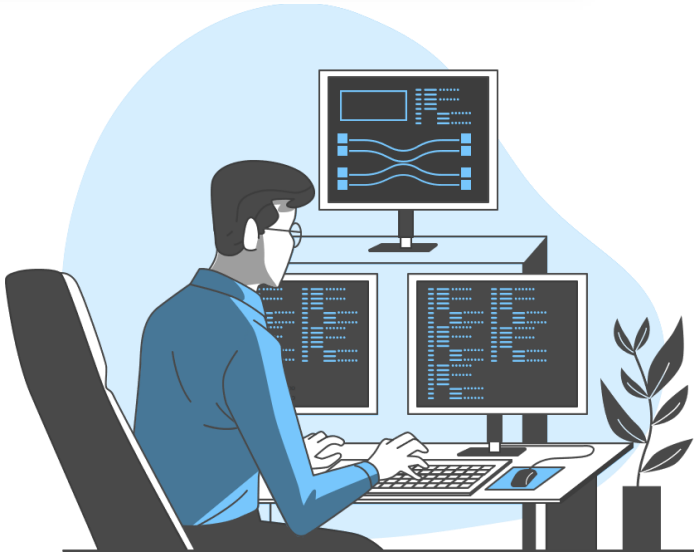
nowait, ordered

- nowait는 암시적 동기화를 하지 않음
- ordered는 루프 실행을 순차적으로 처리

- 추가적인 작업 분할 지시어
- nowait, ordered

09 schedule / task

1. OpenMP Scheduling
2. 코드 설명(schedule 사용)
3. Task, taskwait



학습목표

- OpenMP의 Scheduling을 이해한다
- OpenMP scheduling을 사용할 줄 안다
- Task, taskwait을 이해하고 사용할 줄 안다

Scheduling clauses

- 루프 실행의 분배 방식을 지정
- 기본적인 schedule 정책: 실행 회수 균등 분배
- 작업의 schedule를 지정하기 위해 위해 schedule clause를 사용

- Static [, chunk_size]
  - 반복 실행이 각 스레드에 균일하게 할당
  - ‘chunk_size’가 주어진 경우, 총 반복 실행 회수를 chunk_size로 나누어 chunk 생성한 후 chunk들을 스레드에 라운드-로빈 방식으로 정적 할당

❖ **연산횟수가 스레드 수로 나누어 떨어지지 않으면, 처리를 먼저 끝낸 스레드가 추가 실행**

- Dynamic [, chunk_size]
  - 총 반복실행 회수를 chunk_size로 나누어 chunk 생성
  - ‘chunk’들은 스레드에 동적 할당
    - 작업이 먼저 끝나는 스레드에 다음 chunk 할당
  - ‘chunk_size’가 없으면 디폴트 chunk_size=1

Scheduling clauses

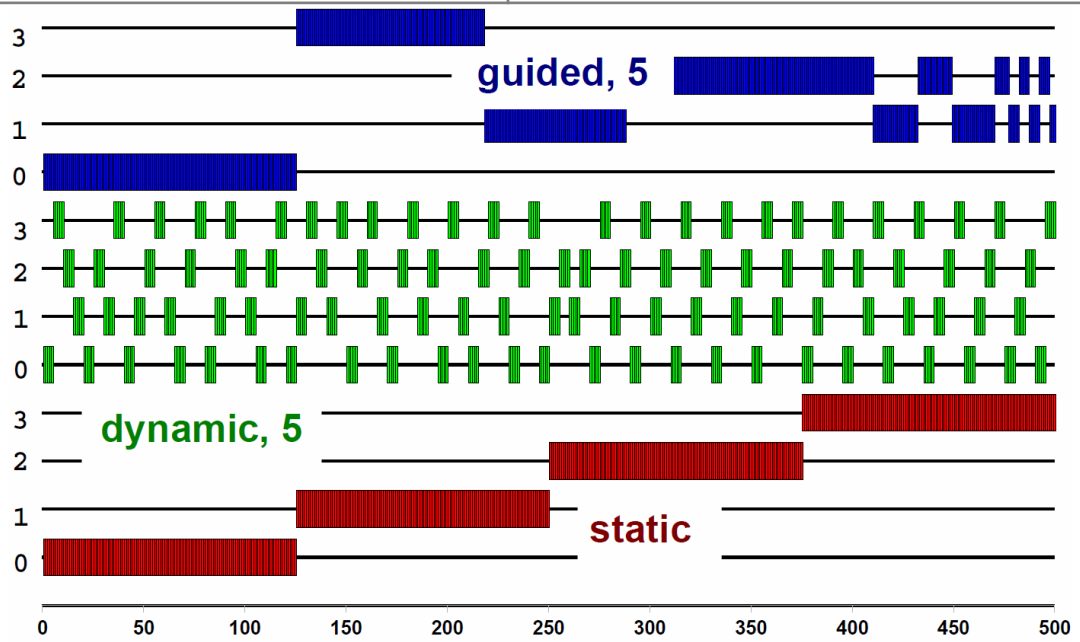
- Guided [, chunk_size]
  - Dynamic scheduling
  - 반복을 진행하면서, chunk 크기가 변함 (default chunk size = 1)

- Runtime
  - 프로그램 실행 중에 환경변수 OMP_SCHEDULE 값을 참조
  - 재 컴파일 없이 다양한 스케줄링 방식 시도 가능
  - Ex) $ export OMP_SCHEDULE="static,1000"
  - Ex) $ export OMP_SCHEDULE="dynamic"

- Auto
  - OpenMP 3.0 이상 지원
  - 컴파일러, 실시간 시스템이 가장 적합하다고 판단하는 스케줄링 설정

SCHEDULE

Fortran	C
program schedule implicit none integer :: a(0:499) !$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; }



Tutorial of IWOMP 2011

Mandelbrot

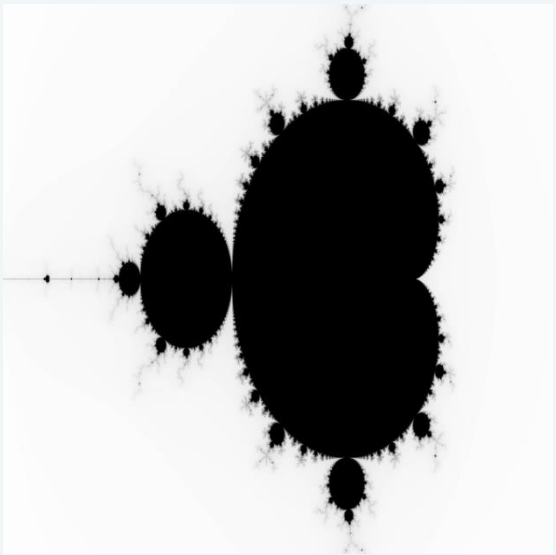
- Mandelbrot is an image computing and display computation
- Pixels of an image are stored in a 2D array
- Each pixel is computed by iterating the complex function

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

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

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

- 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;
       }
   }
}
```

SCHEDULE (Exercise)

Fortran (Serial)

```
program ex

   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
   ENDDO

end
```

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;
       }
   }
}
```

SCHEDULE (Solution)

Fortran (Parallel)

```
program solution

   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
```

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];
```

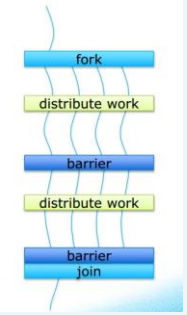
```
#pragma omp parallel for shared(res, maxIterations)
  private(j,z,c,k,temp,lengthsq) schedule(dynamic, 5)
   for(i=0; i < X_RESN; i++) {
       for(j=0; j < Y_RESN; j++) {
          z.real = z.imag = 0.0;
          c.real = X_MIN + j * (X_MAX - X_MIN)/X_RESN;
          c.imag = Y_MAX - i * (Y_MAX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

OpenMP Task

병렬 영역: (암시적) tasks (스레드당 하나) 묶음 생성

- 스레드 팀 발생
- 팀 내의 각 스레드에 task 하나씩 할당
  (tied to thread)
- 모든 task 종료될 때까지 마스터 스레드 대기

```
#pragma omp parallel
{
   #pragma omp for
   for (i = 0; i<N; i++)
   {…}

   #pragma omp for
   for (i = 0; i< N; i++)
   {…}
}
```

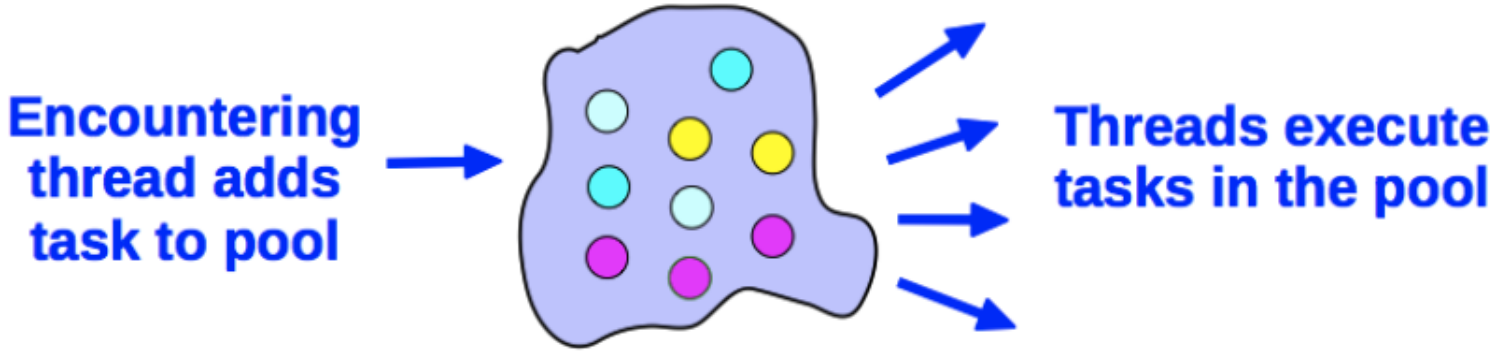


명시적인 task 생성 가능 (OpenMP 3)

