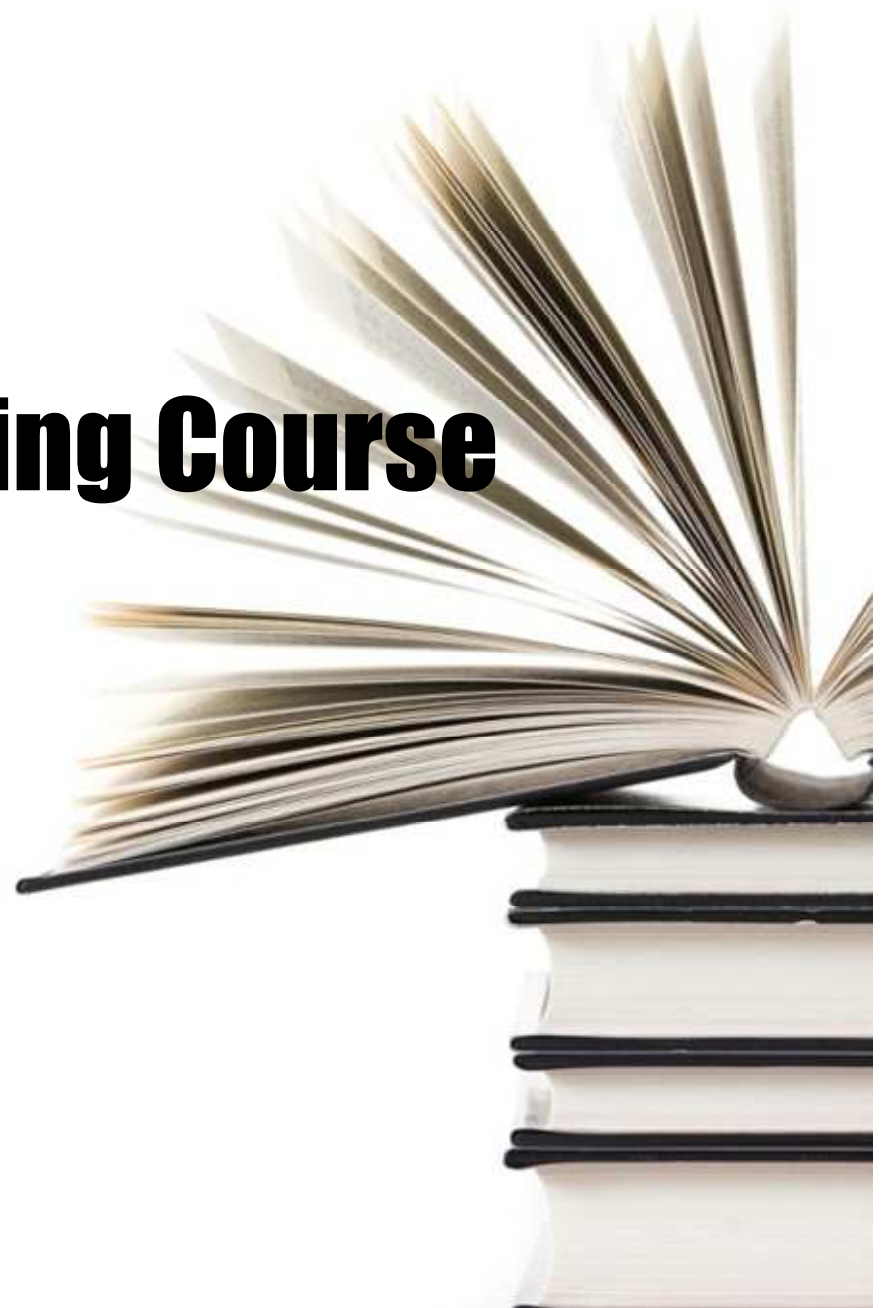


OpenMP Training Course

Part 1 Introduction



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





Resources



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





Agenda (Basic Course)



- 09:30 – 10:00 Motivation & Introduction to Education System
- 10:00 – 12:00 OpenMP Basics I
- 12:00 – 13:30 Lunch
- 13:30 – 14:50 OpenMP Basics II
- 14:50 – 15:00 Break
- 15:00 – 16:30 OpenMP Basics III & Hands-on
- 16:30 – 17:00 Summary



Agenda (Basic Course)



- Motivation
 - Why OpenMP?
 - Evolution of Processor
- Introduction to Education System
 - Check Environment
- OpenMP Basics I
 - Introduction to OpenMP
 - Creating Threads [2.4, 3.2]
 - Data Scope Attributes [2.4]
- OpenMP Basics II
 - Parallel Loops [2.5.1, 2.6.1]
 - Synchronization [2.8.2, 2.8.5, 2.8.3]
- OpenMP Basics III & Hands-on
 - Reduction [2.4, 2.5, 2.6.1]
 - Pi : formula (I), Numerical Integration (II), Monte Carlo (III)
- Summary (Review)



Agenda (Advanced Course)



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



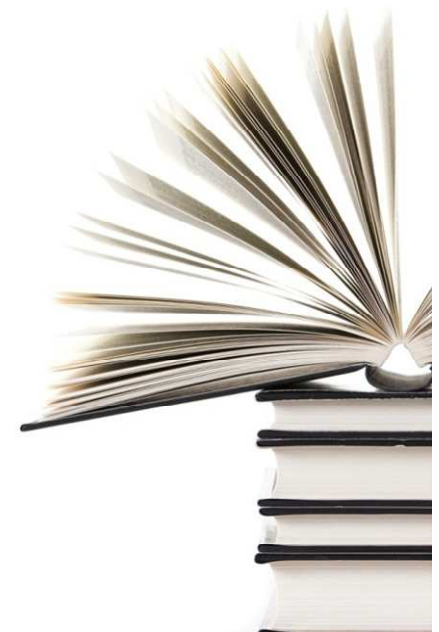
Agenda (Advanced Course)



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

Motivation

- 1. Why Parallel Computing ?**
- 2. Why OpenMP?**
- 3. Evolution of Processor**





Why Parallel Computing ?



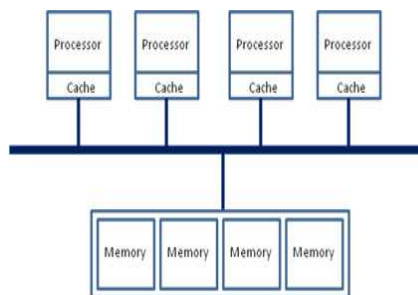
- Limits of serial computing
- Solve larger problems
- Provide concurrency
- Save time and Money
- Use of non-local resources



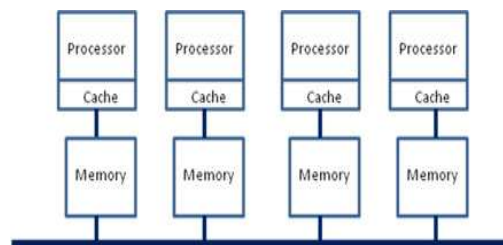
Why Parallel Computing ?



- Shared Memory
 - Single address space for all processors



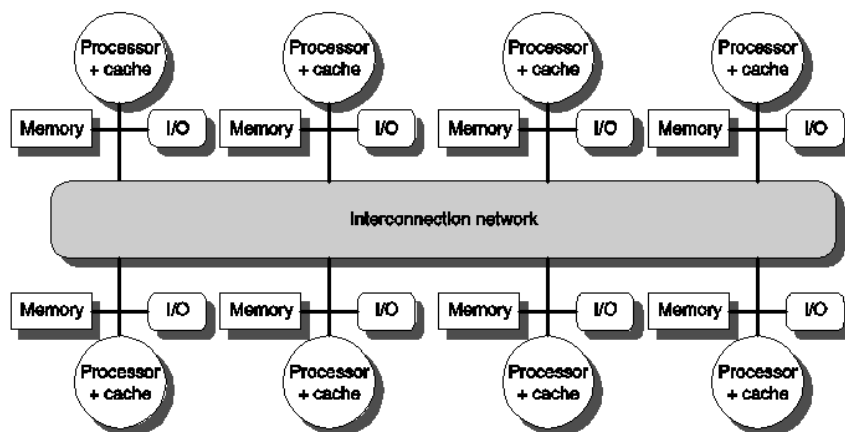
<UMA>



<NUMA>



- Distributed Memory



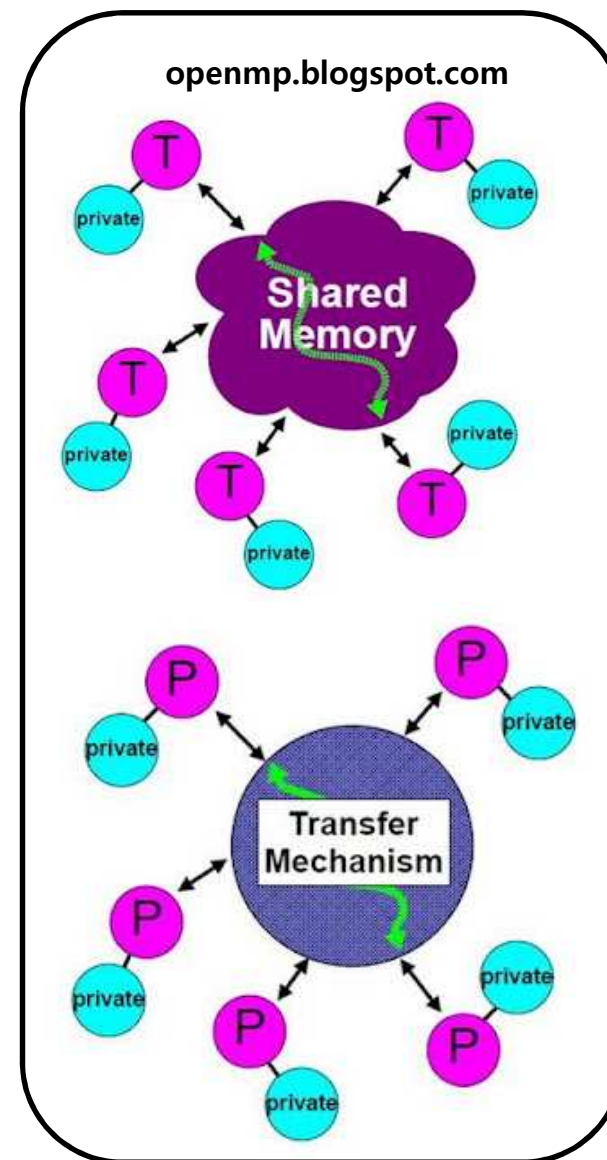


Why Parallel Computing ?



- Shared Memory
 - share a global memory space
 - multiple cores
 - cores can efficiently exchange, share data

- Distributed Memory
 - each node uses its own local memory
 - collection of nodes which have multiple cores
 - communicate between nodes and cores via messages





Why Parallel Computing ?



- Directives-based parallel programming language(OpenMP)
 - Directives tell processor how to distribute data and work across the processors
 - Directives appear as comments in the serial code
 - implemented on shared memory architectures

- Distributed Memory(MPI)
 - Pass messages to send/receive data between processes
 - Each process has its own local variables
 - Can be used on either shared or distributed memory architectures



Why Parallel Computing ?



ABCPL	CORRELATE	GLU	Mentat	Parafraze2	
ACE	CPS	GUARD	Legion	Paralation	pC++
ACT++	CRL	HaSL	Meta Chaos	Parallel-C++	SCHEDULE
Active messages	CSP	Haskell	Midway	Parallaxis	SciTL
Adl	Cthreads	HPC++	Millipede	ParC	SDDA
Adsmith	CUMULVS	JAVAR.	CparPar	ParLib++	SHMEM
ADDAP	DAGGER	HORUS	Mirage	ParLin	SIMPLE
AFAPI	DAPPLE	HPF	MpC	Parmacs	Sina
ALWAN	Data Parallel C	IMPACT	MOSIX	Parti	SISAL
AM	DC++	ISIS.	Modula-P	pC	distributed smalltalk
AMDC	DCE++	JAVAR	Modula-2*	PCN	SMI.
AppLeS	DDD	JADE	Multipol	PCP:	SONiC
Amoeba	DICE.	Java RMI	MPI	PH	Split-C.
ARTS	DIPC	javaPG	MPC++	PEACE	SR
Athapascan-0b	DOLIB	JavaSpace	Munin	PCU	Sthreads
Aurora	DOME	JIDL	Nano-Threads	PET	Strand.
Automap	DOSMOS.	Joyce	NESL	PENNY	SUIF.
bb_threads	DRL	Khoros	NetClasses++	Phosphorus	Synergy
Blaze	DSM-Threads	Karma	Nexus	POET.	Telegraphos
BSP	Ease	KOAN/Fortran-S	Nimrod	Polaris	SuperPascal
BlockComm	ECO	LAM	NOW	POOMA	TCGMSG.
C*.	Eiffel	Lilac	Objective Linda	POOL-T	Threads.h++.
"C* in C	Eilean	Linda	Occam	PRESTO	TreadMarks
C**	Emerald	JADA	Omega	P-RIO	TRAPPER
CarLOS	EPL	WWWinda	OpenMP	Prospero	uC++
Cashmere	ERLANG	ISETL-Linda	Orca	Proteus	UNITY
C4	Express	ParLin	OOF90	pthreads	UC
CC++	Falcon	Eilean	P++	PVM	V
Chu	Filaments	P4-Linda	P3L	PSI	ViC*
Charlotte	FM	POSYBL	Pablo	PSDM	Visifold V-NUS
Charm	FLASH	Objective-Linda	PADE	Quake	VPE
Charm++	The FORCE	LiPS	PADRE	Quark	Win32 threads
Cid	Fork	Locust	Panda	Quick Threads	WinPar
Cilk	Fortran-M	Lparx	Papers	QPC++	XENOOPS
CM-Fortran	FX	Lucid	AFAPI.	Sage++	XPC
Converse	GA	Maisie	Para++	SCANDAL	Zounds
Code	GAMMA	Manifold	Paradigm	SAM	ZPL
COOL	Glenda				



Why OpenMP?



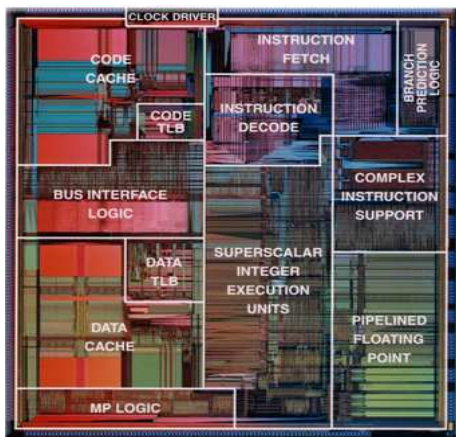
- De-facto and mature standard
- An OpenMP program is portable
- Good performance and scalability
- Requires little programming effort



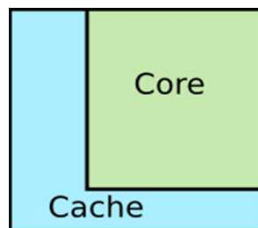
Evolution of Intel Processor



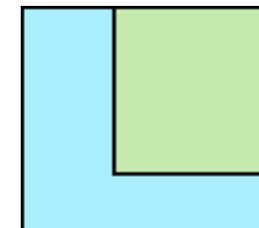
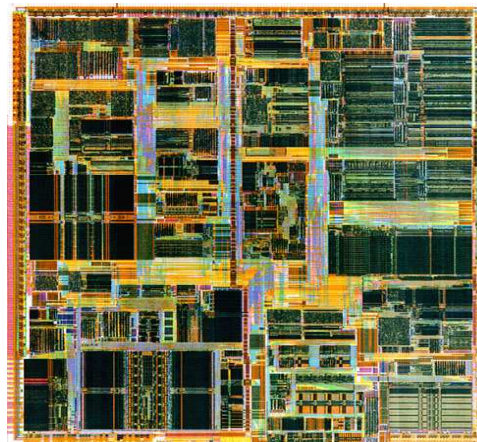
Pentium I



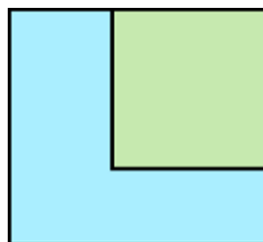
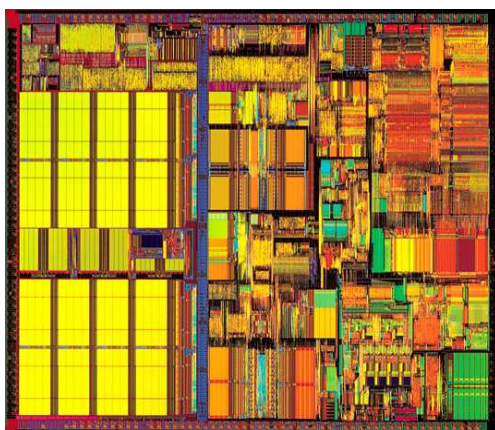
Chip area
breakdown



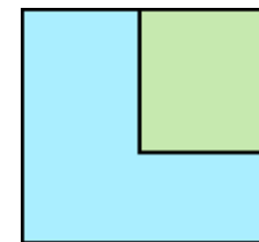
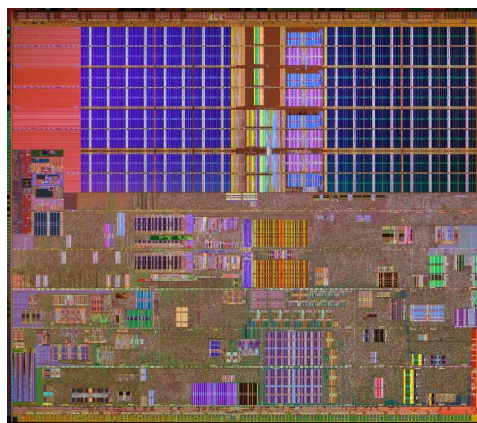
Pentium II



Pentium III



Pentium IV

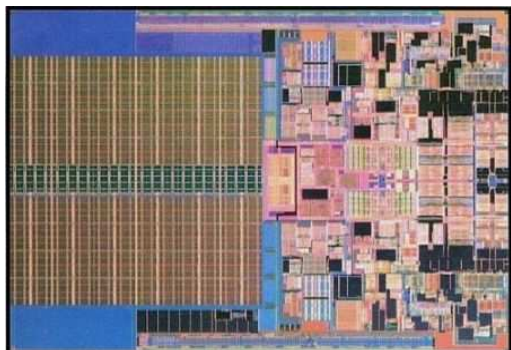




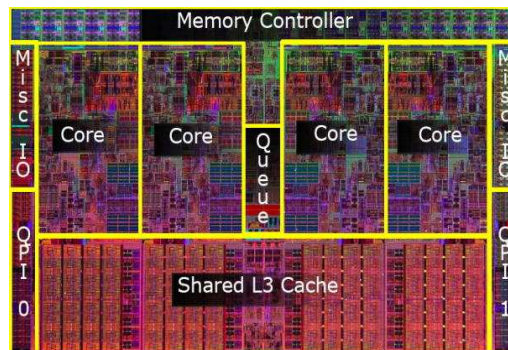
Evolution of Multi-core CPU



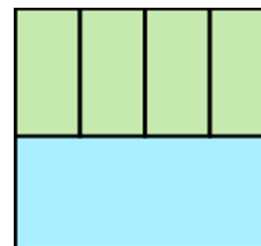
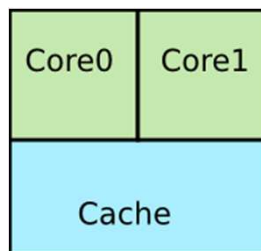
Penryn



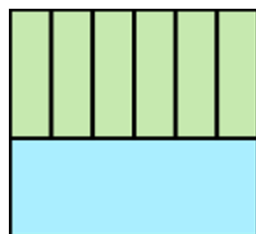
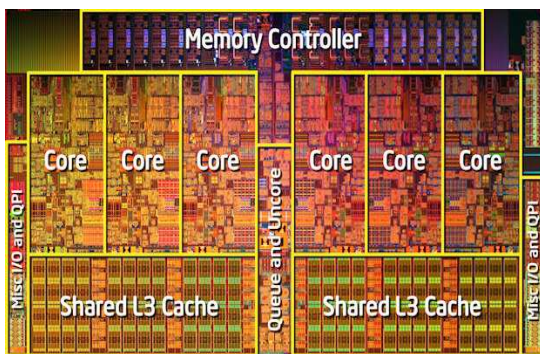
Bloomfield



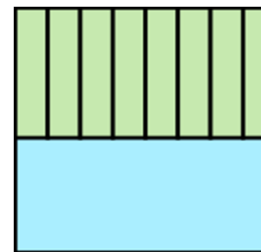
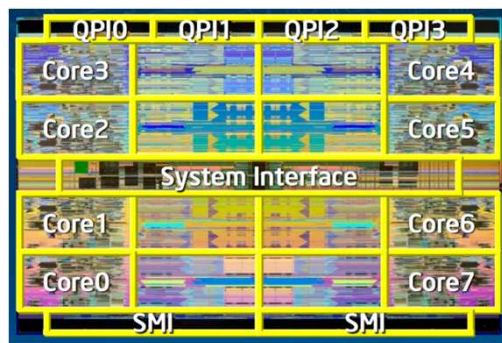
Multi-core



Gulftown

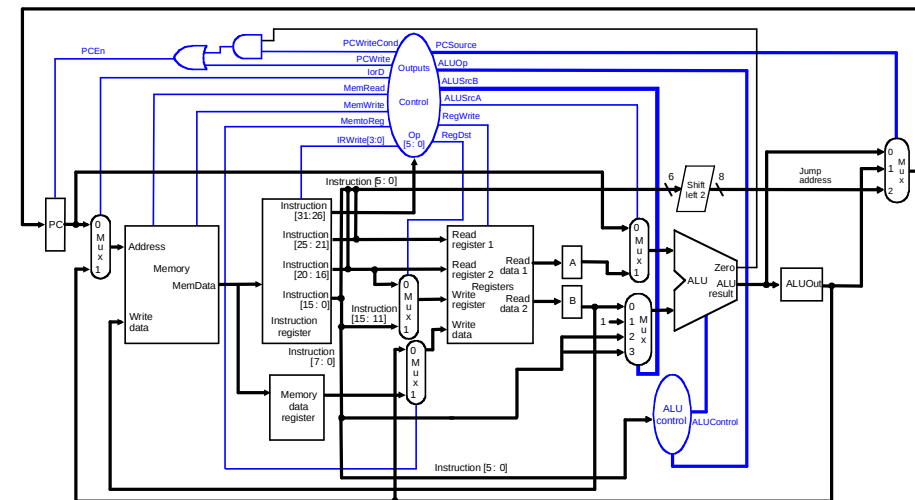
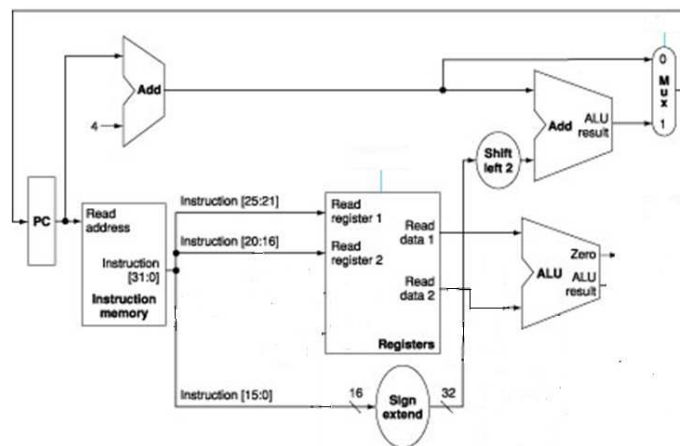
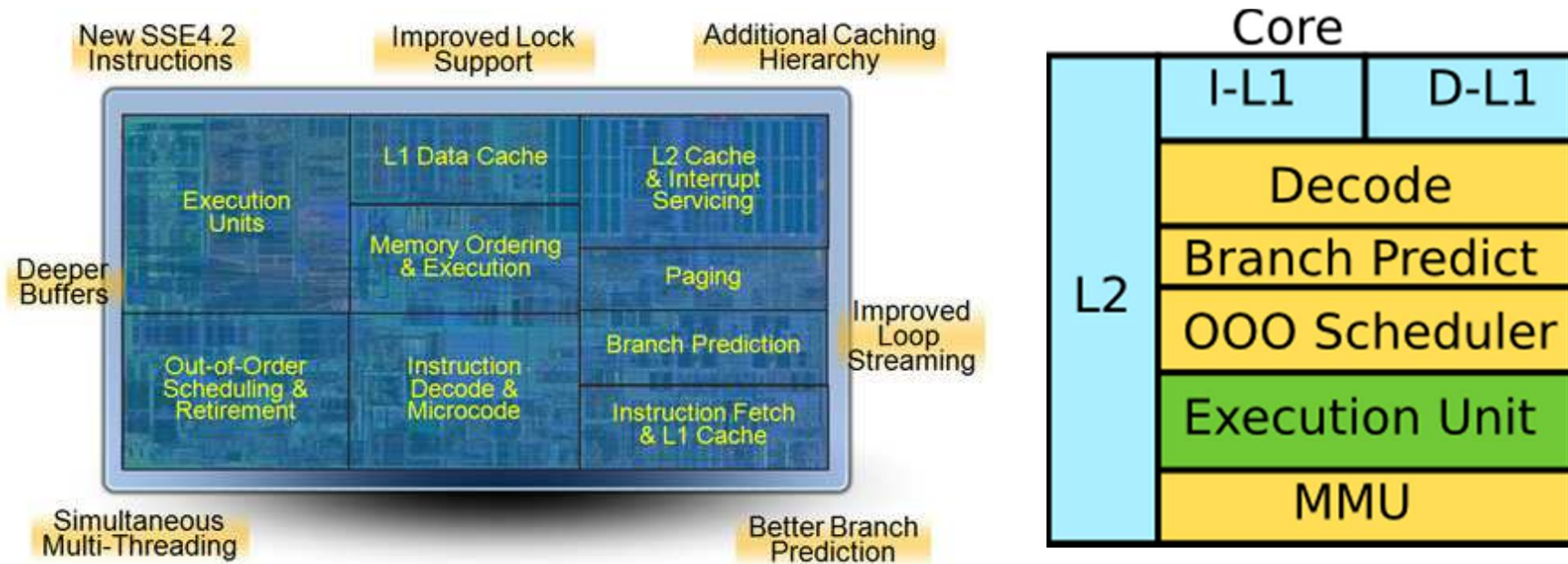


