Non-blocking Collective Operations for MPI

- Towards Coordinated Optimization of Computation and Communication in Parallel Applications -

Torsten Hoefler
Open Systems Lab
Indiana University
Bloomington, IN, USA

Lawrence Livermore National Lab
Livermore, CA, USA
25th August 2008
Outline

2. Why (Non blocking) Collectives?
3. An Implementation - LibNBC
4. And Applications?
5. Ongoing Efforts
Outline

2. Why (Non blocking) Collectives?
3. An Implementation - LibNBC
4. And Applications?
5. Ongoing Efforts
### Fundamental Assumptions (I)

<table>
<thead>
<tr>
<th><strong>We need more powerful machines!</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Solving real-world scientific problems needs huge processing power (more than available)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Capabilities of single PEs have fundamental limits</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>The scaling/frequency race is currently stagnating</td>
</tr>
<tr>
<td>Moore’s law is still valid (number of transistors/chip)</td>
</tr>
<tr>
<td>Instruction level parallelism is limited (pipelining, VLIW, multi-scalar)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Explicit parallelism seems to be the only solution</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Single chips and transistors get cheaper</td>
</tr>
<tr>
<td>Implicit transistor use (ILP, branch prediction) have their limits</td>
</tr>
</tbody>
</table>
### Fundamental Assumptions (II)

#### Parallelism requires communication
- Local or even global data-dependencies exist
- Off-chip communication becomes necessary
- Bridges a physical distance (many PEs)

#### Communication latency is limited
- It’s widely accepted that the speed of light limits data-transmission
- Example: minimal 0-byte latency for $1m \approx 3.3\, ns \approx 13$ cycles on a $4\, GHz$ PE

#### Bandwidth can hide latency only partially
- Bandwidth is limited (physical constraints)
- The problem of “scaling out” (especially iterative solvers)
Assumptions about Parallel Program Optimization

**Collective Operations**
- Collective Operations (COs) are an optimization tool
- CO performance influences application performance
- Optimized implementation and analysis of CO is non-trivial

**Hardware Parallelism**
- More PEs handle more tasks in parallel
- Transistors/PEs take over communication processing
- Communication and computation could run simultaneously

**Overlap of Communication and Computation**
- Overlap can hide latency
- Improves application performance
We need more (functional) parallelism in our algorithms and codes!

This is hard work!

So, how much can we gain?
The LogGP Model

⇒ sending message of size $s$: $L + 2 \cdot o + (s - 1) \cdot G$
Resulting Interconnect Trends

Ongoing Technology Change

- modern interconnects offload communication to co-processors (Quadrics, InfiniBand, Myrinet)
- Ethernet is optimized for lower overhead (e.g., Gamma)
- many Ethernet adapters support protocol offload

$$\Rightarrow L + g + m \cdot G \gg o$$

$$\Rightarrow$$ we prove our expectations with benchmarks of the user CPU overhead
LogGP Model Examples - GigE/TCP

![Graph showing time in microseconds vs. datalsize in bytes for GigE/TCP](image)
LogGP Model Examples - Myrinet/GM

- Myrinet/GM - L + G*s + g
- Myrinet/GM - o

Graph showing the relationship between time in microseconds and dataseize in bytes (s) for Myrinet/GM.
LogGP Model Examples - InfiniBand/OpenIB

- OpenIB - L+G*s+g
- OpenIB - o

Time in microseconds vs. Datasize in bytes (s)
Outline

2. Why (Non blocking) Collectives?
3. An Implementation - LibNBC
4. And Applications?
5. Ongoing Efforts
Isend/Irecv is there - Why Collectives?

- Gorlach, ’04: “Send-Receive Considered Harmful”
- ↔ Dijkstra, ’68: “Go To Statement Considered Harmful”

point to point

```plaintext
if (rank == 0) then
    call MPI_SEND(...)
else
    call MPI_RECV(...)
end if
```

vs. collective

```plaintext
call MPI_GATHER(...)
```

⇒ cmp. math libraries vs. loops
Sparse Collectives/Topological Collectives

“But my algorithm needs nearest neighbor communication!?”
⇒ this is a collective too, just sparse (cf. sparse BLAS)

- sparse communication with neighbors on process topologies
- graph topology can make it generic
- many optimization possibilities (process placing, overlap, message scheduling/forwarding)
- easy to implement
- not part of MPI but fully implemented in LibNBC and proposed to the MPI Forum

⇒ give MPI details about you communication pattern!
Performance Benefits of (Non-Blocking) Collectives

**Blocking/Non-Blocking - Abstraction**
- abstraction enables optimizations
- ease of use, avoids implementation errors
- performance portability

**Non-Blocking - Overlap**
- leverage hardware parallelism (e.g. InfiniBand™)
- overlap similar to non-blocking point-to-point

**Non-Blocking - Pseudo Synchronization**
- avoidance of explicit pseudo synchronization
- limit the influence of OS noise
quantifying the benefits - with loggp

