

# Hybrid MPI and OpenMP Parallel Programming

Jemmy Hu

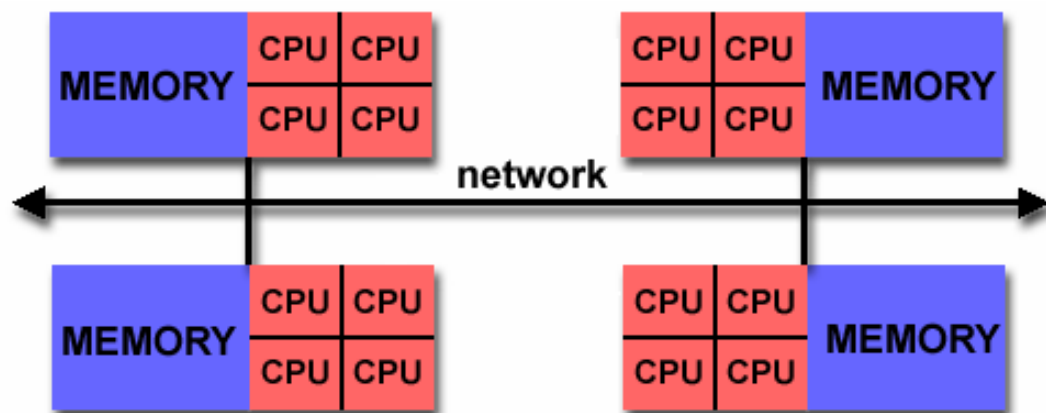
SHARCNET HPTC Consultant

July 8, 2015

# Objectives

- difference between message passing and shared memory models (MPI, OpenMP)
- why or why not hybrid?
- a common model for utilizing both MPI and OpenMP approaches to parallel programming
- example hybrid code
- compile and execute hybrid code on SHARCNET clusters

# Hybrid Distributed-Shared Memory Architecture



- Employ both shared and distributed memory architectures
- The shared memory component is usually a cache coherent SMP node. Processors on a given SMP node can address that node's memory as global.
- The distributed memory component is the networking of multiple SMP nodes. SMPs know only about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another.
- Current trends seem to indicate that this type of memory architecture will continue to prevail: more cpus per SMP node, less memory or bandwidth ratio per cpu.

# MPI

- standard for distributed memory communications
- provides an explicit means to use message passing on distributed memory clusters
- specializes in packing and sending complex data structures over the network
- data goes to the process
- synchronization must be handled explicitly due to the nature of distributed memory

# OpenMP

- a shared memory paradigm, implicit intra-node communication
- efficient utilization of shared memory SMP systems
- easy threaded programming, supported by most major compilers
- the process goes to the data, communication among threads is implicit

# MPI vs. OpenMP

## – Pure MPI Pros:

- Portable to distributed and shared memory machines.
- Scales beyond one node
- No data placement problem

## – Pure MPI Cons:

- Explicit communication
- High latency, low bandwidth
- Difficult load balancing

## – Pure OpenMP Pros:

- Easy to implement parallelism
- Implicit Communication
- Low latency, high bandwidth
- Dynamic load balancing

## – Pure OpenMP Cons:

- Only on shared memory nodes/machines
- Scale within one node
- data placement problem

# Why Hybrid: employ the best from both worlds

- MPI makes inter-node communication relatively easy
- MPI facilitates efficient inter-node scatters, reductions, and sending of complex data structures
- Since program state synchronization is done explicitly with messages, correctness issues are relatively easy to avoid
- OpenMP allows for high performance, and relatively straightforward, intra-node threading
- OpenMP provides an interface for the concurrent utilization of each SMP's shared memory, which is much more efficient than using message passing
- Program state synchronization is implicit on each SMP node, which eliminates much of the overhead associated with message passing

## Overall Goal:

to reduce communication needs and memory consumption, or improve load balance

# Why not Hybrid?

- OpenMP code performs worse than pure MPI code within node
  - all threads are idle except one while MPI communication
  - data placement, cache coherence
  - critical section for shared variables
- Possible waste of effort



# A Common Hybrid Approach

- From sequential code, parallel with MPI first, then try to add OpenMP.
- From MPI code, add OpenMP
- From OpenMP code, treat as serial code.
- Simplest and least error-prone way is to use MPI outside parallel region, and allow only master thread to communicate between MPI tasks.
- Could use MPI inside parallel region with thread-safe MPI.