Beckton





Design for Performance

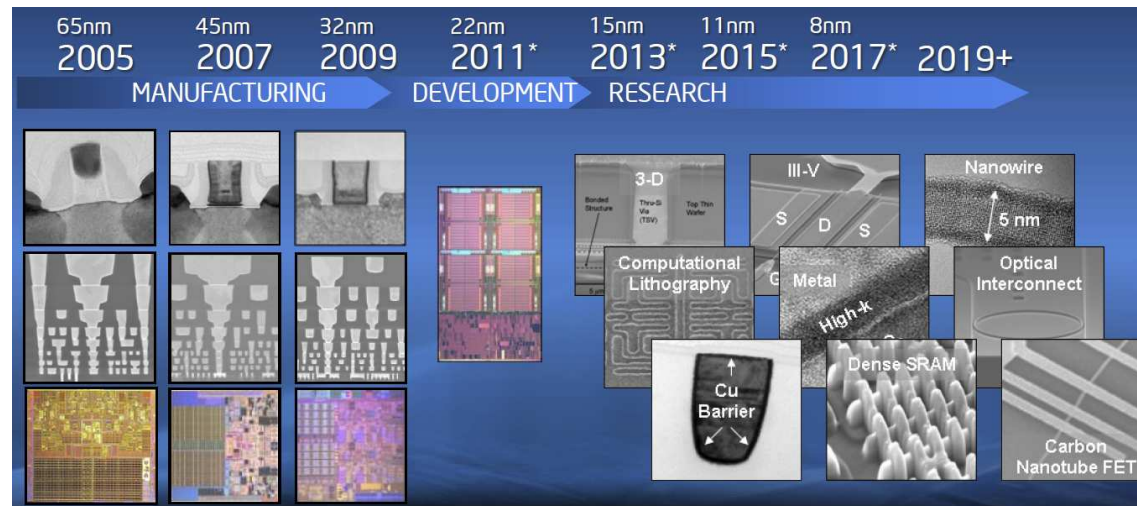




Tick-Tock Development Model



- Moore's Law
 - The number of transistors on a chip will double about every two years





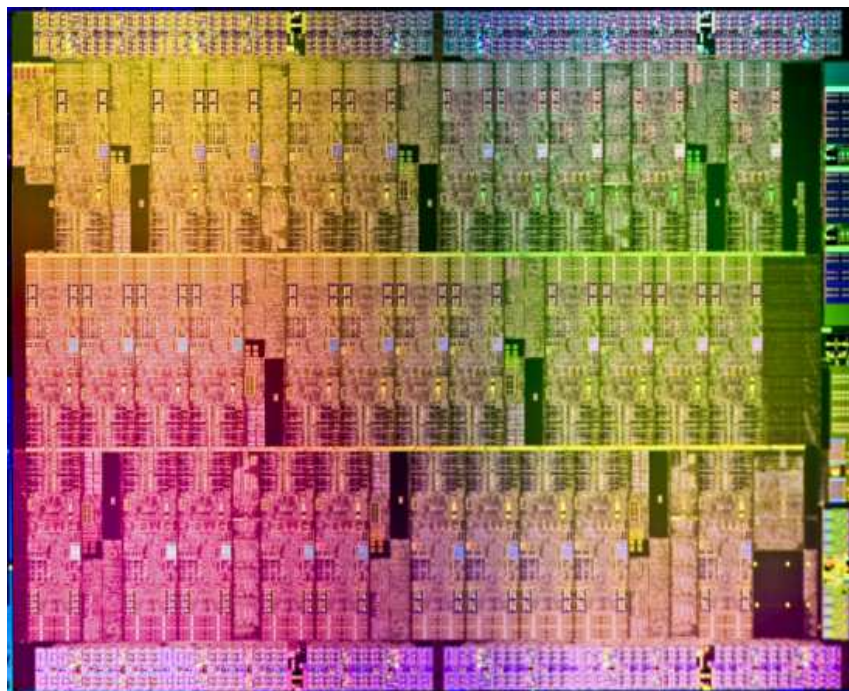
Intel MIC (Many Integrated Core)



- Knights Ferry
 - Software development platform
 - 32 cores, 1.2 GHz
 - 128 threads at 4 threads/core
 - 8MB shared coherent cache
 - 1-2GB GDDR5



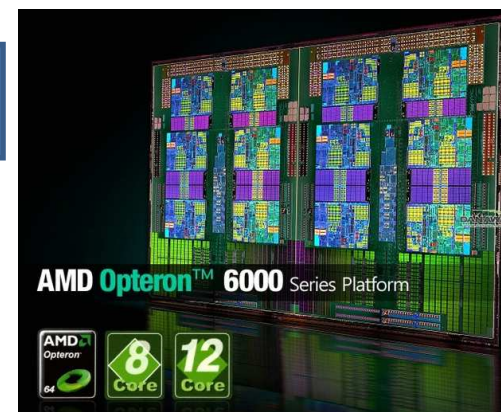
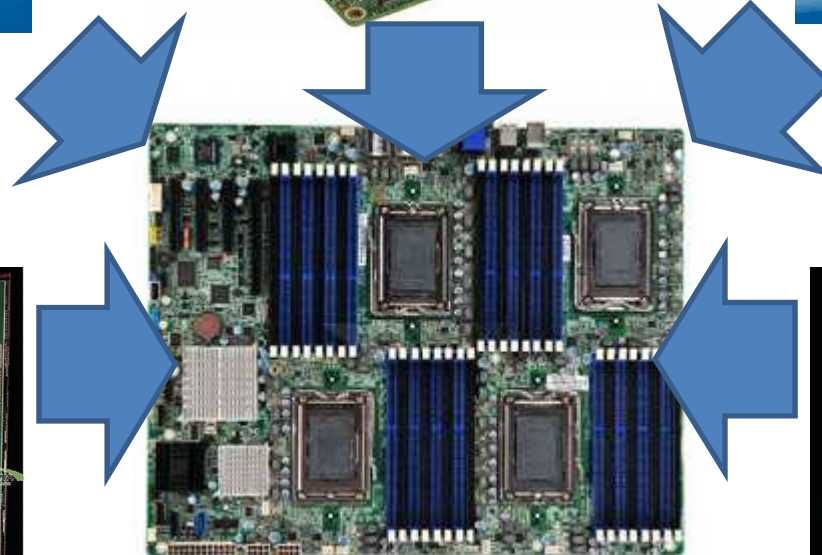
- Knights Corner
 - 1st Intel MIC product
 - 22 nm process
 - 57~61 intel architecture cores
 - 4 threads/core
 - 512bit vector unit/core
 - 6-8 GB GDDR5
 - Scalar unit(x86) & vector unit



- Future Knights Products



Many-core System

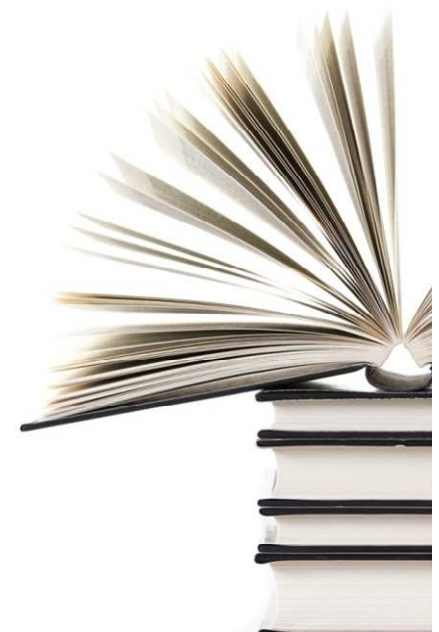


Break!!



Introduction to Education System

- 1. Education System**
- 2. Check Environment**



PuTTY 설정

문류(G)

- 세션
 - 로그인
 - 터미널
 - 키보드
 - 벨
 - 기능
 - 창
 - 모양
 - 특성
 - 변환
 - 선택
 - 색깔
 - 접속
 - 데이터
 - 프락시
 - 텔넷
 - rlogin
 - SSH
 - 시리얼

PuTTY 세션 기본 옵션

접속 대상 정보

Host Name (or IP address) Port

접속 형식:

☐ 생짜 (R) ☐ Telnet ☐ Rlogin ☒ SSH ☐ 시리얼

저장된 세션의 불러오기, 저장, 지움

저장된 세션 (E)

교육용슈퍼컴

기본 설정

교육용슈퍼컴

불러옴(L)

저장(V)

지움(D)

종료시에 창을 닫음 (W):

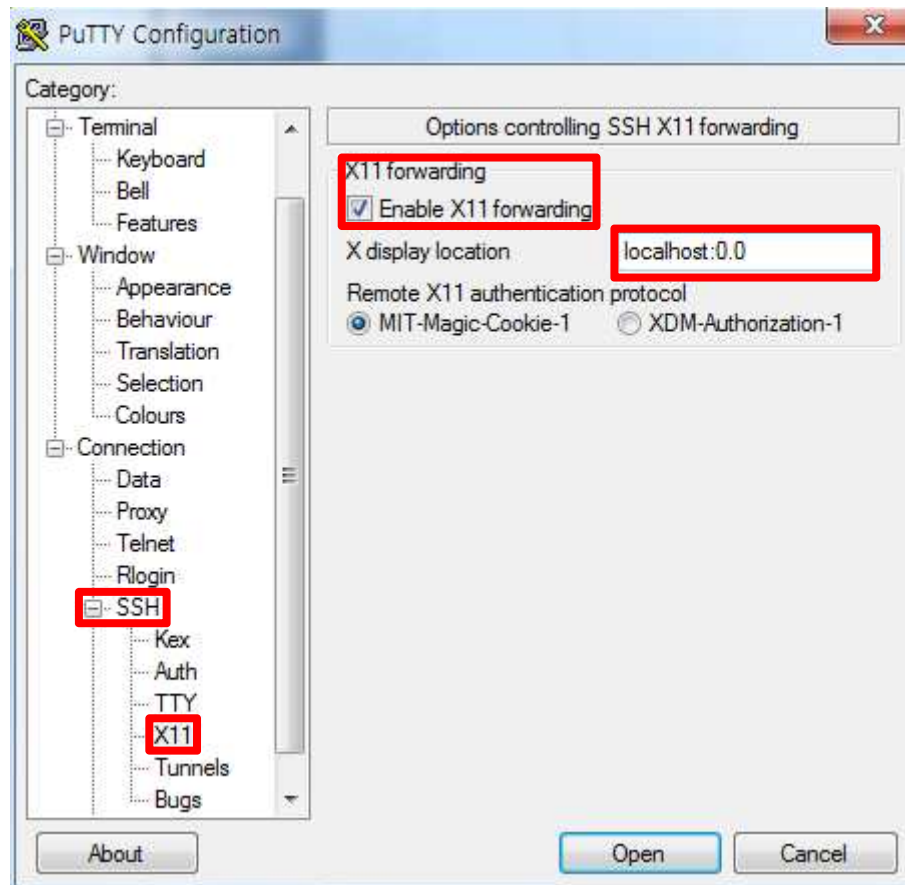
☐ 항상 ☐ 안 닫음 ☒ 접속이 끊겼을 때만

도움 도움 (H) 열기 (O) 취소 (C)

Putty 설정



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



윈도우에서
Xming 실행

➤ PuTTY Security Alert [예(y)] 선택





Putty 설정



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

```
edun20@master01:~  
login as: edun20  
edun20@ password:  
Last login: Wed Sep 21 11:08:31 2011 from 150.183.108.120  
998% [edun20@master01 ~]#
```

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

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





Checking Environment



➤ Move to debugging and computing node

- Use **ssh**
- Usage : ssh s000#
- ex) ssh s0001

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

➤ Compiler Environment

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

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



Checking Environment



Fortran	C
<pre>PROGRAM hello_world !\$OMP PARALLEL PRINT *, 'Hello World' !\$OMP END PARALLEL END</pre>	<pre>#include <stdio.h> int main() { #pragma omp parallel { printf ("Hello World\n"); } return 0; }</pre>
<pre>\$ gfortran -o hello.x hello.f90 \$ gfortran -fopenmp -o hello_omp.x hello.f90 \$./hello.x \$./hello_omp.x</pre>	<pre>\$ gcc -o hello.x hello.c \$ gcc -fopenmp -o hello_omp.x hello.c \$./hello.x \$./hello_omp.x</pre>
<pre>Intel : ifort -o hello.x hello.f90 GCC : gfortran -o hello.x hello.f90 PGI : pgf90 -o hello.x hello.f90</pre>	<pre>Intel : icc -o hello.x hello.c GCC : gcc -o hello.x hello.c PGI : pgcc -o hello.x hello.c</pre>
<pre>Intel : ifort -openmp -o hello_omp.x hello.f90 GCC : gfortran -fopenmp -o hello_omp.x hello.f90 PGI : pgf90 -mp -o hello_omp.x hello.f90</pre>	<pre>Intel : icc -openmp -o hello_omp.x hello.c GCC : gcc -fopenmp -o hello_omp.x hello.c PGI : pgcc -mp -o hello_omp.x hello.c</pre>

for output filename

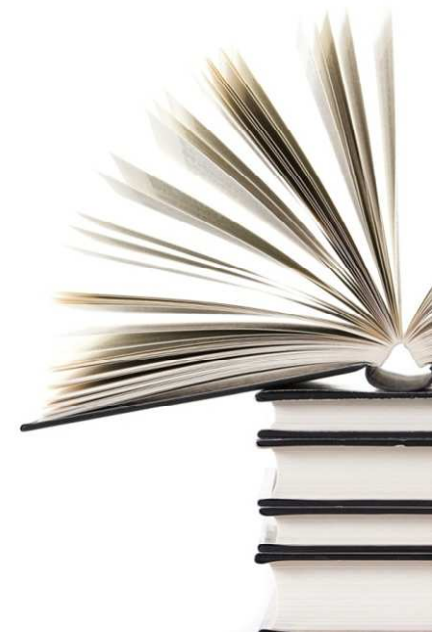
for output filename

for OpenMP

for OpenMP

OpenMP Basics I

- 1. Introduction to OpenMP**
- 2. Create Threads**
- 3. Data Scope Attribute**



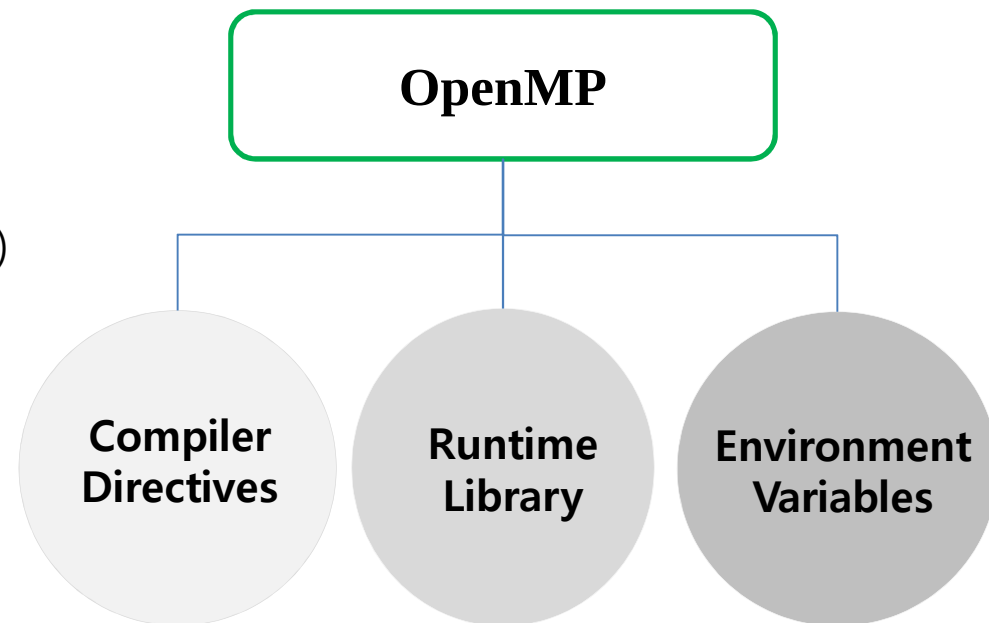


What is OpenMP ?



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

- Consists of
 - Compiler Directives
 - Runtime Library (Functions)
 - Environment Variables





Components of OpenMP (1/2)



- Compiler Directives
 - communicate with the compiler on parallelism
ex) `!$OMP PARALLEL DO`
- Runtime Library (Functions)
 - enables the setting and querying of parallel parameters such as number of participating threads and the thread number
ex) `CALL omp_set_num_threads(128)`
- Environmental Variables
 - define runtime system parallel parameters such as the number of threads
ex) `export OMP_NUM_THREADS=8`



Components of OpenMP (2/2)



Fortran	C
<pre>PROGRAM omp_component INTEGER omp_get_thread_num !\$OMP PARALLEL PRINT *, 'Hello World', omp_get_thread_num() !\$OMP END PARALLEL END</pre> Compiler Directive Runtime Library	<pre>#include <stdio.h> #include <omp.h> int main() { #pragma omp parallel { printf ("Hello World %d\n", omp_get_thread_num()); } return 0; }</pre> Compiler Directive Runtime Library
<pre>\$ gfortran -fopenmp -o omp_component.x omp_component.f90 \$ export OMP_NUM_THREADS=4 \$./omp_component.x</pre> Environment Variable	<pre>\$ gcc -fopenmp -o omp_component.x omp_component.c \$ export OMP_NUM_THREADS=4 \$./omp_component.x</pre> Environment Variable

➤ OpenMP Syntax

	Fortran	C
Compiler Directive	!\$OMP <directive>	#pragma omp <directive>
Runtime Library	omp_	omp_
Environmental Variable	OMP_	OMP_



Creating Threads (1/4)



Fortran	C
<pre>PROGRAM create_thread IMPLICIT NONE INTEGER omp_get_thread_num !\$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</pre>	<pre>#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()); } return 0; }</pre>
<pre>\$ gfortran -fopenmp -o create_thread.x create_thread.f90 \$ export OMP_NUM_THREADS=8 \$./create_thread.x</pre>	<pre>\$ gcc -fopenmp -o create_thread.x create_thread.c \$ export OMP_NUM_THREADS=8 \$./create_thread.x</pre>



Creating Threads (2/4)



➤ Parallel Region

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

➤ Set number of thread

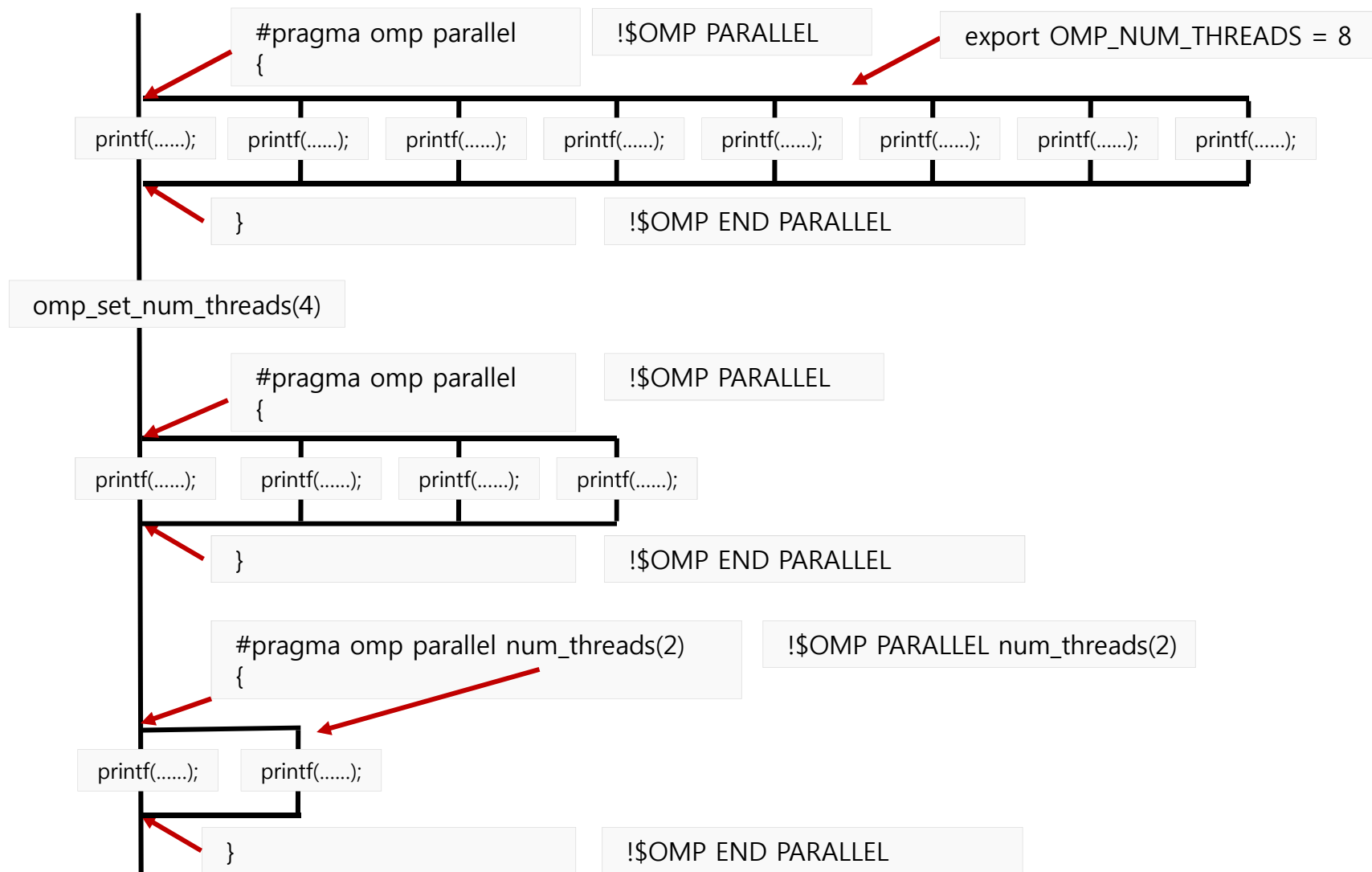
- Environment Variable : export OMP_NUM_THREADS = xxx
- Runtime Library : omp_set_num_threads(xxx)
- Directive : #pragma omp parallel num_threads(xxx)

➤ Runtime libraries related to thread

- omp_set_num_threads(integer) : Affects the number of threads
- omp_get_num_threads() : Returns the number of threads in the current team
- omp_get_thread_num() : Returns the ID of the encountering thread
- omp_get_max_threads()
- omp_get_num_procs()



