Parallel Programming in OpenMP – part III

Outline

- ❑ Runtime library
- ❑ Environment variables
- ❑ OpenMP Future
- ❑ Behind the scenes
- ❑ Summary
- ❑ References

OpenMP Runtime Library

The OpenMP runtime library:
support functions

OpenMP Runtime Library

The OpenMP standard defines an API for library calls, that have a variety of functions:

- query
  - the number of threads/processors
  - thread ID, “in parallel”
- set
  - the number of threads to use
  - scheduling mode
- locking (semaphores)

OpenMP Runtime Library

Name	*Functionality*
omp_set_num_threads	set number of threads
omp_get_num_threads	get number of threads in team
omp_get_max_threads	get max. number of threads
omp_get_thread_num	get thread ID
omp_get_num_procs	get max. number of processors
omp_in_parallel	check whether in parallel region
omp_set_dynamic	activate dynamic thread adjustment
omp_get_dynamic	check for dynamic thread adjustment
	(implementation can ignore this)
omp_set_nested	activate nested parallelism
omp_get_nested	check for nested parallelism
	(implementation can ignore this)
omp_get_wtime	returns wall clock time
omp_get_wtick	number of second between clock ticks

OpenMP Runtime Library

Function prototypes:

```
void omp_set_num_threads(int num_threads)
int  omp_get_num_threads(void)
int  omp_get_max_threads(void)
int  omp_get_thread_num(void)
int  omp_get_num_procs(void)
int  omp_in_parallel(void)

void omp_set_dynamic(int dynamic_threads)
int  omp_get_dynamic(void)
void omp_set_nested(int nested)
int  omp_get_nested(void)

double omp_get_wtime(void)
double omp_get_wtick(void)
```

OpenMP 3.0 Runtime Library

Name	*Functionality*
omp_set_schedule	set the schedule
omp_get_schedule	get the schedule
omp_get_thread_limit	max. number of available threads in the implementation
omp_set_max_active_levels	set the number of nested levels
omp_get_max_active_levels	get the number of nested levels
omp_get_level	returns the current nesting level
omp_get_ancestor_thread_num	returns thread id of the ancestor thread in specified level
omp_get_team_size	get team size at specified level
omp_get_active_level	returns the number of enclosing, active nested parallel regions

for more details see the OpenMP 3.0 specifications

OpenMP Runtime Library

- with the increasing number of features of OpenMP, the number of runtime library functions is growing, too
- OpenMP 5.x has now more than 60 runtime library functions!
- check https://www.openmp.org/specifications/

OpenMP Runtime Library

Usage of omp_get_num_threads() vs omp_get_max_threads():

```
// get the number of threads
threads = omp_get_max_threads();
```

returns value of OMP_NUM_THREADS

```
// get the number of threads
threads = omp_get_num_threads();

#pragma omp parallel
{
#pragma omp master
{ threads = omp_get_num_threads(); }
} // end parallel
```

returns 1- outside a parallel region

returns value of threads in a parallel region

OpenMP Runtime Library

Measuring time:

- It is most useful to compare wall clock times

```
double ts, te;
ts = omp_get_wtime();

do_work();

te = omp_get_wtime() - ts;

printf(“Elapsed time: %lf\n”, te);
```

- clock() returns the accumulated CPU time of all threads!

OpenMP Environment Variables

Controlling OpenMP
via Environment Variables

OpenMP Environment Variables

- OMP_NUM_THREADS = n
  - sets the max. no of threads to n
- OMP_SCHEDULE = schedule[,chunk]
  - schedule: [static | guided | dynamic]
  - chunk: size of chunks (*defaults*: [n/a|1|1])
  - Note: applies to parallel do/for loops only!
- OMP_DYNAMIC = [TRUE | FALSE]
- OMP_NESTED = [TRUE | FALSE]

