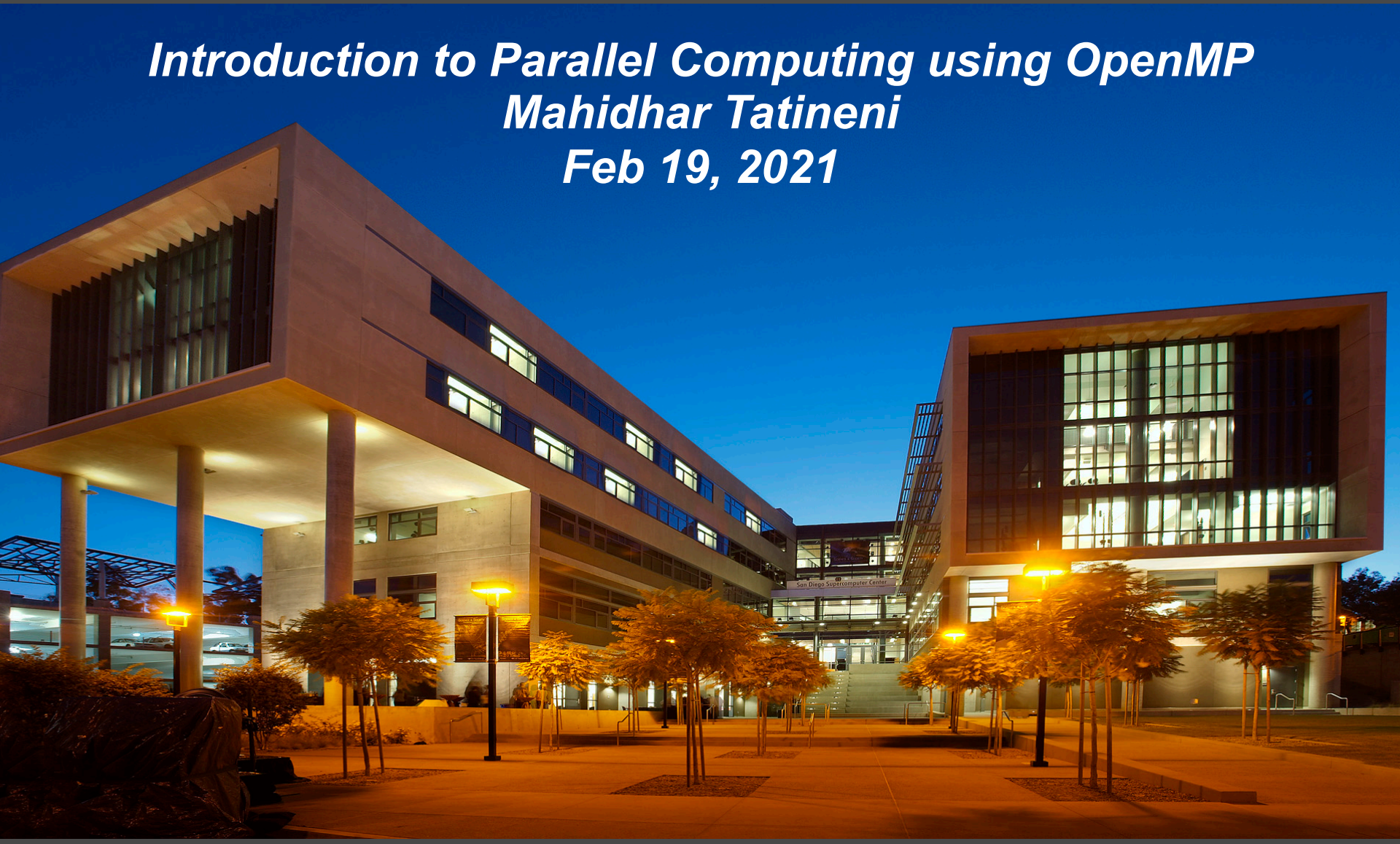
Introduction to Parallel Computing using OpenMP