- 병렬화가 힘들었던 while루프, C++반복자(iterator), 재귀함수의 병렬화 가능
- task 지시어는 수행해야 할 코드를 작업 큐에 넣는 동작 수행
- 스레드 팀과 작업 큐(queue) 연결(binding)
- 작업 큐(queue)에 있는 코드들을 스레드 팀이 순서대로 실행
- task를 작업 큐에 넣는 동작을 할 때는 병렬 영역에서 다수의 스레드 동작, 동기화 문제 발생 가능 -〉 #pragma omp single 지시어 사용으로 해결 가능

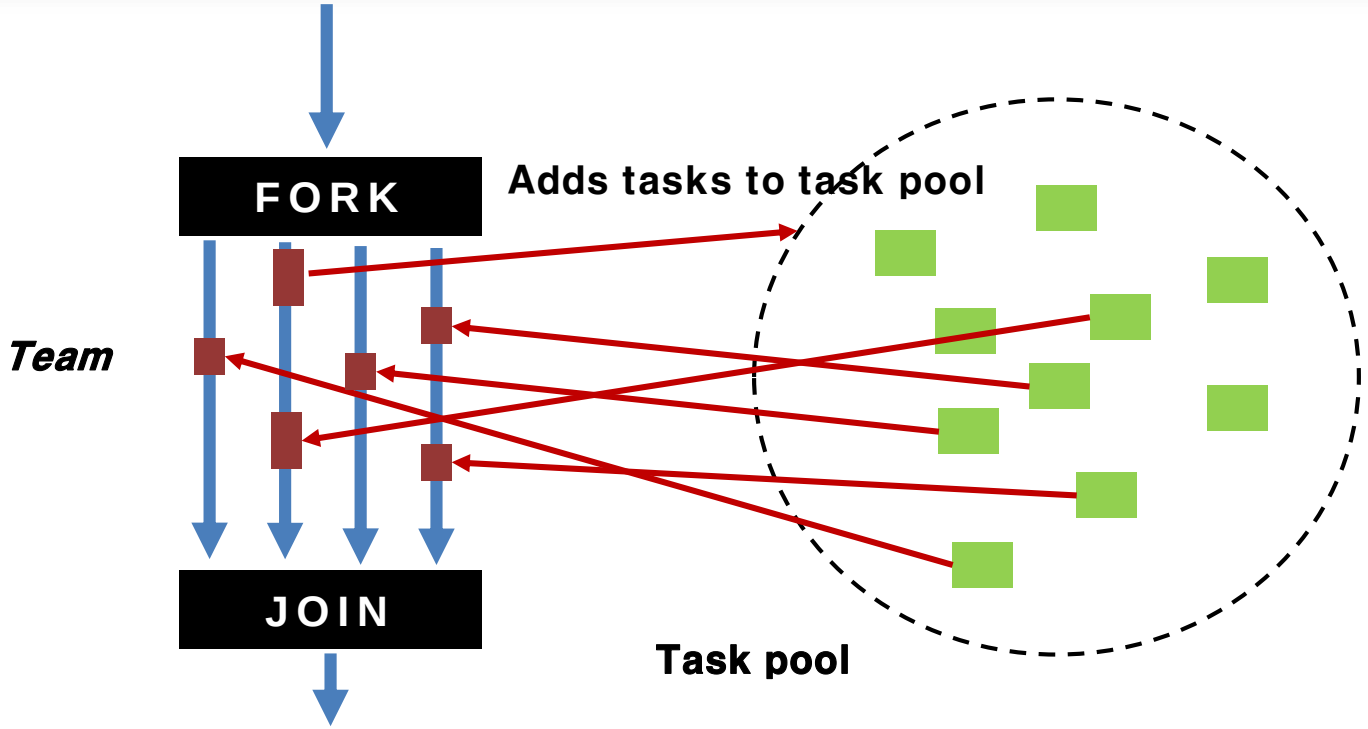
Task Concepts

- task scheduling = 은행 번호표 시스템
- thread = 은행원
- task = 은행에서 처리할 일



Task Construct

- Task는 수행할 코드를 작업 큐에 넣는 동작을 함
- 작업 큐에 있는 작업들은 수행 가능한 스레드에 의해 순서대로 실행됨



Task (1/3)

Fortran	C
Program task1 !$OMP PARALLEL num_threads(32) write(*, '(A)', advance="no") 'A ' write(*, '(A)', advance="no") 'B ' write(*, '(A)', advance="no") 'C ' write(*, '(A)', advance="no") 'D ' write(*, *) ' ‘ !$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”); } }

Task (2/3)

Fortran	C
Program task2 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()); } } }

Task (3/3)

Fortran

```
Program task3
  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
```

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() );
   }
}
}
```

Task : taskwait

Fortran

```
Program task4
  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
```

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() );
   }
}
}
```

Task : 데이터 유효범위

Fortran

```
Program task_data_scope
  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
```

C

```
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?   }
}
}
```

Fortran

```
Program task
USE OMP_LIB or include “omp_lib.h”

  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
```

C

```
int foo()
{
   int a, b, c,d,e;
#pragma omp parallel private(b,d,e)
{
   #pragma omp task private(e)
   {
      a : shared
      b : firstprivate
      c : shared
      d : firstprivate
      e : private
   }
}
}
```

OpenMP Scheduling

- static, dynamic, guided, runtime, auto

task

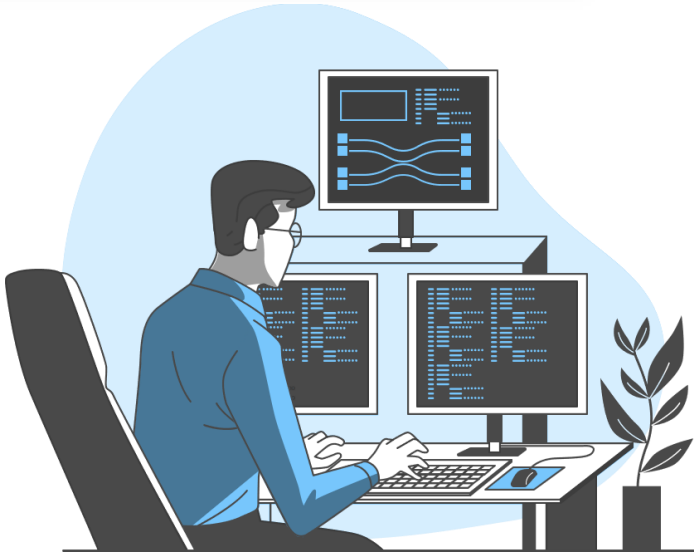
- while 문과 같이 반복 횟수가 컴파일 타임에 결정되지 않는 루프도 병렬화 가능

Outro

- OpenMP Scheduling
- task

10 OpenMP Performance Issue

1. Nested parallel(collapse)
2. Flush, false sharing
3. 데이터 의존성



학습목표

- 다중 루프 병렬화를 위한 collapse 보조 지시어를 이해한다
- Flush, false sharing에 대해서 이해한다
- 데이터 의존성을 이해한다

COLLAPSE

Collapse

- #pragma omp for collapse(loop 개수)로 정의 후 사용
- 중첩 loop문을 이용하기 위해 사용
- 병렬화 효율성 향상 기대

Collapse (1/4)

Fortran

```
program collapse
USE OMP_LIB or include “omp_lib.h”

   interger, parameter  :: M=100, N=100
   integer  :: a(0:M-1, 0:N-1), i, j

   call omp_set_nested(1)
!$OMP PARALLEL DO
   DO j=0, N-1
      !$OMP PARALLEL DO
      DO i=0, M-1
         a(i, j) = a(i, j) * 2
      END DO
      !$OMP END PARALLEL DO
   END DO
!$OMP END PARALLEL DO
End
```

C

```
#include <stdio.h>
#include <omp.h>
#define M 100
#define N 100

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

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

Better performance?

Collapse (2/4)

Fortran

```
program collapse
USE OMP_LIB or include “omp_lib.h”

   interger, 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;
   }
}
}
```

Possible ???

Collapse (3/4)

Fortran

```
program collapse
USE OMP_LIB or include “omp_lib.h”

   interger, parameter  :: N=10000
   integer  :: a(0:N-1, 0:N-1), i, j

   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
```

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
   for(i=0; i<N*N; i++)
      a[i/N][i%N] = a[i/N][j%N] * 2;
}
```

Collapse (4/4)

Fortran

```
program collapse
   interger, 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
```

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;
}
```

flush (1/3)

Fortran	C
program flush1 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	#include <stdio.h> #include <omp.h> #include <unistd.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); } }
$ gfortran –fopenmp –o ok.x flush.90 $ ./ok.x $ gfortran **-O3** –fopenmp –o no.x flush.90 $ ./no.x	$ gcc –fopenmp –o ok.x flush.c $ ./ok.x $ gcc **-O3** –fopenmp –o no.x flush.c $ ./no.x

flush (2/3)

FLUSH Directive

- 메모리 펜스의 역할을 수행
- 컴파일러는 read/write 순서를 reordering하지 않음

```
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	C
program flush2 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	#include <stdio.h> #include <omp.h> #include <unistd.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); } }
$ gfortran **-O3** -fopenmp -o flush.x flush.90 $ ./flush.x	$ gcc **-O3** -fopenmp -o flush.x flush.c $ ./flush.x

Flase Sharing

- CPU 코어 프로세스와 Cache 사이에서 값이 변경되면 캐시 일관성을 유지해야 함
  - 성능 저하 발생
- RAM에 있는 데이터를 Cache로 가져올 때 그 변수의 크기만큼 가져오는 것이 아니라, 캐시라인 크기인 연속된 64바이트의 크기를 한꺼번에 캐시에 올린다.
- Flase Sharing으로 발생하는 부하는 함수 수준에서 6~7배, 어플리케이션 수준에서는 2배 정도 성능 저하 발생

❖ 해결방법

❖ False Sharing이 발생할 변수를 private 선언

False Sharing (1/5)