Creating Threads (3/4)

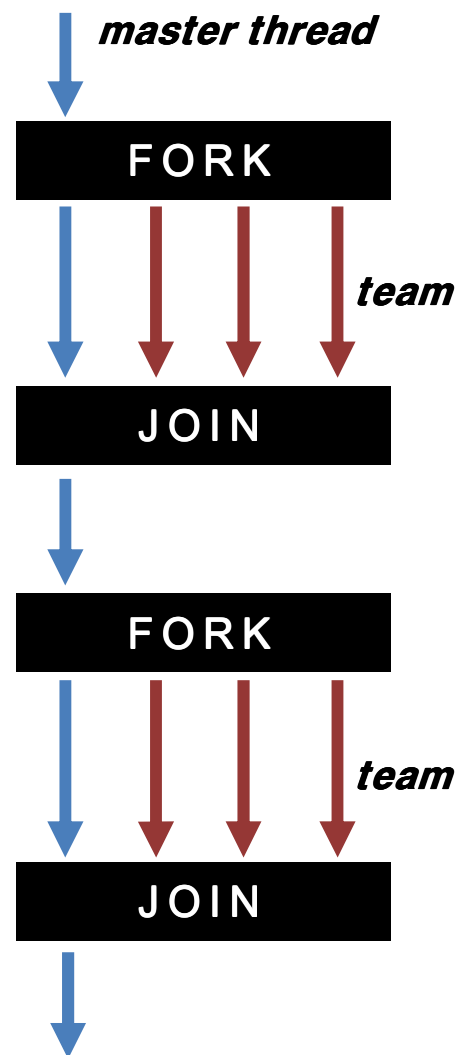




Creating Threads (4/4)



- OpenMP Programming Model
 - Thread-Based
 - Fork-Join Model
- Fork-Join Model
 - The master thread spawns a **team** of threads that joins at the end of the **parallel region**
 - Threads in the same team can **collaborate** to do work





Create Threads (Exercise)



Fortran	C
<pre>PROGRAM create_thread2 INTEGER ?? !! print number of threads PRINT *, "threads =", ??? !! create 3 threads !! print thread id and number of threads PRINT *, "tid =", ???, "threads =", ??? !! end multi threads !! print number of threads PRINT *, "threads =", ??? !! create 5 threads using Environment Variable !! print thread id and number of threads PRINT *, "tid", ???, "threads =", ??? !! end multi threads !! print number of threads PRINT *, "threads =", ??? END</pre>	<pre>#include <stdio.h> #include <omp.h> int main() { // print number of threads printf("threads = %d\n", ???); // create 3 threads { // print thread id and number of threads printf("tid = %d threads = %d\n", ???, ???); } // print number of threads printf("threads = %d\n", ???); // create 5 threads using Environment Variable { // print thread id and number of threads printf("tid = %d threads = %d\n", ???, ???); } // print number of threads printf("threads = %d\n", ???); return 0; }</pre>



Create Threads (solution)



Fortran	C
<pre> PROGRAM create_thread2 IMPLICIT NONE INTEGER omp_get_num_threads, omp_get_thread_num PRINT *, "threads =", omp_get_num_threads() !\$OMP PARALLEL num_threads(3) PRINT *, "tid =", omp_get_thread_num(), & "threads =", omp_get_num_threads() !\$OMP END PARALLEL PRINT *, "threads =", omp_get_num_threads() !\$OMP PARALLEL PRINT *, "tid", omp_get_thread_num(), & "threads =", omp_get_num_threads() !\$OMP END PARALLEL PRINT *, "threads =", omp_get_num_threads() END </pre>	<pre> #include <stdio.h> #include <omp.h> int main() { printf("threads = %d\n", omp_get_num_threads()); #pragma omp parallel num_threads(3) { printf("tid = %d threads = %d\n", omp_get_thread_num(), omp_get_num_threads()); } printf("threads = %d\n", omp_get_num_threads()); #pragma omp parallel { printf("tid = %d threads = %d\n", omp_get_thread_num(), omp_get_num_threads()); } printf("threads = %d\n", omp_get_num_threads()); return 0; } </pre>
<pre> \$ export OMP_NUM_THREADS=5 \$ gfortran -fopenmp -o create_thread2.x create_thread2.f90 \$./create_thread2.x </pre>	<pre> \$ export OMP_NUM_THREADS=5 \$ gcc -fopenmp -o create_thread2.x create_thread2.c \$./create_thread2.x </pre>

Break!!

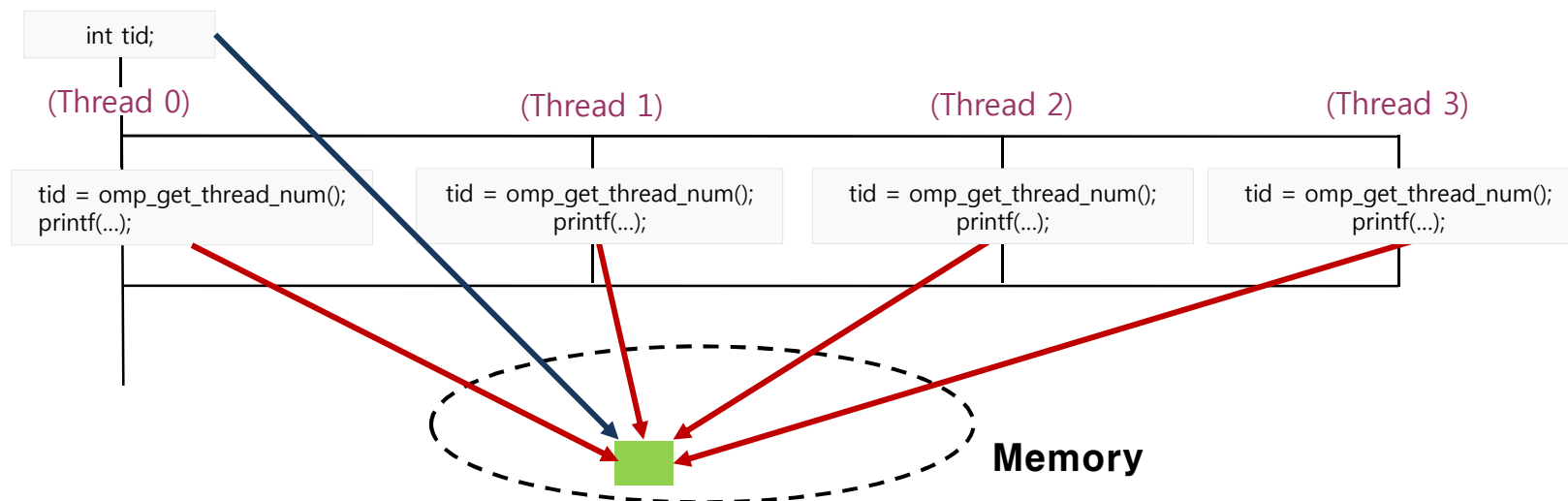




Data Scope Attribute (1/7)

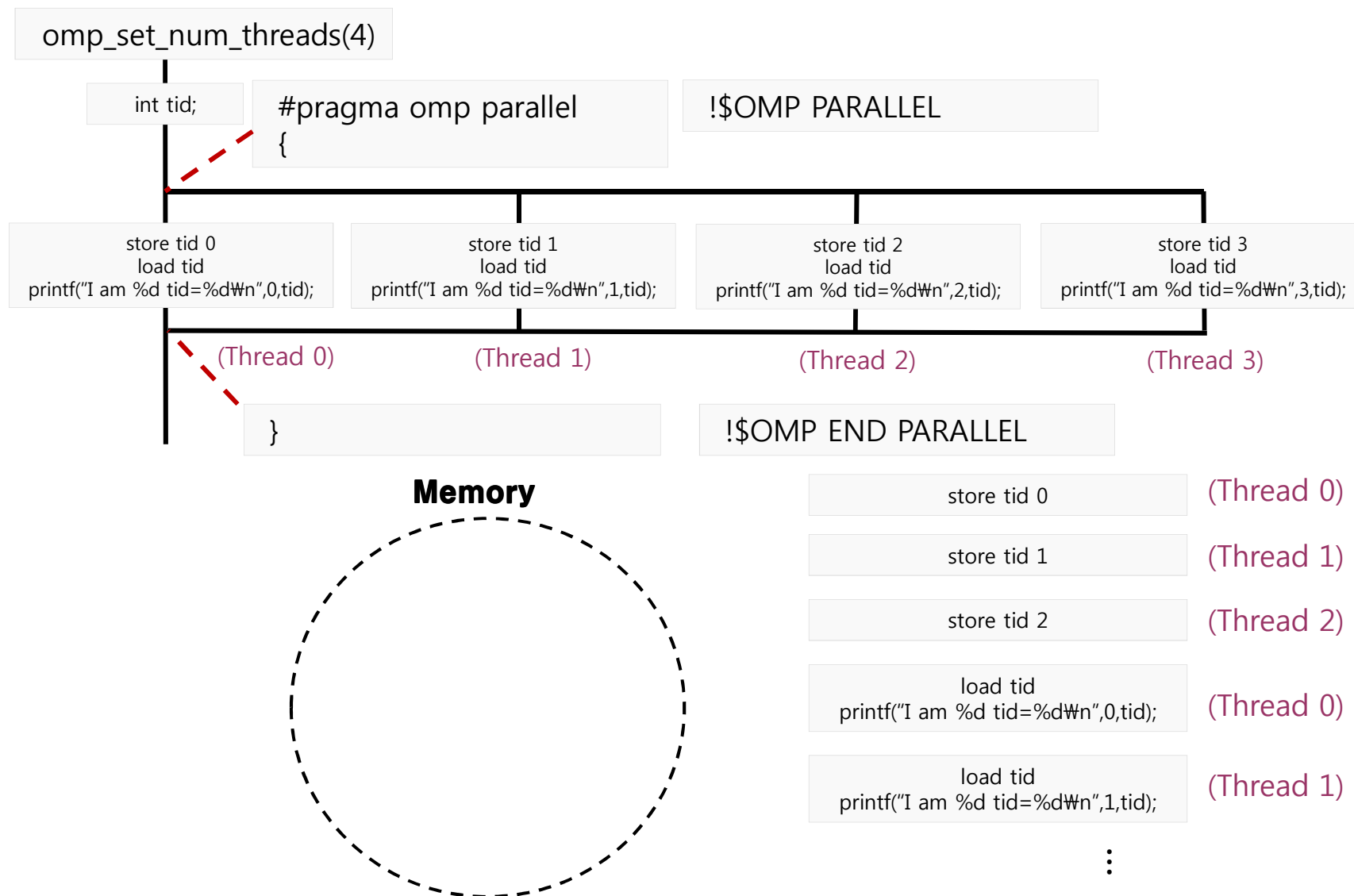


Fortran	C
<pre>PROGRAM hello_wrong INTEGER tid, omp_get_thread_num CALL omp_set_num_threads(4) !\$OMP PARALLEL tid = omp_get_thread_num() PRINT *, ' I am ', omp_get_thread_num(), ' tid = ', tid !\$OMP END PARALLEL END</pre>	<pre>#include <stdio.h> #include <omp.h> int main() { int tid; omp_set_num_threads(4); #pragma omp parallel { tid = omp_get_thread_num(); //sleep(1); printf ("I am %d tid = %d\n", omp_get_thread_num(), tid); } return 0; }</pre>



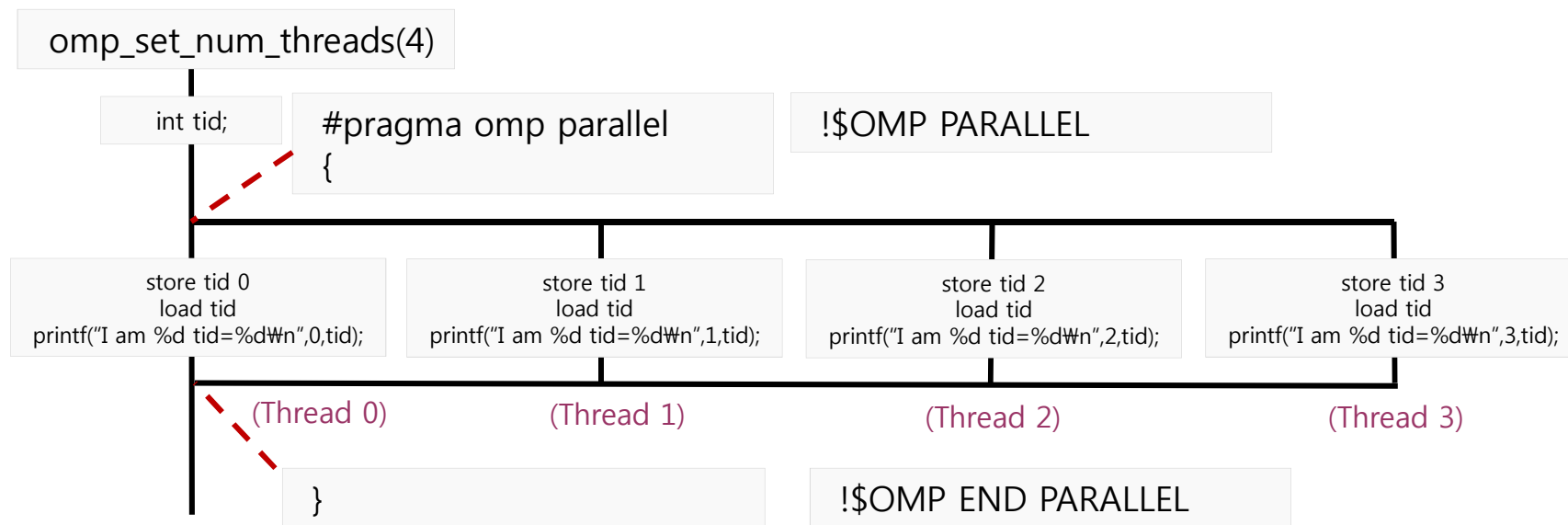


Data Scope Attribute (1/7)





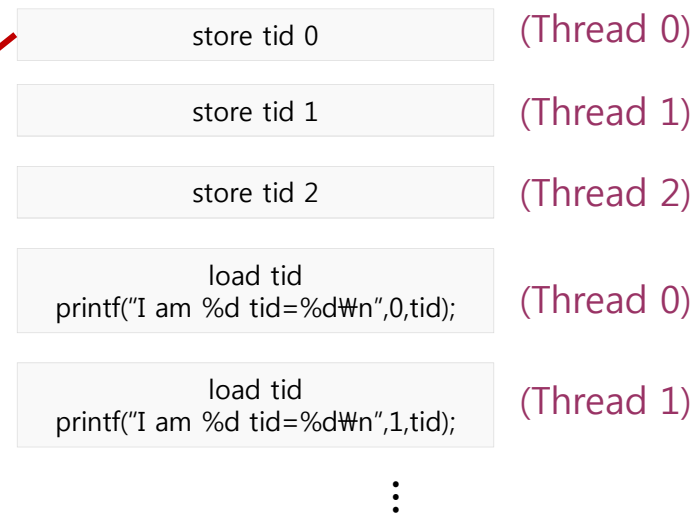
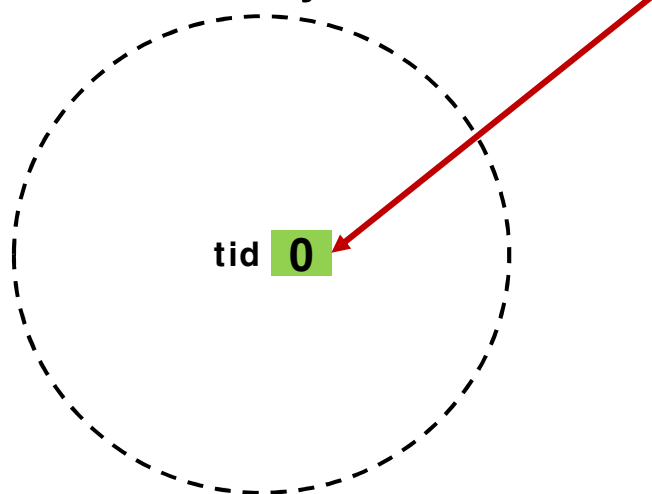
Data Scope Attribute (1/7)



Scenario

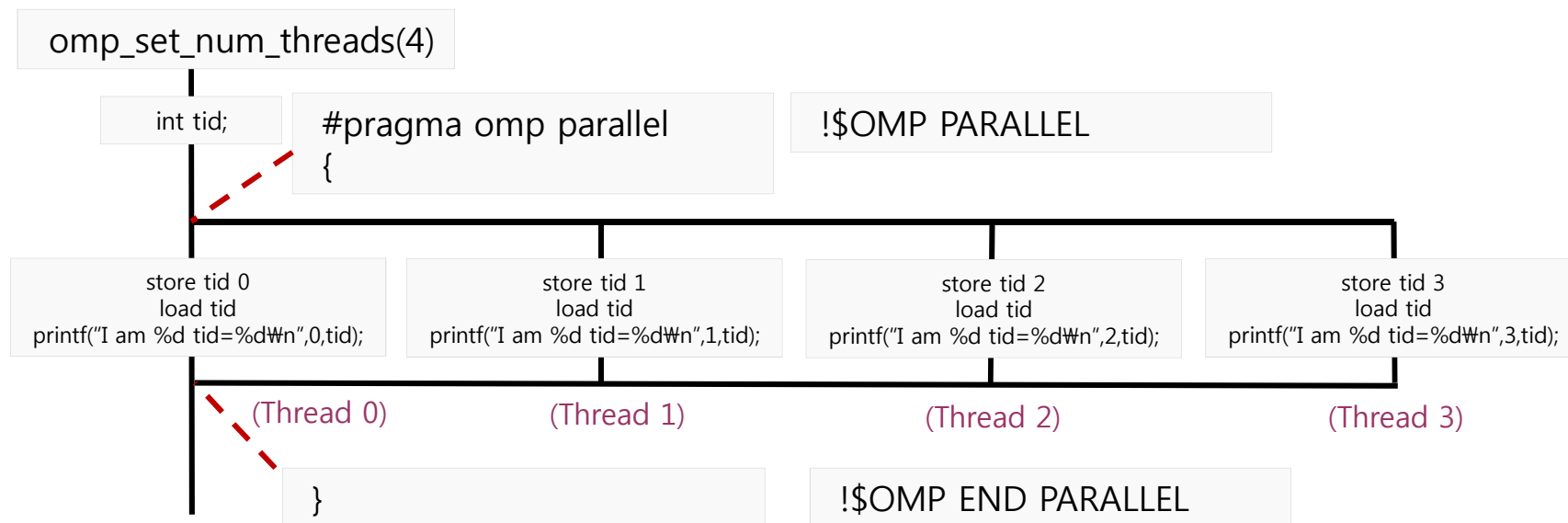
\$./a.out

Memory





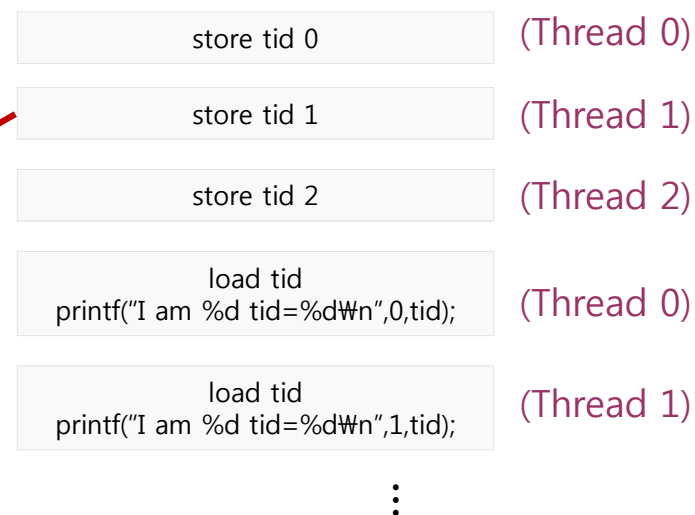
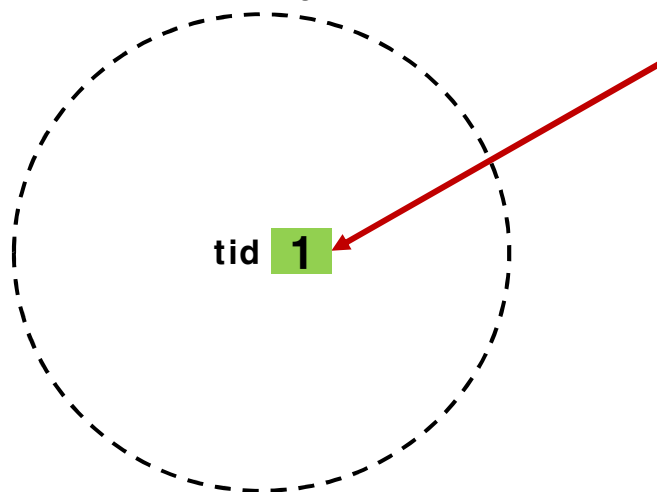
Data Scope Attribute (1/7)

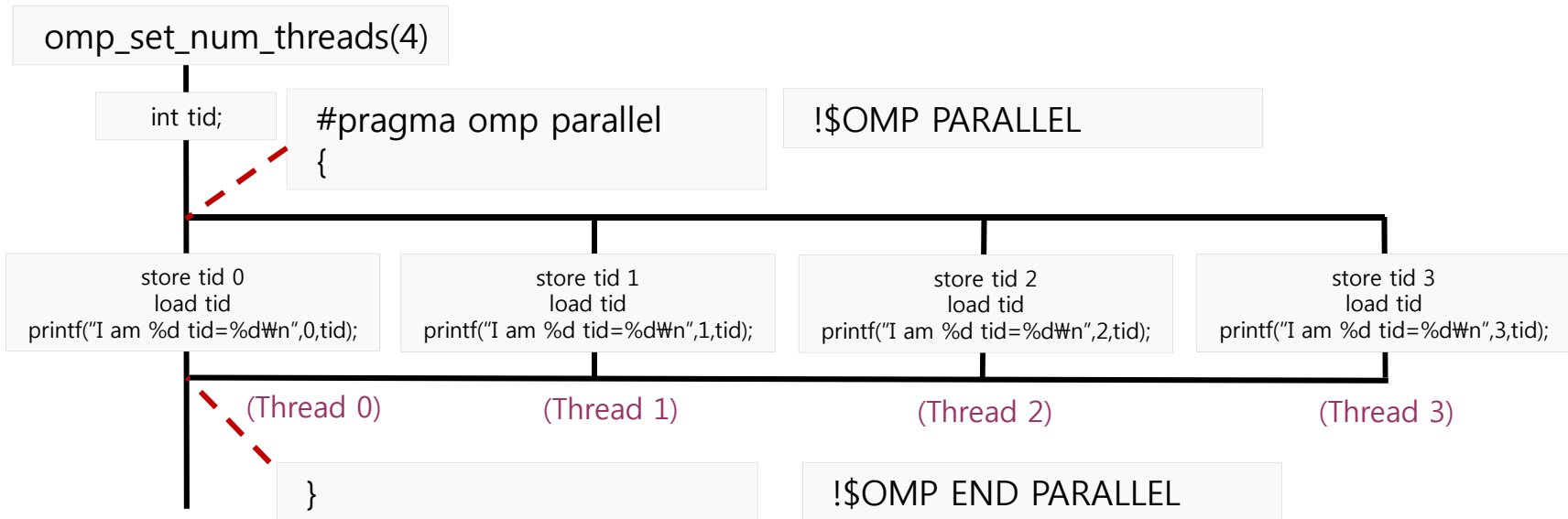


Scenario

\$./a.out

Memory





```
$ ./a.out
```

store tid 1	(Thread 1)
-------------	------------

store tid 2	(Thread 2)
-------------	------------

```
load tid
printf("I am %d tid=%d\n",0,tid);
```

(Thread 0)

```
load tid
printf("I am %d tid=%d\n",1,tid);
```

(Thread 1)

