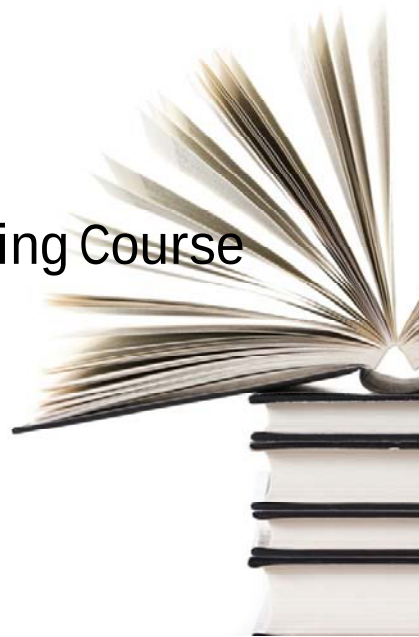


OpenMP Training Course

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



Motivation

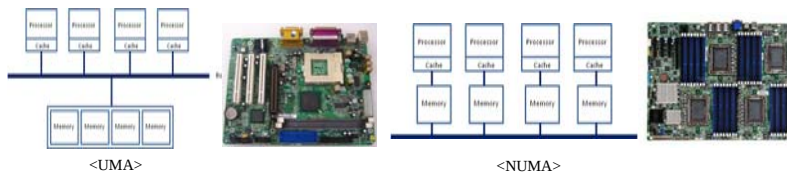
1. Why Parallel Computing ?
2. Why OpenMP ?
3. Check Environment



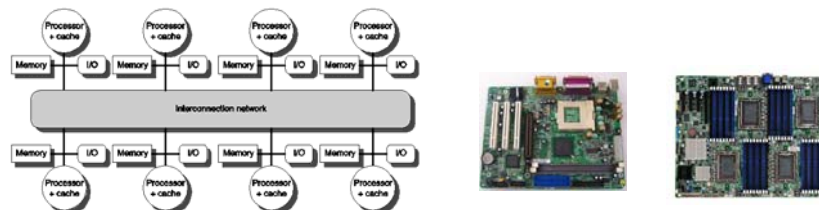
Why Parallel Computing ?



- Shared Memory
 - Single address space for all processors



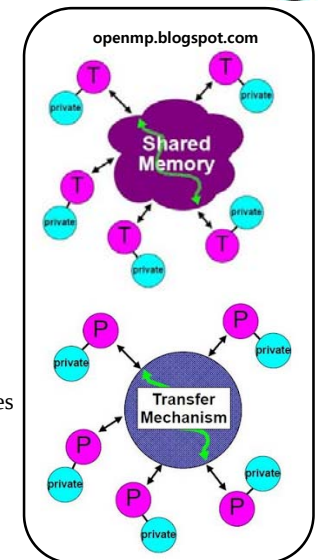
- Distributed Memory



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





OpenMP I

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



5

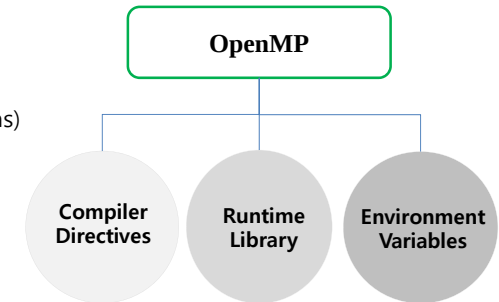


What is OpenMP ?