OpenMP Environment Variables

- OMP_STACKSIZE = size[B|K|M|G]
  - sets the size of the stack of OpenMP threads
  - default unit: Kilobytes
- OMP_WAIT_POLICY = active|passive
  - controls the behaviour of idle threads
  - active: “spinning threads”, i.e. use cycles
  - passive: threads go to sleep
  - the default is implementation dependent

OpenMP Environment Variables

- OMP_PROC_BIND = [true|false|close|spread]
  - controls the binding of threads to cores
  - gives a hint if this should be packed or spread out over the system
- OMP_PLACES = [cores|sockets|<list>]
  - controls the placement of threads
    - cores: place across cores
    - sockets: place on whole sockets
    - or provide a list with core numbers
    - works in combination with binding!

OpenMP Environment Variables

- OMP_MAX_ACTIVE_LEVELS = n
  - controls the max. level for nested parallellism
- OMP_THREAD_LIMIT = n
  - sets the maximum number of threads for an OpenMP program

OpenMP Environment Variables

Notes:

- The defaults are depended on the compiler and runtime environment used.
- You can use OMP_DISPLAY_ENV=true to show the settings at the startup of your program. This is useful to check for differences in runtime implementations!
- On the DTU HPC systems, we set OMP_NUM_THREADS=1 as a default, i.e. you need to adjust the value to your needs!

OpenMP Precedence

- Level of priority:
  1. clauses, e.g. num_threads(...)
  2. library calls, e.g. omp_set_num_threads(...)
  3. environment variables, e.g. OMP_NUM_THREADS
- For a detailed discussion see the OpenMP specifications or check the documentation of your OpenMP implementation.

OpenMP Features

OpenMP development
and standard extensions

OpenMP Features

- New features are discussed in the OpenMP ARB and the community, and made or make it into the standard, e.g. extensions for
  - better performance
  - memory placement (4.0)
  - debugging
  - checks, both at compile- and run-time
  - exception handling (4.0)
  - access to accelerators (e.g. GPUs) (4.0)
  - ...

OpenMP extension: Autoscoping

Courtesy: Dieter an Mey, RWTH Aachen

```
!$omp parallel do &
!$omp &
  lastprivate(INE,IEE,ISE,ISW,IWW,INW,IUE,IUW,IDE,IDW,INU,IND,ISU,ISD)
```

```
 ...
!$omp & omegaz,prode,qdens,qjc,qmqc,redbme,redbpe,renbme, &
!$omp & renbpe,resbme,resbpe,reubme,reubpe,rkdbmk,rkdbpk,rknbmk, &
!$omp & rknbpk,rksbmk,rksbpk,rkubmk,rkubpk,rtdbme,rtdbpe,rtnbme, &
!$omp & rtnbpe,rtsbme,rtsbpe,rtubme,rtubpe,rudbme,rudbmx,rudbmy, &
!$omp & rudbmz,rudbpe,rudbpx,rudbpy,rudbpz,runbme,runbmx,runbmy, &
!$omp & runbmz,runbpe,runbpx,runbpy,runbpz,rusbme,rusbmx,rusbmy, &
!$omp & rusbmz,rusbpe,rusbpx,rusbpy,rusbpz,ruubme,ruubmx,ruubmy, &
!$omp & ruubmz,ruubpe,ruubpx,ruubpy,ruubpz,rvdbme,rvdbmx,rvdbmy, &
!$omp & rvdbmz,rvdbpe,rvdbpx,rvdbpy,rvdbpz,rvnbme,rvnbmx,rvnbmy, &
!$omp & rvnbmz,rvnbpe,rvnbpx,rvnbpy,rvnbpz,rvsbme,rvsbmx,rvsbmy, &
!$omp & rvsbmz,rvsbpe,rvsbpx,rvsbpy,rvsbpz,rvubme,rvubmx,rvubmy, &
!$omp & rvubmz,rvubpe,rvubpx,rvubpy,rvubpz,rwdbme,rwdbmx,rwdbmy, &
!$omp & rwdbmz,rwdbpe,rwdbpx,rwdbpy,rwdbpz,rwnbme,rwnbmx,rwnbmy, &
!$omp & rwnbmz,rwnbpe,rwnbpx,rwnbpy,rwnbpz,rwsbme,rwsbmx,rwsbmy, &
!$omp & rwsbmz,rwsbpe,rwsbpx,rwsbpy,rwsbpz,rwubme,rwubmx,rwubmy, &
!$omp & rwubmz,rwubpe,rwubpx,rwubpy,rwubpz,tc,tdb,tdbm, &
!$omp & tdbp,teb,tkc,tk
!$omp & tknbm,tknbp,tks
!$omp & tkwb,tnb,tnbm,t
!$omp & tubm,tubp,twb,u
!$omp & unb,unbm,unbp,u
!$omp & uubp,uwb,vc,vdb
!$omp & vnb,vnbm,vnbp,v
!$omp & vubp,vwb,wc,wdb
!$omp & wnbm,wnbp,wsb,wsbm,wsbp,wub,wubm,wubp, &
!$omp & wwb,xiaxc,xiaxeb,xiaxwb,xiayc,xiayeb,xiaywb,xiazc, &
!$omp & xiazeb,xiazwb,xibxc,xibxnb,xibxsb,xibynb,xibysb,xibznb, &
!$omp & xibzsb,xicxc,xicxdb,xicxub,xicydb,xicyub,xiczdb,xiczub)
   do i = is,ie
      ---- 1600 lines omitted ----
   end do
```

```
!$omp parallel default(__auto) do
    do i = is,ie
      ---- 1600 lines omitted ----
    end do
```