- collectives scale typically with $O(\log_2 P)$ or $O(P)$ sends
- “wasted” CPU time: $\log_2 P \cdot (L + (s - 1) \cdot G)$
  - Gigabit Ethernet: $L = 15-20 \mu s$
  - InfiniBand: $L = 2-7 \mu s$
  - $1 \mu s \approx 6000$ FLOP on a 3GHz Machine
- synchronization overhead not easy to assess
LogGP Model for Allreduce:

\[ t_{allred} = 2 \cdot (2o + L + m \cdot G) \cdot \left\lceil \log_2 P \right\rceil + m \cdot \gamma \cdot \left\lceil \log_2 P \right\rceil \]
Overlap - Overhead Benchmarks

Allreduce, LAM/MPI 7.1.2/TCP over GigE

CPU Usage (percent) vs. Communicator Size and Data Size
Synchronization - Process Skew

- caused by OS interference or unbalanced application
- worse if system is oversubscribed
- interference multiplies on big systems
- can cause dramatic performance decrease
- all nodes wait for the last

Example

Petrini et. al. (2003) "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q"
Synchronization - Process Skew

- caused by OS interference or unbalanced application
- worse if system is oversubscribed
- interference multiplies on big systems
- can cause dramatic performance decrease
- all nodes wait for the last

Example

Petrini et. al. (2003) "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q"
MPI_Bcast with P0 delayed - Jumpshot
MPI_Ibcast with P0 delayed + overlap - Jumpshot
Outline

2. Why (Non blocking) Collectives?
3. An Implementation - LibNBC
4. And Applications?
5. Ongoing Efforts
LibNBC - Interface

- extension to MPI
- uses NBC_Requests and NBC_Test/NBC_Wait
- IB/OFED optimized Transport Interface
- fully threaded (blocking OFED or MPI_THREAD_MULTIPLE)

Interface

```c
NBC_Ibcast(buf, count, MPI_INT, 0, comm, &req);
/* compute simultaneously to communication */
NBC_Wait(&req);
```

Proposal

Hoefler et. al.: ”Non-Blocking Collective Operations for MPI-2”
LibNBC - Interface

- extension to MPI
- uses NBC_Requests and NBC_Test/NBC_Wait
- IB/OFED optimized Transport Interface
- fully threaded (blocking OFED or MPI_THREAD_MULTIPLE)

```
NBC_Ibcast(buf, count, MPI_INT, 0, comm, &req);
/* compute simultaneously to communication */
NBC_Wait(&req);
```

Proposal

Hoefler et. al.: "Non-Blocking Collective Operations for MPI-2"
Non-Blocking Collectives - Implementation

- implementation available with LibNBC
- written in ANSI-C and uses only MPI-1
- central element: collective schedule
- every coll. algorithm can be represented as a schedule
- trivial addition of new algorithms

Example: dissemination barrier, 4 nodes, node 0:

| send to 1 | recv from 3 | end | send to 2 | recv from 2 | end |

LibNBC download: http://www.unixer.de/NBC
Benchmarks - Gather with InfiniBand on 64 nodes

![Graph showing overhead (usec) vs message size (kilobytes) for different libraries: Open MPI/blocking, LibNBC/Open MPI, and LibNBC/LibOF. Each line represents a different library configuration.](image-url)
Benchmarks - Alltoall with InfiniBand on 64 nodes

- Open MPI/blocking
- LibNBC/Open MPI
- LibNBC/LibOF

![Graph showing overhead vs message size for different MPI implementations.](image-url)
Progression Issues

Threaded Progression
- works with MPI_THREAD_MULTIPLE and InfiniBand™
- thread “blocks” on MPI_Wait or IB file descriptor
- different OS scheduling issues (see Cluster 2008 article)

Manual Progression
- call NBC_Test to progress communication
- is necessary to advance in schedule (rounds)
- necessary frequency depends on the collective

⇒ progression issues are not trivial!
Outline

2. Why (Non blocking) Collectives?
3. An Implementation - LibNBC
4. And Applications?
5. Ongoing Efforts
Independent Computation Exists in Algorithm

1) Linear Solvers - Domain Decomposition
   - iterative linear solvers are used in many scientific kernels
   - often used operation is vector-matrix-multiply
   - matrix is domain-decomposed (e.g., 3D)
   - only outer (border) elements need to be communicated
   - can be overlapped

2) Medical Image Reconstruction - Loop Iteration Pipelining
   - iterations have independent parts
   - communication of iteration $i$ can be overlapped with parts of $i + 1$
1) Linear Solver - Domain Decomposition

- nearest neighbor communication
- can be implemented with sparse/topological collectives
1) Linear Solver - Parallel Speedup (Best Case)

- Cluster: 128 2 GHz Opteron 246 nodes
- Interconnect: Gigabit Ethernet, InfiniBand™
- System size 800x800x800 (1 node ≈ 5300s)
2) Medical Image Reconstruction

- OpenMP + MPI (collectives only) parallelized
- compute $A_{i+1}$ while communicating $c_i$

\[
\text{for each (iteration k)} \{
  \text{for each (subiteration l)} \{
    \text{for (event } i \in S_l) \{
      \text{compute } A_i
      \text{compute } c_i+ = (A_i)^t \frac{1}{A_i f_i^k}
      \text{allreduce } c_i
    \}}
  \}
\]
\[
f_{i+1}^k = f_i^k c_i
\]
\[
f_0^{k+1} = f_{i+1}^k
\]
2) Medical Image Reconstruction (32 Nodes)

![Graph showing time to solution for different thread counts and collective functions.]

