MT-MPI: Multithreaded MPI for Many-Core Environments

Min Si$^{1,2}$ Antonio J. Peña$^2$ Pavan Balaji$^2$
Masamichi Takagi$^3$ Yutaka Ishikawa$^1$

$^1$ University of Tokyo
$^2$ Argonne National Laboratory
$^3$ NEC Corporation
Presentation Overview

• Background and Motivation
• Proposal and Challenges
• Design and Implementation
  – OpenMP Runtime Extension
  – MPI Internal Parallelism
• Evaluation
• Conclusion
Many-core Architectures

• Massively parallel environment

• Intel® Xeon Phi co-processor
  – 60 cores inside a single chip, 240 hardware threads
  – SELF-HOSTING in next generation, NATIVE mode in current version

• Blue Gene/Q
  – 16 cores per node, 64 hardware threads

• Lots of “light-weight” cores is becoming a common model
MPI programming on Many-Core Architectures

**Thread Single mode**

```c
/* user computation */
MPI_Function();
/* user computation */
```

**Funneled / Serialized mode**

```c
#pragma omp parallel
{ /* user computation */ }
MPI_Function();
#pragma omp parallel
{ /* user computation */ }
```

**Multithreading mode**

```c
#pragma omp parallel
{ /* user computation */
  MPI_Function();
  /* user computation */
}
```
Funneled / Serialized mode

- Multiple threads are created for user computation
- Single thread issues MPI

```c
#pragma omp parallel
{ /* user computation */ }
MPI_Function();
```

1. Many threads are IDLE!
2. Single lightweight core delivers poor performance
Our Approach

• Sharing Idle Threads with Application inside MPI

```c
#pragma omp parallel
{ /* user computation */ }

MPI_Function()
{
    #pragma omp parallel
    {
        /* MPI internal task */
    }
}
#pragma omp parallel
{ /* user computation */ }
```
Challenges (1/2)

• Some parallel algorithms are not efficient with insufficient threads, need tradeoff

• But the number of available threads is **UNKNOWN**!
Challenges (2/2)

• Nested parallelism
  – Simply creates new Pthreads, and offloads thread scheduling to OS,
  – Causes threads OVERRUNNING issue

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

Creates $N$ Pthreads!

ONLY use IDLE threads
Design and Implementation

- OpenMP Runtime Extension
- MPI Internal Parallelism
**Guaranteed Idle Threads VS Temporarily Idle Threads**

- **Guaranteed Idle Threads**
  - Guaranteed idle until Current thread exits

- **Temporarily Idle Threads**
  - Current thread does not know when it may become active again

---

**Example 1**

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

**Example 2**

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

**Example 3**

```c
#pragma omp parallel
{
  #pragma omp single nowait
  {...
  }
  #pragma omp critical
  {...
  }
}
```
Expose Guaranteed Idle Threads

- MPI uses Guaranteed Idle Threads to schedule its internal parallelism efficiently (i.e. change algorithm, specify number of threads)

```c
#pragma omp parallel
#pragma omp single
{
    MPI_Function {
        num_idle_threads = omp_get_num_guaranteed_idle_threads();
        if (num_idle_threads < N) {
            /* Sequential algorithm */
        } else {
            #pragma omp parallel num_threads(num_idle_threads)
            { ... }
        }
    }
}
```
Design and Implementation

- OpenMP Runtime Extension
- **MPI Internal Parallelism**

1. Derived Datatype Related Functions
2. Shared Memory Communication
3. Network-specific Optimizations

*Implementation is based on MPICH v3.0.4*
1. Derived Datatype Packing Processing

- MPI_Pack / MPI_Unpack
- Communication using Derived Datatype
  - Transfer non-contiguous data
  - Pack / unpack data internally

```
#pragma omp parallel for
for (i=0; i<count; i++)
{
  dest[i] = src[i * stride];
}
```
2. Shared Memory Communication

- Original sequential algorithm
  - Shared user space buffer between processes
  - Pipelining copy on both sender side and receiver side

- Parallel algorithm
  - Get as many available cells as we can
  - Parallelize large data movement

(a) Sequential Pipelining

(b) Parallel pipelining
Sequential Pipelining VS Parallelism

• Small Data transferring ( < 128K )
  – Threads synchronization overhead > parallel improvement

• Large Data transferring
  – Data transferred using Sequential Fine-Grained Pipelining
  – Data transferred using Parallelism with only a few of threads (worse)
  – Data transferred using Parallelism with many threads (better)
3. InfiniBand Communication

- **Structures**
  - IB context
  - Protection Domain
  - **Queue Pair** (*critical*)
    - 1 QP per connection
  - **Completion Queue** (*critical*)
    - Shared by 1 or more QPs

- **RDMA communication**
  - Post RDMA operation to QP
  - Poll completion from CQ

- **OpenMP contention issue**
Evaluation

1. Derived Datatype Related Functions
2. Shared Memory Communication
3. Network-specific Optimizations

All our experiments are executed on the Stampede supercomputer at the Texas Advanced Computing Center (https://www.tacc.utexas.edu/stampede/).
Derived Datatype Packing

Parallel packing 3D matrix of double

**Packing the X-Z plane with varying Z**

- Speedup vs. Number of Threads
- Graph Data:
  - Fixed matrix volume 1 GB
  - Fixed length of Y: 2 doubles
  - Length of Z: graph legend

**Packing the Y-Z plane with varying Y**

- Speedup vs. Number of Threads
- Graph Data:
  - Fixed matrix volume 1 GB
  - Fixed length of X: 2 doubles
  - Length of Y: graph legend
3D internode halo exchange using 64 MPI processes

Not strong scaling
BUT we are using **IDLE RESOURCES**!
Hybrid MPI+OpenMP NAS Parallel MG benchmark

V-cycle multi-grid algorithm to solve a 3D discrete Poisson equation.

Graph Data:
Class E using 64 MPI processes

Halo exchanges with various dimension sizes from 2 to 514 doubles in class E with 64 MPI processes
Shared Memory Communication

• OSU MPI micro-benchmark

Caused by poor sequential performance due to too small Eager/Rendezvous communication threshold on Xeon Phi. Not by MT-MPI!

Poor pipelining but worse parallelism
One-sided Operations and IB netmod Optimization

• Micro benchmark
  – One to All experiment using 65 processes
    • root sends many MPI_PUT operations to all the other 64 processes (64 IB QPs)

![Graph showing Speedup and BW Improvement against Number of Threads]

Parallelized IB Communication

- QPs only
One-sided Operations and IB netmod Optimization

- **Micro benchmark**
  - One to All experiment using 65 processes
    - root sends many MPI_PUT operations to all the other 64 processes (64 IB QPs)

---

**Profile of the experiment issuing 16000 operations**

<table>
<thead>
<tr>
<th>Nthreads</th>
<th>Time(s)</th>
<th>SP</th>
<th>SP/Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Total</td>
<td>5.8</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Ideal Speedup = 1.61

---

**Parallelized IB Communication**

Ideal Speedup = 3.34
One-sided Graph500 benchmark

• Every process issues many MPI_Accumulate operations to the other processes in every breadth first search iteration.

• Scale $2^{22}$, 16 edge factor, 64 MPI processes
Conclusion

• Many-core Architectures
• Most popular Funneled / Serialized mode in Hybrid MPI + threads programming model
  – Many threads parallelize user computation
  – Only single thread issues MPI calls
• Threads are IDLE during MPI calls!
• We utilize these IDLE threads to parallelize MPI internal tasks, and delivers better performance in various aspects
  – Derived datatype packing processing
  – Shared memory communication
  – IB network communication