Outline

• Introduction
  – Share & Distributed Memory Programming
  – MPI & OpenMP Advantages/Disadvantages
  – MPI vs. OpenMP
  – Why Mixed/Hybrid Programming?
  – MPI Calls
  – Limitations and Problems

• Mixed/Hybrid Programming Model

• Labs: Examples
Shared & Distributed Programming

- **Shared Memory (thread based)**
  - Multiple threads sharing data in the same address space and explicitly synchronizing when needed

- **Distributed Memory (process based)**
  - Distinct processes, explicitly partaking in the pairwise and collective exchange of control and data messages
  - No way to directly access the variables in the memory of another process
MPI Case

• Advantages
  – Portable to distributed and shared memory machines
  – Scales beyond one node
  – No data placement problem

• Disadvantages
  – Difficult to develop and debug
  – High latency, low bandwidth
  – Explicit communication
  – Large granularity
  – Difficult load balancing (dynamic/static)
OpenMP Case

• Advantages
  – Easy to implement parallelism
  – Low latency, high bandwidth
  – Implicit Communication
  – Coarse and fine granularity
  – Dynamic load balancing

• Disadvantages
  – Only on shared memory machines
  – Scale within one node
  – Possible data placement problem
  – No specific thread order
MPI vs. OpenMP Case

- MPI and OpenMP are both suitable for coarse grain parallelism (multiple asynchronous processor)
- OpenMP can be effectively used for fine grain parallelism (vectorization)
- Both MPI and OpenMP can be used to parallelize applications in data parallel and task parallel fashion
- Even though OpenMP is based upon sharing out work, it is possible to assign data to individual threads
- With some care when assigning work, a data parallel mode can be closely approximated in OpenMP
Why Hybrid Programming?

- Hybrid model is an excellent match for the dominant trend in parallel architectures which are made of clusters of multi-core shared memory or SMP (Symmetric Multi-Processor) nodes. Such as quad or dual core Xeon Processor
- Avoid extra communication overhead with MPI within node
- Could have better scalability both pure MPI and OpenMP
Why Hybrid Programming?

- ANADOLU, KARADENIZ, EGE, ...
  - Intel Xeon EM64T arch.
  - dual & quad core nodes (as smp)
  - Front Side Bus!

Hybrid Programming: OpenMP + MPI & Apps.
Hybrid Programming

- Multiple OpenMP threads under each MPI process
  - OpenMP threads can be used within each shared memory node and MPI can be used to communicate across nodes
  - Eliminates message passing within a single shared memory node
  - Nested parallelism is possible in a hybrid model
  - Right approach for DSM architectures comprising of a large number of shared memory SMP nodes
### Hybrid Programming Models

<table>
<thead>
<tr>
<th></th>
<th>MPI</th>
<th>OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid (MPI+OpenMP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>No overlapping of communication and Computation</td>
<td>Overlapping of communication and Computation</td>
</tr>
<tr>
<td></td>
<td>Masteronly (MPI only outside of parallel region)</td>
<td>Funneled (MPI only on master thread)</td>
</tr>
<tr>
<td></td>
<td>Multiple/only (appl. threads inside of MPI)</td>
<td>Multiple (more than one thread may comminicate)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Funneled with Full Load Balancing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple with Full Load Balancing</td>
</tr>
</tbody>
</table>

**Hybrid PI Application**

http://www.compmunity.org/events/ewomp03/omptalks/Tuesday/Session7/T01p.pdf
MPI Routines for Threads

• MPI_INIT_THREAD
  – It allows to request a level of thread support
  – REQUIRED argument can be one of the MPI_THREAD_[SINGLE | FUNNELED | SERIALIZE | MULTIPLE]
  – Returned PROVIDED argument may be less than REQUIRED by the application

C

int MPI_Init_thread(int *argc, char **argv,
                     int required, int *provided);

Fortran

MPI_INIT_THREAD(INTEGER REQUIRED, INTEGER PROVIDED, INTEGER IERROR)
MPI Routines for Threads

- **MPI_INIT_THREAD**
  - The simplest OpenMP+MPI model is MPI_THREAD_SINGLE
  - MPI_INIT could be equal to MPI_INIT_THREAD with required MPI_THREAD_SINGLE in MPP systems and MPI_THREAD_FUNNELED in hybrid systems (NEC SX-5, SX-6, SX-8)
MPI Routines for Threads

• MPI_INIT_THREAD
  – **MPI_THREAD_SINGLE**: There is only one thread in application. There is no OpenMP multithreading in program.
  – **MPI_THREAD_FUNNELED**: There is only one thread that makes MPI calls. All of the MPI calls are made by the master thread. This will happen if all MPI calls are outside OpenMP parallel regions or are in master regions.

Hybrid Programming: OpenMP + MPI & Apps.
MPI Routines for Threads

• MPI_INIT_THREAD
  – **MPI_THREAD_SERIALIZED**: Multiple threads make MPI calls, but only one at a time. This can be enforced in OpenMP by OpenMP SINGLE directive.
  – **MPI_THREAD_MULTIPLE**: Any thread may make MPI calls at any time.
  – All MPI implementations of course support MPI THREAD SINGLE.