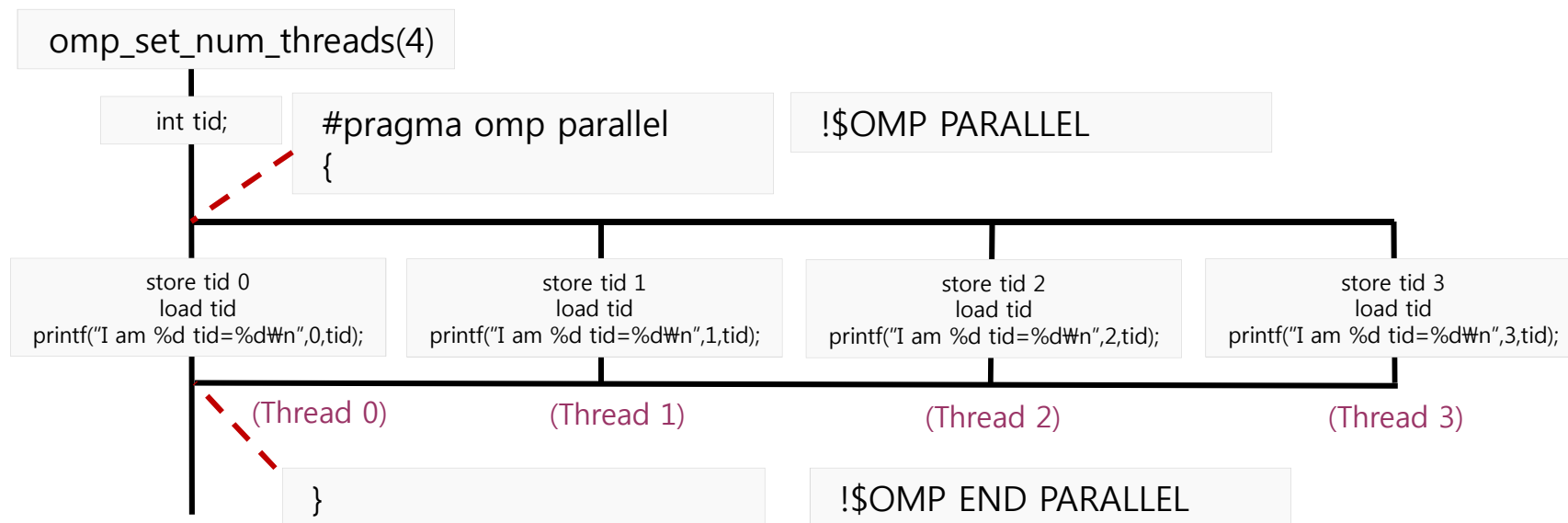
•

•

•



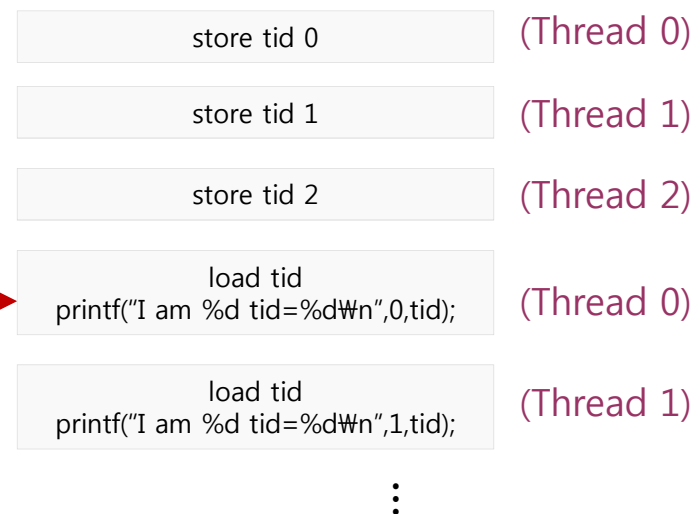
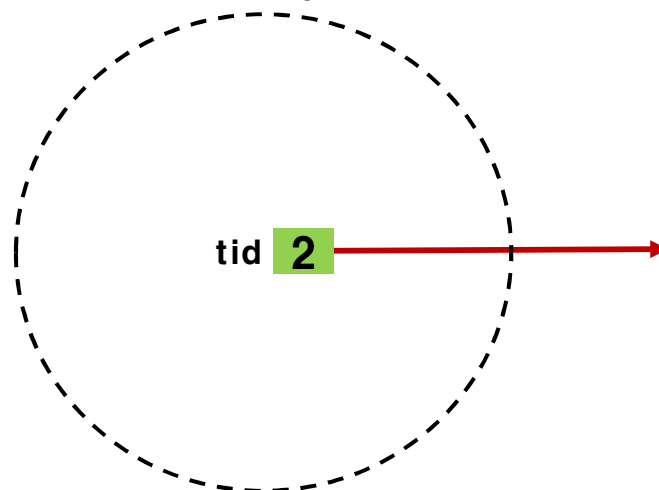
Data Scope Attribute (1/7)



Scenario

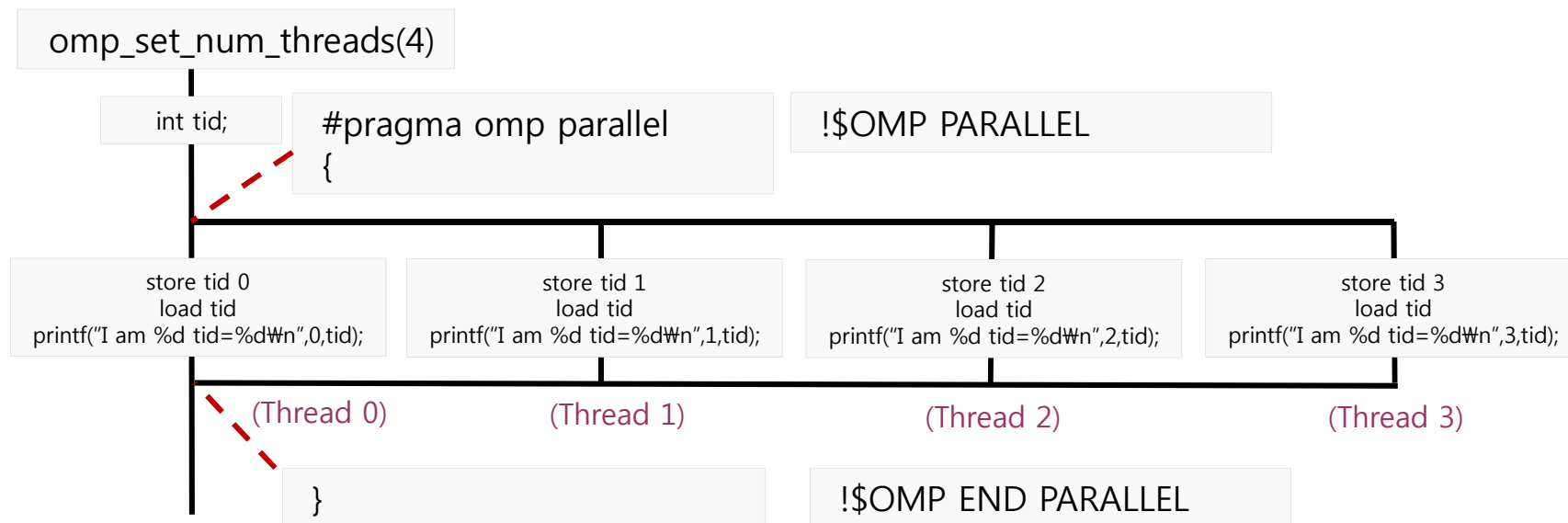
```
$ ./a.out
I am 0 tid = 2
```

Memory





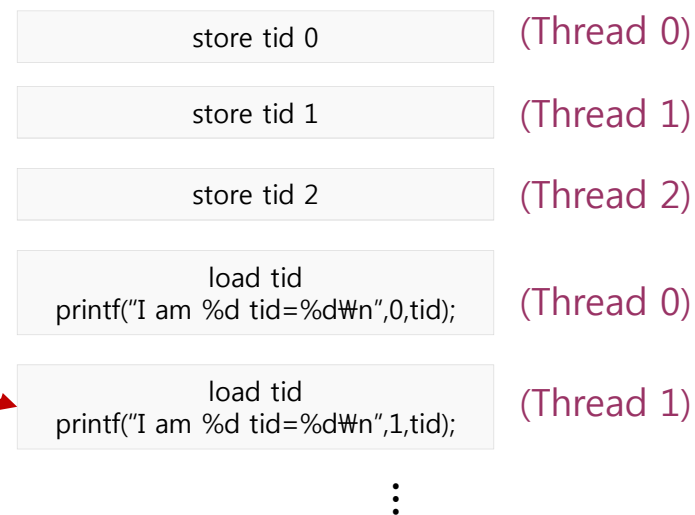
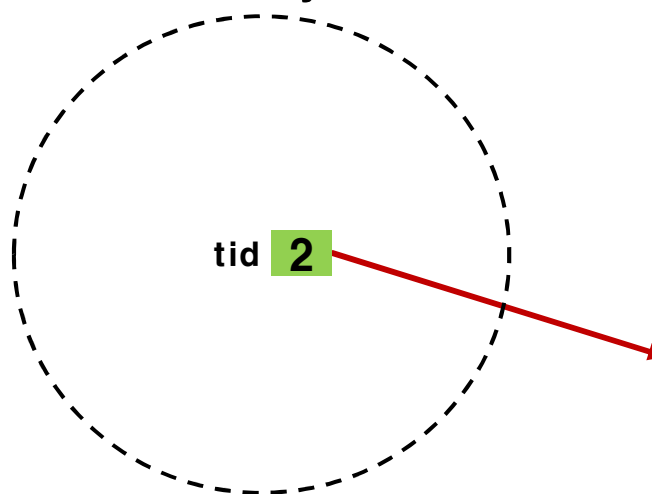
Data Scope Attribute (1/7)



Scenario

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

Memory

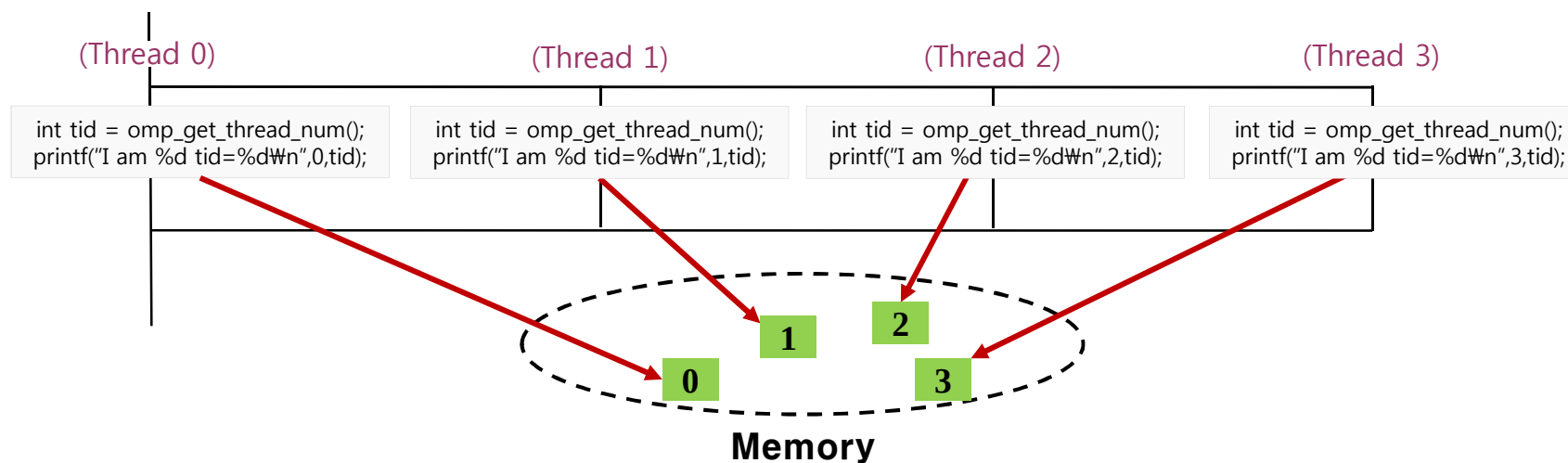




Data Scope Attribute (2/7)



Fortran	C
<pre>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</pre>	<pre>#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); } return 0; }</pre>

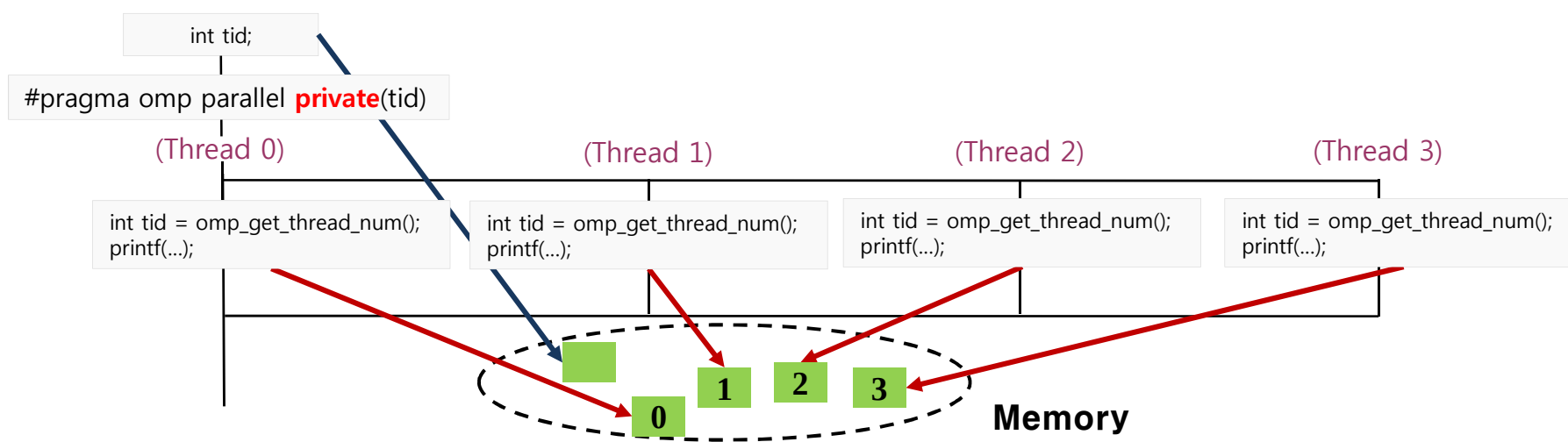




Data Scope Attribute (3/7)



Fortran	C
<pre>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</pre>	<pre>#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); } return 0; }</pre>





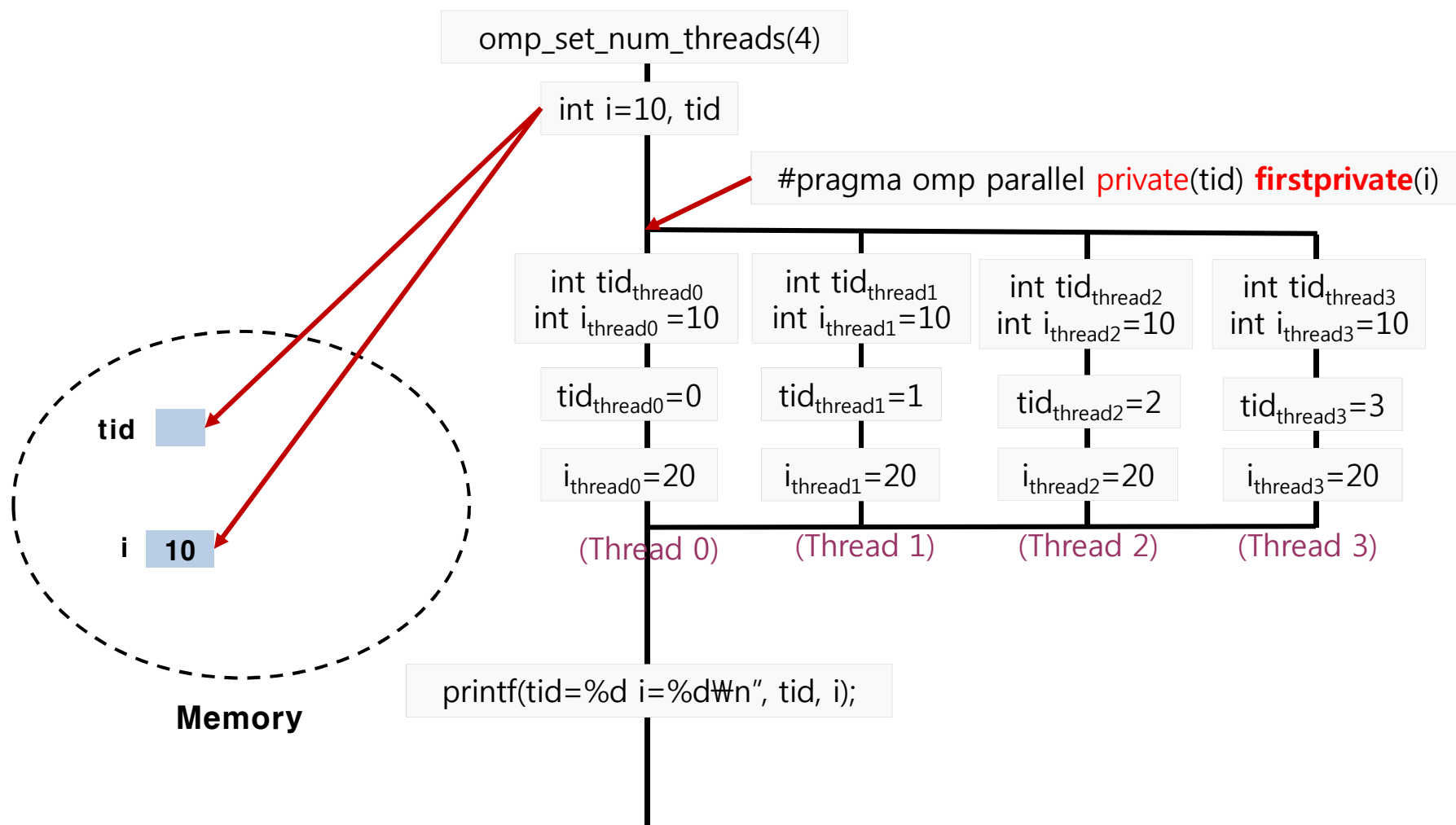
Data Scope Attribute (4/7)



Fortran	C
<pre> PROGRAM 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 </pre>	<pre> #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); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o firstprivate.x firstprivate.f90 \$./firstprivate.x </pre>	<pre> \$ gcc -fopenmp -o firstprivate.x firstprivate.c \$./firstprivate.x </pre>

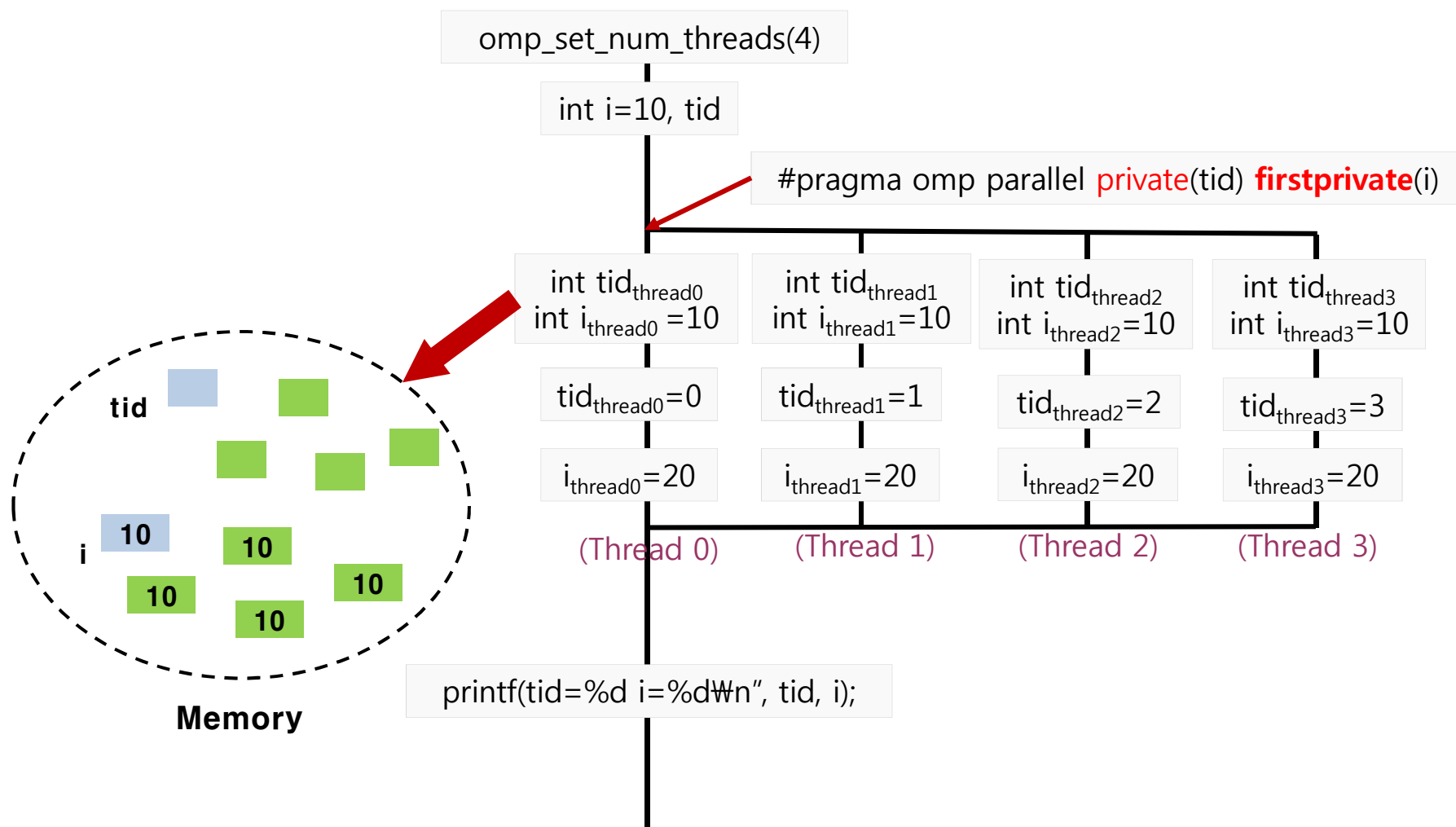


Data Scope Attribute (5/7)



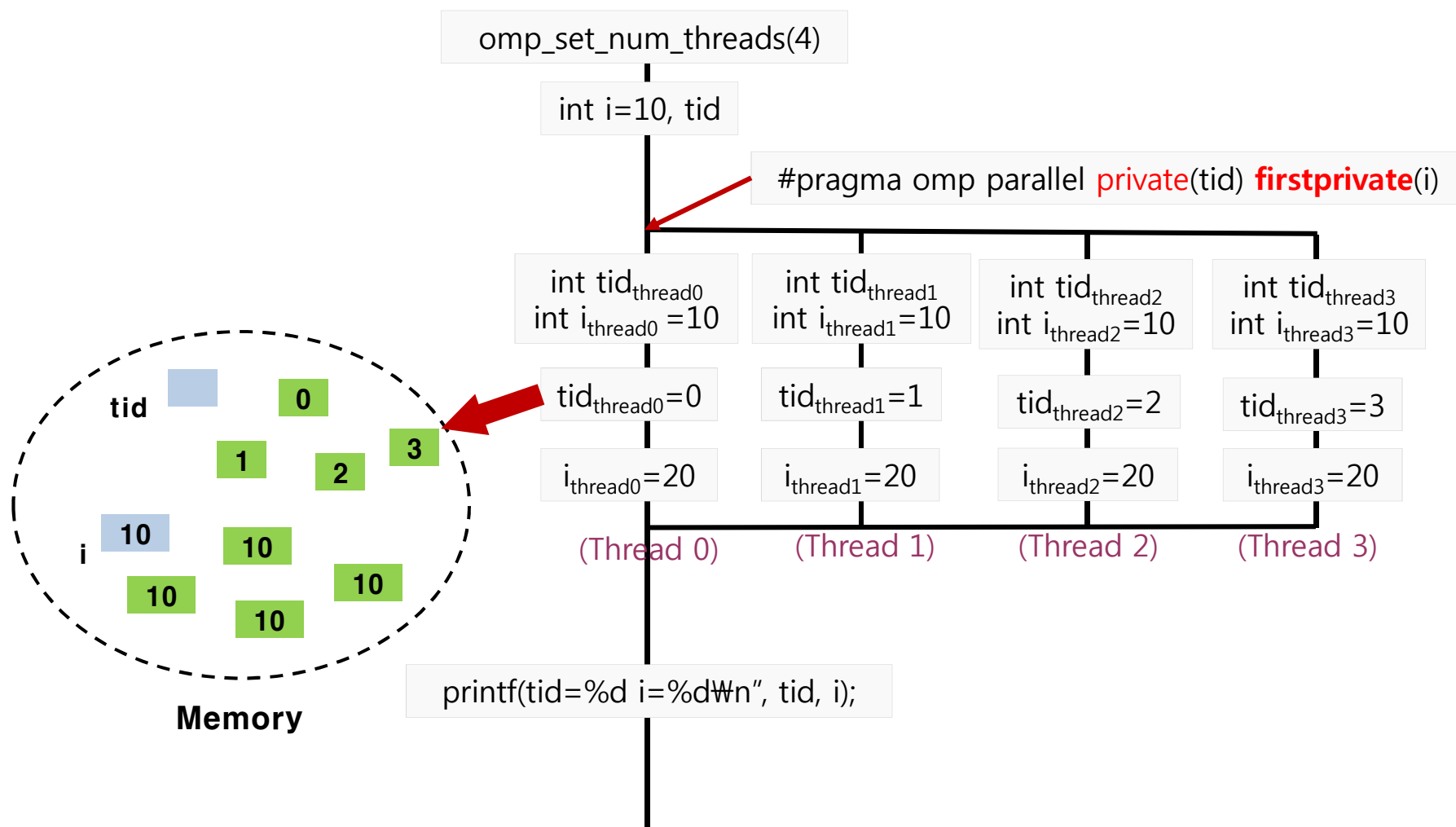


Data Scope Attribute (5/7)