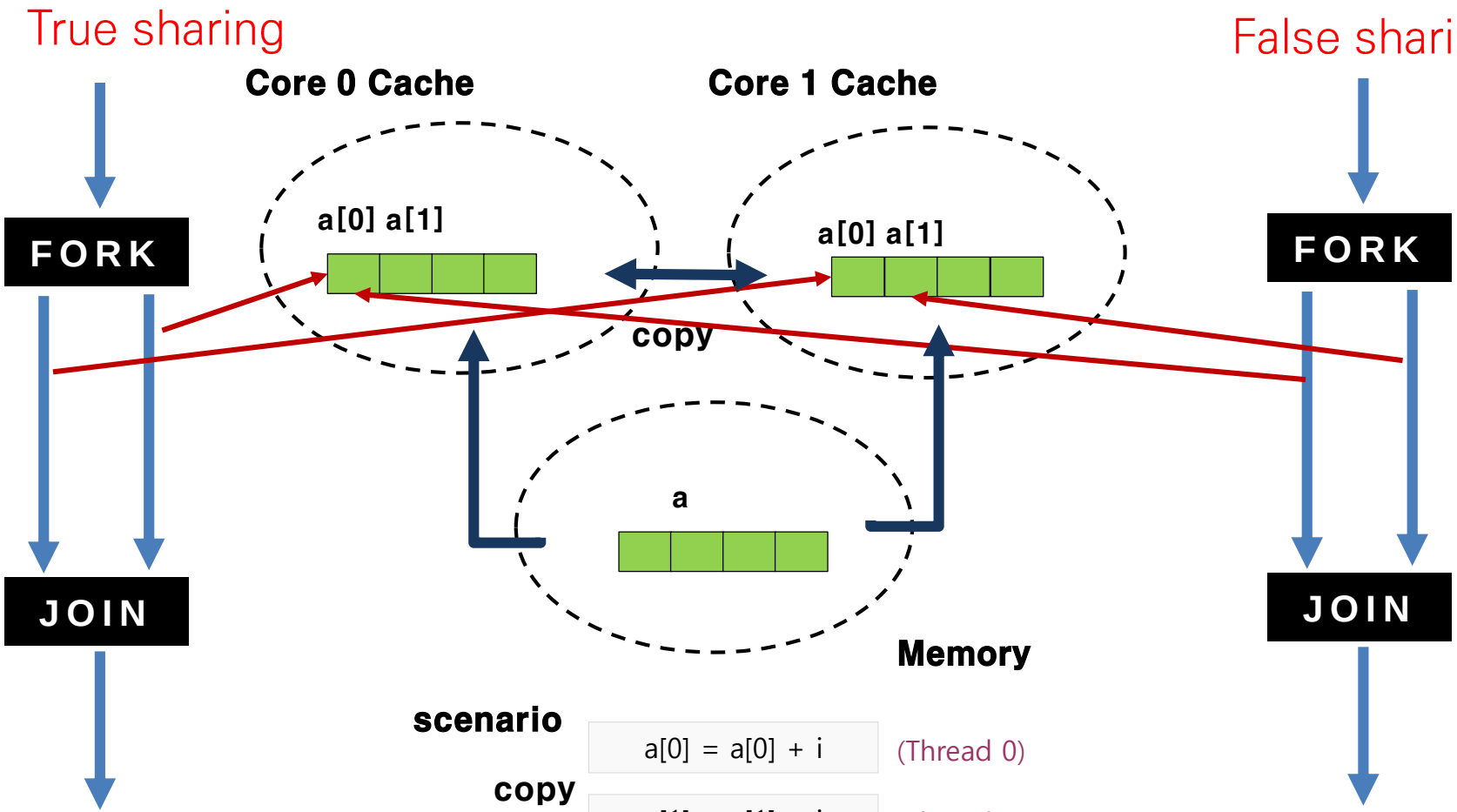
Fortran

```
program false_sharing
   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
```

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);
}
```

False Sharing (2/5)



False Sharing (3/5)

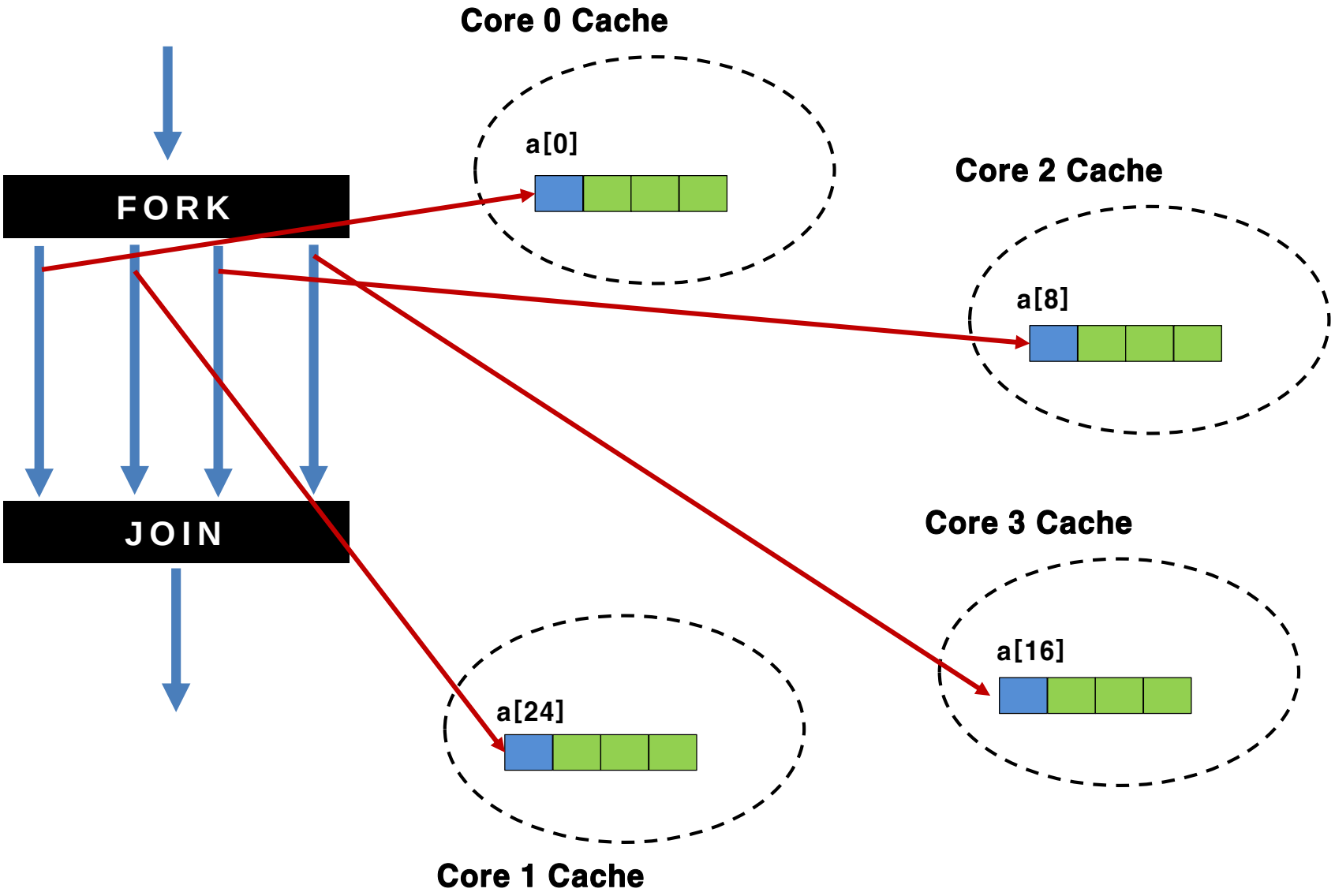
Fortran

```
program false_sharing_solved
   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
```

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);
}
```

False Sharing (4/5)



False Sharing (5/5)

Fortran

```
program use_local_sum
   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
```

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 = sum + local_sum;
}
   printf("sum = %ld\n", sum);
}
```

데이터 의존성 (1/2)

```
a(i) = i
...
DO i = 2, 5
  a(i) = a(i)*a(i-1)
ENDDO
```

순차실행

병렬실행

i	a(i)
1	1
2	2
3	6
4	24
5	120

time

Thread 0	Thread 1
a(2)=a(2)*a(1)=2*1=2	
	a(4)=a(4)*a(3)=4*3=12
a(3)=a(3)*a(2)=3*2=6	
	a(5)=a(5)*a(4)=5*12=60

데이터 의존성 (2/2)

데이터 의존성

- 서로 다른 문장이 동일한 메모리 위치에 접근(racing)할 때 발생
- Loop-carried 데이터 의존성

shared 변수에서만 발생

루프에서 read만 되는 변수에는 의존성 문제 없음

한 번의 반복에서만 접근되는 변수에는 의존성이 없음

➔ 여러 반복에서 접근되는 shared 변수

데이터 의존성의 분류(1/2)

Loop-carried 의존성

- flow 의존성(true dependence or RAW)
  - 순차 실행에서 S1 → S2의 순서로 실행될 때
    - S1 : a(2) = a(1) + b(2) (write)
    - S2 : a(3) = a(2) + b(3) (read)

```
DO i = 2, N
  a(i) = a(i-1) + b(i)
ENDDO
```

  - S1이 write, S2가 read (RAW)
- anti 의존성 : S1이 read, S2가 write (WAR)
- output 의존성 : S1, S2 모두 write (WAW)

데이터 의존성의 분류(2/2)