- 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



6



Components of OpenMP



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>	<pre>#include <stdio.h> #include <omp.h> int main() { #pragma omp parallel { printf ("Hello World %d\n", omp_get_thread_num()); } return 0; }</pre>
<pre>\$ gfortran -fopenmp -o omp_component.x omp_component.f90 \$ export OMP_NUM_THREADS=4 \$./omp_component.x</pre>	<pre>\$ gcc -fopenmp -o omp_component.x omp_component.c \$ export OMP_NUM_THREADS=4 \$./omp_component.x</pre>

➤ OpenMP Syntax

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

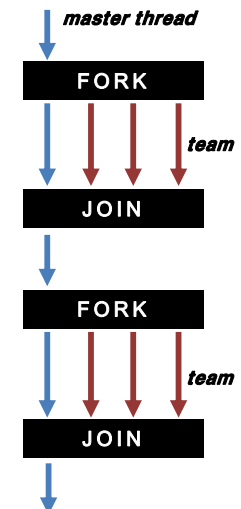
7



Creating Threads



- 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



8

Creating Threads



Parallel Region

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

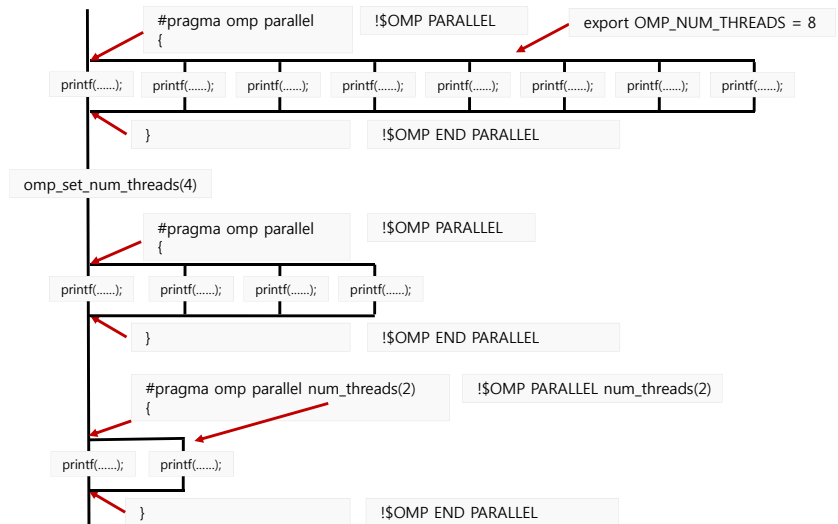
Set number of thread

- 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



Creating Threads

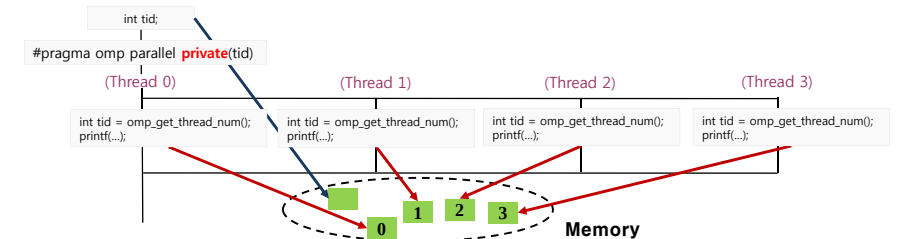


Fortran	C
<pre>PROGRAM create_thread IMPLICIT NONE INTEGER omp_get_thread_num !\$OMP PARALLEL PRINT *, 'Hello World', omp_get_thread_num() !\$OMP END PARALLEL print *, '' CALL omp_set_num_threads(4) !\$OMP PARALLEL PRINT *, 'Hello World', omp_get_thread_num() !\$OMP END PARALLEL print *, '' !\$OMP PARALLEL 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>