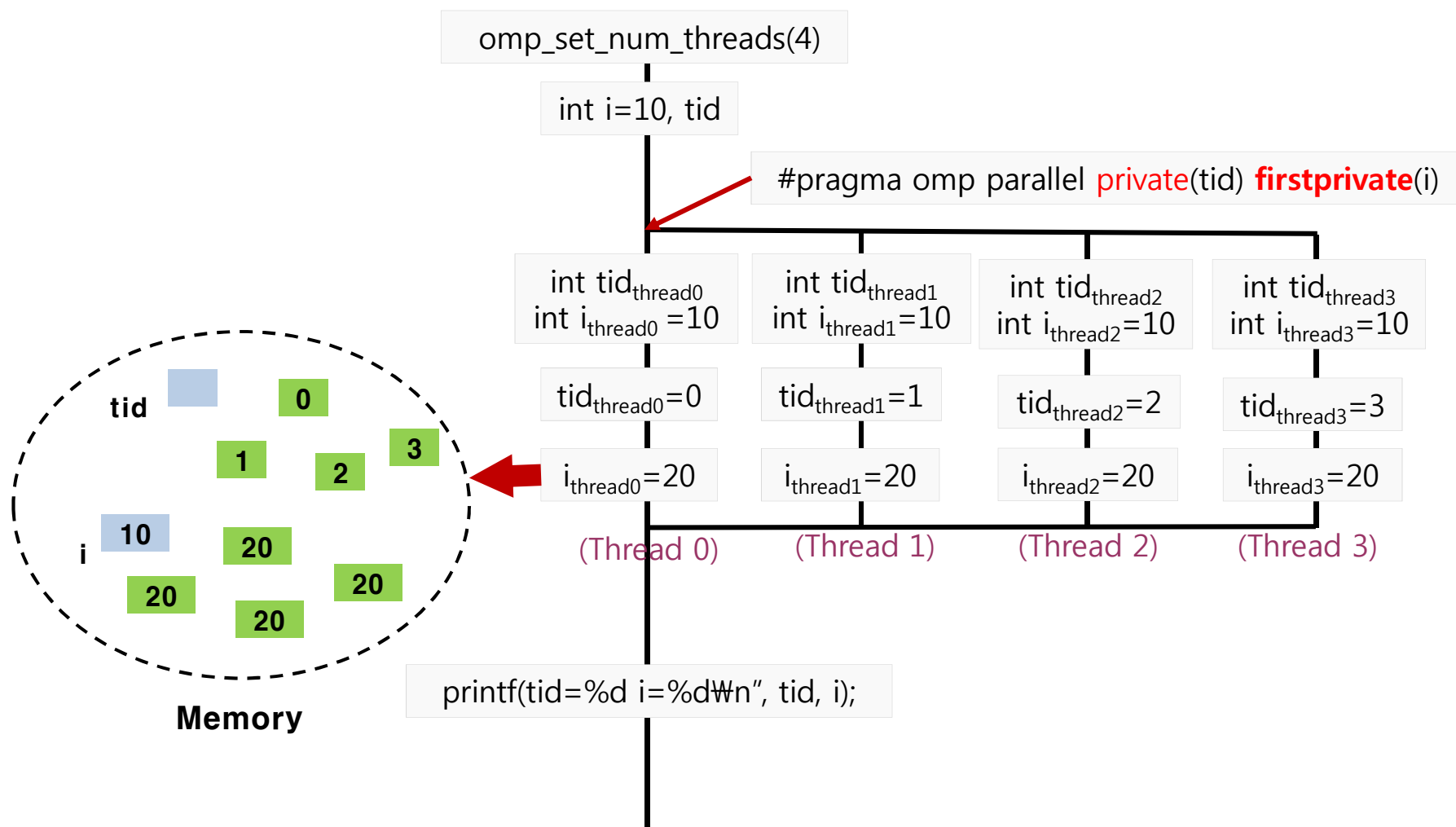


Data Scope Attribute (5/7)



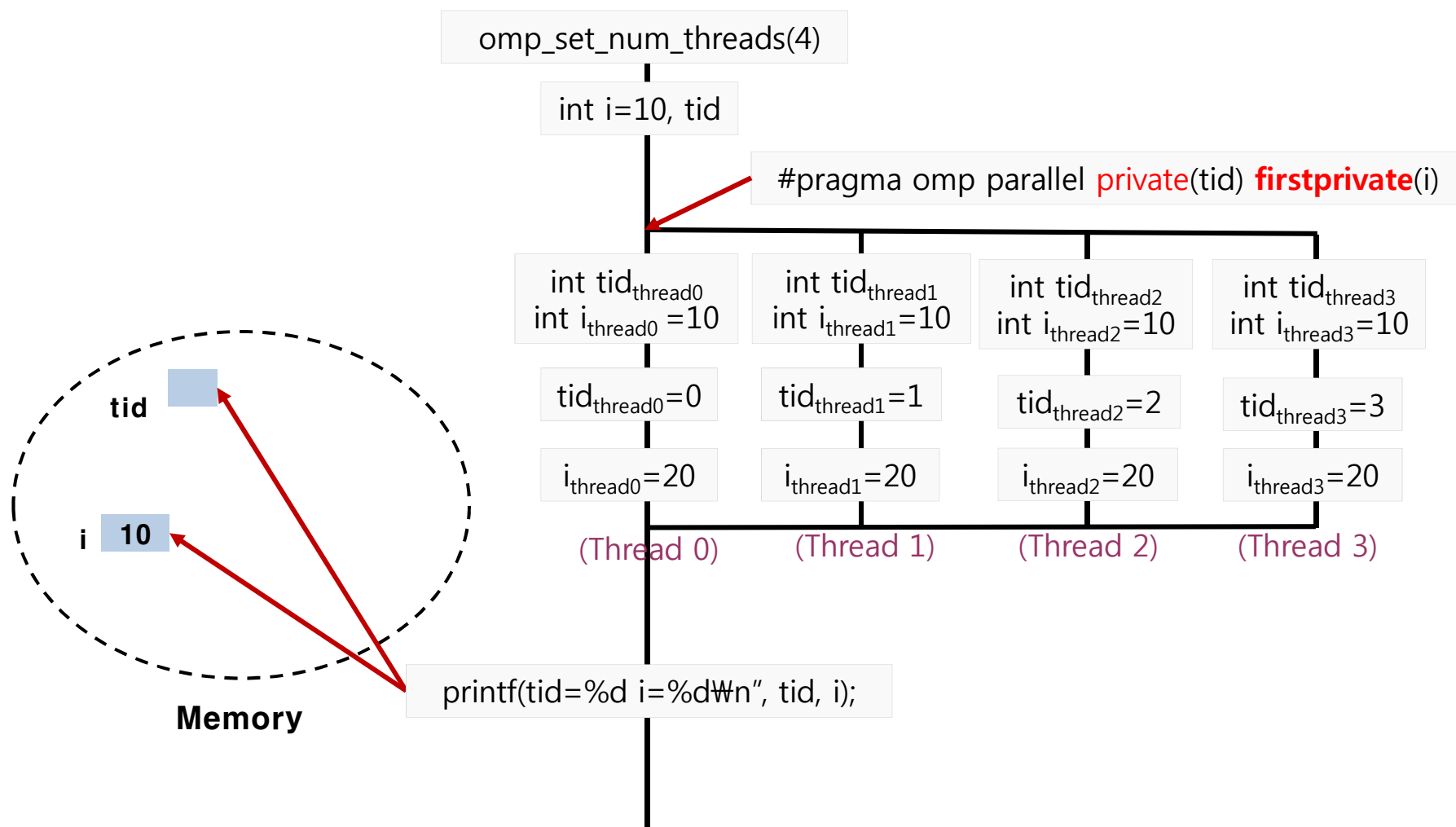


Data Scope Attribute (5/7)





Data Scope Attribute (5/7)

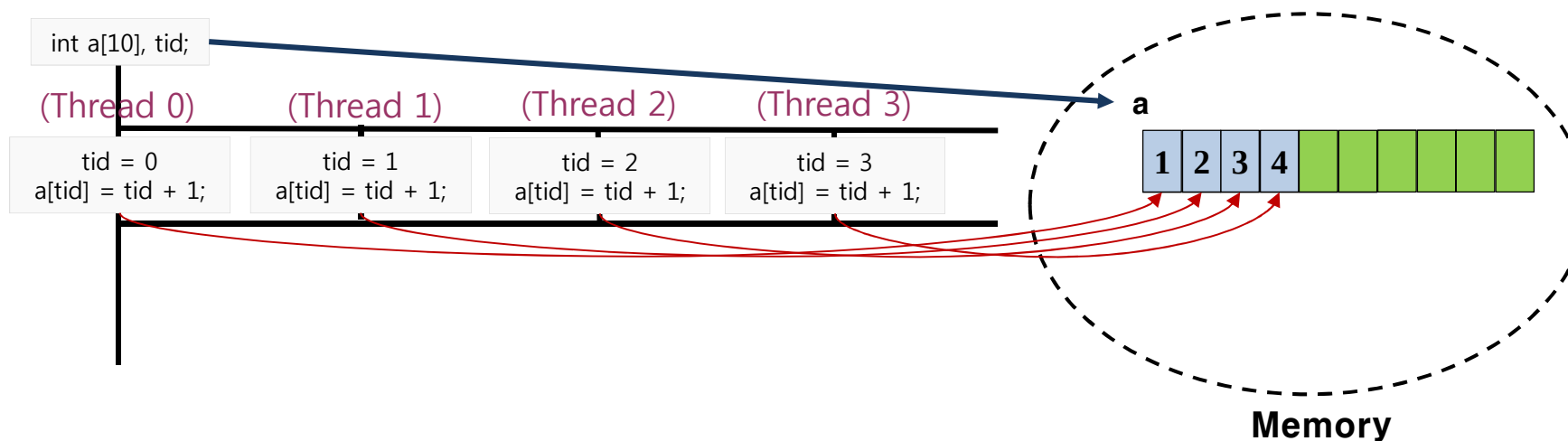




Data Scope Attribute (6/7)



Fortran	C
<pre>PROGRAM 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</pre>	<pre>#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]); return 0; }</pre>

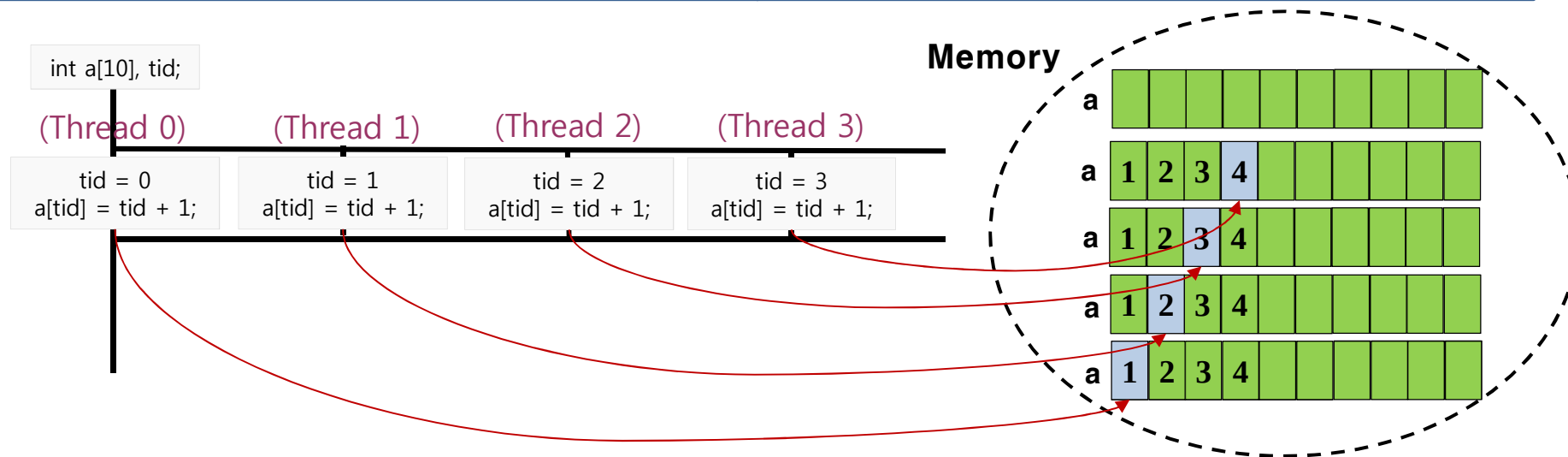




Data Scope Attribute (6/7)



Fortran	C
<pre>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</pre>	<pre>#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]); return 0; }</pre>





Thread and Process



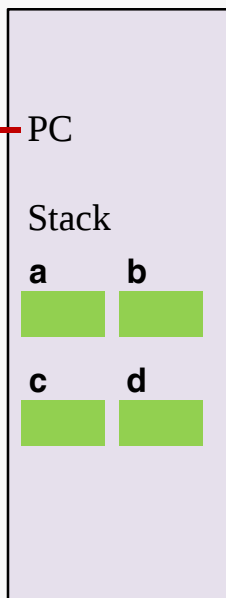
```
void foo()
{
    int a, b, c, d;
    a = 10;
    b = 5;
    c = 2;
    d = 4;

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

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

Process 1

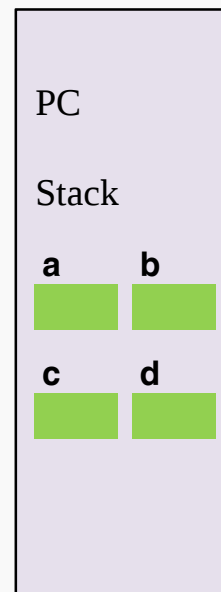
Thread 1



```
int a, b, c, d;
a=10, b=5, c=2, d=4;
#pragma omp ...
{
    c = a + 10;
    d = b + 5;
    a = a + 1;
}
```

Process 2

Thread 1



```
int a, b, c, d;
a=10, b=5, c=2, d=4;
#pragma omp ...
{
    c = a + 10;
    d = b + 5;
    a = a + 1;
}
```



Thread and Process



Process 1

```
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 -fopenmp -o foo.x foo.c
$ ./foo.x
```

Thread 1

PC

Stack

a	b
10	5
c	d
2	4

```
int a, b, c, d;
a=10, b=5, c=2, d=4;
#pragma omp ...
{
    c = a + 10;
    d = b + 5;
    a = a + 1;
}
```



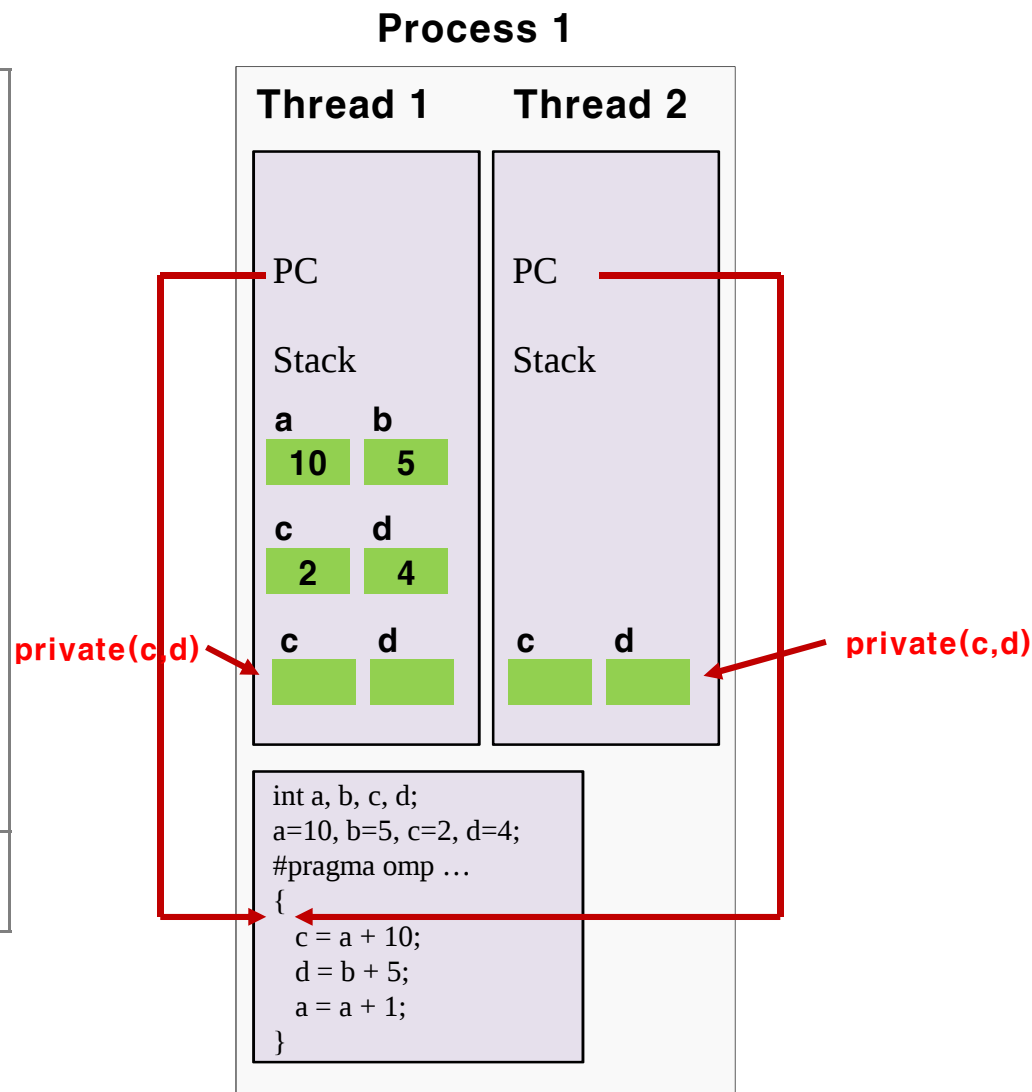
Thread and Process



```
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 -fopenmp -o foo.x foo.c
$ ./foo.x
```





Thread and Process

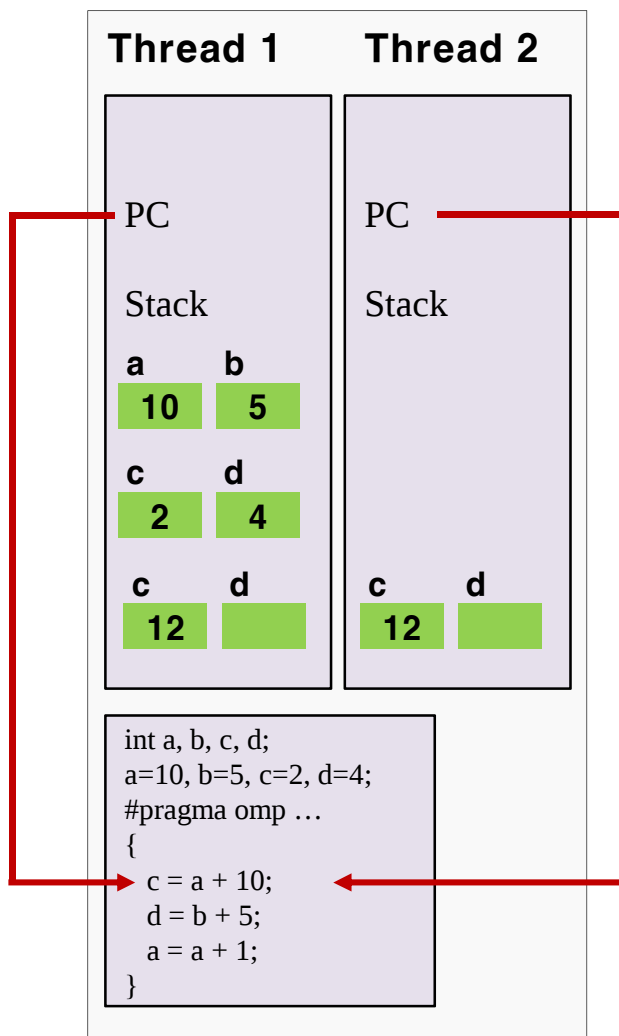


```
void foo()
{
    int a, b, c, d;
    a = 10;
    b = 5;
    c = 2;
    d = 4;

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

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

Process 1





Thread and Process

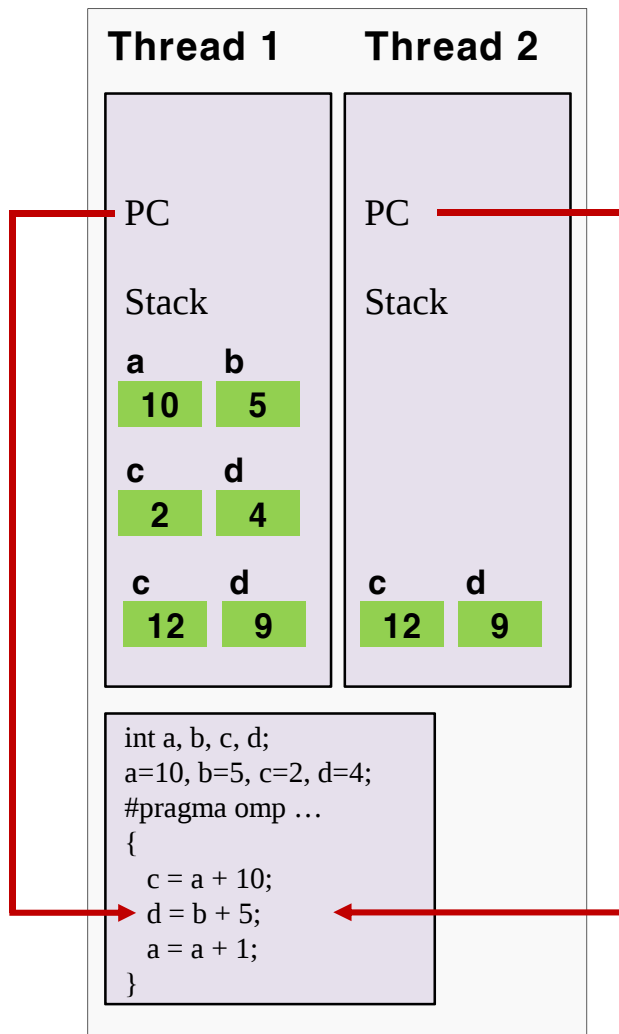


```
void foo()
{
    int a, b, c, d;
    a = 10;
    b = 5;
    c = 2;
    d = 4;

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

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

Process 1





Thread and Process

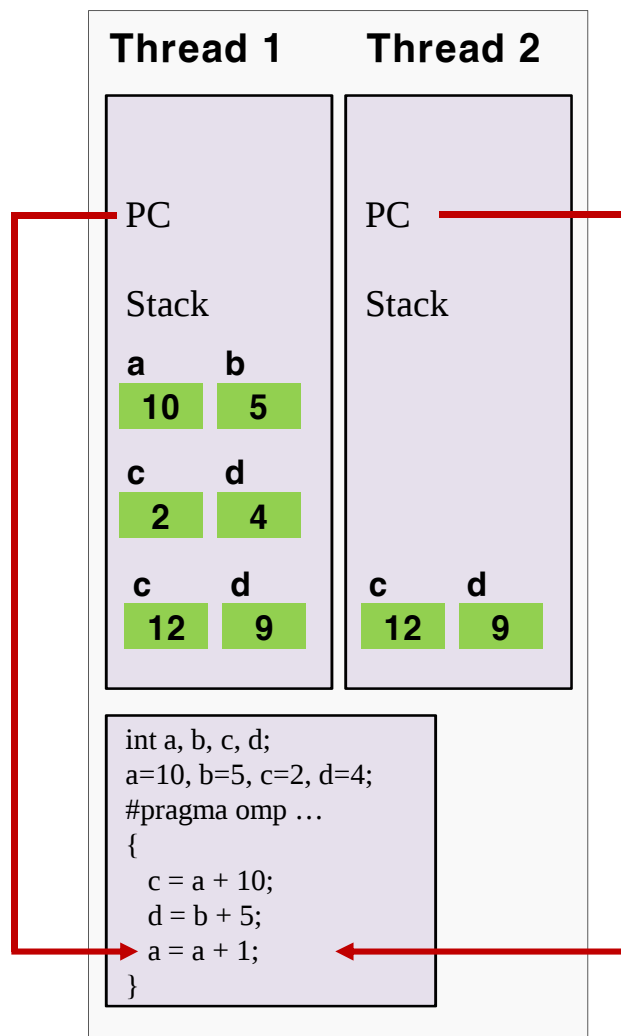