```
      DO i = 2, N-1
10      x = d(i) + 1
20      a(i) = a(i+1) + x
30      b(i) = b(i) + b(i-1) + d(i-1)
40      c(2) = 2*i
      ENDDO
```

Memory location	처음 상태			나중 상태			Loop-carried?	Dataflow 종류
	라인	반복	접근	라인	반복	접근		
x	10	i	쓰기	20	i	읽기	No	Flow
x	10	i	쓰기	10	i+1	쓰기	Yes	Output
x	20	i	읽기	10	i+1	쓰기	Yes	Anti
a(i+1)	20	i	읽기	20	i+1	쓰기	Yes	Anti
b(i)	30	i	쓰기	30	i+1	읽기	Yes	Flow
c(2)	40	i	쓰기	40	i+1	쓰기	Yes	Output

데이터 의존성의 제거

의존성 제거

- 데이터 유효 범위 변경
- 간단한 소스코드 변경
- anti, output 의존성은 항상 제거 가능
- flow 의존성 제거
  - 가능 or 불가능
  - 근본적인 알고리즘 변경이 요구될 수도 있음

의존성 제거 과정에서 가급적 다른 의존성에 영향을 주지 말아야 함

데이터 의존성의 제거

flow 의존성 제거 : Loop skewing

- Loop-carried flow 의존성을 non-loop-carried로 변환

```
DO i=2, N
   b(i) = b(i) + a(i-1)
   a(i) = a(i) + c(i)
END DO
```

```
i = i : b(i) = b(i) + a(i-1)
        a(i) = a(i) + c(i)       ! Write
i=i+1 : b(i+1) = b(i+1) + a(i)  ! Read RAW(flow dependency)
        a(i+1) = a(i+1) + c(i+1)
```

의존성 제거

```
b(2) = b(2) + a(1)
DO i=2,N-1
   a(i) = a(i) + c(i)
   b(i+1) = b(i+1) + a(i)
END DO
a(N) = a(N) + c(N)
```

OpenMP 병렬화

```
b(2) = b(2) + a(1)
!$OMP PARALLEL DO SHARED(a,b,c)
DO i=2,N-1
   a(i) = a(i) + c(i)
   b(i+1) = b(i+1) + a(i)
END DO
a(N) = a(N) + c(N)
```

데이터 의존성의 제거

flow 의존성 제거 : Loop skewing

- flow.f90

```
PROGRAM FLOW
IMPLICIT NONE
INTEGER(KIND=8),PARAMETER::N=1000000000
REAL::A(N),B(N),C(N),X,X_TMP(N)
INTEGER(KIND=8)::STIME,ETIME,C_RATE
INTEGER::I
CALL RANDOM_NUMBER(A)
CALL RANDOM_NUMBER(B)
CALL RANDOM_NUMBER(C)
CALL SYSTEM_CLOCK(STIME,C_RATE)
DO I=2,N
  B(I)=B(I)+A(I-1)
  A(I)=A(I)+C(I)
END DO
CALL SYSTEM_CLOCK(ETIME)
PRINT*,'Elasped Time : (flow)', (ETIME-STIME)/DBLE(C_RATE)
```

데이터 의존성의 제거

flow 의존성 제거 : Loop skewing

```
CALL SYSTEM_CLOCK(STIME)
B(2)=B(2)+A(1)
DO I=2,N-1
  A(I)=A(I)+C(I)
  B(I+1)=B(I+1)+A(I)
END DO
A(N)=A(N)+C(N)
CALL SYSTEM_CLOCK(ETIME)
PRINT*,'Elasped Time : (Loop skewing)', (ETIME-STIME)/DBLE(C_RATE)

CALL SYSTEM_CLOCK(STIME)
B(2)=B(2)+A(1)
!$OMP PARALLEL DO
DO I=2,N-1
  A(I)=A(I)+C(I)
  B(I+1)=B(I+1)+A(I)
END DO
A(N)=A(N)+C(N)
CALL SYSTEM_CLOCK(ETIME)
PRINT*,'Elasped Time : (OpenMP)', (ETIME-STIME)/DBLE(C_RATE)
END PROGRAM FLOW
```

학습 요약

collapse 보조 지시어

- 다중 루프 병렬화를 위해 사용

False sharing

- 멀티 코어 CPU에서 캐시 일관성 때문에 발생
- False sharing이 발생하는 병렬 성능이 좋지 않음
- False sharing이 발생하지 않도록 코드 작성을 해야 함

데이터 의존성

- 데이터 의존성이 있으면 루프 병렬화가 불가능
- 데이터 의존성을 제거한 후 병렬화를 시도해야 함

Outro

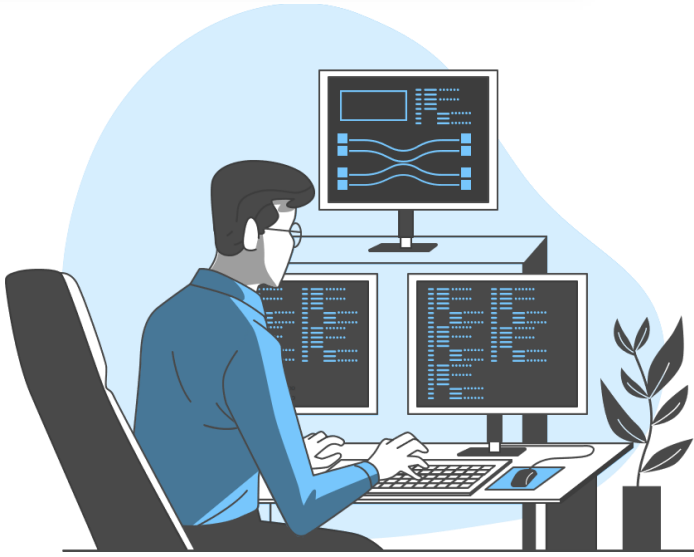
- collapse 보조 지시어
- False sharing
- 데이터 의존성

11 Hands-On

1. Pi 계산

2. 여러 파일 동시에 읽기

3. 2D FDM



학습목표

- 예제를 통해 병렬화 기법을 이해한다.
  - Pi 계산
  - 파일 여러 개 읽기
  - 2D FDM

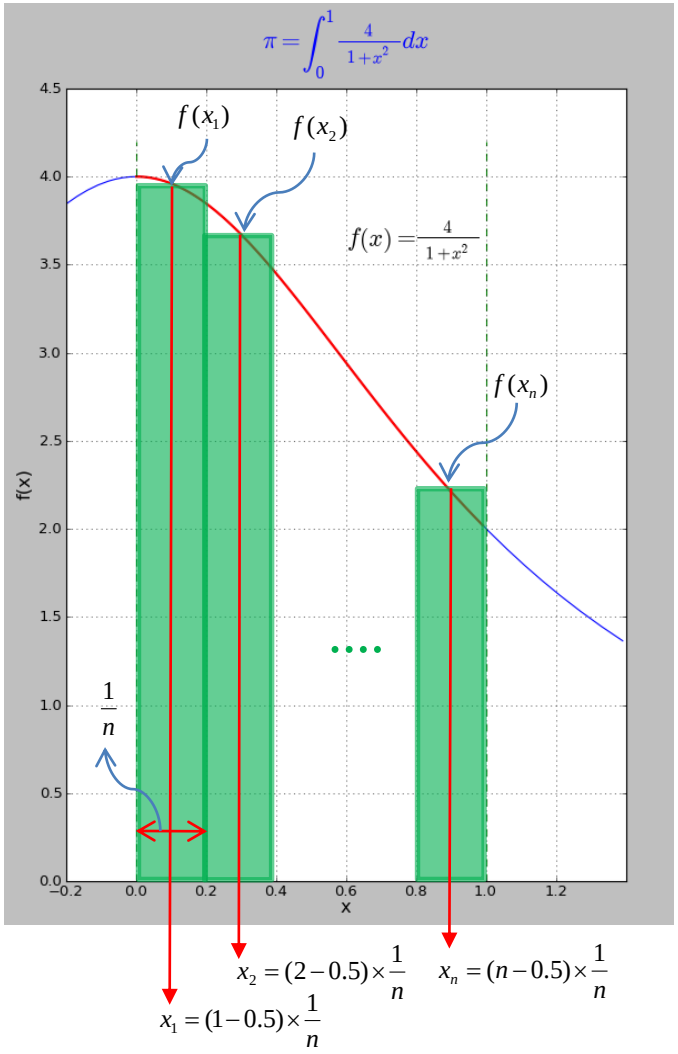
Pi calculation for Numerical integration

$$\int_0^1 \frac{4}{1+x^2}\,dx \qquad x = \tan\theta \qquad dx = \sec^2\theta\ d\theta$$

$$\Rightarrow \int_0^{\frac{\pi}{4}} \frac{4}{1+\tan^2\theta}\sec^2\theta\ d\theta$$

$$\Rightarrow \int_0^{\frac{\pi}{4}} 4\times\cos^2\theta\frac{1}{\cos^2\theta}\,d\theta$$

$$\Rightarrow \int_0^{\frac{\pi}{4}} 4\ d\theta = \pi$$



ex1 : Numerical Integration Pi (C)

Hands-On

```
// KISTI educational PI calculation
// Copyright(c) 2012 KISTI Supercomputing Center

#include <stdio.h>
#include <math.h>

int main() {
    const long num_step=100000000;
    long i;
    double sum, step, pi, x;

    step = (1.0/(double)num_step);
    sum=0.0;
    printf("-------------------------------------------------------------");
    for(i=0; i<num_step; i++) {
        x = ((double)i - 0.5) * step;
        sum += 4.0/(1.0+x*x);
    }

    pi = step * sum;

    printf("PI = %17.15lf (Error = %e)\n", pi, fabs(acos(-1.0)-pi));
    printf("-------------------------------------------------------------");
    return 0;
}
```

ex1 : Numeracal Integration Pi (Fortran)

Hands-On

```
!! KISTI educational PI calculation
!! Copyright (c) 2012 KISTI Supercomputing Center

program Pi_cal
implicit none
  integer(kind=8), parameter :: num_step=100000000   !! Number of steps
  integer :: i
  double precision :: sum, step, pi, x

  step = (1.0d0/dble(num_step))
  sum = 0.0d0
  write(*,200)
  do i=0, num_step-1
      x = (dble(i)-0.5d0)*step
      sum = sum + 4.0d0/(1.0d0+x*x)
  enddo
  pi = step * sum

  write(*, 100) pi, dabs(dacos(-1.0d0)-pi)
  write(*, 200)