Data Scope Attribute

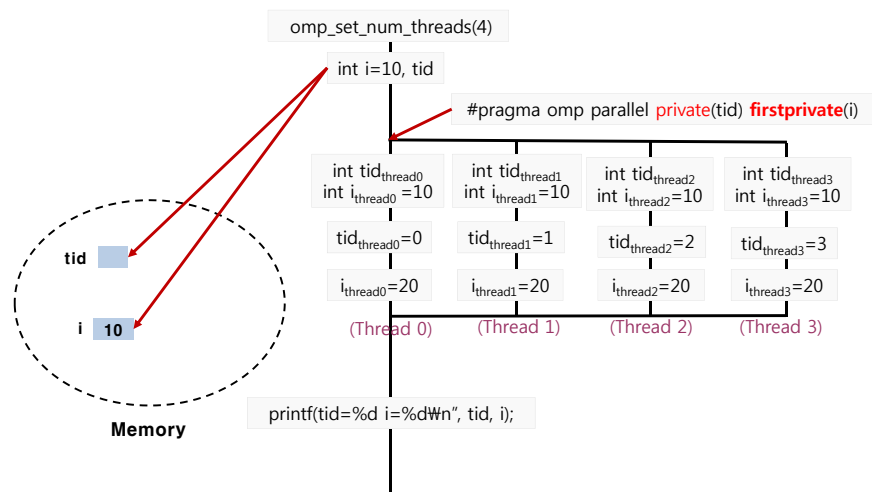


Fortran	C
<pre>PROGRAM hello_right INTEGER tid, omp_get_thread_num CALL omp_set_num_threads(4) !\$OMP PARALLEL PRIVATE(tid) tid = OMP_GET_THREAD_NUM() PRINT *, 'I am ', OMP_GET_THREAD_NUM(), ' tid = ', tid !\$OMP END PARALLEL END</pre>	<pre>#include <stdio.h> #include <omp.h> int main() { int tid; omp_set_num_threads(4); #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>