# Hybrid – Program Model

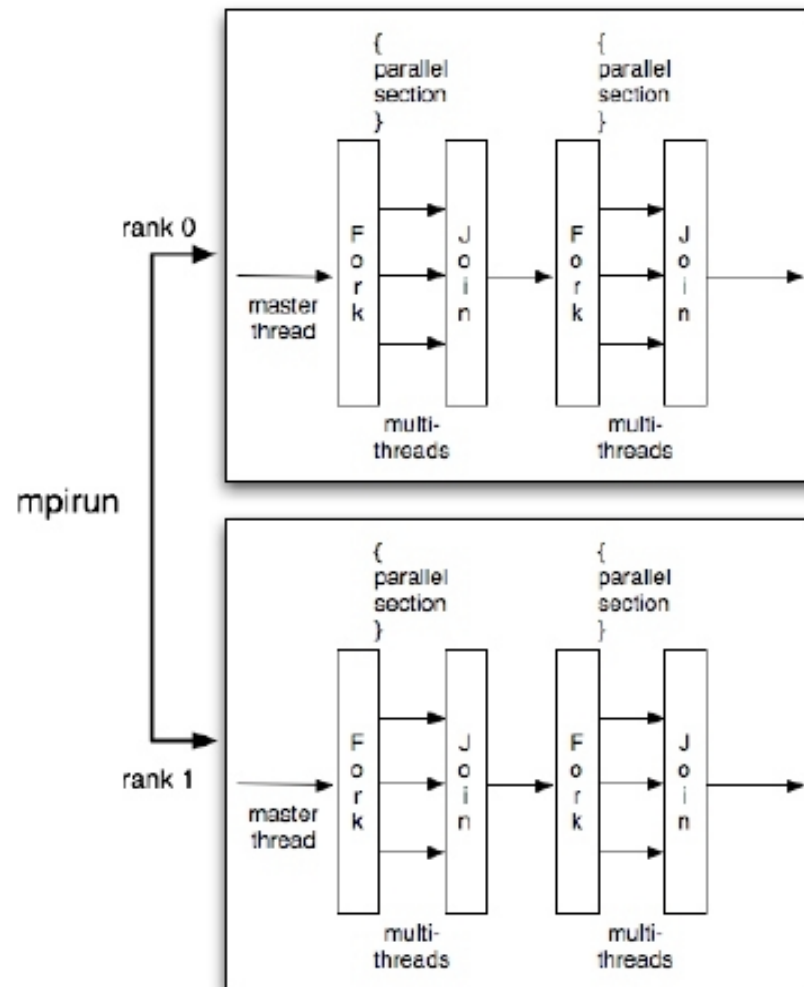
- Start with MPI initialization
- Create OMP parallel regions within MPI task (process).
  - Serial regions are the master thread or MPI task.
  - MPI rank is known to all threads
- Call MPI library in serial and parallel regions.
- Finalize MPI