OpenMP extension: Autoscoping

- available with the Oracle Studio compilers, only! No further development!
- if the compiler can't autoscope, you will get a message why it failed
  - use -xvpara to see the messages
  - the failure message is on the .o file as well, make it visible with the er_src command
- was a proposed extension for an upcoming OpenMP standard (didn't make it ...)
- Hint: use the Studio compiler to autoscope, and put the result into your code

OpenMP: Behind the scenes

What the compiler does
with your code

OpenMP: Behind the scenes

```
#define MAX_SIZE 8000000
int main() {
    double GlobSum;            /* A global variable */
    double array[MAX_SIZE];
    int nthreads;
    int i;
    /* Initialize things */
    for (i=0; i<MAX_SIZE; i++) array[i] = i;
    GlobSum = 0;
    nthreads = omp_get_max_threads();
    printf("Threads: %d\n", nthreads );
    #pragma omp parallel for private(i) \
            reduction(+ : GlobSum)
    for(i=0; i<MAX_SIZE;i++)
        GlobSum = GlobSum + array[i];

    return(EXIT_SUCCESS);
}
```

OpenMP: Behind the scenes

- Used the OMPi compiler to generate the intermediate code shown on the next slides.
- The actual implementation differs from compiler to compiler, and probably also from version to version (improvements).

OpenMP: Behind the scenes

```
int main() {
    ...
    int     i;
    _omp_initialize();

    for (i = 0; i < 8000000; i++) array[i] = i;
    GlobSum = 0;
    nthreads = omp_get_max_threads();
    printf("Threads: %d\n", nthreads);

/* #pragma omp parallel for private(i) reduction(+: GlobSum) */
   {
    _OMP_PARALLEL_DECL_VARSTRUCT(main_parallel_0);
    _OMP_PARALLEL_INIT_VAR(main_parallel_0, GlobSum);
    _OMP_PARALLEL_INIT_VAR(main_parallel_0, array);
    _omp_create_team((-1), _OMP_THREAD, main_parallel_0,
        (void *) &main_parallel_0_var); /* create team of
                                         * threads */
    _omp_destroy_team(_OMP_THREAD->parent);
    }

    return 0;
}
```