100 format("PI =", F17.15, "(Error =", E11.5,")")
200 format("-----------------------------------------")
  stop
end program
```

Gmsh (https://gmsh.info/)

Gmsh

A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities

Christophe Geuzaine and Jean-François Remacle

Download | Documentation | Licensing | Screenshots | Links | References |

Gmsh is an open source 3D finite element mesh generator with a built-in CAD engine and post-processor. Its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities. Gmsh is built around four modules: geometry, mesh, solver and post-processing. The specification of any input to these modules is done either interactively using the graphical user interface, in ASCII text files using Gmsh's own scripting language (`.geo` files), or using the C++, C, Python or Julia Application Programming Interface (API).

See this general presentation for a high-level overview of Gmsh and recent developments, the screencasts for a quick tour of Gmsh's graphical user interface, and the reference manual for a more thorough overview of Gmsh's capabilities, some frequently asked questions and the documentation of the C++, C, Python and Julia API.

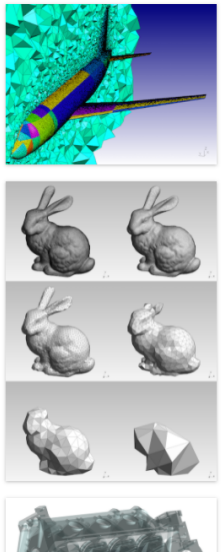
The source code repository contains many examples written using both the built-in script language and the API (see e.g. the tutorials and the and demos).

Download

Gmsh is distributed under the terms of the GNU General Public License (GPL):

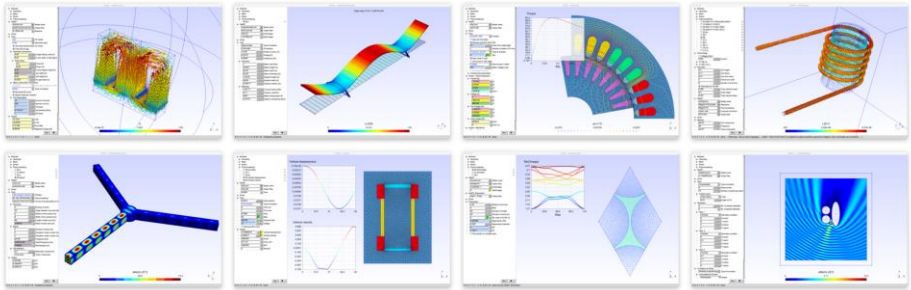
- **Current stable release (version 4.8.4, 28 April 2021):**
  - **Download Gmsh for Windows, Linux or macOS**
  - **Download the source code**
  - **Download the Software Development Kit (SDK) for Windows, Linux or macOS**
  - **Download both Gmsh and the SDK with pip:** `pip install --upgrade gmsh`

Make sure to read the tutorials before sending questions or bug reports.



ONELAB

Open Numerical Engineering LABoratory



ex 2 : Read Mesh File(C)

Hands-On

```
/* Read_Partitioned_Mesh_1.c */
#include <stdio.h>
#include <string.h>
void ReadMesh(char *Input, char* Output);
int main(void)
{
  int i;
  char Inputfile[20], Outputfile[20];
#pragma omp parallel sections private(Inputfile, Outputfile) num_threads(4)
{
  #pragma omp section
  {
    sprintf(Inputfile,"t1_1.msh");
    sprintf(Outputfile,"t1Out_1.msh");
    ReadMesh(Inputfile, Outputfile);
  }
  #pragma omp section
  {
    sprintf(Inputfile,"t1_2.msh");
    sprintf(Outputfile,"t1Out_2.msh");
    ReadMesh(Inputfile, Outputfile);
  }
```

ex 2 : Read Mesh File(C)

Hands-On

```
/* Read_Partitioned_Mesh_1.c */
#include <stdio.h>
#include <string.h>
void ReadMesh(char *Input, char* Output);
int main(void)
{
  int i;
  char Inputfile[20], Outputfile[20];
#pragma omp parallel sections private(Inputfile, Outputfile) num_threads(4)
{
   …
  #pragma omp section
  {
    sprintf(Inputfile,"t1_3.msh");
    sprintf(Outputfile,"t1Out_3.msh");
    ReadMesh(Inputfile, Outputfile);
  }
  #pragma omp section
  {
    sprintf(Inputfile,"t1_4.msh");
    sprintf(Outputfile,"t1Out_4.msh");
    ReadMesh(Inputfile, Outputfile);
  }
}
  return 0;
}
```

```
! Read_Partitioned_Mesh.f90
program main
implicit none
integer:: i
character(len=20)::Inputfile, Outputfile
character(len=1)::num
!$omp parallel do private(num, Inputfile, Outputfile), num_threads(4)
do i=1,4
  write(num,'(i1)') i
  Inputfile='t1_'//num//'.msh'
  Outputfile='t1Out_'//num//'.msh'
  call ReadMesh(trim(Inputfile), trim(Outputfile),10+i)
end do
!$omp end parallel do
end program main

!====================================
subroutine ReadMesh(Input, Output,Fid)
implicit none
character(len=*),intent(in)::Input,Output
integer,intent(in)::Fid
integer::i,k,tmp1,tmp2,tmp3,tmp4
integer::numnode, numele,eletype,num_l_ele_type, num_tri_ele_type,outFid
real(kind=8),allocatable::coord(:,:)
integer,allocatable::connect(:,:)
…
```

```
! Read_Partitioned_Mesh_1.f90
program main
implicit none
integer:: i
character(len=20)::Inputfile, Outputfile
character(len=1)::num

Inputfile='t1_1.msh'
Outputfile='t1Out_1.msh'
call ReadMesh(trim(Inputfile), trim(Outputfile),10)

Inputfile='t1_2.msh'
Outputfile='t1Out_2.msh'
call ReadMesh(trim(Inputfile), trim(Outputfile),10)

Inputfile='t1_3.msh'
Outputfile='t1Out_3.msh'
call ReadMesh(trim(Inputfile), trim(Outputfile),10)

Inputfile='t1_4.msh'
Outputfile='t1Out_4.msh'
call ReadMesh(trim(Inputfile), trim(Outputfile),10)

end program main
```

```
! Read_Partitioned_Mesh_1_openmp.f90
program main
implicit none
integer:: i
character(len=20)::Inputfile, Outputfile
character(len=1)::num

!$omp parallel sections private(Inputfile, Outputfile) num_threads(4)
!$omp section
Inputfile='t1_1.msh'
Outputfile='t1Out_1.msh'
call ReadMesh(trim(Inputfile), trim(Outputfile),10)

!$omp section
Inputfile='t1_2.msh'
Outputfile='t1Out_2.msh'
call ReadMesh(trim(Inputfile), trim(Outputfile),11)

!$omp section
Inputfile='t1_3.msh'
Outputfile='t1Out_3.msh'
call ReadMesh(trim(Inputfile), trim(Outputfile),12)

!$omp section
Inputfile='t1_4.msh'
Outputfile='t1Out_4.msh'
call ReadMesh(trim(Inputfile), trim(Outputfile),13)
!$omp end parallel sections

end program main
```

ex 2 : Read Mesh File(Fortran)

Hands-On

```
#include <stdio.h>
#include <string.h>
void ReadMesh(char *Input, char* Output);
int main(void)
{
  int i;
  char Inputfile[20], Outputfile[20];
#pragma omp parallel for private(i,Inputfile,Outputfile) num_threads(4)
  for(i=1;i<=4;i++)
  {
    sprintf(Inputfile,"t1_%1d.msh",i);
    sprintf(Outputfile,"t1Out_%1d.msh",i);
    ReadMesh(Inputfile, Outputfile);
  }
  return 0;
}
```

Laplace's equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

- Finite Difference Formulations

$$\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} = 0$$

  - $\beta \equiv \Delta x / \Delta y$

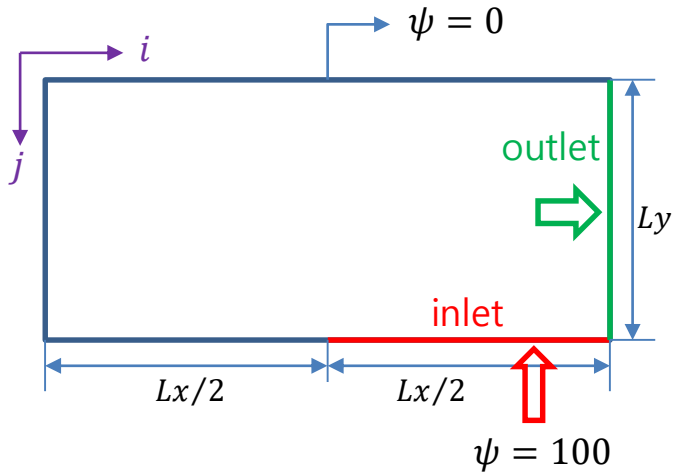
$$u_{i+1,j} + u_{i-1,j} + \beta^2 u_{i,j+1} + \beta^2 u_{i,j-1} - 2(1+\beta^2)u_{i,j} = 0$$

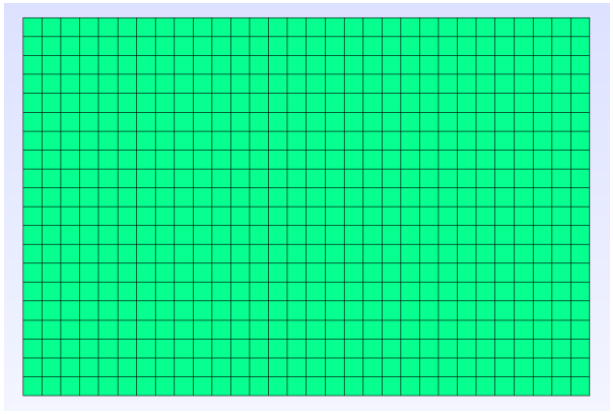
- Jacobi Iteration Method

$$u_{i,j}^{k+1} = \frac{1}{2(1+\beta^2)}\left[u_{i+1,j}^{k} + u_{i-1,j}^{k} + \beta^2\left(u_{i,j+1}^{k} + u_{i,j-1}^{k}\right)\right]$$

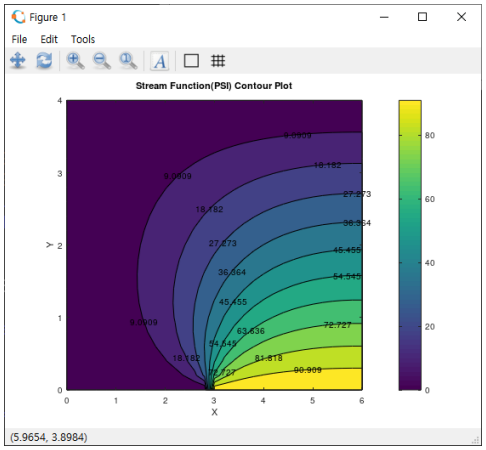
Stream_Function_Example-1 참조

- https://github.com/vishalk2/Computational-Fluid-Dynamics-CFD/tree/main/FDM/Elliptic/Stream_Function_Examples/Example_1
- Matlab file ➔ C, Fortran으로 변환





- M : # of nodes in i-direction
- N : # of nodes in j-direction



- 원본(matlab file) 실행 결과

