Techniques for Enabling Highly Efficient Message Passing on Many-Core Architectures

Min Si
PhD student at University of Tokyo, Tokyo, Japan
Advisor: Prof. Yutaka Ishikawa, Prof. Reiji Suda

Guest graduate student at Argonne National Laboratory, IL, USA
Supervisor: Dr. Pavan Balaji

Email: msi@il.is.s.u-tokyo.ac.jp
Homepage: http://sudalab.is.s.u-tokyo.ac.jp/~msi/
Background

- Complexity in scientific applications
- Trends of hardware change
- Popular programming models and existing challenges
NWChem in Chemistry

- **Current applications have been looking at small-to-medium molecules consisting of 20-100 atoms**
  - Amount of computation per data element is reasonably large, so scientists have been reasonably successful decoupling computation and data movement

- **For Exascale systems, scientists want to study molecules of the order of a 1000 atoms or larger**
  - Coulomb interactions between the atoms is much stronger in the problems today than what we expect for Exascale-level problems
  - Larger problems will need to support both **short-range** and **long-range** components of the coulomb interactions (possibly using different solvers)
Irregular Sparse Computation in NWChem

- Diversity in the amount of computation per data element is going to increase substantially.
- Regularity of data and/or computation would be substantially different.

*Current computation pattern*

More than 50% time idling in CCSD(T) for $W^{21}$

<table>
<thead>
<tr>
<th>Number of Cores</th>
<th>Time (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1704</td>
<td>RMA: 5</td>
</tr>
<tr>
<td>3072</td>
<td>DGEMM: 4</td>
</tr>
<tr>
<td>6144</td>
<td></td>
</tr>
<tr>
<td>12288</td>
<td></td>
</tr>
</tbody>
</table>

Courtesy Pavan Balaji (Argonne)

Task load balancing? Communication complexity?
Genome Analysis in Bioinformatics

- **Sequence alignment**
- **Sequence assembly**
  - **Reconstruct** long DNA sequences by merging many small fragments
- **Gene mapping**

Hard to read whole genomes in current sequencing technology. Instead, read many small fragments, called "reads".

**Sequencing**

- Break into small "reads" (e.g., AGTTCCCTGGAAACGGTGAC...)
- DNA Samples → Reads

**Assembly**

- Search & merge overlapping reads
- De Bruijn graph
- Remove erroneous links
- Output long contigs

**Larger raw data & overlapping reads**
- Human Genome: 2TB ~ 3TB DNA reads
- Metagenome: PB ~ EB+ level DNA reads

[Adapted from National Human Genome Research Institute]
Massive Data Movement in Kiki Genome Assembly

Basic edge merging algorithm

1. Send local DNA unit to that node;
2. Search matching unit on that node;
3. Merge two units on that node;
4. Return merged unit.

ACGCGATTCAG

Large amount of outstanding data movement

Hard to balance task load
- 10^6+ outstanding msgs / rank
- 2.3TB sample was assembled on 18,000 cores for 4 days, 90%+ of time idling
Particle Tracing and Graph in Parallel Visualization

- Particle tracing
  - e.g., For Rayleigh–Taylor instability
    - Interface between a heavy fluid overlying a light fluid

  Mushroom cloud: RTI at the interface between hot less-dense and cold more-dense air

- Irregular graph visualization
  - Completely data-driven
  - Possible optimization is unclear but is interesting to investigate!

Semi-regular Communication in Particle tracing:
Exchange particles in 4D time-space neighborhoods

Irregular Task load

Courtesy Tom Peterka (Argonne)
GFMC in Nuclear Physics

- Green’s Function Monte Carlo
  - The “gold standard” for \textit{ab initio} calculations in nuclear physics at Argonne (Steve Pieper, PHY)

- Irregular pattern for load balancing
  - A non-trivial master/slave algorithm, with assorted work types and priorities
  - multiple processes create work dynamically
  - large work units

