Module 11: The “lastprivate” Clause

Lecture 21: Clause and Routines

The Lecture Contains:

Previous Next

Module 11: The “lastprivate” Clause

Lecture 21: Clause and Routines

The “lastprivate” Clause

Example: “lastprivate” Clause

```
void useless() {
int tmp = 0;
#pragma omp parallel for firstprivate(tmp)lastprivate(tmp)
for (int j = 0; j ¡ 1000; ++j)
tmp += j;
printf(“%d”, tmp);
}
```

- Each thread gets its own code with initial value 0.Is there something still wrong with the code ?
- Tmp is defined as its value at the “last sequential” iteration (i .e., for j=999)

Data Scope Attribute Clauses

“shared” Clause

Purpose: The shared clause declares variables in its list to be shared among all the threads in the team

Format: Shared (list)

- A shared variable exists in only one memory and all the threads can read or write the same address
- Its programmers responsibility to ensure that multiple threads properly access shared variables (such as critical sections)

“default” Clause

Purpose: The default clause allows user to specify a default scope for all variables in the parallel region

Format: Default(shared | none)

- Using NONE as a default requires that the programmer explicitly scope all the variables
- The C/C++ OpenMP does not include private or firstprivate as a possible default
- Only the Fortran API supports default(private)
- Note that the default storage attribute is DEFAULT(SHARED) (so no need to use it)

Previous Next

Reduction

How to handle this case?

Example

```
double ave = 0.0, A [ MAX ]; int i;
for (i = 0; i < MAX; i++) {
ave + = A[i];
}
ave = ave/MAX;
```

- We are combining values into a single accumulation variable (ave) there is a true dependence between loop iterations that can not be trivially removed
- This is a very common situation.it is called a “reduction”
- Support for reduction operations is included in most parallel programming environments.

Data Scope Attribute Clauses

“reduction” Clause

Purpose: Reduction

Format: reduction (operation : list)

- A local copy of each list variable is made and initialized depending on the “op” (e.g. 0 for “+”)
- Compiler finds standard reduction expressions containing “op” and uses them to update the local copy.
- Local copies are reduced into a single value and combined with the original global value
- Variables in the list must be named scalar variables. They can not be array or structure type variables.
- Reduction variables must be shared in the enclosing context.
- Note that the value of a reduction variable is undefined from the moment the first thread reaches the clause till the operation has completed

Previous Next

Module 11: The “lastprivate” Clause

Lecture 21: Clause and Routines

Reduction

Example: Reduction

```
double ave = 0.0, A [ MAX ]; int i;
#pragma omp parallel for reduction (+ : ave)
for (i = 0; i < MAX; i++) {
ave + = A[i];
}
ave = ave/MAX;
```

Data Scope Attribute Clauses

“copyin” Clause

Purpose: The copyin clause provides a means for assigning the same value to threadprivate variables for all threads in the team.

Format: copyin (list)

- The master thread variable is used as the copy source. The team threads are initialized with its value upon entry into the parallel construct.

Loop Work-sharing Construct: Schedule Clause

Format

```
schedule ( static | dynamic | guided [, chunk ] )
schedule (runtime)
```

- The schedule clause affect how loop iterations are mapped into threads

Static [,chunk]

- Loop iterations are divided into pieces of size chunk and then statically assigned to threads
- In absence of chunk size iterations are evenly (if possible) divided contiguously among the threads.
- Pre-determined and predictable by the programmer
- Least work at runtime: scheduling done at compile-time

Loop Work-sharing Construct: Schedule Clause

Dynamic [, chunk]

- Fixed portions of work; size is controlled by the value of chunk
- When a thread finishes one chunk, It is dynamically assigned another
- The default chunk size is 1.
- Most work at runtime: complex scheduling logic used at run-time

Guided [,chunk]

- Special case of dynamic to reduce scheduling overhead
- The size of the block starts large and shrinks down to size chunk as the calculation proceeds
- Default chunk size is 1

Runtime

- Iteration scheduling scheme is set at runtime through environment variable OMP SCHEDULE or the runtime library

Questions

Given loop of length 16 with 4 threads:

How the iterations will be assigned in static schedule with no chunk and chunk=2 ?

What will be the change in case of dynamic scheduling?

Run-Time Library Routines

OMP SET NUM THREADS: Sets the number of threads that will be used in the next parallel region. Must be a positive number.

Format

void omp set num threads(int num threads)

- This routine can only be called from the serial portion of the code.
- This call has precedence over the OMP NUM THREADS

Previous Next

Run-Time Library Routines

OMP GET NUM THREADS: Returns the number of threads that are currently in the team executing the parallel region from which it is called.

Format
int omp get num threads(void)

- If this call is made from a serial portion of the program, or a nested parallel region that is serialized, it will return 1.
- The default number of threads is implementation dependent

OMP GET MAX THREADS: Returns the maximum value that can be returned by a call to the OMP GET NUM THREADS function.