```
void foo()
{
    int a, b, c, d;
    a = 10;
    b = 5;
    c = 2;
    d = 4;

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

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

Process 1



Thread 1

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

Thread 2

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 2

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



Thread and Process



Process 1

```
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 -fopenmp -o foo.x foo.c
$ ./foo.x
```

Thread 1

PC

Stack

a	b
??	5
c	d
2	4

```
int a, b, c, d;
a=10, b=5, c=2, d=4;
#pragma omp ...
{
    c = a + 10;
    d = b + 5;
    a = a + 1;
}
```



Data Scope Attribute (Exercise)



Fortran	C												
<pre>PROGRAM data_scope_exercise INTEGER :: i=10, a(0:11), tid, ... !! create 4 threads !! assign a rray index + I to each element of array !! ex) a(0) = 10, a(1) = 11, a(2) = 12, ... , a(11) = 21 DO i=0, 11 PRINT *, 'a(', i, ') =', a(i) END DO END</pre>	<pre>#include <stdio.h> #include <omp.h> int main() { int i=10, a[12], tid; // create 4 threads // assign array index + i to each element of array // ex) a[0] = 10, a[1] = 11, a[2] = 12, ..., a[11] = 21 for(i=0; i<12; i++) printf("a[%d]=%d\n", i, a[i]); return 0; }</pre>												
a(0) a(1) a(2) a(3) a(11) <table><tr><td>10</td><td>11</td><td>12</td><td>13</td><td>.....</td><td>21</td></tr></table>	10	11	12	13		21	a[0] a[1] a[2] a[3] a[11] <table><tr><td>10</td><td>11</td><td>12</td><td>13</td><td>.....</td><td>21</td></tr></table>	10	11	12	13		21
10	11	12	13		21								
10	11	12	13		21								



Data Scope Attribute (Solution)

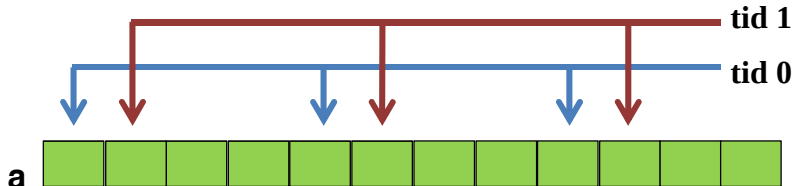
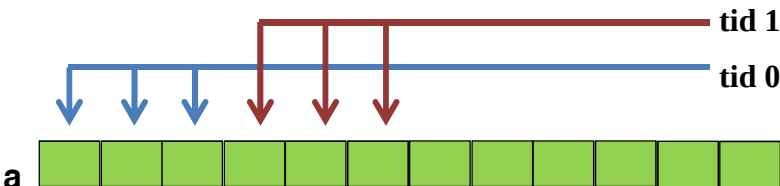


Fortran	C
<pre> PROGRAM data_scope_sol INTEGER :: i=10, a(0:11), tid, omp_get_thread_num CALL omp_set_num_threads(4) !\$OMP PARALLEL shared(a) private(tid) firstprivate(i) tid = omp_get_thread_num() i = i + tid a(tid+0) = i+0; a(tid+4) = i+4; a(tid+8) = i+8 !\$OMP END PARALLEL !!! or !\$OMP PARALLEL shared(a) private(tid) firstprivate(i) tid = omp_get_thread_num() * 3 i = i + tid a(tid+0) = i+0; a(tid+1) = i+1; a(tid+2) = i+2 !\$OMP END PARALLEL DO i=0, 11 PRiNT *, 'a(', i, ') = ', a(i) END DO END </pre>	<pre> #include <stdio.h> #include <omp.h> int main() { int i=10, a[12], tid; omp_set_num_threads(4); #pragma omp parallel shared(a) private(tid) firstprivate(i) { tid = omp_get_thread_num(); i = i + tid; a[tid+0] = i+0; a[tid+4] = i+4; a[tid+8] = i+8; } // or #pragma omp parallel shared(a) private(tid) firstprivate(i) { tid = omp_get_thread_num() * 3; i = i + tid; a[tid+0] = i+0; a[tid+1] = i+1; a[tid+2] = i+2; } for(i=0; i<12; i++) printf("a[%d]=%d\n", i, a[i]); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o data_scope_sol.x data_scope_sol.f90 \$./data_scope_solution.x </pre>	<pre> \$ gcc -fopenmp -o data_scope_sol.x data_scope_sol.c \$./data_scope_solution.x </pre>



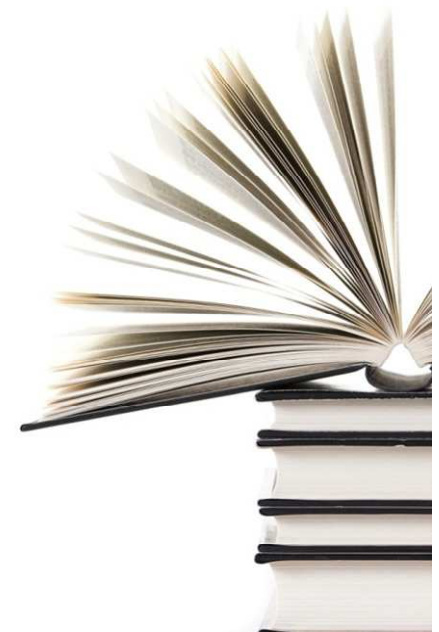
Data Scope Attribute (Solution)



Fortran	C
<pre>[method 1] tid = omp_get_thread_num() i = i + tid a(tid+0) = i+0; a(tid+4) = i+4; a(tid+8) = i+8 [method 2] tid = omp_get_thread_num() * 3 i = i + tid a(tid+0) = i+0; a(tid+1) = i+1; a(tid+2) = i+2</pre>	<pre>[method 1] tid = omp_get_thread_num(); i = i + tid; a[tid+0] = i+0; a[tid+4] = i+4; a[tid+8] = i+8; [method 2] tid = omp_get_thread_num() * 3; i = i + tid; a[tid+0] = i+0; a[tid+1] = i+1; a[tid+2] = i+2;</pre>
[method 1] 	[method 2] 

OpenMP Basics II

- 1. Parallel Loops**
- 2. Synchronization**





Parallel Loops



Fortran	C
<pre>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</pre>	<pre>#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); return 0; }</pre>
<pre>\$ gfortran -fopenmp -o serial_loop.x serial_loop.f90 \$./serial_loop.x</pre>	<pre>\$ gcc -fopenmp -o serial_loop.x serial_loop.c \$./serial_loop.x</pre>



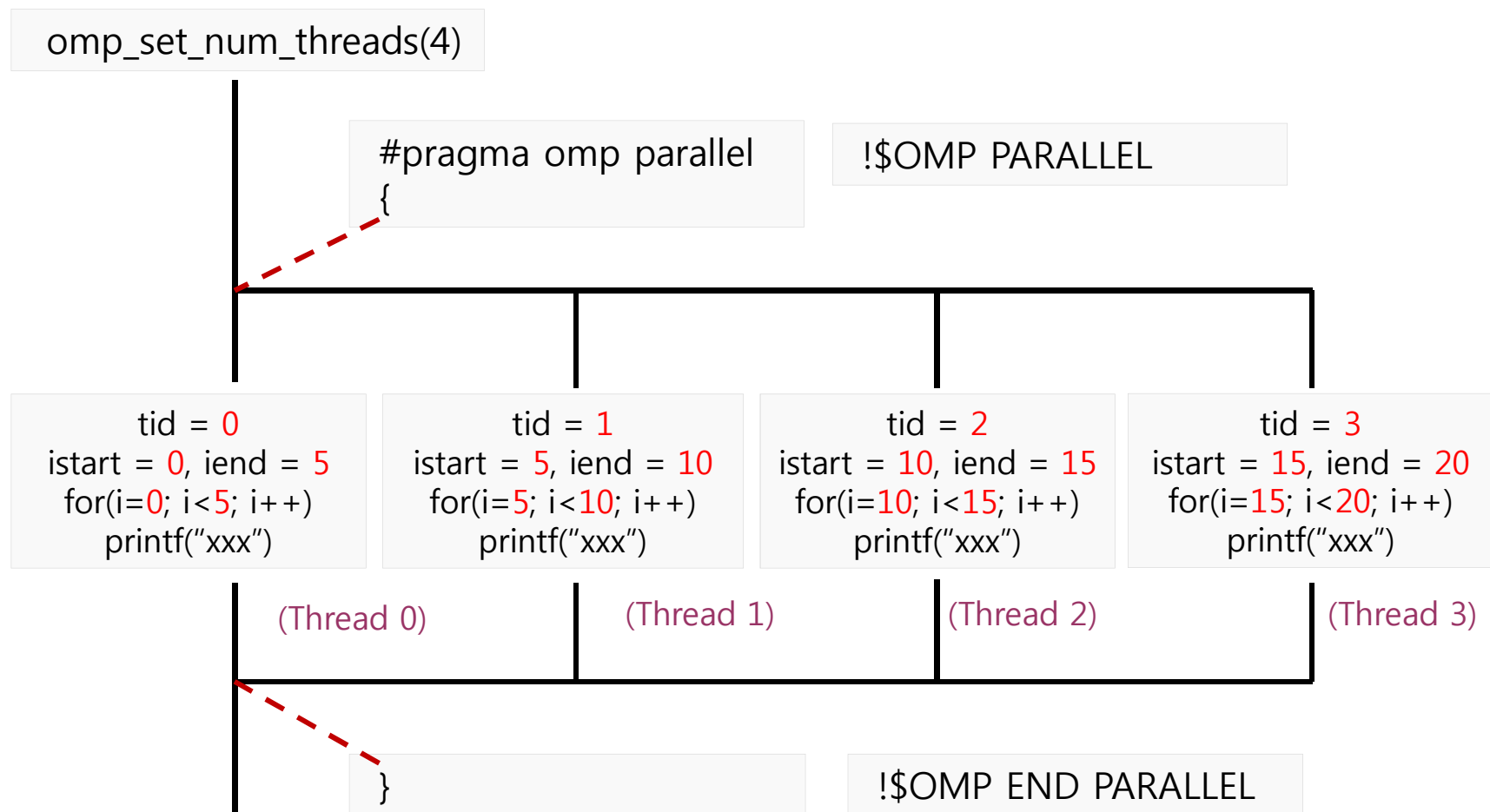
Parallel Loops



Fortran		C	
<pre> 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 </pre>		<pre> #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); } return 0; } </pre>	
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



Parallel Loops





Parallel Loops



Fortran	C
<pre> PROGRAM parallel_loop1 INTEGER, PARAMETER :: N=20 INTEGER::tid, i, istart, iend, omp_get_thread_num CALL omp_set_num_threads(4) !\$OMP PARALLEL private(i, 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 </pre>	<pre> #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(i, 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); } return 0; } </pre>
<pre> \$ gfortran -fopenmp -o parallel_loop1.x parallel_loop1.f90 \$./parallel_loop1.x </pre>	<pre> \$ gcc -fopenmp -o parallel_loop1.x parallel_loop1.c \$./parallel_loop1.x </pre>



Parallel Loops



Fortran	C
<pre> PROGRAM parallel_loop2 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(i, 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 </pre>	<pre> #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(i,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); } return 0; } </pre>
<pre> \$ gfortran -fopenmp -o parallel_loop2.x parallel_loop2.f90 \$./parallel_loop2.x </pre>	<pre> \$ gcc -fopenmp -o parallel_loop2.x parallel_loop2.f90 \$./parallel_loop2.x </pre>



Parallel Loops



Fortran	C
<pre>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(i,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</pre>	<pre>#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(i,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); } return 0; }</pre>

Break!!





Parallel Loops



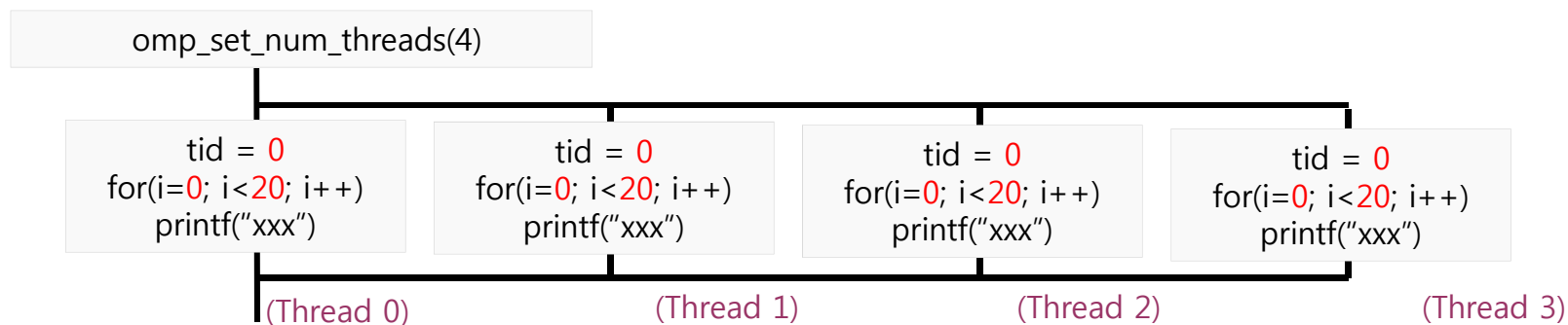
Fortran	C
<pre> PROGRAM parallel_for INTEGER, PARAMETER :: N=20 INTEGER :: tid, i, omp_get_thread_num CALL omp_set_num_threads(4) !\$OMP PARALLEL private(tid) !!! i : shared?? tid = omp_get_thread_num() !\$OMP DO DO i=0, N-1 PRINT *, 'Hello World', tid, i END DO !\$OMP END DO !!![optional] !\$OMP END PARALLEL END </pre>	<pre> #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); } return 0; } </pre>
<pre> \$ gfortran -fopenmp -o parallel_for.x parallel_for.f90 \$./parallel_for.x </pre>	<pre> \$ gcc -fopenmp -o parallel_for.x parallel_for.c \$./parallel_for.x </pre>



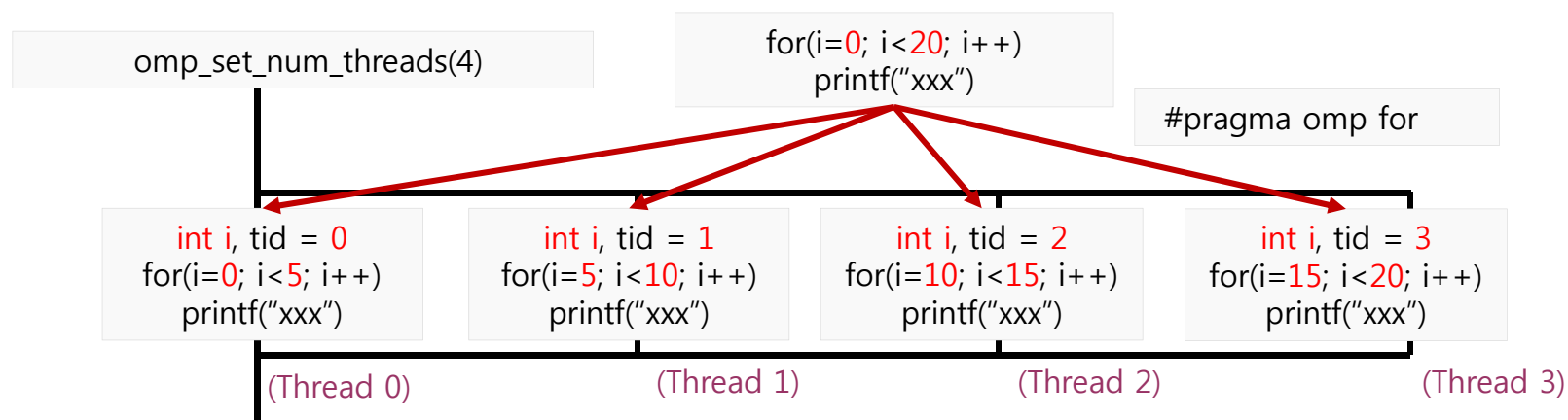
Parallel Loops



➤ Without #pragma omp for



➤ With #pragma omp for





Parallel Loops



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

➤ Work Sharing

- Distribution of work across threads
- **Do/for**, Sections/section, Single, Task construct, etc.
- cf) SPMD (Single Program Multiple Data), WORKSHARE clause in Fortran



Parallel Loops



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

- OpenMP shortcut
 - Combined parallel/worksharing construct



Parallel Loops - Inner(dot) Product (Serial)



Fortran	C
<pre> 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 </pre>	<pre> #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; } for (i=0; i<N; i++) sum += a[i] * b[i]; printf("sum = %d\n", sum); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o inner_product.x inner_product.f90 \$ inner_product.x </pre>	<pre> \$ gcc -fopenmp -o inner_product.x inner_product.f90 \$ inner_product.x </pre>



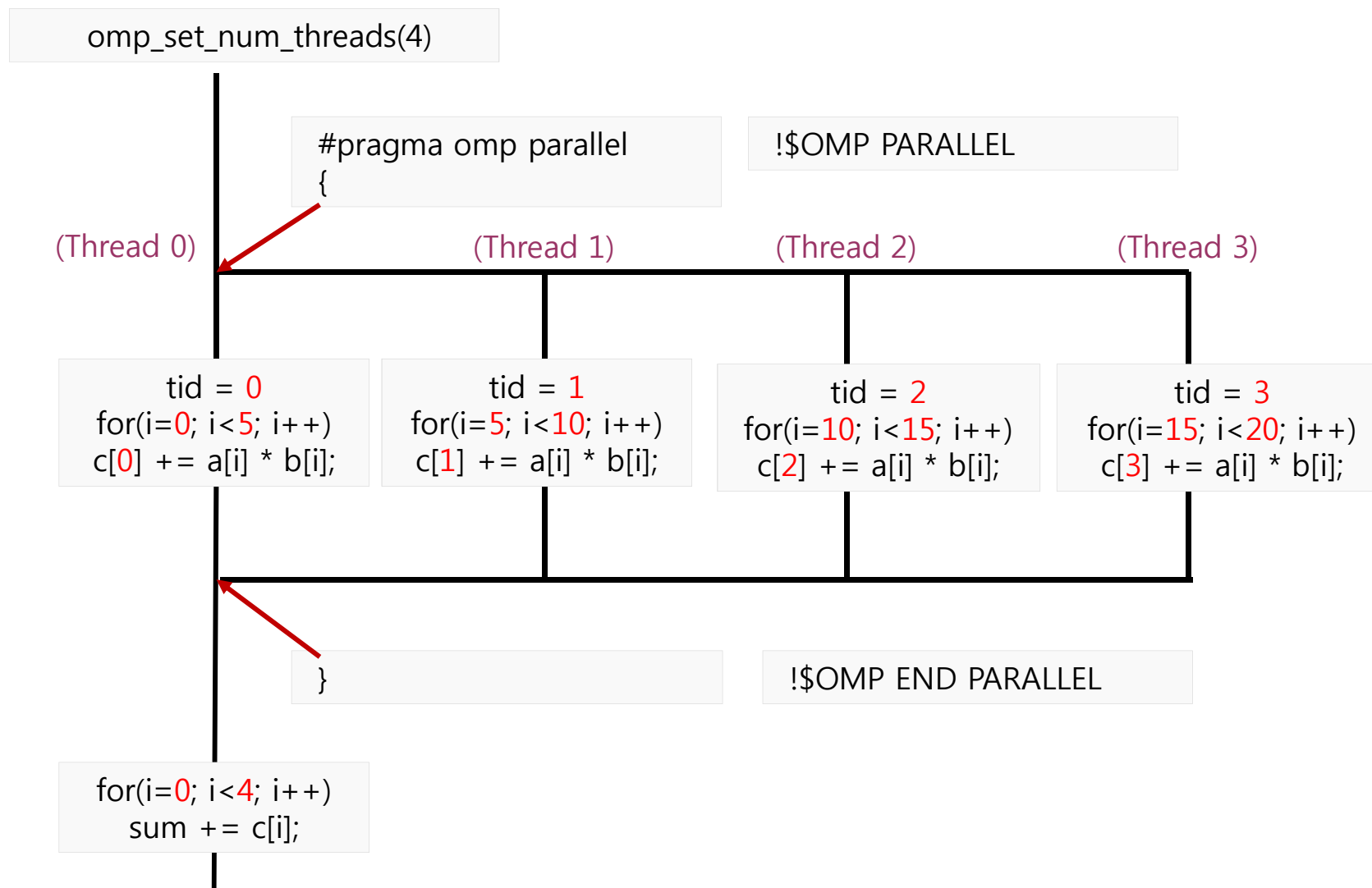
Parallel Loops - Inner(dot) Product (Parallel)



Fortran	C
<pre>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</pre>	<pre>#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); } return 0; }</pre>



Parallel Loops





Parallel Loops - BLAS1 Inner(dot) Product



Fortran	C
<pre> PROGRAM inner_product_omp 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 </pre>	<pre> #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); return 0; } </pre>
<pre> \$ gcc -fopenmp -o inner_product.x inner_product.f90 \$./inner_product_omp.x </pre>	<pre> \$ gcc -fopenmp -o inner_product_omp.x inner_product_omp.c \$./inner_product_omp.x </pre>



Synchronization - BLAS1 Inner(dot) Product



Fortran	C
<pre>PROGRAM inner_product_omp2 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 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</pre>	<pre>#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); return 0; }</pre>