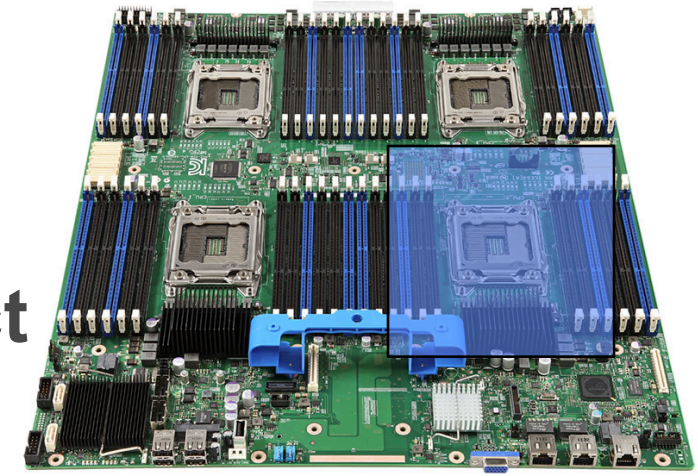
Mahidhar Tatineni

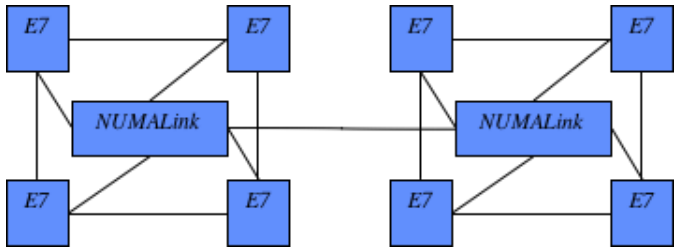
Feb 19, 2021

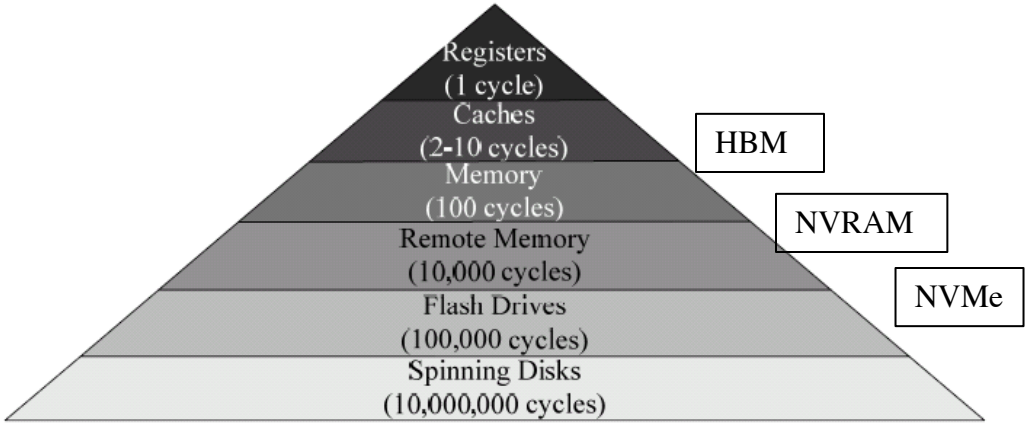


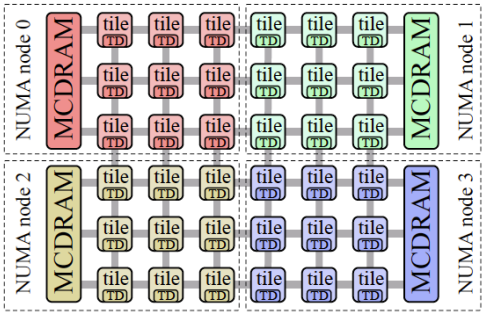
Current Supercomputer Architectures

- **Multi-socket server nodes**
  - NUMA
  - Accelerators
- **High performance interconnect**
  - e.g. Infiniband, OmniPath
- ***Scalable parallel approach needed to achieve performance***