## Program hybrid

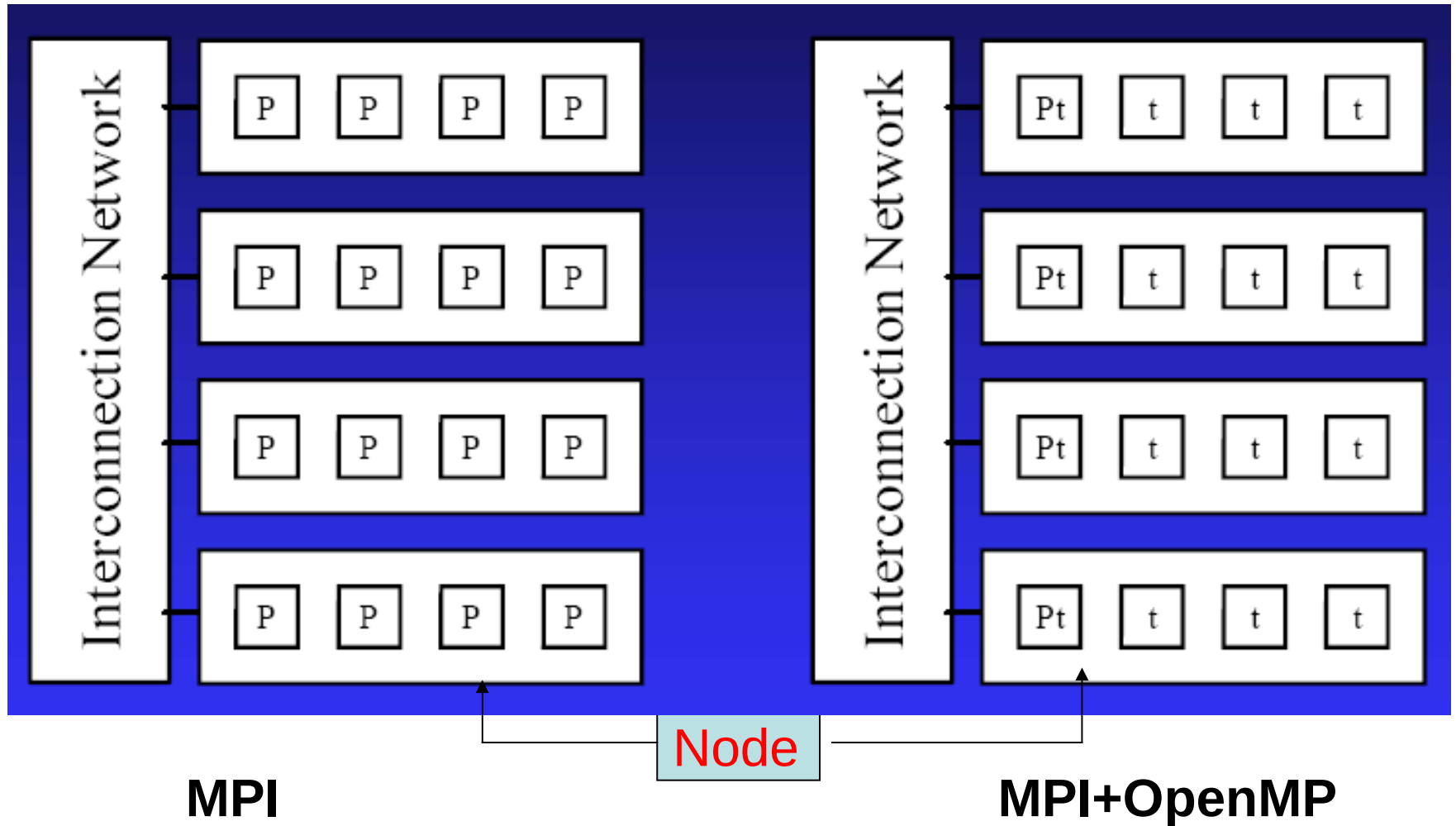
```
call MPI_INIT (ierr)
call MPI_COMM_RANK (...)
call MPI_COMM_SIZE (...)
... some computation and MPI
communication
... start OpenMP within node
!$OMP PARALLEL DO PRIVATE(i)
!$OMP&                               SHARED(n)
  do i=1,n
    ... computation
  enddo
!$OMP END PARALLEL DO
... some computation and MPI
communication
call MPI_FINALIZE (ierr)
end
```

# Hybrid MPI+OpenMP Programming

Each MPI process spawns multiple OpenMP threads



# MPI vs. MPI+OpenMP



16 cpus across 4 nodes  
16 MPI processes

16 cpus across 4 nodes  
1 MPI process and 4 threads per node

# Example: Calculating $\pi$

- Numerical integration

$$\int_0^1 \frac{4}{1+x^2} dx = \pi$$

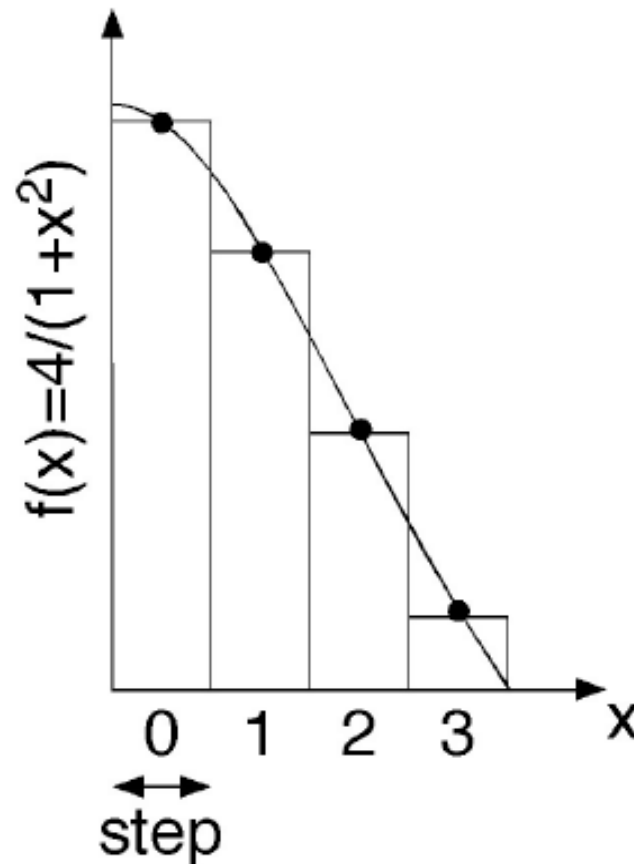
- Discretization:

$$\Delta = 1/N: \text{step} = 1/\text{NBIN}$$

$$x_i = (i+0.5)\Delta \quad (i = 0, \dots, N-1)$$

$$\sum_{i=0}^{N-1} \frac{4}{1+x_i^2} \Delta \cong \pi$$

```
#include <stdio.h>
#define NBIN 100000
void main() {
    int i; double step,x,sum=0.0,pi;
    step = 1.0/NBIN;
    for (i=0; i<NBIN; i++) {
        x = (i+0.5)*step;
        sum += 4.0/(1.0+x*x);}
    pi = sum*step;
    printf("PI = %f\n",pi);
}
```



# pi – MPI version

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>          /* MPI header file */
#define NUM_STEPS 10000000

int main(int argc, char *argv[]) {
    int nprocs;
    int myid;
    double start_time, end_time;
    int i;
    double x, pi;
    double sum = 0.0;
    double step = 1.0/(double) NUM_STEPS;

    /* initialize for MPI */
    MPI_Init(&argc, &argv);    /* starts MPI */
    /* get number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    /* get this process's number (ranges from 0 to nprocs - 1) */
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

```
/* do computation */
for (i=myid; i < NUM_STEPS; i += nprocs) {    /* changed */
    x = (i+0.5)*step;
    sum = sum + 4.0/(1.0+x*x);
}
sum = step * sum;                            /* changed */
MPI_Reduce(&sum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);/* added */

/* print results */
if (myid == 0) {
    printf("parallel program results with %d processes:\n", nprocs);
    printf("pi = %g (%17.15f)\n", pi, pi);
}

/* clean up for MPI */
MPI_Finalize();

return 0;
}
```

# OpenMP, reduction clause

```
#include <stdio.h>
#include <omp.h>
#define NBIN 100000
int main(int argc, char *argv[ ]) {
    int l, nthreads;
    double x, pi;
    double sum = 0.0;
    double step = 1.0/(double) NUM_STEPS;

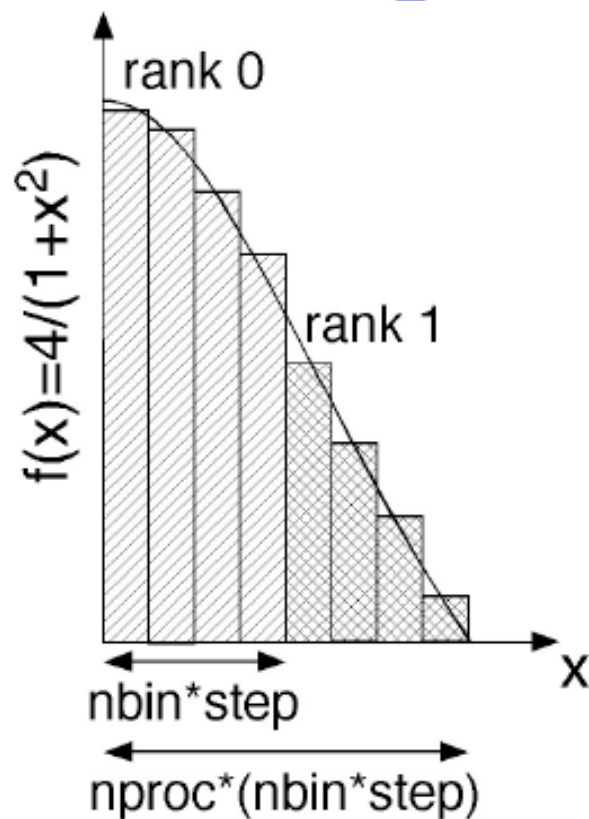
    /* do computation -- using all available threads */
    #pragma omp parallel
    {
        #pragma omp for private(x) reduction(+:sum) schedule(runtime)
        for (i=0; i < NUM_STEPS; ++i) {
            x = (i+0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
        }
        #pragma omp master
        {
            pi = step * sum;
        }
    }
    printf("PI = %f\n",pi);
}
```