```
/* stream.c */
#include <stdio.h>
#include <math.h>
#include <omp.h>
const double LX=6.0, LY=4.0;  // length of domain along x-, y-direction
const double EPSILON=1e-8;    // tolerance
const long int MAX_ITER=1000000000;

void Jacobi_iter(int N, int M, double psi_new[N][M],double beta,double beta_1);
void pre_post(int N, int M, double psi_new[N][M], double dx, double dy);

int main(void)
{
   const int M=90*3+1, N=60*3+1; // # of grid points starting from 0 in x-, y-
direction
//  const int M=271, N=181; // # of grid points starting from 0 in x-, y-direction

  double dx, dy, beta, beta_1;
  int i, j, divide;
  double psi_new[N][M];
  double STime, ETime;

  dx=LX/(M-1),  dy=LY/(N-1);
  beta=(dx/dy);
  beta_1=1.0/(2.0*(1.0+beta*beta));
```

```
#pragma omp parallel
{
  #pragma omp for private(i,j)
  for(i=0;i<N;++i)
    for(j=0;j<M;++j)
    {
      psi_new[i][j]=0.0;
    }

  // boundary conditions
  divide = (int)(M-1)*0.5;
  #pragma omp for private(i)
  for(i=0;i<divide;++i)
    psi_new[N-1][i]=0.0; // bottom(left)
  #pragma omp for private(i)
  for(i=divide;i<M;++i)
    psi_new[N-1][i]=100.0;  // bottom(right)
  #pragma omp for private(i)
  for(i=0;i<N;++i)
    psi_new[i][0]=0.0;  // left wall
  #pragma omp for private(i)
  for(i=0;i<M;++i)
    psi_new[0][i]=0.0;  // upper wall
}
```

```
// Jacobi_ieration
  Jacobi_iter(N,M,psi_new,beta,beta_1);

  // write mesh & post-processihng file
  pre_post(N, M, psi_new, dx, dy);
  return 0;
}
```

```
void Jacobi_iter(int N, int M,double psi_new[N][M],double beta,double beta_1)
{
  long int iter;
  double error=1.0;
  int i, j;
  double psi_old[N][M];

  for(iter=1;iter<MAX_ITER;++iter)
  {
    error=0.0;
  #pragma omp parallel
  {
    #pragma omp for private(i,j)
    for(i=0;i<M;++i)
      for(j=0;j<N;++j)
        psi_old[j][i]=psi_new[j][i];

    #pragma omp for private(i,j)
    for(i=1;i<M-1;++i)
      for(j=1;j<N-1;++j)
        psi_new[j][i]=beta_1*(psi_old[j][i+1]+psi_old[j][i-1]+
                              beta*beta*(psi_old[j+1][i]+psi_old[j-1][i]));
```

```
    // Right Neumann Boundary Condition
    #pragma omp for private(j)
    for(j=0;j<N;++j)
      psi_new[j][M-1]=psi_new[j][M-2];

    #pragma omp for private(i,j) reduction(+:error)
    for(i=0;i<M;++i)
      for(j=0;j<N;++j)
        error += (psi_new[j][i]-psi_old[j][i])*(psi_new[j][i]-psi_old[j][i]);
  }
    error = sqrt(error/(M*N));
    if(iter%1000==0) printf("Iteration = %ld, Error= %lg\n",iter,error);

    if(error<=EPSILON) break;
  }
  printf("Iteration = %ld, Error= %lg\n",iter,error);
}
```

```
! stream.f90
MODULE GLOBAL
IMPLICIT NONE
REAL(KIND=8),PARAMETER::LX=6.0D0, LY=4.0D0  !length of domain along x-,y-direction
INTEGER,PARAMETER::M=90*3+1, N=60*3+1         ! #of grid points starting from 1 in
x-,y-direction
REAL(KIND=8),PARAMETER::EPSILON=1E-8  ! TOLERANCE
INTEGER(KIND=8),PARAMETER::MAX_ITER=1000000000
REAL(KIND=8)::PSI_NEW(N,M),PSI_OLD(N,M)
END MODULE GLOBAL

PROGRAM MAIN
USE GLOBAL
IMPLICIT NONE
REAL(KIND=8)::DX,DY
REAL(KIND=8)::BETA, BETA_1
INTEGER::I,J
INTEGER::DIVIDE

DX=LX/(M-1);    DY=LY/(N-1)
BETA=(DX/DY)
BETA_1=1.0/(2*(1+BETA*BETA))
```

```
!$OMP PARALLEL DO
DO I=1,M
  DO J=1,N
    PSI_NEW(J,I)=0.0D0
    PSI_OLD(J,I)=0.0D0
  END DO
END DO
!$OMP END PARALLEL DO

!BOUNDARY CONDITIONS 1
DIVIDE=(M-1)*(5.0/10.0)
PSI_NEW(N,1:DIVIDE)=0.0D0  ! BOTTOM BOUNDARY
PSI_NEW(N,DIVIDE+1:M)=100.0D0    ! BOTTOM BOUNDARY
PSI_NEW(:,1)=0.0D0    ! LEFT BOUNDARY
PSI_NEW(1,:)=0.0D0    ! TOP BOUNDARY

! Jacobi_iteration
CALL JACOBI_ITER(DX,DY,BETA,BETA_1)

! write mesh & post-processing file
CALL PRE_POST(DX,DY)

END PROGRAM MAIN
```

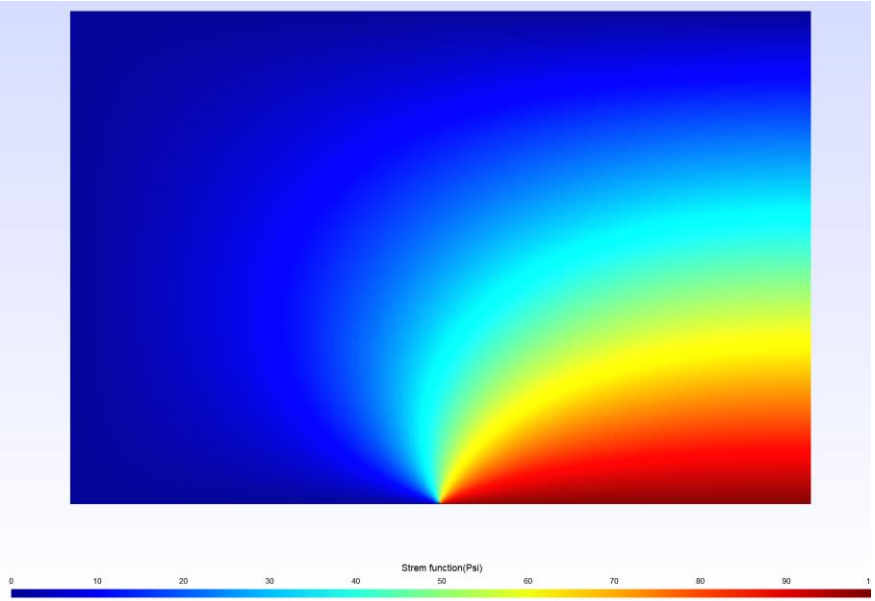
```
SUBROUTINE JACOBI_ITER(DX,DY,BETA,BETA_1)
USE GLOBAL
INTEGER::ITER=1
REAL(KIND=8)::ERROR=1.0D0
INTEGER::I,J
REAL(KIND=8)::DX,DY
REAL(KIND=8)::BETA, BETA_1

DO WHILE((ERROR>EPSILON) .AND. (ITER<=MAX_ITER))
  ERROR=0.0D0
!$OMP PARALLEL
  !$OMP DO
  DO I=1,M
    DO J=1,N
      PSI_OLD(J,I)=PSI_NEW(J,I)
    END DO
  END DO
  !$OMP END DO
  !$OMP DO
  DO I=2,M-1
    DO J=2,N-1
      PSI_NEW(J,I)=BETA_1*(PSI_OLD(J,I+1)+PSI_OLD(J,I-1) +
BETA*BETA*(PSI_OLD(J+1,I)+PSI_OLD(J-1,I)))
    END DO
  END DO
  !$OMP END DO
```

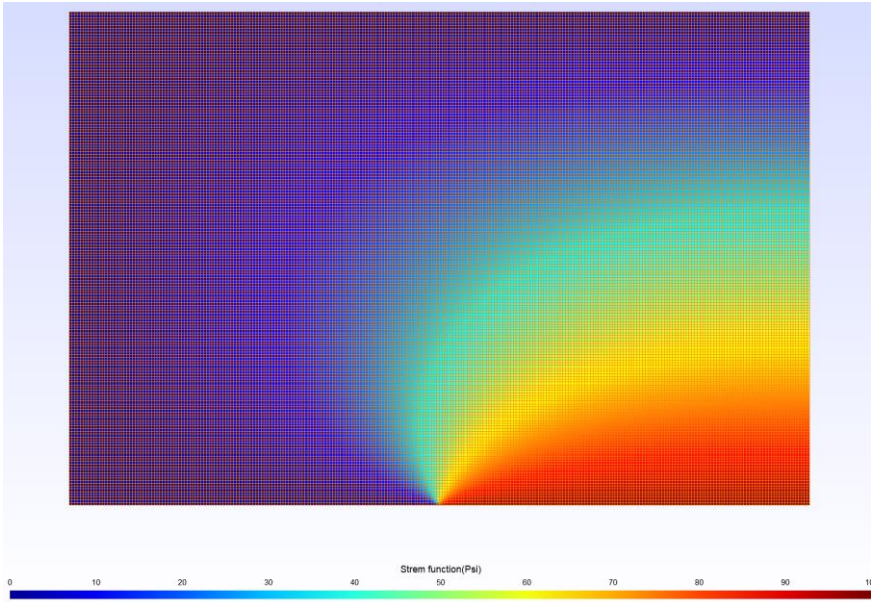
```
  ! Right Neumann Boundary Condition
  !$OMP DO
  DO J=1,N
    PSI_NEW(J,M)=PSI_NEW(J,M-1)
  END DO
  !OMP END DO