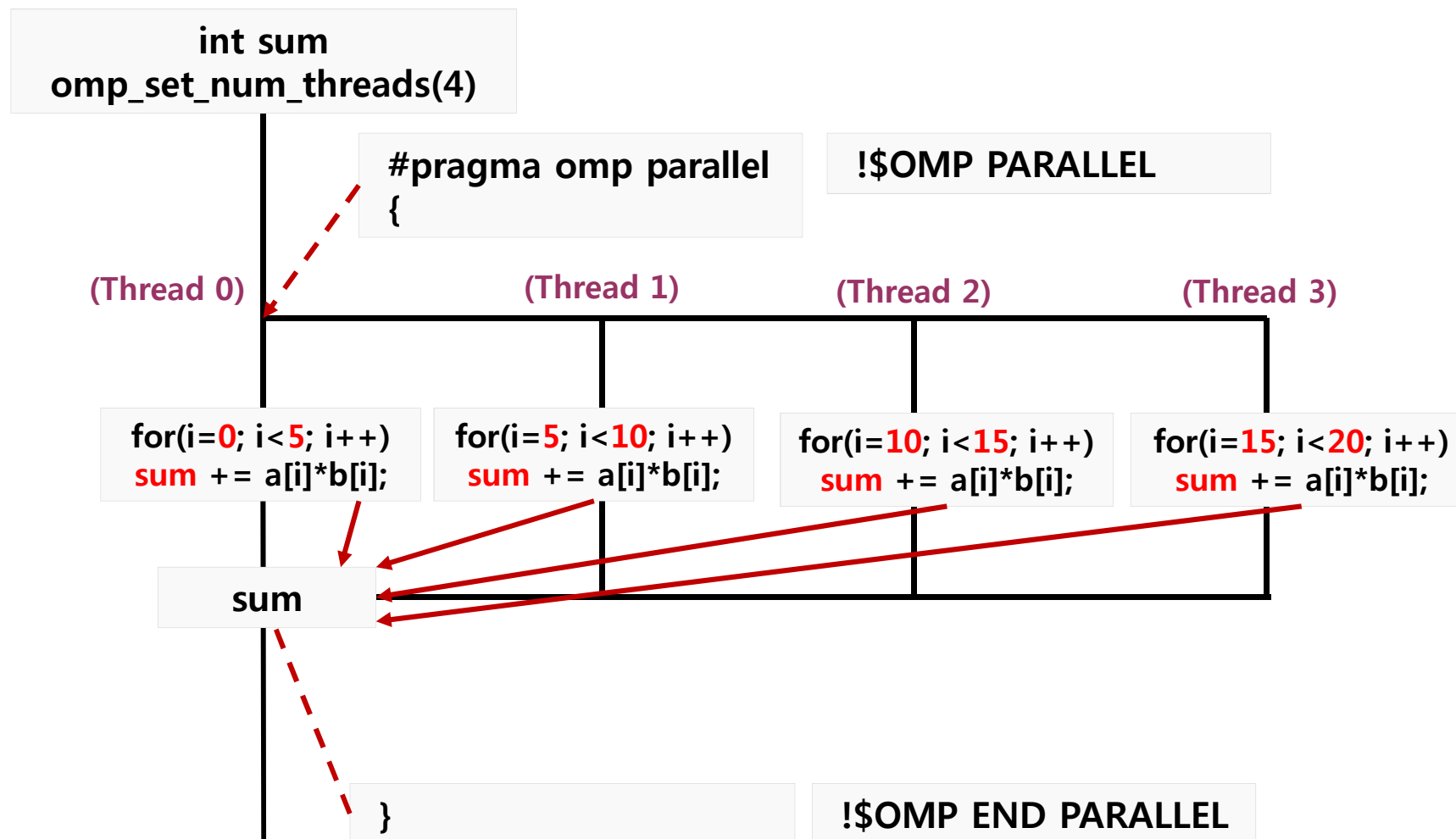
Synchronization - BLAS1 Inner(dot) Product



Fortran	C
<pre> PROGRAM wrong_inner_prod 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 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 </pre>	<pre> #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); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o wrong_inner_prod.x wrong_inner_prod.f90 \$./ wrong_inner_prod.x (Different results value when repeated run.) </pre>	<pre> \$ gcc -fopenmp -o wrong_inner_prod.x wrong_inner_prod.c \$./wrong_inner_prod.x (Different results value when repeated run.) </pre>



Synchronization





Synchronization - BLAS1 Inner(dot) Product



Fortran	C
<pre> PROGRAM inner_prod_critical IMPLICIT NONE 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 END DO CALL omp_set_num_threads(4) !\$OMP PARALLEL !\$OMP DO DO i=0, N-1 !\$OMP CRITICAL sum = sum + A(i) * B(i) !\$OMP END CRITICAL END DO !\$OMP END PARALLEL PRINT *, "sum =", sum END </pre>	<pre> #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); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o inner_prod_critical.x inner_prod_critical.f90 \$./inner_prod_critical.x </pre>	<pre> \$ gcc -fopenmp -o inner_prod_critical.x inner_prod_critical.c \$./inner_prod_critical.x </pre>



Synchronization - BLAS1 Inner(dot) Product



Fortran	C
<pre> PROGRAM inner_prod_atomic IMPLICIT NONE 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 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 </pre>	<pre> #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); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o inner_prod_atomic.x inner_prod_atomic.f90 \$./inner_prod_atomic.x </pre>	<pre> \$ gcc -fopenmp -o inner_prod_atomic.x inner_prod_atomic.c \$./inner_prod_atomic.x </pre>



Synchronization



Fortran		C	
<pre>!\$OMP PARALLEL DO i=0, 99 !\$OMP CRITICAL(n1) CALL sub(a,b) !\$OMP END CRITICAL END DO !\$OMP END PARALLEL</pre>	<pre>!\$OMP PARALLEL DO i=0, 99 !\$OMP ATOMIC a = a + b END DO !\$OMP END PARALLEL</pre>	<pre>#pragma omp parallel { for (i=0; i<100; i++) { !pragma omp critical(n1) func1(a,b); } }</pre>	<pre>#pragma omp parallel { for (i=0; i<100; i++) { !pragma omp atomic a += b; } }</pre>

- Synchronization
 - Imposing order constraints and protecting access to shared data
- High level synchronization
 - **critical, atomic, barrier**, ordered
- Low level synchronization
 - flush, locks
- critical vs atomic
 - critical : Only one thread at a time can enter a critical region, distinguished by name
 - atomic : Only applies to the update of a memory location
 - Like mini-critical section



BARRIER



Fortran	C
<pre>PROGRAM barrier INTEGER omp_get_thread_num CALL omp_set_num_threads(4) !\$OMP PARALLEL PRINT *, 'A tid = ', omp_get_thread_num() !\$OMP BARRIER PRINT *, 'B tid = ', omp_get_thread_num() !\$OMP END PARALLEL END</pre>	<pre>#include <stdio.h> #include <omp.h> int main() { omp_set_num_threads(4); #pragma omp parallel { printf ("A tid = %d \n", omp_get_thread_num()); #pragma omp barrier printf ("B tid = %d \n", omp_get_thread_num()); } return 0; }</pre>

➤ BARRIER

- The BARRIER directive **synchronizes all threads in the team**
- When a BARRIER directive is reached, a thread will wait at that point until all other threads have reached that barrier. All threads then resume executing in parallel the code that follows the barrier
- must be reached by all threads in team before any thread can proceed



Synchronization (Exercise)



Fortran	C
<pre>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 sum = sum + local_sum END DO !! join threads print *, 'sum =', sum END</pre>	<pre>#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); return 0; }</pre>



Synchronization (Solution)



Fortran	C
<pre> 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 </pre>	<pre> #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; // compile error sum += local_sum; } printf("sum = %d\n", sum); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o sync_solution.x sync_solution.f90 \$./sync_solution.x </pre>	<pre> \$ gcc -fopenmp -o sync_solution.x sync_solution.c \$./sync_solution.x </pre>



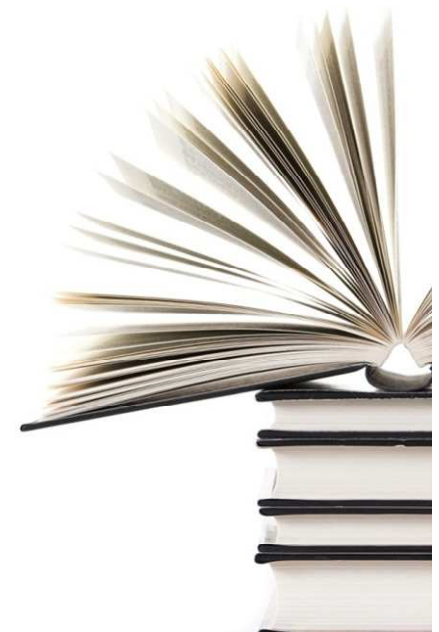
Synchronization (Solution) - Using Environmental Variable



Fortran	C
<pre>PROGRAM sync_solution IMPLICIT NONE INTEGER, parameter :: N=100 INTEGER:: i, sum=0, local_sum !\$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</pre>	<pre>#include <stdio.h> #define N 100 int main() { int i, sum = 0, local_sum; #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 += local_sum; } printf("sum = %d\n", sum); return 0; }</pre>
<pre>\$ gfortran -fopenmp -o sync_solution2.x sync_solution2.f90 \$ export OMP_NUM_THREADS=7 \$./ sync_solution2.x</pre>	<pre>\$ gcc -fopenmp -o sync_solution2.x sync_solution2.c \$ export OMP_NUM_THREADS=7 \$./ sync_solution2.x</pre>

OpenMP Basics III

1. Reduction





Reduction - BLAS1 Inner(dot) Product



Fortran	C
<pre> PROGRAM sync_atomic IMPLICIT NONE 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 ENDDO CALL omp_set_num_threads(4) !\$OMP PARALLEL !\$OMP DO DO i=0, N-1 !\$OMP ATOMIC sum = sum + A(i) * B(i) ENDDO !\$OMP END PARALLEL PRINT *, "sum =", sum END </pre>	<pre> #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); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o sync_atomic.x sync_atomic.f90 \$./sync_atomic.x </pre>	<pre> \$ gcc -fopenmp -o sync_atomic.x sync_atomic.c \$./sync_atomic.x </pre>



Reduction - BLAS1 Inner(dot) Product



Fortran	C
<pre> PROGRAM reduction IMPLICIT NONE 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 </pre>	<pre> #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); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o reduction.x reduction.f90 \$./reduction.x </pre>	<pre> \$ gcc -fopenmp -o reduction.x reduction.c \$./reduction.x </pre>



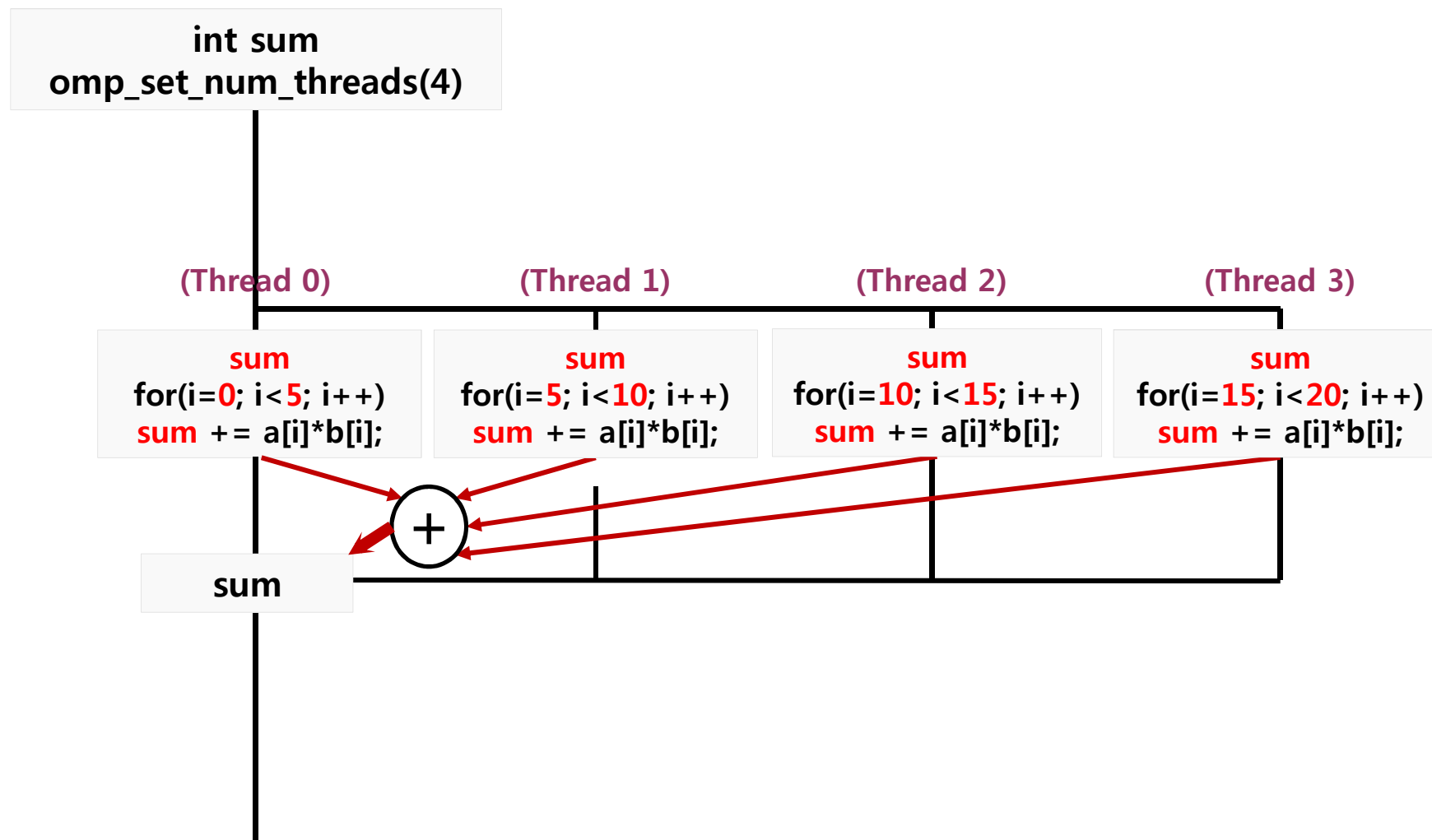
Reduction - BLAS1 Inner(dot) Product



Fortran	C
<pre> PROGRAM reduction_shortcut IMPLICIT NONE 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 </pre>	<pre> #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); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o reduction_short.x reduction_short.f90 \$./reduction_short.x </pre>	<pre> \$ gcc -fopenmp -o reduction_short.x reduction_short.c \$./reduction_short.x </pre>



Reduction





Reduction



➤ 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



➤ 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



Reduction (Exercise1) - Factorial



Fortran	C
<pre>PROGRAM reduction_exercise INTEGER, PARAMETER :: N=10 INTEGER :: i, fac=1 DO i=1, N fac = fac * i END DO PRINT *, "factorial =", fac END</pre>	<pre>#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); return 0; }</pre>
<pre>\$ gfortran -o factorial.x factorial.f90 \$./factorial.x</pre>	<pre>\$ gcc -o factorial.x factorial.c \$./factorial.x</pre>



Reduction (Solution)



Fortran	C
<pre> PROGRAM reduction_solution IMPLICIT NONE 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 </pre>	<pre> #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); return 0; } </pre>
<pre> \$ gfortran -fopenmp -o factorial_sol.x factorial_sol.f90 \$./factorial_sol.x </pre>	<pre> \$ gcc -fopenmp -o factorial_sol.x factorial_sol.c \$./factorial_sol.x </pre>



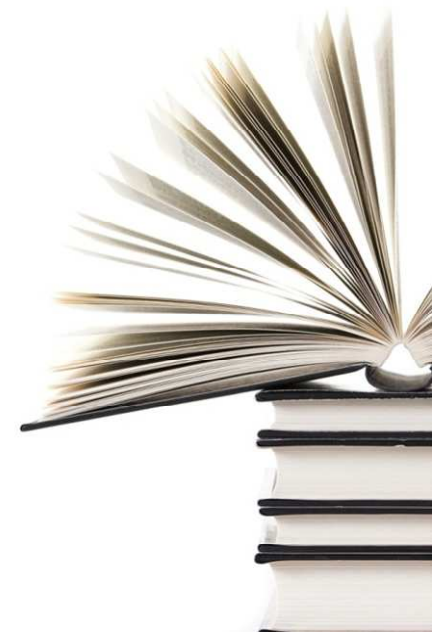
Reduction (Exercise2)



- 1. Modify inner(dot) product using reduction
- 2. Compare time among atomic, local_sum and reduction

Summary

- 1. Review**
- 2. common problems**





Review



- OpenMP Compile Option
 - GCC : -fopenmp, Intel : -openmp, PGI : -mp
- 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



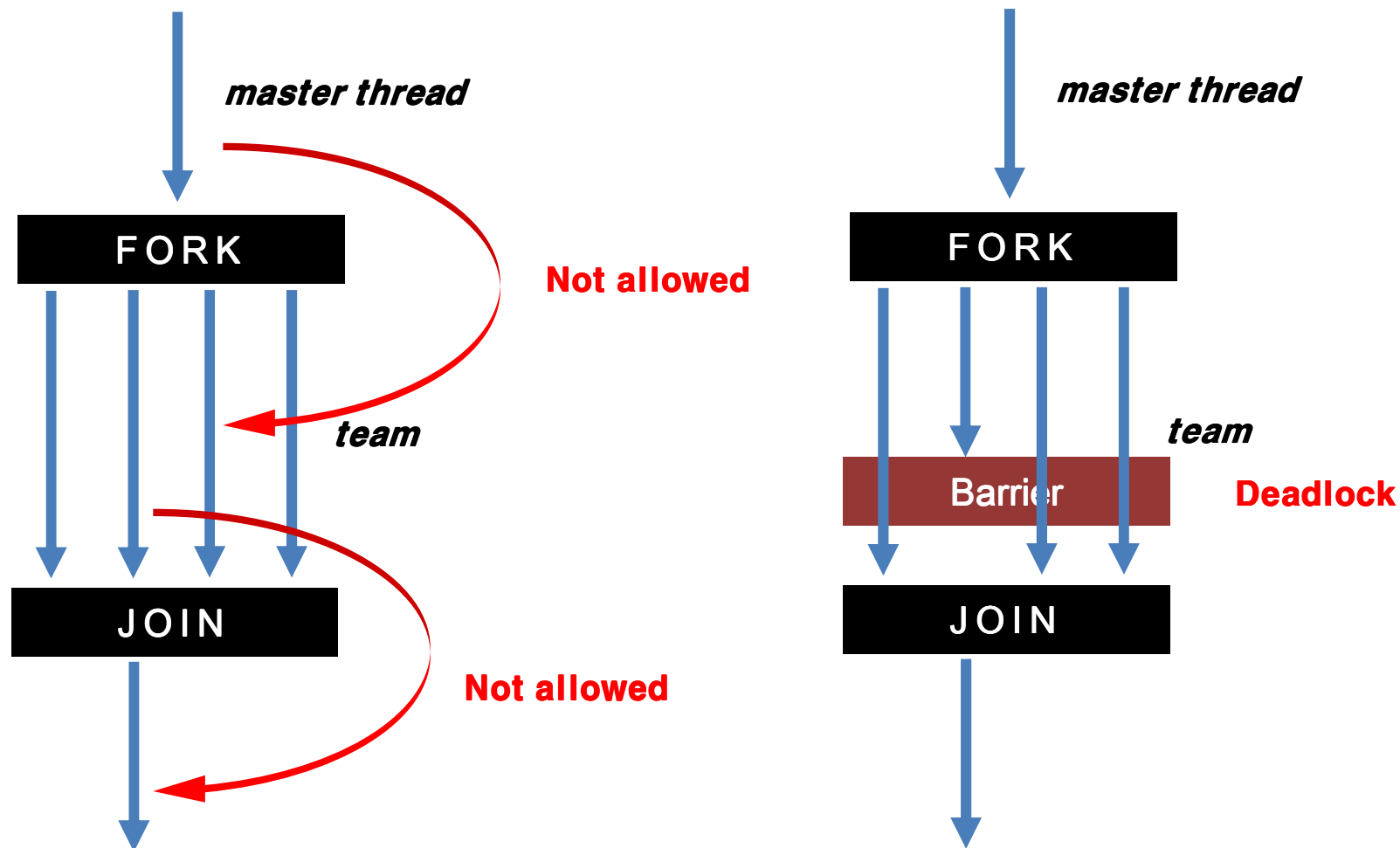
Parallel Block (1/2)



Fortran	C
<pre>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</pre>	<pre>#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 } } return 0; }</pre>



Parallel Block (2/2)



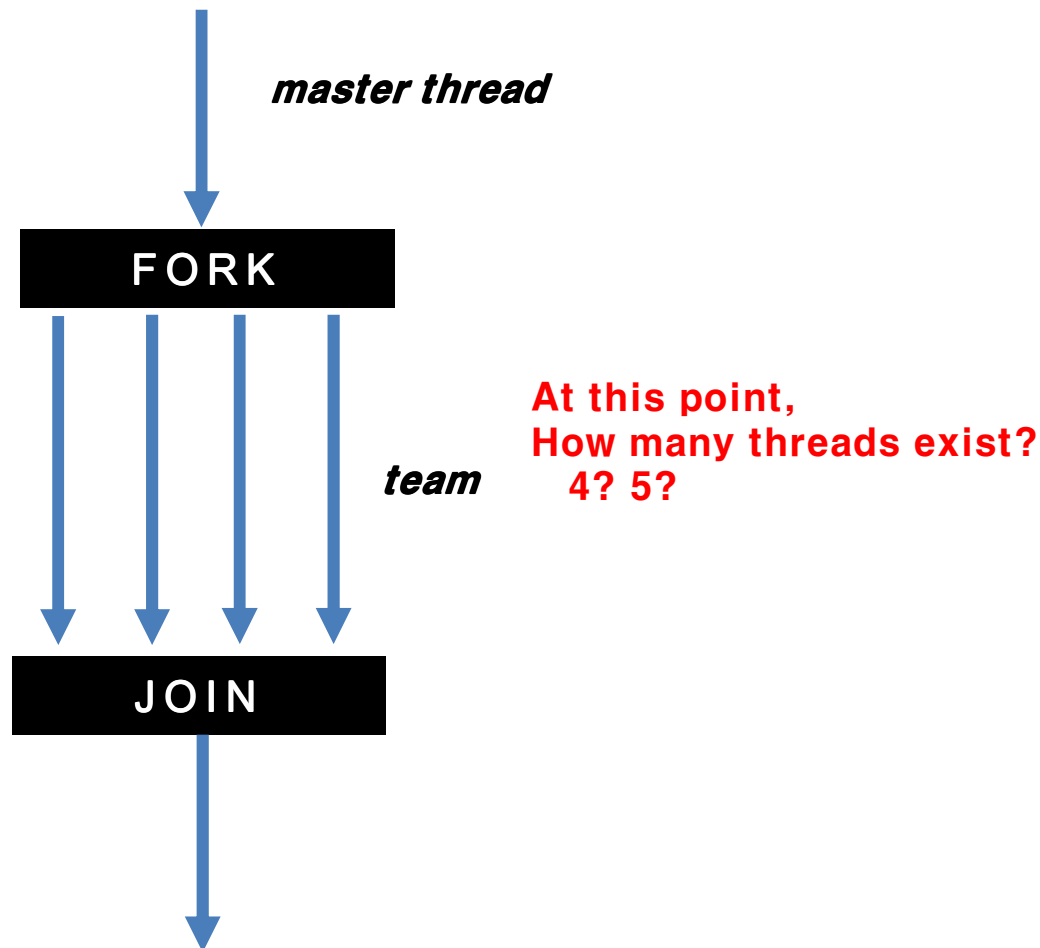
Only `exit()` is possible inside parallel region



Fork-Join Model

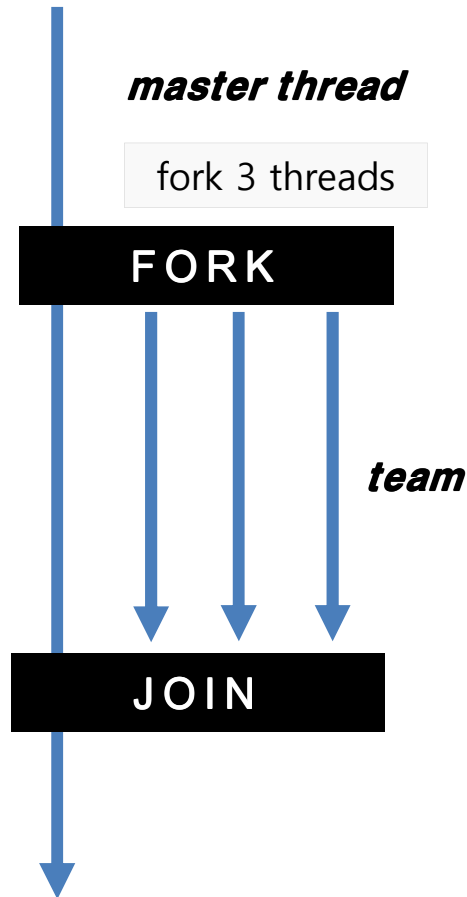


- Fork-Join Model
 - Master thread forks new threads at the beginning parallel regions