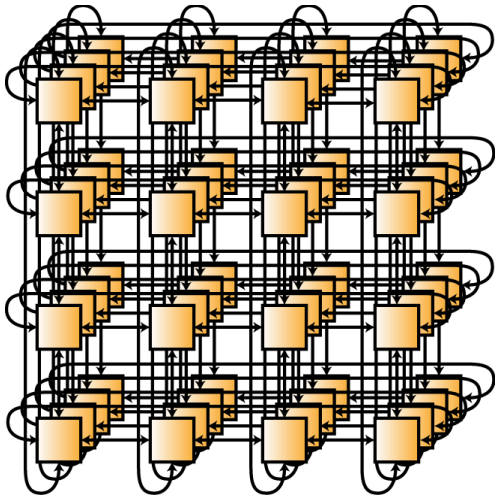


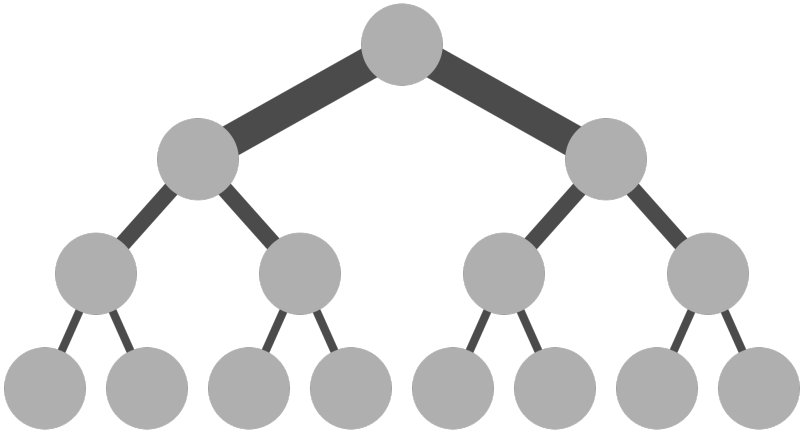




Network Topologies

- **Mesh, Torus, Hypercube**
- **Tree based**
  - Fat-tree
  - Clos
- **Dragonfly**
- **Metrics**
  - Bandwidth
  - Diameter, Connectivity
  - Bisection bandwidth





Parallel Computing

- **Executing instructions concurrently on physical resources (not time slicing)**
  - Multiple tightly coupled resources (e.g. cores) collaboratively solving a single problem
- **Benefits**
  - Capacity
    - Memory, storage
  - Performance
    - More instructions per unit of time (FLOPS)
    - Data streaming capability
- **Cost and Complexity**
  - Coordinate tasks and resources
  - Use resources efficiently

Memory, Communication, and Execution Models

- **Shared**
  - Communication model: shared memory
- **Distributed**
  - Communication model: exchange messages
- **Execution Models**
  - Fork-Join (e.g. Thread Level Parallelism)
- **Parallelism enabled by decomposing work**
  - Tasks can be executed concurrently
  - Some tasks can have dependencies

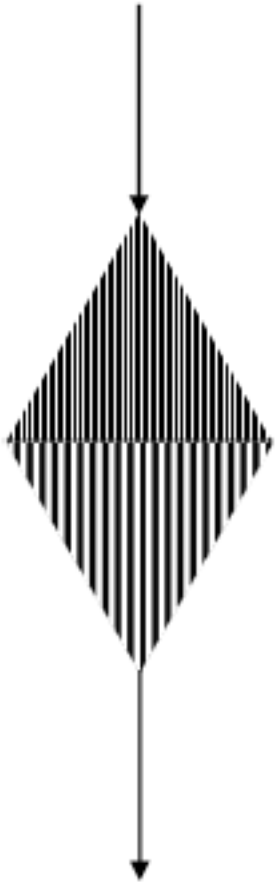
What is OpenMP?

- **High level parallelism abstraction based on thread**
  - Easy to use
  - Suitable for an incremental approach
- **A specification and evolving standard**
  - "a portable, scalable model … for developing portable parallel programs"
  - http://openmp.org
  - GNU, Intel, PGI, etc.
- **A set of**
  - Compiler directives
  - Library routines
  - Environment variables
  - Supports C/C++ and Fortran

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

OpenMP Models

- **Fork/Join Execution**
  - Process starts single threaded (master thread)
  - Forks child threads activated in parallel regions (team)
  - The team synchronizes and threads are disbanded
    - – barrier
  - Overhead is mitigated by reusing threads
  - Master thread continues execution of serial phases
- **Work decomposition**
  - Programming constructs
  - Scope and compound statements
  - Declarative in loops
  - Mapping to threads can be static or dynamic
  - Barriers and synchronization automatically inserted