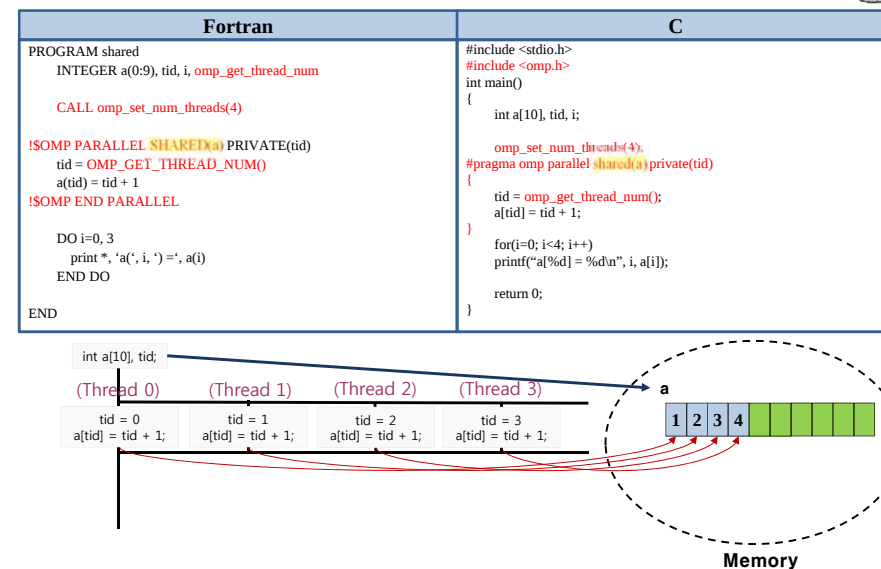
Data Scope Attribute



13



Data Scope Attribute



14



OpenMP II

1. Parallel Loops
2. Synchronization



15

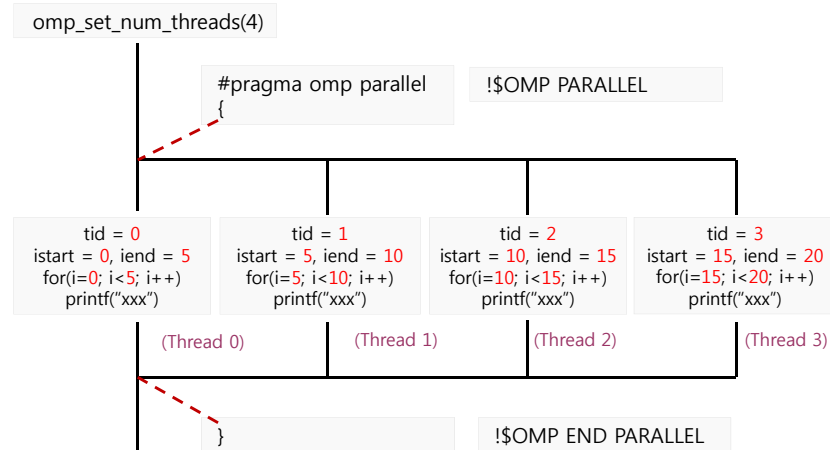


Parallel Loops

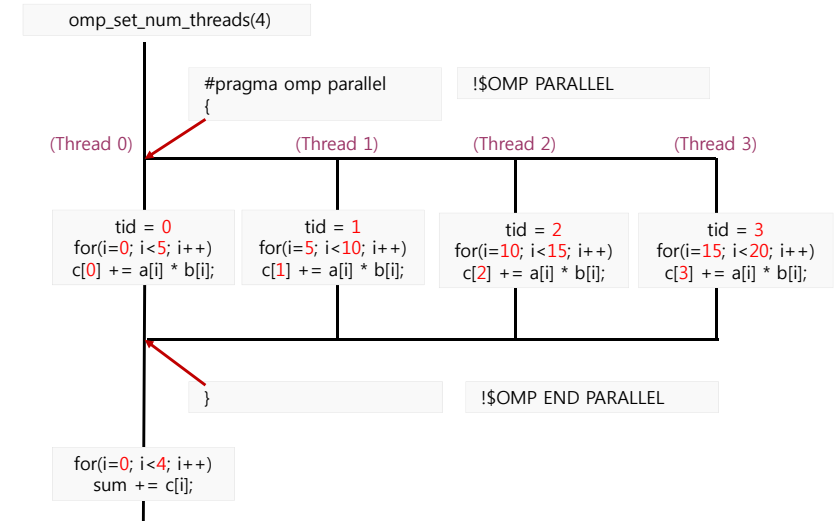


Fortran	C
<pre> PROGRAM parallel_loop2 INTEGER, PARAMETER :: N=20 INTEGER :: tid, i, istart, iend, nthreads, & omp_get_thread_num, omp_get_num_threads CALL omp_set_num_threads(4) !SOMP PARALLEL private(i, tid, istart, iend) tid = omp_get_thread_num() nthreads = omp_get_num_threads() istart = tid * N / nthreads iend = (tid+1) * N / nthreads - 1 DO i=istart, iend PRINT *, 'Hello World', tid, i END DO !SOMP 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>

16



17



18



Fortran		C	
!\$OMP PARALLEL DO i=0, 99 !\$OMP CRITICAL(n1) CALL sub(a,b) !\$OMP END CRITICAL END DO !\$OMP END PARALLEL	!\$OMP PARALLEL DO i=0, 99 !\$OMP ATOMIC a = a + b END DO !\$OMP END PARALLEL	#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; } }

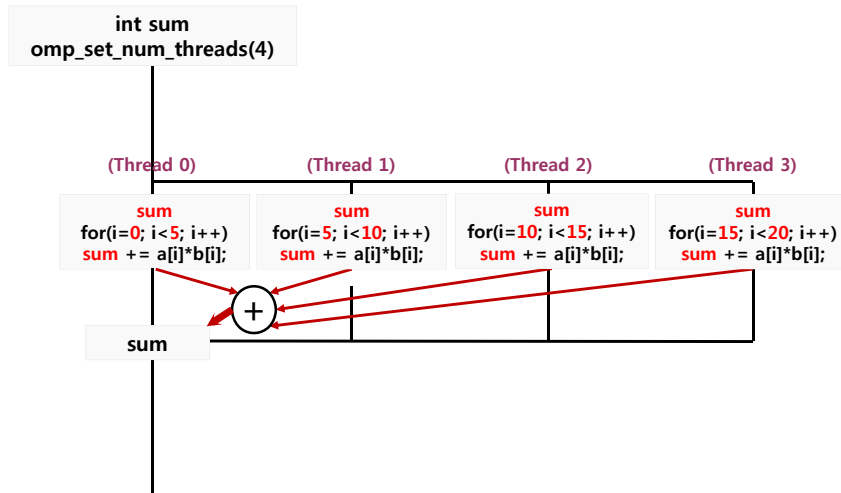
- Synchronization
 - Imposing order constraints and protecting access to shared data
- High level synchronization
 - **critical, atomic, barrier**, ordered
- Low level synchronization
 - flush, locks
- critical vs atomic
 - critical : Only one thread at a time can enter a critical region, distinguished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