# MPI+OpenMP Calculation of $\pi$

---

- Each MPI process integrates over a range of width  $1/nproc$ , as a discrete sum of  $nbin$  bins each of width  $step$
- Within each MPI process,  $nthreads$  OpenMP threads perform part of the sum as in `omp_pi.c`



# MPI\_OpenMP version

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>          /* MPI header file */
#include <omp.h>          /* OpenMP header file */
#define NUM_STEPS 100000000
#define MAX_THREADS 4

int main(int argc, char *argv[]) {
    int nprocs, myid;
    int tid, nthreads, nbin;
    double start_time, end_time;
    double pi, Psum=0.0, sum[MAX_THREADS]={0.0};
    double step = 1.0/(double) NUM_STEPS;

    /* initialize for MPI */
    MPI_Init(&argc, &argv);    /* starts MPI */
    /* get number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    /* get this process's number (ranges from 0 to nprocs - 1) */
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);

    nbin= NUM_STEPS/nprocs;
```

```
#pragma omp parallel private(tid)
```

```
{
```

```
    int i;
```

```
    double x;
```

```
    nthreads=omp_get_num_threads();
```

```
    tid=omp_get_thread_num();
```

```
    for (i=nbin*myid+tid; i < nbin*(myid+1); i+= nthreads) { /* changed*/
```

```
        x = (i+0.5)*step;
```

```
        sum[tid] += 4.0/(1.0+x*x);
```

```
    }
```

```
}
```

```
for(tid=0; tid<nthreads; tid++) /*sum by each mpi process*/
```

```
    Psum += sum[tid]*step;
```

```
MPI_Reduce(&Psum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);/* added */
```

```
if (myid == 0) {
```

```
    printf("parallel program results with %d processes:\n", nprocs);
```

```
    printf("pi = %g (%17.15f)\n",pi, pi);
```

```
}
```

```
MPI_Finalize();
```

```
return 0;
```

```
}
```

# Compile and Run

- Compile (default intel compilers on SHARCNT systems)

```
mpicc -o pi-mpi pi-mpi.c
```

```
cc -openmp -o pi-omp pi-omp.c
```

```
mpicc -openmp -o pi-hybrid pi-hybrid.c
```

- Run (sqsub)

```
sqsub -q mpi -n 8 --ppn=4 -r 10m -o pi-mpi.log ./pi-mpi
```

```
sqsub -q threaded -n 8 -r 10m -o pi-omp.log ./pi-omp
```

```
sqsub -q mpi -n 8 --ppn=1 --tpp=4 -r 10m -o pi-hybrid.log ./pi-hybrid
```

Example codes and results are in:

</home/jemmyhu/CES706/Hybrid/pi/>

# Results

- MPI

MPI uses 8 processes:

pi = 3.14159 (3.141592653589828)

- OpenMP

OpenMP uses 8 threads:

pi = 3.14159 (3.141592653589882)

- Hybrid

mpi process 0 uses 4 threads

mpi process 1 uses 4 threads

mpi process 1 sum is 1.287 (1.287002217586605)

mpi process 0 sum is 1.85459 (1.854590436003132)

Total MPI processes are 2

pi = 3.14159 (3.141592653589738)

# Summary

- Computer systems in High-performance computing (HPC) feature a hierarchical hardware design (multi-core nodes connected via a network)
- OpenMP can take advantage of shared memory to reduce communication overhead
- Pure OpenMP performs better than pure MPI within node is a necessity to have hybrid code better than pure MPI across node.
- Whether the hybrid code performs better than MPI code depends on whether the communication advantage outcomes the thread overhead, etc. or not.
- There are more positive experiences of developing hybrid MPI/OpenMP parallel paradigms now. It's encouraging to adopt hybrid paradigm in your own application.

# References

- [http://openmp.org/sc13/HybridPP\\_Slides.pdf](http://openmp.org/sc13/HybridPP_Slides.pdf)
- <https://www.cct.lsu.edu/~estrabd/intro-hybrid-mpi-openmp.pdf>
- [http://www.cac.cornell.edu/education/Training/parallelMay2011/Hybrid\\_Talk-110524.pdf](http://www.cac.cornell.edu/education/Training/parallelMay2011/Hybrid_Talk-110524.pdf)