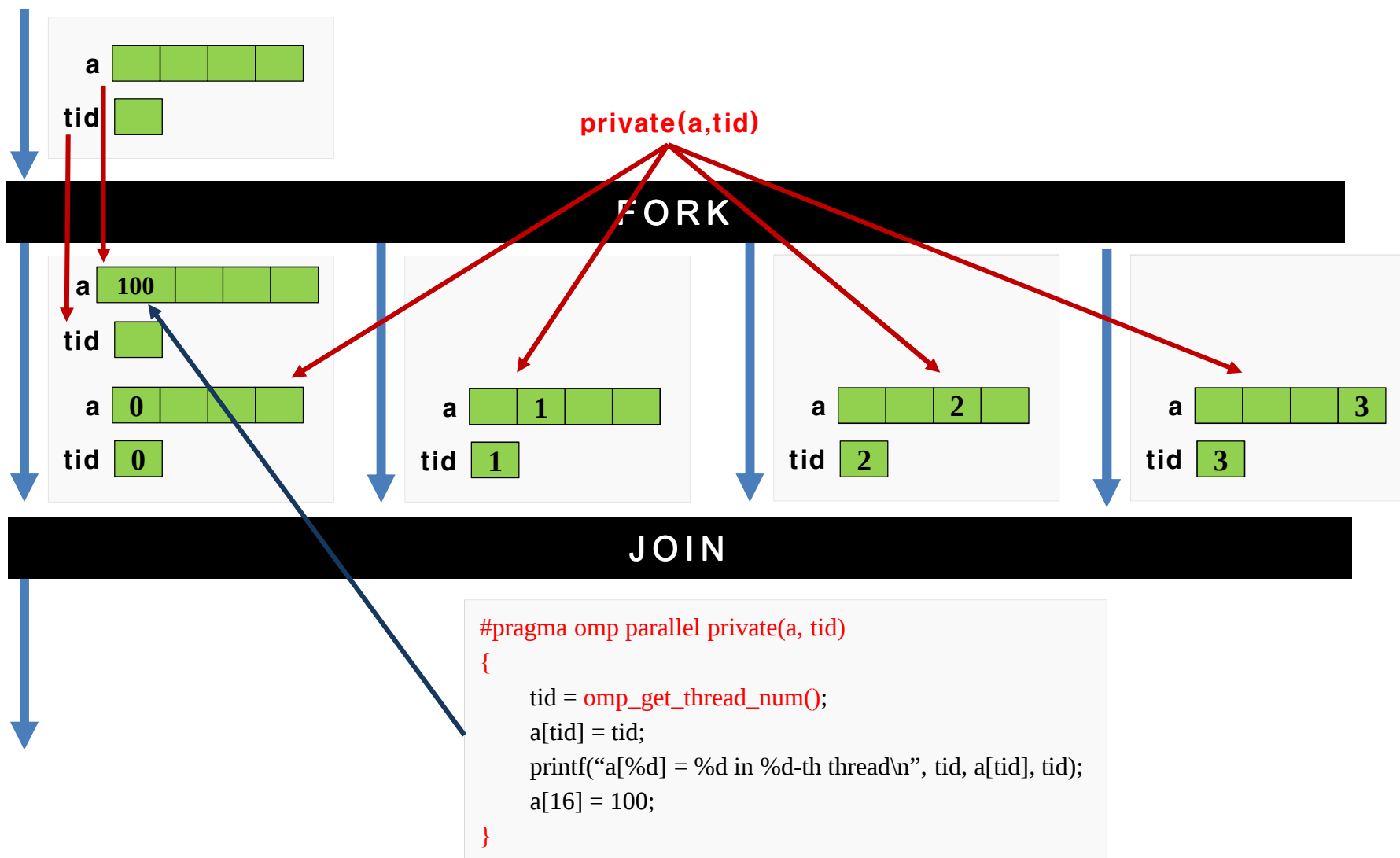
Fork-Join Model



Fortran	C
<pre>PROGRAM fork_join_test INTEGER :: a(0:3),tid INTEGER 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</pre>	<pre>#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]); return 0; }</pre>



Fork-Join Model





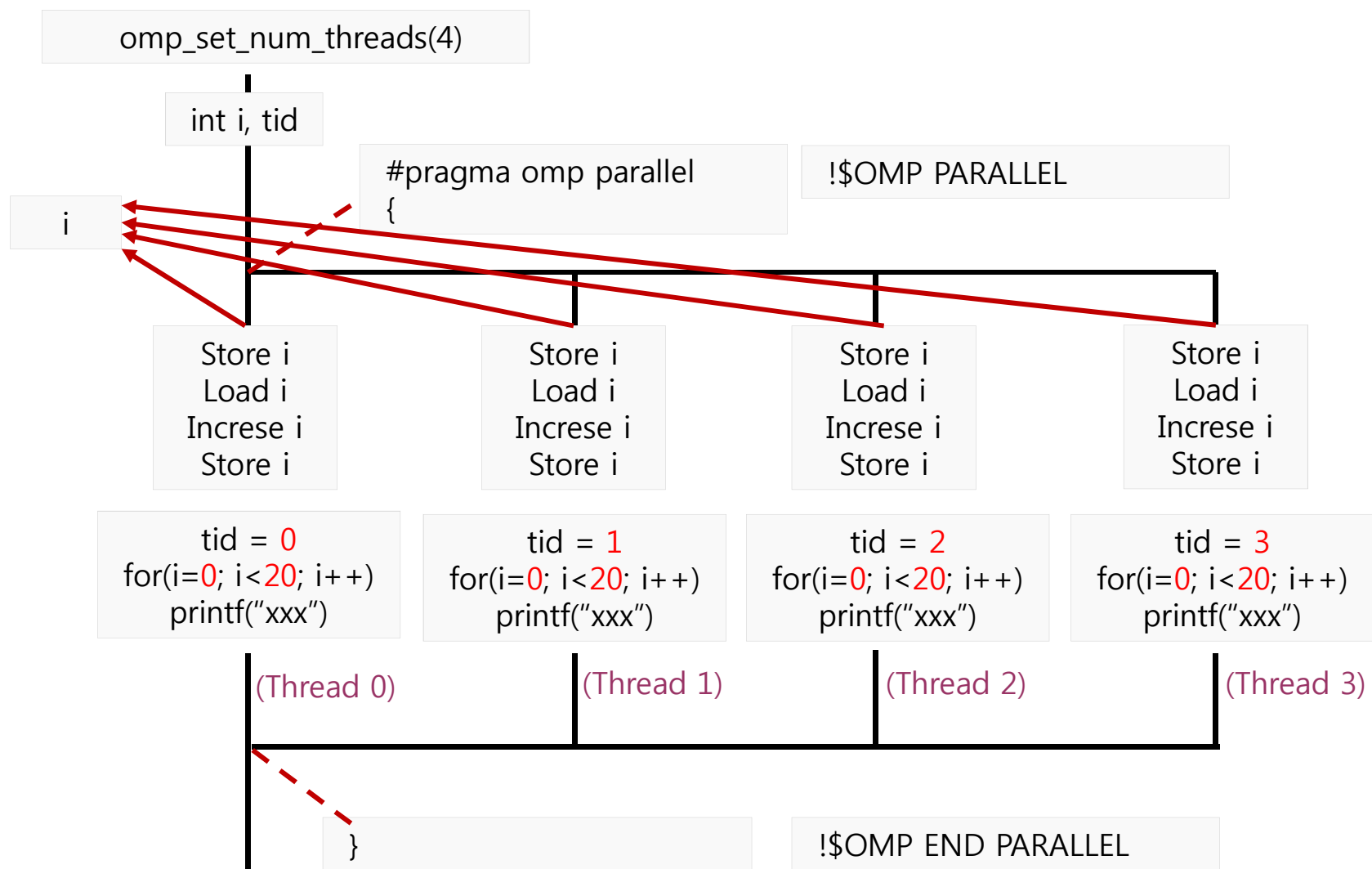
Parallel Loops (1/3)



Fortran	C
<pre>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</pre>	<pre>#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); } return 0; }</pre>



Parallel Loops (2/3)





Parallel Loops (3/3)



Fortran	C
<pre>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 !! ??? !! nested parallel (tomorrow) DO i=0, N-1 PRINT *, 'Hello World', tid, i END DO !\$OMP END PARALLEL END</pre>	<pre>#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 (tomorrow) for (i=0; i<N; i++) printf("Hello World %d %d\n", tid, i); } return 0; }</pre>



Synchronization : critical option(name) (1/2)

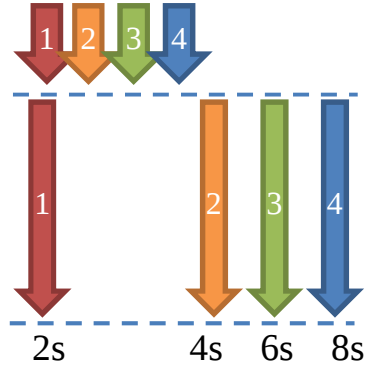


Fortran	C
<pre> PROGRAM sync_test IMPLICIT NONE CALL omp_set_num_threads(4) !\$OMP PARALLEL !\$OMP CRITICAL (n1) CALL sleep(1) !\$OMP END CRITICAL (n1) !\$OMP CRITICAL (n2) CALL sleep(1) !\$OMP END CRITICAL (n2) !\$OMP END PARALLEL END </pre>	<pre> #include <stdio.h> #include <omp.h> int main() { omp_set_num_threads(4); #pragma omp parallel { #pragma omp critical (n1) { sleep(1); } #pragma omp critical (n2) { sleep(1); } } return 0; } </pre>
<pre> \$ gfortran -fopenmp -o sync_test.x sync_test.f90 \$ time ./ sync_test.x real 0m5.014s </pre>	<pre> \$ gcc -fopenmp -o sync_test.x sync_test.c \$ time ./ sync_test.x real 0m5.024s </pre>



Synchronization : critical option(name) (2/2)



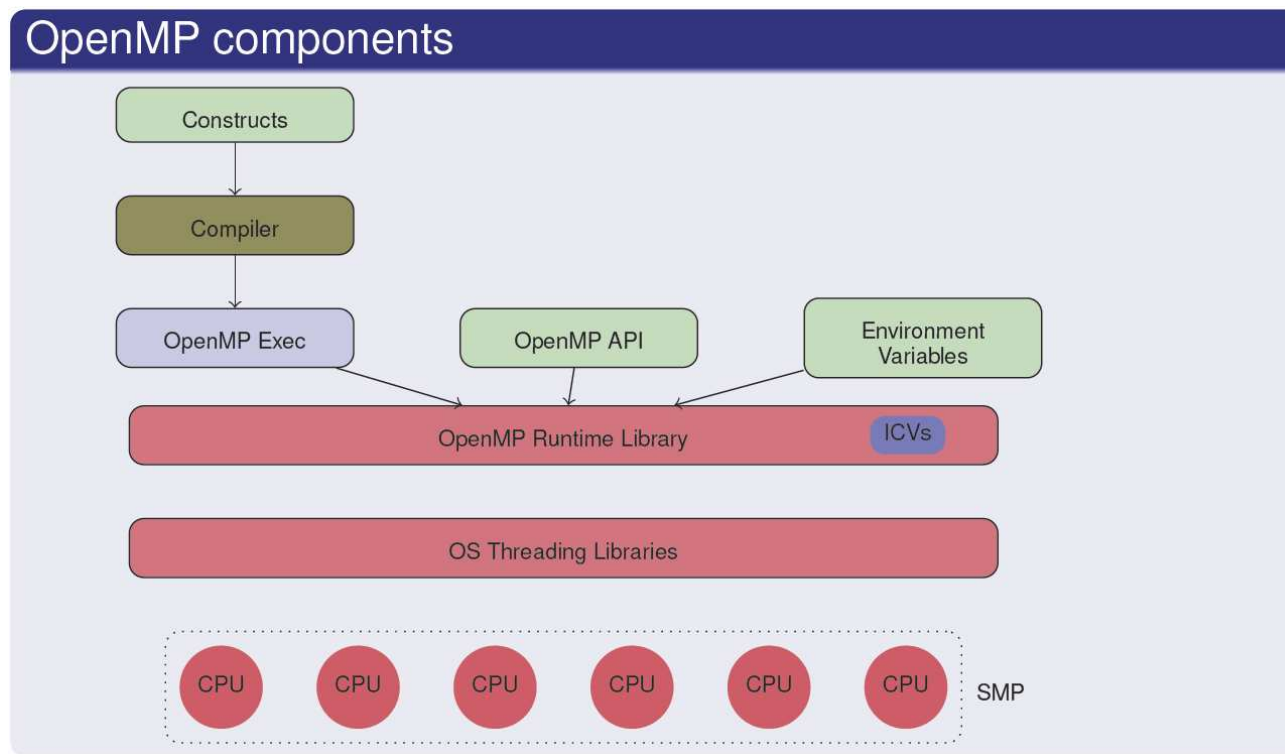
Fortran	C
<pre> PROGRAM sync_test IMPLICIT NONE CALL omp_set_num_threads(4) !\$OMP PARALLEL !\$OMP CRITICAL (n1) CALL sleep(1) !\$OMP END CRITICAL (n1) !\$OMP CRITICAL (n1) CALL sleep(1) !\$OMP END CRITICAL (n1) !\$OMP END PARALLEL END </pre>	<pre> #include <stdio.h> #include <omp.h> int main() { omp_set_num_threads(4); #pragma omp parallel { #pragma omp critical (n1) { sleep(1); } #pragma omp critical (n1) { sleep(1); } } return 0; } </pre> 
<pre> \$ gfortran -fopenmp -o sync_test.x sync_test.f90 \$ time ./ sync_test.x real 0m8.015s </pre>	<pre> \$ gcc -fopenmp -o sync_test.x sync_test.c \$ time ./ sync_test.x real 0m8.013s </pre>



ICV (Internal Control Variable)



- The number of threads : nthreads-var
- The dynamic adjustment of threads : dyn-var
- Runtime schedule type and chunk size : run-sched-var
- To enable or to disable nested parallelism : nested-var



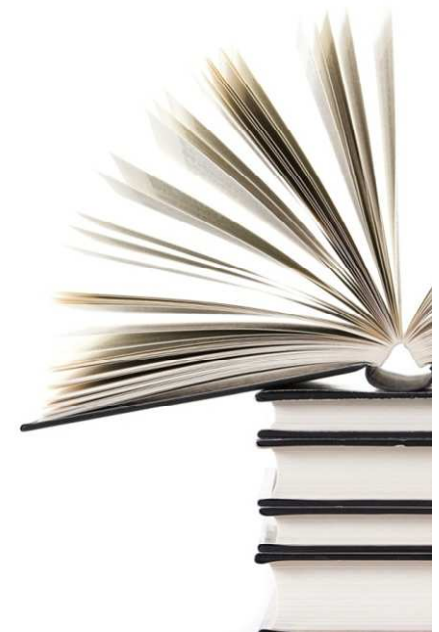
Ref. : Parallel Programming with OpenMP, BSC (Barcelona Supercomputing Center), Spain

Break!!



Hands-on

- 1. Pi : fomula**
- 2. Pi : Numerical Integration**
- 3. Pi : Monte-carlo**





ex1. Formula Pi

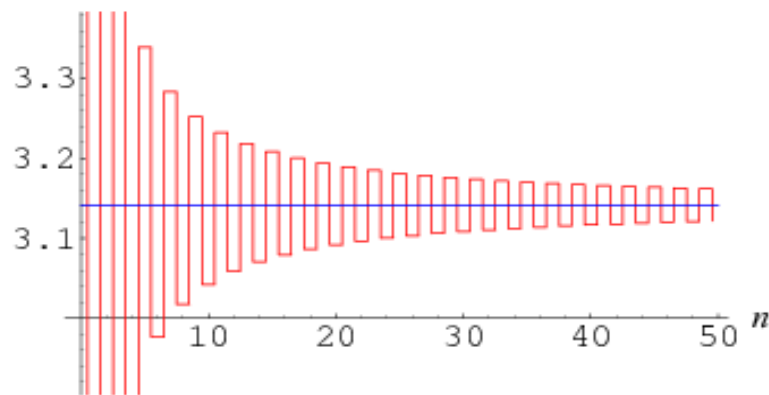


➤ Pi calculation for Formulas

1. There are many formulas of π of many series.
2. Gregory and Leibniz found

$$\frac{\pi}{4} = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{2k-1} = 1 - \frac{1}{3} + \frac{1}{5} - \dots$$

$$4 \sum_{k=1}^n \frac{(-1)^{k+1}}{2k-1}$$





ex1. Formula Pi



Fortran	C
<pre>!! KISTI educational PI calculation !! Copyright (c) 2012 KISTI Supercomputing Center PROGRAM Pi_cal3 IMPLICIT NONE !! Number of count INTEGER(KIND=8), PARAMETER :: num_cnt=100000000 INTEGER(KIND=8) :: i DOUBLE PRECISION :: pi WRITE(*,200) pi = 0.0d0 DO i=0, num_cnt-1 pi = pi + 4.0d0 * (dble(-1)**i/ dble(2*i+1)) ENDDO WRITE(*, 100) pi, dabs(dacos(-1.0d0)-pi) WRITE(*, 200) 100 FORMAT("PI =, F17.15, "(Error =, E11.5,)"") 200 FORMAT("-----") STOP END PROGRAM</pre>	<pre>// KISTI educational PI calculation // Copyright(c) 2012 KISTI Supercomputing Center #include <stdio.h> #include <math.h> int main() { const long num_cnt=100000000; long i; double pi; printf("-----\n"); pi = 0.0; for (i=0; i<num_cnt; i++) pi += 4.0 * (double)pow(-1,i) / (double)(2*i+1); printf("PI = %17.15lf (Error = %e)\n", pi, fabs(acos(-1.0)-pi)); printf("-----\n"); return 0; }</pre>
<pre>\$ gfortran -o formular_pi.x formula_pi.f90 \$./formular_pi.x</pre>	<pre>\$ gcc -o formular_pi.x formula_pi.c -lm \$./formular_pi.x</pre>



ex2. Numerical Integration Pi



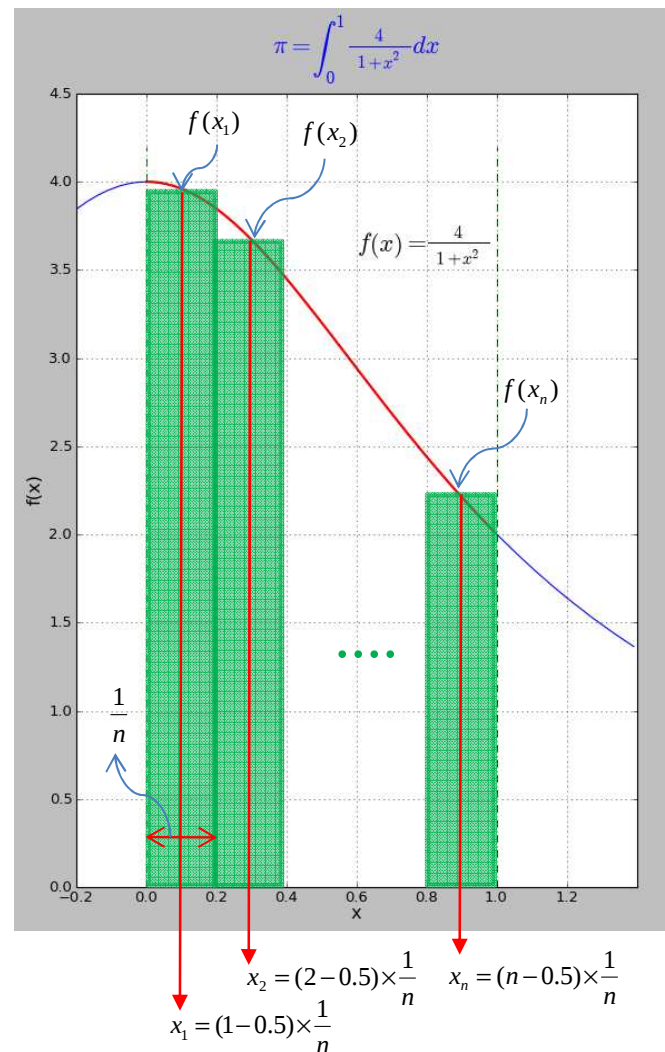
- Pi calculation for Numerical integration

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

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

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

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





ex2. Numerical Integration Pi



Fortran	C
<pre>!! KISTI educational PI calculation !! Copyright (c) 2012 KISTI Supercomputing Center PROGRAM Pi_cal IMPLICIT NONE !! Number of steps INTEGER(KIND=8), PARAMETER :: num_step=100000000 INTEGER :: i DOUBLE PRECISION :: sum, step, pi, x step = (1.0d0/dbl(num_step)) sum = 0.0d0 WRITE(*,200) DO i=0, num_step-1 x = (dbl(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</pre>	<pre>// 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("-----\n"); 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 = %5lf (Error = %e)\n", pi, fabs(acos(-1.0)-pi)); printf("-----\n"); return 0; }</pre>
<pre>\$ gfortran -fopenmp -o pi_cal.x pi_cal.f90 \$./pi_cal.x</pre>	<pre>\$ gcc -fopenmp -o pi_cal.x pi_cal.c -lm \$./pi_cal.x</pre>



ex3. Monte carlo Pi



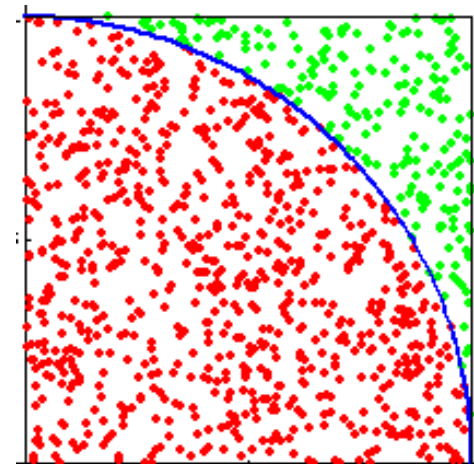
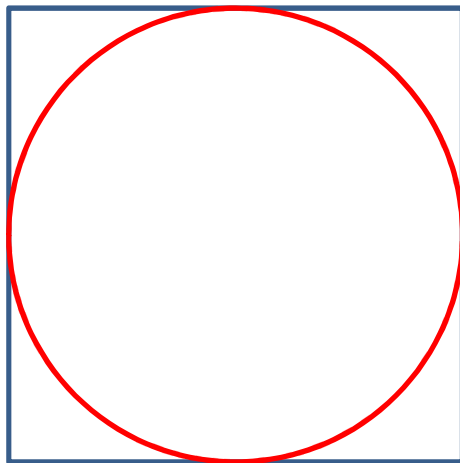
■ Monte carlo Method for Pi

1. A circle with radius $r=1$ inscribed within a square.

2. The area of the circle is $\pi r^2 = \pi 1^2 = \pi$

3. And the area of the square is $(2r)^2 = 2^2 = 4$

4. $\pi \approx 4 \times \frac{\text{number of points of in the Circle}}{\text{total number of points}}$





ex3. Monte carlo Pi



Fortran	C
<pre>!! KISTI educational PI calculation !! Copyright (c) 2012 KISTI Supercomputing Center PROGRAM Pi_cal3 IMPLICIT NONE INTEGER(KIND=8), PARAMETER :: num_cnt=100000000 INTEGER(KIND=8) :: i, cnt DOUBLE PRECISION :: pi, x, y, r WRITE(*,200) pi = 0.0d0; cnt = 0; r = 0.0d0; DO i=0, num_cnt-1 CALL random_number(x) CALL random_number(y) r = sqrt(x*x + y*y) IF (r <= 1) THEN cnt = cnt + 1 ENDIF ENDDO pi = 4.0d0 * dble(cnt) / dble(num_cnt) WRITE(*, 100) pi, dabs(dacos(-1.0d0)-pi) WRITE(*, 200) 100 FORMAT("PI =", F17.15, "(Error =", E11.5,)") 200 FORMAT("-----") STOP END PROGRAM</pre>	<pre>// KISTI educational PI calculation // Copyright(c) 2012 KISTI Supercomputing Center #include <stdio.h> #include <stdlib.h> #include <math.h> int main() { const long num_step=100000000; long i, cnt; double pi, x, y, r; printf("-----\n"); pi = 0.0; cnt = 0; r = 0.0; for (i=0; i<num_step; i++) { x = rand() / (RAND_MAX+1.0); y = rand() / (RAND_MAX+1.0); r = sqrt(x*x + y*y); if (r<=1) cnt += 1; } pi = 4.0 * (double)(cnt) / (double)(num_step); printf("PI = %17.15lf (Error = %e)\n", pi, fabs(acos(-1.0)-pi)); printf("-----\n"); return 0; }</pre>
<pre>\$ gfortran -fopenmp -o pi_monte.x pi_monte.f90 \$./pi_monte.x</pre>	<pre>\$ gcc -fopenmp -o pi_monte.x pi_monte.c -lm \$./pi_monte.x</pre>

Q&A

A large, three-dimensional red question mark is positioned to the right of the 'Q&A' text. It has a thick, blocky design with a circular base and a curved top. The entire graphic is set against a light gray, tilted rectangular background.