19



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>

20



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>

OpenMP Programming in Intel Xeon Phi Coprocessor

- `/proc/cpuinfo`, `/proc/meminfo`
- `top` command
- `numactl --hardware`
- `cpuinfo`
- `dmidecode -t processor/memory/cache`
- LIKWID tool suite
 - <http://code.google.com/p/likwid>
- `hwloc`
 - <http://www.open-mpi.org/projects/hwloc>

```
[doko76@m8 ~]$ cat /proc/cpuinfo
===== Processor composition =====
Processor name : Intel(R) Xeon(R) E5630
Packages(sockets) : 2
Cores : 8
Processors(CPUs) : 16
Cores per package : 4
Threads per core : 2
===== Processor identification =====
Processor Thread Id. Core Id. Package Id.
0 0 0 0
.....
===== Placement on packages =====
Package Id. Core Id. Processors
0 0,1,9,10 (0,8)(2,10)(4,12)(6,14)
1 0,1,9,10 (1,9)(3,11)(5,13)(7,15)
===== Cache sharing =====
Cache Size Processors
L1 32 KB
(0,8)(1,9)(2,10)(3,11)(4,12)(5,13)(6,14)(7,15)
L2 256 KB
(0,8)(1,9)(2,10)(3,11)(4,12)(5,13)(6,14)(7,15)
L3 12 MB (0,2,4,6,8,10,12,14)(1,3,5,7,9,11,13,15)
```

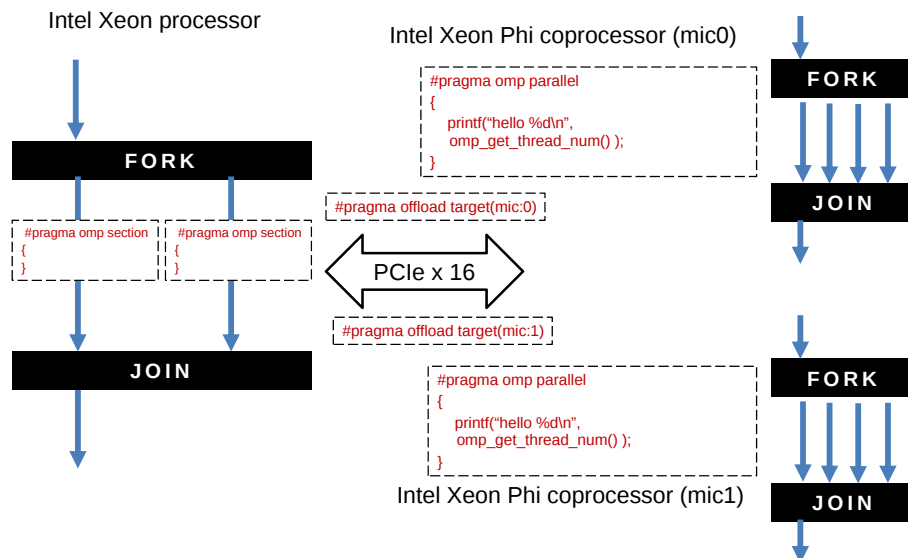
Programming Model

- Native OpenMP on the host
 - ex. `icc -openmp source.c`
- Serial host with OpenMP offload
- OpenMP on the host with OpenMP offload
- Native OpenMP on the Xeon Phi coprocessor
 - ex. `icc -mmic -openmp source.c`