  !$OMP DO REDUCTION(+:ERROR)
  DO I=1,M
    DO J=1,N
      ERROR=ERROR+(PSI_NEW(J,I)-PSI_OLD(J,I))*(PSI_NEW(J,I)-PSI_OLD(J,I))
    END DO
  END DO
  !$OMP END DO
!$OMP END PARALLEL
  ERROR=SQRT(ERROR/(M*N))
  ITER=ITER+1   !ITERATOR
  IF(MOD(ITER,1000)==0) PRINT*,'ITERATION = ',ITER, 'ERROR=',ERROR
END DO
PRINT*,'ITERATION = ',ITER, 'ERROR=',ERROR

END SUBROUTINE JACOBI_ITER
```

ex 4 : 2D FDM(C)

- M=271, N=181
- Compile
  - gfortran stream.f90
  - gfortran stream.f90 -fopenmp
    - 최적화 옵션 사용 안 함

# of threads	Walltime(sec.)	Speed-up
1 (serial)	777	1x
2	424	1.8x
4	217	3.6x
8	113	6.9x
16	71	10.9x
32	48	16.2x





예제 실습

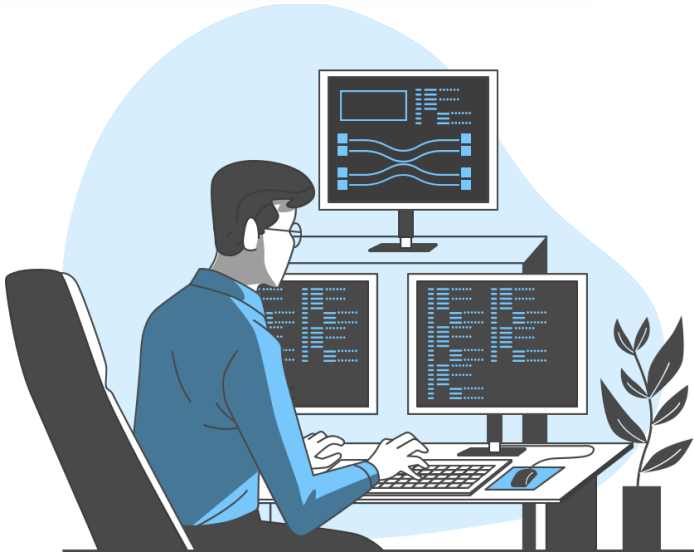
- Pi 값 계산
  - Reduction 보조 지시어 사용
- 여러 파일 동시에 읽기
  - #pragma omp for를 이용해서 스레드 별로 파일을 읽어 들이게 함
  - #pragma omp sections를 이용해서 스레드 별로 파일을 읽어 들이게 함
- 2D FDM
  - reduction이용
  - 스레드 개수를 변경 시켜가면서 병렬 성능 측정

예제 실습

- Pi 값 계산
- 여러 파일 동시에 읽기
- 2D FDM

12 Summary

1. Parallel block, fork-join model
2. Parallel loop
3. review



학습목표

- 전체 강의 내용을 복습한다

Parallel Block (1/2)

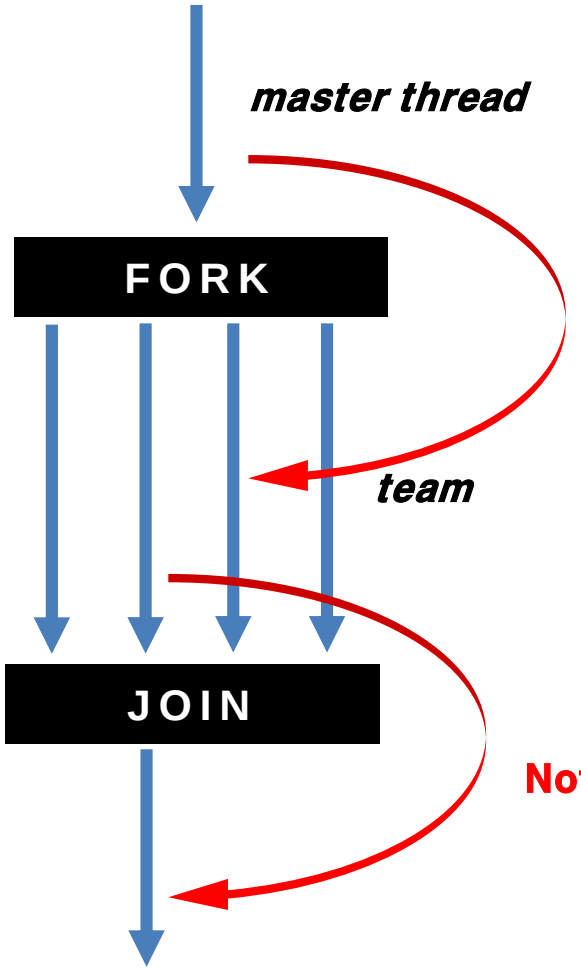
Fortran

```
PROGRAM wrong_parallel_block
   implicit none
   INTEGER omp_get_thread_num

   call omp_set_num_threads(4)

   goto 100
!$OMP PARALLEL
100 if ( omp_get_thread_num() == 1 ) &
       goto 200
!$OMP END PARALLEL

200 continue

!$OMP PARALLEL
   if ( omp_get_thread_num() == 1 )
  then
     !$OMP BARRIER
   end if
!$OMP END PARALLEL

END
```

C

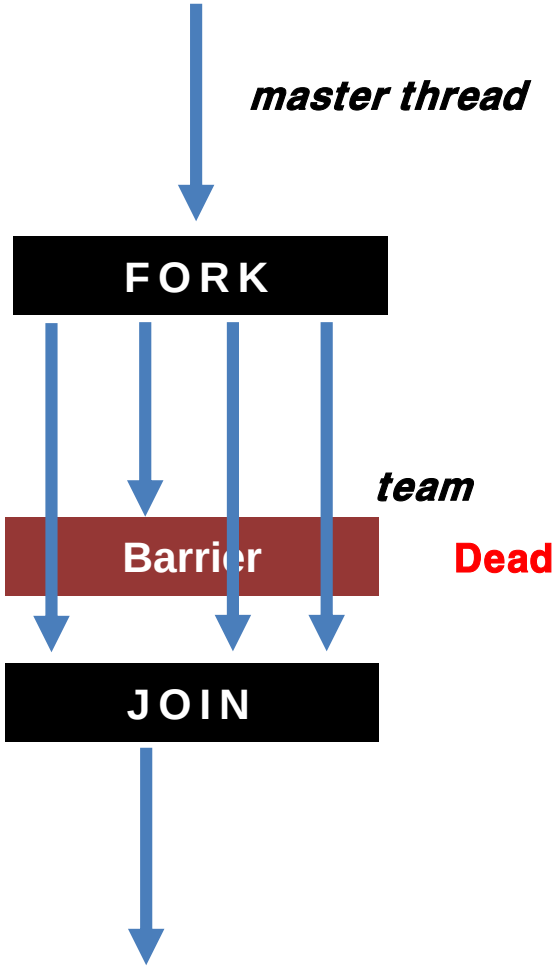
```
#include <stdio.h>
#include <omp.h>