- MPI_Allreduce()
- NBC_Iallreduce()
- NBC_Iallreduce() (thread)

Time to Solution (s)

1 thread 2 threads 3 threads 4 threads
Data-parallel Computations

Automated Pipelining with C++ Templates

- loop tiling
- automated overlap with window of outstanding communications
- optimizing tiling factor and window size
Data-parallel Examples

1) Parallel Data Transformation (e.g., Compression)
   - scatter from source, transformation, gather to destination
   - scatter/gather step pipelined
   - example uses bzip2 algorithm

2) 3d Fast Fourier Transformation
   - 1d-distribution identical to “normal” parallel 3d-FFT
   - start communication as early as possible
   - start MPI_Ialltoall as soon as first xz-plane is ready
   - calculate next xz-plane
   - start next communication accordingly ...
   - collect multiple xz-planes (tile factor)
1) Parallel Compression

```c
my_size = 0;
for (i=0; i < N/P; i++) {
    my_size += compress(i, outptr);
    outptr += my_size;
}
gather(sizes, my_size);
gatherv(outbuf, sizes);
```

```c
for (i=0; i < N/P; i++) {
    my_size = compress(i, outptr);
    gather(sizes, my_size);
    igatherv(outbuf, sizes, hndl[i]);
    if (i > 0) waitall(hndl[i-1], 1);
}
waitall(hndl[N/P], 1);
```
1) Parallel Compression Communication Overhead

![Graph showing communication overhead for different MPI/NBC and OF/NBC configurations across different node counts (64, 32, 16, 8). The Y-axis represents communication overhead in seconds, and the X-axis represents node count. The graph compares MPI/BL, MPI/NBC, and OF/NBC configurations.]
Transformation in z Direction

Data already transformed in y direction

1 block = 1 double value (3x3x3 grid)
Transformation in z Direction

Transform first xz plane in z direction

pattern means that data was transformed in y and z direction
Transformation in z Direction

start MPI_Ialltoall of first xz plane and transform second plane

cyan color means that data is communicated in the background
Transformation in z Direction

start MPI_Ialltoall of second xz plane and transform third plane

data of two planes is not accessible due to communication
Transformation in x Direction

start communication of the third plane and ...

we need the first xz plane to go on ...
Transformation in x Direction

... so MPI_Wait for the first MPI_Ialltoall!

and transform first plane (new pattern means xyz transformed)
Transformation in x Direction

Wait and transform second xz plane

first plane’s data could be accessed for next operation
Transformation in x Direction

wait and transform last xz plane

done! → 1 complete 1D-FFT overlaps a communication
2) $1024^3$ 3d-FFT over InfiniBand

<table>
<thead>
<tr>
<th>Method</th>
<th>1 ppn</th>
<th>2 ppn</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI/BL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI/NB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NBC/NB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

P=128, “Coyote”@LANL - 128/64 dual socket 2.6GHz Opteron nodes
2) $1024^3$ 3d-FFT on XT4

![Graph showing FFT time for different numbers of processes using MPI/BL and NBC/NB methods. The x-axis represents the number of processes: 32, 64, 128. The y-axis represents FFT time in seconds.]

“Jaguar”@ORNL - Cray XT4, dual socket dual core 2.6GHz Opteron
2) $1024^3$ 3d-FFT on XT4 (Communication Overhead)

"Jaguar"@ORNL - Cray XT4, dual socket dual core 2.6GHz Opteron
2) $640^3$ 3d-FFT InfiniBand (Communication Overhead)

![Chart showing FFT Communication Time (s) for different data sizes and communication methods.]

- **MPI/BL**: Red bars
- **MPI/NBC**: Green bars
- **OF/NBC**: Blue bars

“Odin”@IU - dual socket dual core 2.0GHz Opteron InfiniBand
Outline

2. Why (Non blocking) Collectives?
3. An Implementation - LibNBC
4. And Applications?
5. Ongoing Efforts
Ongoing Work

**LibNBC**
- optimized collectives and modeling
- more low-level transports (e.g., MX)
- analyze offloading/onloading collectives

**MPI-Forum (MPI-3) Efforts**
- proposed non-blocking collectives
- proposed sparse collective
- several proposals to enhance library support

**Applications**
- work on more applications (apply C++ templates?)
- → interested in collaborations (ask me!)
Discussion

THE END

Questions?

Thank you for your attention!