Compiler Directives

- **Compiler directives are the main mechanisms for introducing parallelism. Functionality enabled includes:**
  - Spawning a parallel region
  - Diving code among threads
  - Distributing loop iterations over threads
  - Serialization of parts of the code
  - Synchronization of work
- **Example:**

```
#pragma omp parallel default(shared) private(beta,pi)
```

Parallel Region Construct

```
!$OMP PARALLEL [clause ...]
          IF (scalar_logical_expression)
          PRIVATE (list)
          SHARED (list)
          DEFAULT (PRIVATE | FIRSTPRIVATE | SHARED | NONE)
          FIRSTPRIVATE (list)
          REDUCTION (operator: list)
          COPYIN (list)
          NUM_THREADS (scalar-integer-expression)

  code block
!$OMP END PARALLEL
```

Data Scope Attribute Clauses

- PRIVATE – variables in the list are private to each thread.
- SHARED – variables in the list are shared between all threads.
- DEFAULT – default scope for all variables in a parallel region.
- FIRSTPRIVATE – variables are private and initialized according to value prior to entry into parallel or work sharing construct.
- LASTPRIVATE – variables are private, the value from the last iteration or section is copied to original variable object.
- Others – COPYIN, COPYPRIVATE
- REDUCTION – reduction on variables in the list

Simple C Example

```
#include <omp.h>

int main(void) {

   int number_of_threads, thread_id;

   #pragma omp parallel private(thread_id)
   {
      thread_id = omp_get_thread_num();

      #pragma omp master
      {
         number_of_threads = omp_get_num_threads();
      }

      #pragma omp barrier
      printf("hello, shared memory world from thread %d of \
         %d\n", thread_id, number_of_threads);
   }

   return 0;

}
```

Simple FORTRAN Example

```
      program hello_omp

      use, intrinsic :: iso_fortran_env
      implicit none

      integer :: number_of_threads, thread_id
      integer :: omp_get_num_threads, omp_get_thread_num

!$omp parallel private(thread_id)
      thread_id = omp_get_thread_num()
!$omp master
      number_of_threads = omp_get_num_threads()
!$omp end master
!$omp barrier
      write(unit=output_unit, fmt=*) "hello, shared memory&
         & world from thread ", thread_id, " of ",&
         & number_of_threads
!$omp end parallel

      stop

      end program hello_omp
```

Compiling on Expanse

- **Load a compiler module. For example:**

  module load gcc/10.2.0

- **Compile commands:**
  - Fortran: gfortran -fopenmp -o hello_fortran hello-omp.f90
  - C: gcc -fopenmp -o hello_c hello-omp.c

Sample job script

```
#!/bin/bash
#SBATCH --job-name="hello"
#SBATCH --output="hello.%j.%N.out"
#SBATCH --partition=shared
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH –account=XYZ123
#SBATCH --export=ALL
#SBATCH -t 00:10:00

module load gcc
#Export the number of OpenMP threads
export OMP_NUM_THREADS=8

#Run the openmp code
./hello_fortran
./hello_c
```

Work-Sharing

- **Schedule:**
  - Static – Loop iterations are statically divided (chunk or as close to even as possible)
  - Dynamic – Loop iterations are divided in size chunk, and dynamically scheduled among threads. When a thread finishes one chunk it is dynamically assigned another
  - Guided – Similar to dynamic but chunk size is proportionally reduced based on work remaining.
  - Runtime – set at runtime by environment variables
  - Auto – set by compiler or runtime system.

Number of Threads

- **Number of threads will be determined in the following order of precedence:**
  - Evaluation of the IF clause
  - Setting of NUM_THREADS clause
  - omp_set_num_threads() library function
  - OMP_NUM_THREADS environment variable
  - Default – usually ends up being the ***number of cores on the node*** (!)

- **The last factor can accidentally lead to oversubscription of nodes in hybrid MPI/OpenMP codes.**

Simple OpenMP Program – Compute PI

- Find the number of tasks and taskids (omp_get_num_threads, omp_get_thread_num)
- PI is calculated using an integral. The number of intervals used for the integration is fixed at 128000.
- Use OpenMP loop parallelization to divide up the compute work.
- Introduce concept of private and shared variables.
- OpenMP reduction operation used to compute the sum for the final integral.