Format
int omp get max threads(void)

- This routine can only be called from the serial portion of the code.
- This call has precedence over the OMP NUM THREADS

OMP GET THREAD NUM: Returns the thread number of the thread, within the team, making this call. This number will be between 0 and OMP GET NUM THREADS-1. The master thread of the team is thread 0

Format
int omp get thread num(void)

- If called from a nested parallel region, or a serial region, this function will return 0

OMP GET THREAD LIMIT: New with OpenMP 3.0. Returns the maximum number of OpenMP threads available to a program.

Format
int omp get thread limit(void)

OMP GET NUM PROCS: Returns the number of processors that are available to the program.

Format
int omp get num procs(void)

Previous Next

Run-Time Library Routines

OMP IN PARALLEL: May be called to determine if the section of code which is executing is parallel or not.

Format

int omp in parallel(void)

- For FORTRAN, this function returns TRUE if is called from the dynamic extent of a region executing in parallel, and FALSE otherwise. For C/C++, it will return a non-zero integer if parallel and zero otherwise

OMP SET DYNAMIC: Enables or Disables dynamic adjustment(by the run time system) of the number of threads available for the execution of parallel regions.

Format

int omp set dynamic(int dynamic threads)

- Must be called from serial section of the program
- If dynamic threads evaluated to non-zero, then the mechanism is enabled, otherwise it is disabled

OMP GET DYNAMIC: Used to determine thread adjustment is enabled or not

Format

int omp get dynamic(void)

- For C/C++, non-zero will be returned if dynamic thread adjustment is enabled, and zero otherwise
- For FORTRAN, this function returns TRUE if dynamic thread adjustment is enabled and FALSE otherwise

Previous Next

Run-Time Library Routines

OMP SET NESTED: Used to enable or disable nested parallelism.

Format
int omp set nested(int nested)

- The default is for nested parallelism to be disabled
- For C/C++, if nested evaluates to non-zero, nested parallelism is enabled;otherwise is disabled

OMP GET NESTED: Used to determine if nested parallelism is enabled or not.

Format
int omp get nested(void)

- The default is for nested parallelism to be disabled
- For C/C++, if non-zero value returned then nested parallelism is enabled;otherwise is disabled

OMP INIT LOCK: This subroutine initializes a lock associated with the lock variable.

Format
void omp init lock(omp lock t *lock)
void omp init nest lock(omp nest nest lock t *lock)

- The initial state is unlocked

OMP DESTROY LOCK: This subroutine disassociates the given lock variable from any locks.

Format
void omp destroy lock(omp lock t *lock)
void omp destroy nest lock(omp nest nest lock t *lock)

- It is illegal to call this routine with a lock variable that is not initialized

Previous Next

Run-Time Library Routines

OMP SET LOCK: This subroutine forces the executing thread to wait until the specified lock is available. A thread is granted ownership of a lock when it becomes available.

Format

void omp set lock(omp lock t *lock)
void omp set nest lock(omp nest nest lock t *lock)

- It is illegal to call this routine with a lock variable that is not initialized.

OMP UNSET LOCK: This subroutine releases the lock from the executing subroutine

Format

void omp unset lock(omp lock t *lock)
void omp unset nest lock(omp nest nest lock t *lock)

- It is illegal to call this routine with a lock variable that is not initialized.

OMP TEST LOCK: This subroutine attempts to set a lock, but does not block if the lock is unavailable.

Format

void omp test lock(omp lock t *lock)
void omp test nest lock(omp nest nest lock t *lock)

- It is illegal to call this routine with a lock variable that is not initialized.
- For C/C++ non zero is returned if the lock was set successfully,otherwise zero is returned

Previous Next

OMP GET WTIME: Provides a portable wall clock timing routine and returns a double precision floating point value equal to the number of elapsed seconds since some point in the past. Usually used in ”pairs” with the value of the first call substracted from the value of the second call to obtain the elapsed time for ablock of code (per thread times)

Format

double omp get wtime(void)

OMP GET WTICK: Returns a double precision floating point value equal to the number of seconds between successive clock ticks.

Format

double omp get wtick(void)

Environment Variables

- OpenMP provides the following environment variables for controlling the execution of parallel code
- All environment variable names are uppercase. The values assigned to them are not case sensitive

List of Variables

- OMP_SCHEDULE
- OMP_NUM THREADS
- OMP_DYNAMIC
- OMP_NESTED
- OMP_STACKSIZE

Uses Example

setenv OMP_SCHEDULE “guided, 4”
setenv OMP_NUM THREADS 8

Previous Next

Thank You

Questions ?

References:

- OpenMP Home (link)
- OpenMP Book Using OpenMP
  Portable Shared Memory Parallel Programming
  Barbara Chapman, Gabriele Jost and Ruud van der Pas (link)
- OpenMP Tutorial by Blaise Barney, Lawrence Livermore National Laboratory (link)
- An Overview of OpenMP - Ruud van der Pas - Sun Microsystems (link)
- Hands-On Introduction to OpenMP, Mattson and Meadows, from SC08 (Austin) (link)
- OpenMP wikipedia page
- More resources can be found at (link)

Previous Next