int main()
{
   omp_set_num_threads(4);
    goto L1;
#pragma omp parallel
{
L1:
   if( omp_get_thread_num() == 1 )
      goto L2;
}
L2:
#pragma omp parallel
{
   if( omp_get_thread_num() == 1 ) {
      #pragma omp barrier
   }
}
}
```

Parallel Block (2/2)



Not allowed

Not allowed

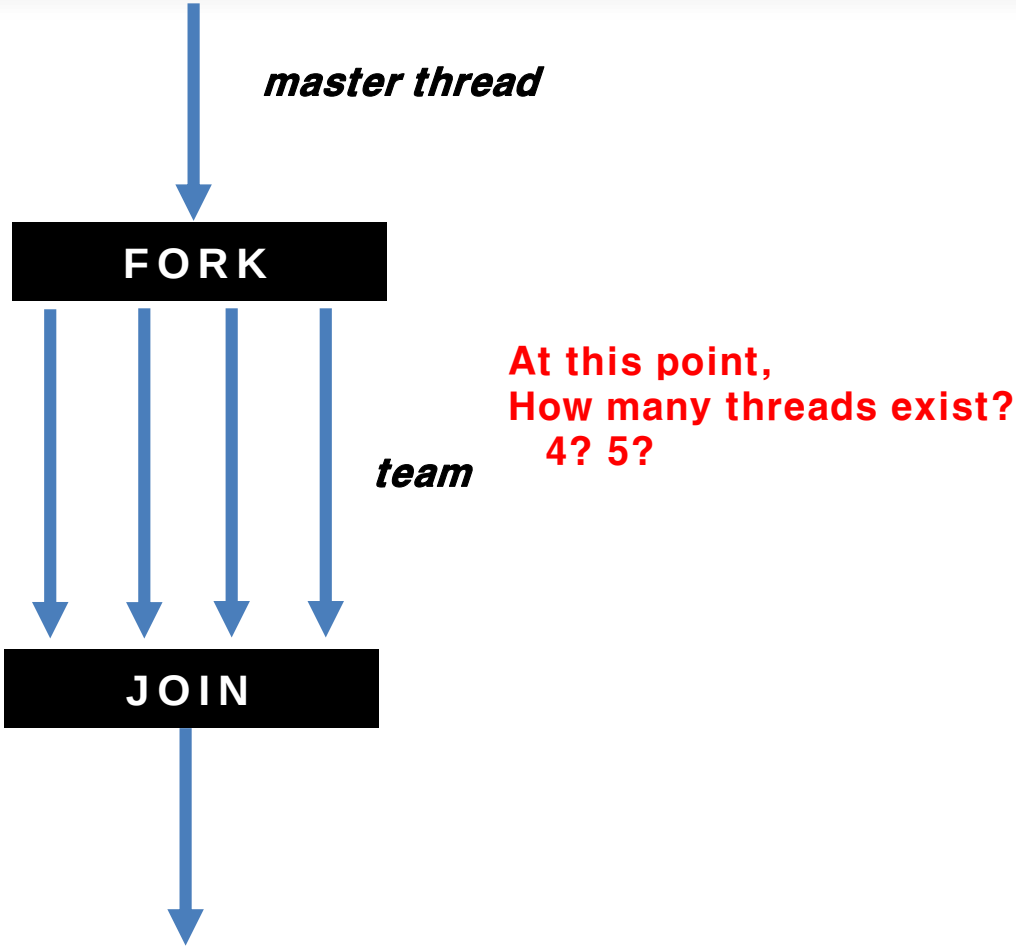


Deadlock

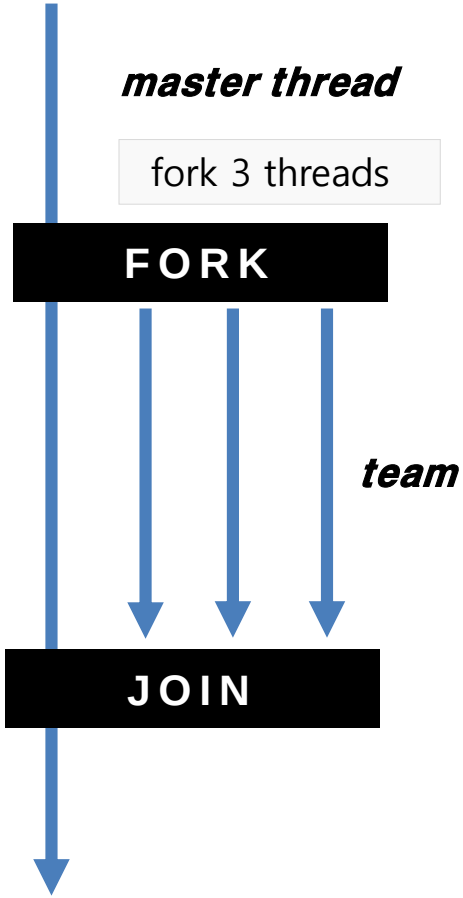
Only exit() is possible inside parallel region

Fork-Join Model

- 마스터 스레드는 병렬 영역의 시작 부분에서 새로운 스레드를 생성함



Fork-Join Model

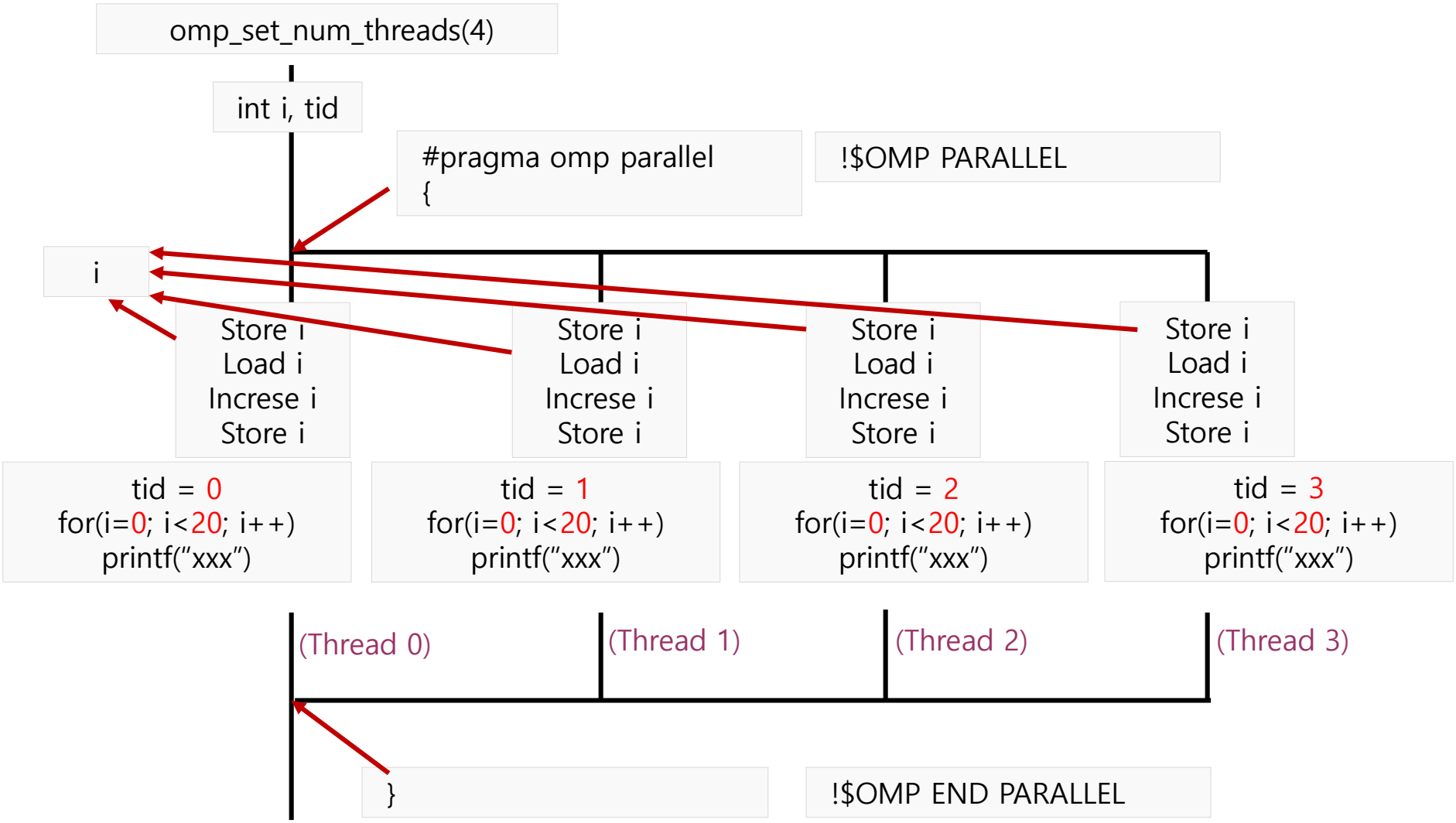


Fortran	C
PROGRAM fork_join_test INTEGER :: a(0:3),tid,omp_get_thread_num a=0 call omp_set_num_threads(4) !$OMP PARALLEL private(a, tid) tid = omp_get_thread_num() a(tid) = tid print *, 'a(', tid, ') =', a(tid), 'in', tid, '-th thread' a(0) = 100 !$OMP END PARALLEL print *, 'a(0) = ', a(0) END	#include <stdio.h> #include <omp.h> int main() { int a[4] = { 0 }, tid; omp_set_num_threads(4); #pragma omp parallel private(a,tid) { tid = omp_get_thread_num(); a[tid] = tid; printf("a[%d] = %d in %d-th thread\n", tid, a[tid], tid); a[0] = 100; } printf("a[0] = %d\n", a[0]); }

Parallel Loops (1/3)

Fortran	C
PROGRAM wrong_parallel_loop implicit none INTEGER, PARAMETER :: N=20 INTEGER :: tid, i, omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL private(tid) tid = omp_get_thread_num() !$OMP DO DO i=0, N-1 PRINT *, 'Hello World', tid, i END DO !$OMP END PARALLEL END	#include <stdio.h> #include <omp.h> #define N 20 int main() { int tid, i; omp_set_num_threads(4); #pragma omp parallel private(tid) { tid = omp_get_thread_num(); #pragma omp for for(i=0; i<N; i++) printf("Hello World %d %d\n", tid, i); } }

Parallel Loops (2/3)



Parallel Loops (3/3)

Fortran	C
PROGRAM nested_parallel_do implicit none INTEGER, PARAMETER :: N=20 INTEGER :: tid, i, omp_get_thread_num call omp_set_num_threads(4) !$OMP PARALLEL private(tid) tid = omp_get_thread_num() **!$OMP PARALLEL DO !! ???** **!! netsted parallel (tomorrow)** DO i=0, N-1 PRINT *, 'Hello World', tid, i END DO !$OMP END PARALLEL END	#include <stdio.h> #include <omp.h> #define N 20 int main() { int tid, i; omp_set_num_threads(4); #pragma omp parallel private(tid) { tid = omp_get_thread_num(); **#pragma omp parallel for // ???** **// nested parallel** for(i=0; i<N; i++) printf("Hello World %d %d\n", tid, i); } }

OpenMP Compile Option

- GCC : -fopenmp, Intel : -qopenmp, Cray : -h omp

OpenMP Components and Syntax

- Compiler Directives, Runtime Library (functions), Environment Variables

Create Threads

- Parallel region
- Set/Get # of thread, Get thread ID
- Fork/Join model

Data Scope

- shared, private, firstprivate

Parallel Loop

- do/for
- Work Sharing

Synchronization

- critical, atomic, barrier

Reduction

Create Threads (Nested Parallelism)

Synchronization

- nowait, ordered

Schedule

Task

- task, taskwait

OpenMP Performance

- Nested Parallel (Collapse)
- Flush
- False Sharing
- Data Dependency

Hands-on

Q&A


Thank
you!