BDMPI: Conquering Big Data with Small Clusters using MPI

Dominique LaSalle and George Karypis
University of Minnesota, Minneapolis, MN, USA

November 18, 2013
What is Big Data?

- Depends on your compute system:
  - Laptop/PC
  - Server
  - Cluster
  - Data Center

- Data > DRAM
Existing Solutions

Big Data Solutions

- MapReduce/Hadoop
- GraphChi
- Giraph
- Hama
- Custom Solution
Distributed and Out-of-core Computing

Distributed Algorithms
- Minimize communication between processes.
- Extract independent tasks to perform in parallel.
- Organized into a series of compute and collective/point-to-point communication steps.

Out-of-Core Algorithms
- Minimize reads and writes to disk.
- Extract independent tasks to perform serially.
- Organized into a series of compute and disk read/write steps.
Insights

The Graph Ordering Problem

- How can a graph be efficiently re-order in an out-of-core fashion?
- How can a graph be efficiently re-order in a distributed fashion?

General Applications

- How can we treat a remote process as a disk?
  - Already supported by MPI’s one sided communication (exchange fread/fwrite for MPI_get/MPI_put).
- Can we treat the disk as a remote process?
  - Need to handle remote computations/data movement.
How it Works

BDMPI

- Transparent layer between an MPI program and an MPI runtime.
- For a problem of size $n$ and a compute cluster with $p$ processing nodes each with $m$ memory:
  1. Divide the data into $t = \frac{n}{m}$ blocks.
  2. Spawn a master process on each compute node.
  3. Spawn $\frac{t}{p}$ slave processes on each compute node.
- Allow only one slave process to run at a time on each compute node.
  - That process will run until it blocks on a communication operation.
Why it Works

Node-Level Cooperative Multi-Tasking

- Processes run until blocking for a collective communication or receive operation.
- Cost of loading data from disk is amortized over large blocks of computation.
- Since only one process runs at a time, the thrashing associated with multiple processes attempting to gain residency is avoided.
Usage

- `bdmpiexec`

```bash
mpiexec -np 80
    programe [arg1] [arg2] ...
```

```bash
bdmpiexec -np 4 [-nr 2] -ns 20
    programe [arg1] [arg2] ...
```

- Executes `mpi` program on a cluster with four nodes as if it were on a cluster of 80 computes nodes.

- `libbdmpi`
  - Provides `MPI_X` functions.
  - Replace `#include <mpi.h>` with `#include <bdmpi.h>`.
**BDMPI API**

**MPI Subset Implemented by BDMP**

BDMPI_Init, BDMP Finalize

BDMPI_Comm_size, BDMPI_Comm_rank, BDMPI_Comm_dup, BDMPI_Comm_free, BDMPI_Comm_split

BDMPI_Send, BDMPI_Isend, BDMPI_Recv, BDMPI_Irecv, BDMPI_Sendrecv

BDMPI_Probe, BDMPI_Iprobe, BDMPI_Test, BDMPI_Wait, BDMPI_Get_count

BDMPI_Barrier

BDMPI_Bcast, BDMPI_Reduce, BDMPI_Allreduce, BDMPI_Scan, BDMPI_Gather[v], BDMPI_Scatter[v], BDMPI_Allgather[v], BDMPI_Alltoall[v]
Implementation

Communication Model

Node 1

Master

Slave

Slave

Node 2

Master

Slave

Slave
Master-Slave Communication

```
int main() {
  ...
  MPI_Init()
  ...
  MPI_Isend()
  ...
  MPI_Recv()
  ...
  MPI_Finalize()
  ...
}
```
Message Buffering

- Small messages buffered in memory.
- Large messages buffered on disk.

Send and ISend

- Message buffering allows sending process to continue executing without blocking.

Recv and IRecv

- If the master has already buffered the message, no blocking occurs.
- Otherwise the process becomes blocked, and another process is allowed to run.
Benchmarks

PageRank
- Memory heavy operation.
- Multiplying a sparse matrix by a vector.

KMeans Clustering
- Multiplying a sparse matrix by a dense matrix (100 clusters).

SGD
- Matrix factorization $A = UV$ (20 factors).
- Element-wise random traversal.
- SGD-row
  - Row-wise traversal.
  - Better locality than regular SGD.
Test Codes

- **Serial-OOC** - Custom out-of-core solutions.
- **MPI** - MPI codes ran using MPICH.
- **GraphChi** - Kyrola et. al. 2012.
- **Hadoop**
  - Mahout for KMeans.
- **BDMPI**
  - **BDMPI** - MPI codes ran using the BDMPI runtime.
  - **BDMPI-mlock** - MPI codes + munlock()/mlock().
  - **BDMPI-OOC** - MPI codes + fread()/fwrite().
Experiment Setup

Our Cluster
- Four machine cluster:
  - Intel i7 @ 3.4 GHz
  - 4 GB of DRAM
  - Seagate Barracuda 7200 RPM 1.0 TB (300GB swap and /scratch partitions)

Our Datasets
- PageRank - 6.6B edges, ordered randomly (50GB CSR).
- KMeans - 30M \( \times \) 83K with 7.3B non-zeros (56GB CSR).
- SGD - 3.8M \( \times \) 284K with 12.8B non-zeros (50GB CSR).
Single Node Results
Cluster Results

<table>
<thead>
<tr>
<th>Task</th>
<th>Hadoop</th>
<th>MPI</th>
<th>BDMPI</th>
<th>BDMPI-mlock</th>
<th>BDMPI-OOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>PageRank</td>
<td>234.93</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KMeans</td>
<td>1196.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SGD</td>
<td>&gt;3000</td>
<td></td>
<td></td>
<td>196.31</td>
<td>1196.7</td>
</tr>
</tbody>
</table>

Minutes / Iteration

![Cluster Results Graph]

**Motivation**
- Distributed and Out-of-core Computing

**BDMPI**
- Overview
- Usage
- Implementation

**Results**
- Experiments
- Single Node
  - **Cluster**
  - Scaling

**Conclusion**
Scaling Results

- **PageRank**
- **KMeans**
- **SGD**

Speedup on Four Nodes

- MPI
- BDMPI
- BDMPI-mlock
- BDMPI-OOC

Graph showing performance comparison between different methods and their speedup on four nodes.
**Conclusion**

**BDMPI**

- Utilizes existing MPI interface.
  - Turns existing MPI applications into distributed out-of-core applications.
  - Leverages 20 years worth of experience.
- Achieves speeds comparable to custom out-of-core solutions.
- Scales well across multiple machines.