OpenMP Host with OpenMP Offload (1/2)

Fortran	C
<pre> program section_offload use omp_lib implicit none !\$omp parallel !\$omp sections !\$omp section !DIRS offload begin target(mic) !\$omp parallel print *, 'hello ', omp_get_thread_num() !\$omp end parallel !DIRS end offload !\$omp section !DIRS offload begin target(mic) !\$omp parallel print *, 'hello ', omp_get_thread_num() !\$omp end parallel !DIRS end offload !\$omp end sections !\$omp end parallel end </pre>	<pre> #include <stdio.h> int main() { #pragma omp parallel { #pragma omp sections { #pragma omp section { #pragma offload target(mic:0) #pragma omp parallel { printf("hello %d\n", omp_get_thread_num()); } } } #pragma omp section { sleep(4); #pragma offload target(mic:1) #pragma omp parallel { printf("hello %d\n", omp_get_thread_num()); } } } } </pre>

OpenMP Host with OpenMP Offload (2/2)



OpenMP on Xeon Phi Coprocessor

- Default Setting
 - OpenMP on host without Hyper-Threading : # of cores
 - OpenMP on host with Hyper-Threading : # of cores x 2
 - OpenMP on Xeon Phi in native mode : # of cores x 4
 - OpenMP on Xeon Phi in offload mode : # (of cores -1) x 4
 - With norespect modifier : # of cores x 4
- Environment Setting
 - `$ export OMP_NUM_THREADS = # of threads on host`
 - `$ export MIC_ENV_PREFIX = MIC`
 - `$ export MIC_OMP_NUM_THREADS = # of threads on Xeon Phi`
 - `$ export MIC_KMP_AFFINITY = compact/scatter/balanced`
 - `$ export MIC_2_OMP_NUM_THREADS = # of threads on 2nd Xeon Phi`
 - `$ export MIC_3_ENV = "OMP_NUM_THREADS=xxx | KMP_AFFINITY=xxx"`
- Runtime Library
 - `omp_set_num_threads_target, omp_set_nested_target, etc.`

Offload and OpenMP (1/4)

Fortran	C
<pre> program sum_offload use omp_lib implicit none integer(8) :: i, total_sum = 0 !DIR\$ offload begin target(mic) !\$omp parallel do reduction (+:total_sum) do i=1, N total_sum = total_sum + i end do !\$omp end parallel do !DIR\$ end offload print *, 'sum = ', total_sum end </pre>	<pre> #include <stdio.h> int main() { long i, sum=0; #pragma offload target(mic) #pragma omp parallel for reduction (+:sum) for(i=1; i<=N; i++) sum += i; printf("sum = %ld\n", sum); } </pre>
Fortran	C
<pre> !\$omp parallel do reduction (+:total_sum) !DIR\$ offload begin target(mic) do i=1, N total_sum = total_sum + i end do !DIR\$ end offload !\$omp end parallel do </pre>	<pre> #pragma omp parallel for reduction (+:sum) #pragma offload target(mic) #pragma omp single for(i=1; i<=N; i++) sum += i; } </pre>

Offload and OpenMP (2/4)

Fortran	C
<pre> program sum_offload2 use omp_lib implicit none integer(8) :: i, total_sum = 0 !\$omp parallel !\$omp single !DIR\$ offload begin target(mic) print *, 'check' !DIR\$ end offload !\$omp end single !\$omp parallel do reduction (+:total_sum) do i=1, N total_sum = total_sum + i end do !\$omp end parallel do !\$omp end parallel end </pre>	<pre> #include <stdio.h> int main() { long i, sum=0; #pragma omp parallel { #pragma omp single #pragma offload target(mic) { printf("check\n"); } #pragma omp for reduction (+:sum) for(i=0; i<N; i++) sum += i; } } </pre>