OpenMP: Behind the scenes

```
void *main_parallel_0(void *_omp_thread_data){
  int     _omp_dummy = _omp_assign_key(_omp_thread_data);
  double  (*array)[8000000] = &_OMP_VARREF(main_parallel_0,array);
  {
    int     i;
    double  GlobSum = 0;
    int     _omp_start, _omp_end, _omp_incr, _omp_last_iter = 0;
    int     _omp_for_id = _omp_module.for_ofs + 0;
    int     (*_omp_sched_bounds_func) (int, int, int, int,
                                int, int *, int *, int, int, int *);
    /* static with chunksize or runtime */
    int     _omp_init_start, _omp_nchunks, _omp_c = 0,
            _omp_chunksize;
    _omp_incr = (1);
    _omp_init_directive(_OMP_FOR, _omp_for_id, 0,
                        _omp_incr, 0, 115);
    _omp_sched_bounds_func = _omp_static_bounds;
    _omp_static_bounds_default(8000000, 0, _omp_incr,
                                  &_omp_start, &_omp_end);
    ...
```

OpenMP: Behind the scenes

```
    ...

    while ((*_omp_sched_bounds_func) (8000000, 0, _omp_for_id,
         _omp_incr, -1, &_omp_start, &_omp_end, 1, 0, &_omp_c)) {
      if (_omp_start < (8000000) && _omp_end == (8000000))
        _omp_last_iter = 1;

      for (i = _omp_start; i < _omp_end; i++) {
        GlobSum = GlobSum + (*(array))[i];
      }              /* for */
    }

    if (_omp_last_iter) { /* lastprivate assignments */ }

    /* reduction operation (+:GlobSum) */
    othread_set_lock(&_omp_module.reduction_lock[0]);
    _OMP_VARREF(main_parallel_0, GlobSum) += GlobSum;
    othread_unset_lock(&_omp_module.reduction_lock[0]);
  }
  return 0;
}
```

OpenMP vs POSIX threads

A possible POSIX threads solution:

```
main() {
  int i,retval;
  pthread_t tid;

  /* Initialize things */
  pthread_attr_init(&attr);
  pthread_mutex_init (&my_mutex, NULL);
  pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);

  for (i=0; i<MAX_SIZE; i++) array[i] = i;
  GlobSum = 0;

  for(i=0;i<ThreadCount;i++) {
    index[i] = i;
    retval = pthread_create(&tid,&attr,SumFunc,
                            (void *)index[i]);
    thread_id[i] = tid;
  }
  for(i=0;i<ThreadCount;i++)
    retval = pthread_join(thread_id[i],NULL);
}
```

OpenMP vs POSIX threads

```
void *SumFunc(void *parm){
  int i,me,chunk,start,end;
  double LocSum;

  /* Decide which iterations belong to me */
  me =  (int) parm;
  chunk = MAX_SIZE / ThreadCount;
  start = me * chunk;
  end = start + chunk; /* C-Style - actual element + 1 */
  if ( me == (ThreadCount-1) ) end = MAX_SIZE;

  /* Compute sum of our subset*/
  LocSum = 0;
  for(i=start;i<end;i++ ) LocSum = LocSum + array[i];

  /* Update the global sum and return */
  pthread_mutex_lock (&my_mutex);
  GlobSum = GlobSum + LocSum;
  pthread_mutex_unlock (&my_mutex);
}
```

Note: Variable definitions are omitted in this example!

OpenMP Summary

Short summary
of the three lectures

OpenMP Summary

- OpenMP: a parallel programming model for multi-core computers
- compiler directives, support functions, environment variables
- easy to implement, also “little by little”
- next lecture: “OpenMP & Performance”
  - special guest: Ruud van der Pas, Oracle

OpenMP References

- Useful Websites:
  - http://www.openmp.org/
  - check for webinars and tutorials
- Tutorial from LLNL:
  - https://hpc.llnl.gov/tuts/openMP/
- OpenMP specifications:
  - https://www.openmp.org/specifications/
  - C/C++ reference card for OpenMP 4.5
  - FORTRAN reference card for OpenMP 4.5