\textit{Courtesy Rusty Lusk (Argonne)}
**Complexity in Hardware Design**

1996 ASCI Red

**Terascale**

2008 IBM Roadrunner

**Petascale**

Increasing power per processor

2012 MIRA

Hit the power wall, multi-core started

2016 Cori

10Petascale

2017-2018 Summit

2018-2019 Aurora

100Petascale

Exascale

**Complexity of processors and memory design**

- Heterogeneous (i.e., CPU+GPU/Manycore)
- Fat node performance (many threads/cores)
- On-package memory
- I/O Burst buffer
- ...
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

Blue Gene/Q

<table>
<thead>
<tr>
<th>Node resources</th>
<th>Mira</th>
<th>Aurora</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Cores/#Threads</td>
<td>16/64</td>
<td>60+/240+</td>
</tr>
<tr>
<td>Memory</td>
<td>16GB</td>
<td>32GB (High Bandwidth Memory)</td>
</tr>
</tbody>
</table>

- Large amount of simple and low frequency cores
- Other on-chip resources are growing at a lower rate...
Scientific Programming models (1)

Hybrid MPI+Threads model

- To fully utilize the hardware resources
  - Massive parallelism in computation
  - On-chip resource sharing
- To handle complex & irregular computation
  - Dynamic & fine-grained task scheduling

- Large amount of low frequency cores
- Limited other on-chip resources (e.g., memory)
Hybrid MPI + threads modes

Traditional Thread Single mode

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

Multithreading mode

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

Funneled / Serialized mode (most widely used)

- Multithreaded user computation
- Still single thread issues MPI calls

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

- Multiple threads are created for user computation
- But only single thread issues MPI

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

- Large amount of IDLE threads
- Single lightweight core delivers poor performance
Scientific Programming models (2)

One-sided programming

- PGAS-like applications (e.g., Global Arrays for NWChem)
- CESAR project (Next generation Nuclear Reactor Modeling)

- For better resource sharing
  - Memory sharing across nodes on distributed memory systems
- To handle complex & irregular computation
  - Dynamic, data-driven communication

Perform DGEMM in local buffer
Problem Statement

- MPI one-sided operations are not truly one-sided!
  - Some operations can be supported by hardware (e.g., PUT/GET on IB, Cray)
  - Other operations still have to be handled by software (e.g., 3D accumulates of double precision data)

Software implementation of one-sided operations means that the target process has to make an MPI call to make progress. Not TRULY asynchronous!
Research Contribution

- Enable **highly efficient message passing** on many-core architectures for various kinds of scientific applications

I. Multithreaded MPI for hybrid MPI+ threads model
   - Sharing Idle Threads with application inside MPI
   - Optimizing MPI internal processing by **massive parallelism**

II. Process-based Asynchronous Progress for MPI one-sided programming
   - Flexible & Portable & Low overhead
   - Improve SW-handled RMA operations without affecting HW-handled RMA.
MT-MPI
Multithreaded MPI for Many-Core Environments