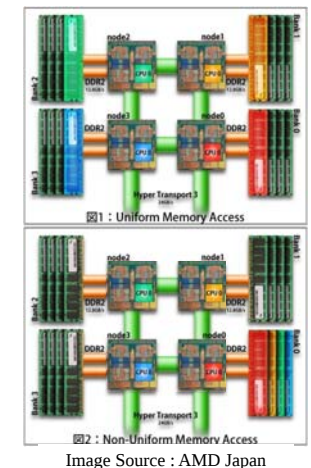
Offload and OpenMP (3/4)

Fortran	C
<pre> program sum_offload3 use omp_lib implicit none integer(8) :: i, total_sum = 0, local_sum=0 !\$omp parallel private(local_sum) local_sum = 0 !\$omp single !DIR\$ offload begin target(mic) print *, 'check' !DIR\$ end offload !\$omp end single nowait !\$omp parallel do schedule(dynamic) do i=1, N local_sum = local_sum + i end do !\$omp end parallel do !\$omp end parallel End </pre>	<pre> #include <stdio.h> int main() { long i, sum=0, local_sum; #pragma omp parallel private(local_sum) { local_sum = 0; #pragma omp single nowait #pragma offload target(mic) { printf("check\n"); } #pragma omp for schedule(dynamic) for(i=0; i<N; i++) local_sum += i; } } </pre>

First Touch

- “Touch” means “write”, not “allocate”

Fortran	C
<pre> program first_touch use omp_lib implicit none integer(8) :: i double precision :: A(N) !\$omp parallel do do i=1, N A(i) = 0.0 end do !\$omp end parallel do End </pre>	<pre> #include <stdio.h> int main() { int i; double *A; A = (double *) malloc(sizeof(double) * N); #pragma omp parallel for for(i=0; i<N; i++) A[i] = 0.0; } </pre>



Thread Affinity

- Current Support for Thread Affinity (Environment Variable)
 - Intel : KMP_AFFINITY
 - GNU : GOMP_CPU_AFFINITY
 - Oracle : SUNW_PROCBIND
 - PGI : MP_BIND and MP_BLIST
 - IBM : XLSPMOPPTS(PROC, PROC_CORE, L2CACHE, MCM)
- Thread Affinity Interface
 - High-level affinity interface : scatter, compact, etc.
 - Mid-level affinity interface : specify process id
 - Low-level affinity interface : sched_{set/get}_affinity
- GNU (mid-level)
 - \$ export GOMP_CPU_AFFINITY = 0,1,2,3
 - \$ export GOMP_CPU_AFFINITY = 1-16
- Intel
 - \$ export KMP_AFFINITY = modifier, type, permute, offset
- taskset
 - \$ taskset -c 0,2 ./a.out or \$ taskset 0x0005 ./a.out
- Numactl
 - \$ numactl --physcpubind=0,3 ./a.out

```
#include <stdio.h>
#include <sched.h>
int main()
{
    #pragma omp parallel
    {
        printf("cpuid : %d\n", sched_getcpu() );
    }
}
```

MIC Thread Affinity

- The optimal number of threads
 - Intel MIC maintains 4 hardware contexts per core
 - Round-robin execution policy,
- Use OpenMP application with NCORE cores
 - On host only for 2 x ncore
 - On MIC Native mode for 4 x ncore
 - Offload 4 x (ncore-1) because OpenMP runtime avoids the core OS runs
- Two ways to specify thread affinity
 - OMP_NUM_THREADS, KMP_AFFINITY
 - kmp_set_defaults("KMP_AFFINITY=compact")
 - omp_set_num_threads(244);

2014-02-26

Korea-Japan HPC Winter School

34

Thread Affinity Choices

- Intel OpenMP Supports the following Affinity Type:
 - Compact** assign threads to consecutive h/w contexts on same physical core
 - Scatter** assign consecutive threads to different physical cores maximize access to M