MPI Routines for Threads

(Strategy: In a threaded barrier, master process synchronizes processes)

• MPI Calls inside OMP_MASTER
  – MPI_THREAD_FUNNELED required
  – Support also MPI outside parallel region
  – OMP_BARRIER is needed since there is no synchronization with OMP_MASTER
  – It implies all other threads are sleeping!
MPI Routines for Threads

- MPI Calls inside OMP_MASTER ...

```c
 !$OMP PARALLEL
 !$OMP DO
   do i=1,1000
     a(i) = buf(i)
   end do
 !$OMP END DO NOWAIT
 !$OMP BARRIER
 !$OMP MASTER
   call MPI_RECV(buf,...)
 !$OMP END MASTER
 !$OMP BARRIER
 !$OMP DO
   do i=1,1000
     c(i) = buf(i)
   end do
 !$OMP END DO NOWAIT
 !$OMP END PARALLEL
```

```c
 #pragma parallel
 {
   #pragma for nowait
   for (i=0; i<1000; i++)
     a[i] = buf[i];
   
   #pragma omp barrier
   #pragma omp master
     MPI_Recv(buf,...);
   #pragma omp barrier
   
   #pragma for nowait
   for (i=0; i<1000; i++)
     c[i] = buf[i];
 }
 #pragma end parallel
```

http://www.hlrs.de/organization/amt/services/models/openmp
MPI Routines for Threads

• MPI Calls inside OMP_SINGLE | MASTER
  – MPI_THREAD_SERIALIZED required
  – OMP_BARRIER is needed since OMP_SERIAL only guaranties synchronization at the end
  – It also implies all other threads are sleeping!

```
!$OMP BARRIER
!$OMP SINGLE
  call MPI_xxx(...)  
!$OMP END SINGLE
```
• Overlap communication and computation
  – At least **MPI_THREAD_FUNNELED** required. While master or single thread is making MPI calls, other threads are computing

```
!$OMP PARALLEL
    if (my_thread_rank < 1) then
        call MPI_xxx(...)
    else
        do some computation
    endif
!$OMP END PARALLEL
```
MPI Routines for Threads

• Overlap communication and computation
  – At least MPI_THREAD_FUNNELED required. While master or single thread is making MPI calls, other threads are computing

```c
!$OMP PARALLEL
  if (my_thread_rank < 1) then
    call MPI_xxx(...)
  else
    do some computation
  endif
!$OMP END PARALLEL
```
Advantages & Disadvantages

- THREAD_SINGLE
  - Easy to program
  - other threads are sleeping while master thread calls MPI routines

- THREAD_FUNNELED
  - load balance is necessary
  - useful for dynamic task distribution and problematic for domain decomposition programming
Advantages & Disadvantages

• Results
  – There is no optimal OpenMP+MPI model
    • depends on your application needs for communication
    • capability for SMP parallelization
    • your available working hours for hybrid programming

• Is the MPI Library thread safe?

• From which code line I can call MPI routines?

• In which code block I can use parallel regions?
Thread Info for MPI Process

- **MPI_QUERY_THREAD**
  - Returns provided level of thread support
  - Thread level may be set via environment variable!

- **MPI_IS_THREAD_MAIN**
  - Returns true if this is the thread that invoked MPI_INIT or MPI_INIT_THREAD
  - Indicates that which thread is master
Limitations & Problems

- OpenMP has less scalability due to implicit parallelism while MPI allows multi-dimensional blocking.
- All threads are idle except one while MPI communication
  - Need to overlap computation and communication for better performance.
  - Critical Section necessary for shared variables.
Limitations & Problems

• Cache coherence, data placement
• Natural one level parallelism problems.
• Pure OpenMP code performs worse than pure MPI within node
• Lack of optimized OpenMP compilers and libraries
• Not necessarily faster than pure MPI or pure OpenMP
Limitations & Problems

- Partly depends on code, architecture, partly on how the programming models interact
  - How many times is a new team of threads created?
  - Does machine/OS put new threads where they best reuse data in cache?
  - How much of the MPI code is multithreaded?
  - How well does the code use the comm. hardware?
• PI Calculation
  - To demonstrate the bottom to top approach in MPI/OpenMP program development, we provide a simple program for the calculation of the π number using an easily parallelizable trapezoidal rule integral evaluation
    - MPI (mpi_pi.c) / OpenMP (omp_pi.c) and Hybrid (hybrid_pi.c) versions
Lab: Example 1

- **INTEL (MPI)**

  **Set environment**

  ```
  source /RS/progs/intel/ict/3.0.1/mpi/3.0/bin64/mpivars.sh
  ```

  **Compile code**

  ```
  mpiicc mpi_pi.c -o mpi_pi.x
  ```

  **Run code**

  ```
  mpirun -np 8 ./mpi_pi.x
  ```
Hybrid Programming: OpenMP + MPI & Apps.

Lab: Example 1

- INTEL (OpenMP)

  Set environment

  ```bash
  export OMP_NUM_THREADS=8
  ```

  Compile code

  ```bash
 icc -openmp omp_pi.c -o omp_pi.x
  ```

  Run code

  ```bash
  ./omp_pi.x
  ```
INTEL (OpenMP + MPI)

Set environment

```
export OMP_NUM_THREADS=4
source /RS/progs/intel/mpi/3.1/bin64/mpivars.sh
```

Compile code

```
mpiicc -openmp hybrid_pi.c -o hybrid_pi.x
```

Run code

```
mpirun -np 2 hybrid_pi.x
```
Lab: Example 2

- 2D Heat Equation Calculation
  - Write Hybrid MPI + OpenMP program using the given Heat2D MPI program
  - Write correct OpenMP pragma lines
  - Write MPI Thread Strategy with MPI_THREAD_FUNNELED
  - Write Hybrid code with at least one of the MPI_INIT_THREAD
    ✓ MULTIPLE
    ✓ SERIALIZE