OpenMP Program to Compute PI

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
{
int nthreads, tid;
int i, INTERVALS;
double n_1, x, pi = 0.0;

INTERVALS=128000;
/* Fork a team of threads giving them their own
      copies of variables */
#pragma omp parallel private(nthreads, tid)
  {
  /* Obtain thread number */
  tid = omp_get_thread_num();
  printf("Hello from thread = %d\n", tid);
```

```
 /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }

  }  /* All threads join master thread and disband */

  n_1 = 1.0 / (double)INTERVALS;

/* Parallel loop with reduction for calculating PI */
#pragma omp parallel for private(i,x)
shared(n_1,INTERVALS) reduction(+:pi)
  for (i = 0; i < INTERVALS; i++)
    {
    x = n_1 * ((double)i  - 0.5);
    pi += 4.0 / (1.0 + x * x);
    }
    pi *= n_1;
    printf ("Pi = %.12lf\n", pi);
}
```

OpenMP result: 1-D Heat Equation

$ sbatch -A use300 pi_openmp.sb

```
[mahidhar@login01 openmp_examples]$ more pi.1336540.exp-1-06.out
Hello from thread = 6
Hello from thread = 8
Hello from thread = 11
Threre are 16 threads!
Hello from thread = 4
Hello from thread = 2
Hello from thread = 1
Hello from thread = 0
Hello from thread = 15
Hello from thread = 5
Hello from thread = 13
Hello from thread = 14
Hello from thread = 9
Hello from thread = 3
Hello from thread = 7
Hello from thread = 12
Hello from thread = 10
PI = 3.139384000000, Err = 2.208654e-03, Time = 0.045669
[mahidhar@login01 openmp_examples]$
```

More Work-Share Constructs

- **SECTIONS** directive – enclosed sections are divided among the threads.
- **WORKSHARE** directive – divides execution of block into units of work, each of which is executed once.
- **SINGLE** directive – Enclosed code is executed by only one thread.

Example SECTIONS code

```
!$OMP PARALLEL SHARED(A,B,C,D), PRIVATE(I)

!$OMP SECTIONS

!$OMP SECTION
      DO I = 1, N
         C(I) = A(I) + B(I)
      ENDDO

!$OMP SECTION
      DO I = 1, N
         D(I) = A(I) * B(I)
      ENDDO

!$OMP END SECTIONS NOWAIT

!$OMP END PARALLEL
```

Synchronization Constructs

- **MASTER** directive – Specifies region is executed only by the master thread.
- **CRITICAL** directive – Region of the code that is executed one thread at a time.
- **BARRIER** directive – synchronize all threads
- **TASKWAIT** directive – wait for all child tasks to complete
- **ATOMIC** directive – specific memory location updated atomically (not let all threads write at the same time)

Simple Application using OpenMP: 1-D Heat Equation

- $\partial T/\partial t = \alpha(\partial^2 T/\partial x^2)$; $T(0) = 0$; $T(1) = 0$; $(0 \leq x \leq 1)$
  $T(x,0)$ is know as an initial condition.

- Discretizing for numerical solution we get:
  $T^{(n+1)}_i - T^{(n)}_i = (\alpha \Delta t/\Delta x^2)(T^{(n)}_{i-1} - 2T^{(n)}_i + T^{(n)}_{i+1})$
  (*n* is the index in time and *i* is the index in space)

Fortran OpenMP Code: 1-D Heat Equation

```
 PROGRAM HEATEQN
      implicit none
      integer :: iglobal, itime, nthreads
      real*8 :: xalp,delx,delt,pi
      real*8 :: T(0:100,0:10)
      integer:: id
      integer:: OMP_GET_THREAD_NUM,
        OMP_GET_NUM_THREADS

!$OMP PARALLEL SHARED(nthreads)
!$OMP MASTER
      nthreads = omp_get_num_threads()
      write (*,*) 'There are', nthreads, 'threads'
!$OMP END MASTER
!$OMP END PARALLEL
      if (nthreads.ne.3) then
        write(*,*)"Use exactly 3 threads for this case"
        stop
      endif
      delx = 0.1d0
      delt = 1d-4
      xalp = 2.0d0
```