Published Paper
Core Concept of Multithreaded MPI

- Sharing Idle Threads with Application inside MPI
- Parallelizing MPI internal processing

```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**

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

- Number of available threads is **UNKNOWN!**
Challenges (2/2)

- Nested parallelism
  - Simply creates new Pthreads, and offloads thread scheduling to OS

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

**Threads Oversubscription**

Should ONLY use IDLE threads. However, it is UNKNOWN!
Design Overview

- **Modification in OpenMP runtime**
  - Expose number of IDLE threads
    - Guaranteed Idle Threads
    - Temporarily Idle Threads

- **Modification in MPI**
  - Parallelize internal tasks
    - Use `num_idle_threads` for *tradeoff* between sequential and parallelism algorithms
    - Use `num_idle_threads` for specifying `num_threads` in nested parallel region to *avoid threads overrunning issue*
MPI Internal Parallelism

DDT packing/unpacking

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

InfiniBand communication

Sender

Shared Buffer

Cell[0]

Cell[1]

Cell[2]

Cell[3]

Receiver

User Buffer

User Buffer

Shared memory communication

IB CTX

P0

HCA

PD

CQ

QP

QP

IB CTX

P1

HCA

PD

CQ

QP

IB CTX

Min Si, Postdoc Interview, 2015-10-26
Evaluation on Stampede

Hybrid MPI+OpenMP NAS MG (Class E, 64 processes) using parallelized DDT packing/unpacking

One-sided Graph500 (Scale $2^{22}$, 64 processes) using parallelized InfiniBand communication

OSU P2P BW using parallelized shared memory communication
CASPER
Process-based Asynchronous Progress Model for MPI RMA

Papers

Invited Talk
Message Passing Models

- Regular two-sided communication
  - Process 0
    - Send (data)
    - Receive (data)
  - Process 1
    - Receive (data)
    - Send (data)

- Irregular one-sided communication (Remote Memory Access)
  - Process 0
    - Put (data)
    - Get (data)
    - Acc (data)
  - Process 1
    - Computation

- Not TRULY asynchronous!
  - Non-contiguous Accumulate in MPI
Traditional Approaches of Asynchronous Progress

- **Thread-based approach**
  - Every process has a communication dedicated background thread
  - Background thread polls progress

- **Interrupt-based approach**
  - Assume all hardware resources are busy with user computation on target processes
  - Utilize hardware interrupts to awaken a kernel thread

**Cons:**
- Waste 50% computing cores or oversubscribe cores
- Overhead of multithreading safety

**DMMAP-based ASYNC overhead on Cray XC30**

Min Si, Postdoc Interview, 2015-10-26
Casper  Process-based ASYNC Progress

- Multi- and many-core architectures
  - “Infinite cores”
  - Not all of the cores are always keeping busy

- Process-based asynchronous progress
  - Dedicating arbitrary number of cores to “ghost processes”
  - Ghost process intercepts all RMA operations to the user processes

Pros:
- No overhead caused by multithreading safety or frequent interrupts
- Flexible core deployment
- Portable PMPI* redirection

![Diagram showing communication and computation processes](attachment:image.png)
Basic Design of Casper

- **Three primary functionalities**
  1. Transparently replace `MPI_COMM_WORLD` by `COMM_USER_WORLD`
  2. **Shared memory mapping** between local user and ghost processes by using MPI-3 `MPI_Win_allocate_shared`.
  3. **Redirect RMA operations** to ghost processes

* **MPI_WIN_ALLOCATE_SHARED** : Allocates window that is shared among all processes in the window’s group, usually specified with MPI_COMM_TYPE_SHARED communicator.

---

Min Si, Postdoc Interview, 2015-10-26
Challenges

- Ensuring Correctness and Performance
  - Lock Permission Management
  - Self Lock Consistency
  - Managing Multiple Ghost Processes
  - Multiple Simultaneous Epochs

- Asynchronous progress
- Transparent & Portable
- Correctness
- Performance

Applications

Casper

MPICH
CrayMPI
Intel MPI
MVAPICH
Evaluation on Cray XC30 (1)

RMA implementation in Cray MPI v6.3.1

<table>
<thead>
<tr>
<th></th>
<th>HW-handled OP</th>
<th>ASYNC. mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original mode</td>
<td>NONE</td>
<td>Thread</td>
</tr>
<tr>
<td>DMAPP mode</td>
<td>Contig. PUT/GET</td>
<td>Interrupt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Number of Application Processes (ppn=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulate on Cray XC30 (SW)</td>
<td></td>
</tr>
<tr>
<td>Original MPI</td>
<td>8.87</td>
</tr>
<tr>
<td>Thread-based async</td>
<td>17.22</td>
</tr>
<tr>
<td>DMAPP (Interrupt-based async)</td>
<td>53.16</td>
</tr>
<tr>
<td>Casper</td>
<td>5.09</td>
</tr>
</tbody>
</table>

PUT on Cray XC30 (HW in DMAPP mode)

<table>
<thead>
<tr>
<th></th>
<th>Number of Application Processes (ppn=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original MPI</td>
<td>5.09</td>
</tr>
<tr>
<td>Thread-based async</td>
<td>17.22</td>
</tr>
<tr>
<td>DMAPP (HW PUT)</td>
<td>17.04</td>
</tr>
<tr>
<td>Casper</td>
<td>7.07</td>
</tr>
</tbody>
</table>

No impact on HW-handled operations.

Casper provides asynchronous progress for SW-handled operations.
Evaluation on Cray XC30 (2)

- NWChem Quantum Chemistry Application
  - Computational chemistry application suite composed of many types of simulation capabilities.
  - ARMCI-MPI (Portable implementation of Global Arrays over MPI RMA)
Evaluation on Cray XC30 (3)

- Typical Get-Compute-Update mode in GA programming