```
********** Initial Conditions
*********************************
      pi = 4d0*datan(1d0)
      do iglobal = 0, 10
        T(0,iglobal) = dsin(pi*delx*dfloat(iglobal))
      enddo
********** Iterations
*****************************************
      do itime = 1 , 3
       write(*,*)"Running Iteration Number ", itime
!$OMP PARALLEL DO PRIVATE(iglobal)
SHARED(T,xalp,delx,delt,itime)
       do iglobal = 1, 9
        T(itime,iglobal)=T(itime-1,iglobal)+
     +  xalp*delt/delx/delx*
     +  (T(itime-1,iglobal-1)-2*T(itime-1,iglobal)+T(itime-
1,iglobal+1))
       enddo
!$OMP BARRIER
      enddo
      do iglobal = 0, 10
        write(*,*)iglobal,T(3, iglobal)
      enddo
      END
```

OpenMP result: 1-D Heat Equation

```
$ sbatch heat_openmp.sb
```

Sample output file:

```
There are        3 threads
Running Iteration Number        1
Running Iteration Number        2
Running Iteration Number        3
         0  0.000000000000000E+000
         1  0.307205621017285
         2  0.584339815421976
         3  0.804274757358271
         4  0.945481682332598
         5  0.994138272681972
         6  0.945481682332598
         7  0.804274757358271
         8  0.584339815421977
         9  0.307205621017285
        10  7.797843424221369E-316
```

Run Time Library Routines

- Setting and querying number of threads
- Querying thread identifier, team size
- Setting and querying dynamic threads feature
- Querying if in parallel region and at what level
- Setting and querying nested parallelism
- Setting, initializing and terminating locks, nested locks.
- Querying wall clock time and resolution.

Environment Variables

- OMP_SCHEDULE – e.g set to “dynamic”
- OMP_NUM_THREADS
- OMP_DYNAMIC (TRUE or FALSE)
- OMP_PROC_BIND (TRUE or FALSE)
- OMP_NESTED (TRUE of FALSE)
- OMP_STACKSIZE – size of stack for created threads
- OMP_THREAD_LIMIT

General OpenMP Performance Considerations

- Avoid or minimize use of BARRIER, CRITICAL (complete serialization here!), ORDERED regions, and locks. Can use NOWAIT clause to avoid redundant barriers.

- Parallelize at a high level, i.e. maximize the work in the parallel regions to reduce parallelization overhead.

- Use appropriate loop scheduling – static has low synchronization overhead but can be unbalanced, dynamic (and guided) have higher synchronization overheads but can improve load balancing.

- Avoid false sharing (more about it in following slide)!

What is False Sharing?

- Most modern processors have a cache buffer between slow memory and high speed registers of the CPU.
- Accessing a memory location causes a “cache line” to be copied into the cache.
- In an OpenMP code two processors may be accessing two different elements in the same cache line. On writes this will lead to “cache line” being marked invalid (because cache coherency is being maintained).
- This will lead to an increase in memory traffic even though the write is to different elements (hence the term **false sharing**).
- This can have a drastic performance impact if such updates are occurring frequently in a loop.

False Sharing Example

Code snippet:

```
double global=0.0, local[NUM_THREADS];
#pragma omp parallel num_threads(NUM_THREADS)
{
 int tid = omp_get_thread_num();
 local[tid] = 0.0;
 #pragma omp for
 for (i = 0; i < N; i++)
 local[tid] += x[i];
 #pragma omp atomic
 global += local[me];
}
```

False Sharing - Solutions

- **Three options**
  - Compiler directives to align individual variables on cache line boundaries
  - Pad arrays/data structures to make sure array elements begin on cache line boundary.
  - Use thread local copies of data (assuming the copy overhead is small compared to overall run time).

Homework!

- **Download matrix multiply example from LLNL site:**
  - https://computing.llnl.gov/tutorials/openMP/samples/Fortran/omp_mm.f

  (wget https://computing.llnl.gov/tutorials/openMP/samples/Fortran/omp_mm.f on Expanse)
- **Compile (ifort -fopenmp omp_mm.f) and run the example. See if you can vary the environmental variables, scheduling to get better performance!**
- **This is very quick intro. Lot of ongoing developments. Detailed specifications at:**
  - https://www.openmp.org/specifications/
- **Excellent tutorials from LLNL:**
  - https://computing.llnl.gov/tutorials/openMP/