```
for i in I blocks:
  for j in J blocks:
    for k in K blocks:
      GET block a from A
      GET block b from B
      c += a * b /*computing*/
      ACC block c to C
  end do
end do
```

**Pseudo code**

Perform DGEMM in local buffer
Evaluation on Cray XC30 (4)

- "Gold standard" CCSD(T)

- Water molecular (H$_2$O)$_{21}$

---

**Core deployment (24 cores per node)**

<table>
<thead>
<tr>
<th></th>
<th># COMP.</th>
<th># ASYNC.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original MPI</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>Casper</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>Thread-ASYNC (oversubscribed)</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Thread-ASYNC (dedicated)</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

---

**NWChem CCSD(T) for W21=(H$_2$O)$_{21}$ with pVDZ**

- Original MPI: 14.1
- Casper: 8.2
- Thread(O): 4.6
- Thread(D): 4.1

Reduced!

---

More accuracy
More computation
Summary

- Applications & hardware architectures are becoming more complex
- Parallelism & Resource sharing & Dynamic computation are important!
- Two most popular programming models used in modern applications

1. Hybrid MPI+Threads model
   - Problem:
     - Many IDLE threads in COMM.
     - Single lightweight core performs COMM.

2. One-sided programing
   - Problem:
     - Lack asynchronous progress in MPI RMA

Solutions

Multithreaded MPI
- Parallelizing MPI communication by utilizing user IDLE threads

Process-based Asynchronous Progress
- Provide Portable & Efficient & Flexible asynchronous progress for MPI RMA
Future Research Plan: BEEHIVE

Over-decomposition

Beehive dynamic execution runtime

Decoupling logical tasks from physical cores

High Performance
- Intelligent latency hiding
- Migration for better load balance

Fault Resilience
- Lightweight checkpointing
- Dynamic migration

Power Efficiency
- Computation and data consolidation
Under investigation: Optimization for High Performance

- **Intelligent Latency Hiding**
  - Context switch when blocking in communication / IO.
  - Yield to a “Ready-To-Go” process

- **Load Balancing**
  - Migrate processes from busy core to relatively idle core
Thank you
Backup Slides
Selected Publications

MPI optimization for many-core architectures (Ph.D. research)

Low level communication facility for many-core architectures (Master research)
Guaranteed Idle Threads VS Temporarily Idle Threads

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

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

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

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

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

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

Example 1

Example 2

Example 3
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)
        {
            ...
        }
    }
}
```
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)
Internal steps in CCSD(T) task

<table>
<thead>
<tr>
<th>Self-consistent field (SCF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Four-index transformation (4-index)</td>
</tr>
<tr>
<td>CCSD iteration</td>
</tr>
<tr>
<td>(T) portion</td>
</tr>
</tbody>
</table>

(T) Portion profiling for \( W_{21} \) with Original MPI

<table>
<thead>
<tr>
<th>Time (h)</th>
<th>Number of Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1704</td>
<td>16</td>
</tr>
<tr>
<td>3072</td>
<td>10</td>
</tr>
<tr>
<td>6144</td>
<td>8</td>
</tr>
<tr>
<td>12288</td>
<td>6</td>
</tr>
</tbody>
</table>

CCSD(T) internal steps in varying water problems

(T) portion consistently dominates the entire cost by close to 80%.