Improving the Performance of MPI Derived Datatypes by Optimizing Memory-Access Cost

Surendra Byna[†] William Gropp[‡] Xian-He Sun[†] Rajeev Thakur[‡]

[†]Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616.
[‡]Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439

Introduction

- The MPI Standard supports derived datatypes, which allow users to describe noncontiguous memory layouts and communicate noncontiguous data with a single communication function
- However, many MPI implementations perform poorly with derived datatypes
- Users resort to their own implementations of packing data into a contiguous buffer and then calling MPI_Send
- Such usage clearly defeats the purpose of having derived datatypes in the MPI Standard
- Noncontiguous communication occurs commonly in many applications and improving the performance of derived datatypes is essential

Possible Solutions

- The performance of derived datatypes can be improved in two ways.
  - Improving the data structures used to store derived datatypes internally in the MPI implementation
  - Optimizing packing noncontiguous data into a contiguous buffer by exploiting the advanced memory hierarchies of today's computer architectures.
- Research has already been done on the first solution, mainly in using data structures that allow a stack-based approach to parse a datatype, rather than making recursive function calls, which are expensive
- We chose to optimize the performance based on the data access pattern and the memory architecture of the machine

Solution to Improve the Performance

- Performance is improved at 2 levels
- When MPI_Type_commit is called – Find the current data access cost and possibility of optimization. If optimization is possible, find all the loop optimization parameters (Tile size, array padding etc.,)
- When MPI communication function is called at the source process, If the optimization tag is set, use the loop optimizations to pack data into contiguous buffer before sending data to network buffer

Optimizing Memory Access Cost

- When MPI_Type_commit function is called, retrieve the data access pattern and apply loop transformations to optimize the memory access cost

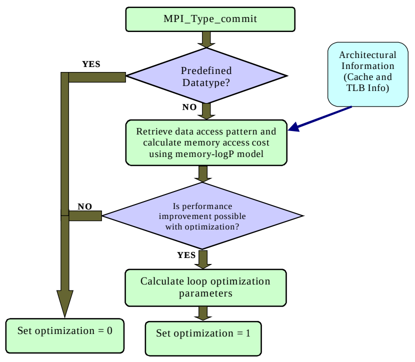
MPI_Type_commit
Predefined Datatype?
YES
NO
Architectural Information (Cache and TLB Info)
Retrieve data access pattern and calculate memory access cost using memory-logP model
Is performance improvement possible with optimization?
NO
YES
Calculate loop optimization parameters
Set optimization = 0
Set optimization = 1

Figure 1. Data access cost prediction and optimization

Modeling Memory Communication

- Total communication cost = Network communication cost + Memory communication cost
- *Memory communication* cost is the cost incurred in the transmission of data to/from user space from/to the local network buffer (or shared memory buffer).
- *Network communication* cost is the cost incurred in the movement of data between source and target network buffers.
- Memory access cost is a function of data size, access pattern, architecture of memory hierarchy and compiler
- Memory communication is modeled with memory-logP model

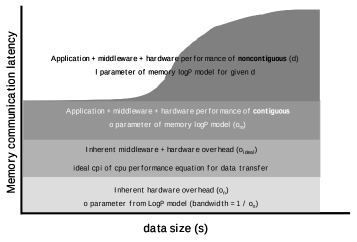
Memory communication latency
Application + middleware + hardware performance of noncontiguous (d)
l parameter of memory logP model for given d
Application + middleware + hardware performance of contiguous
o parameter of memory logP model (o_m)
Inherent middleware + hardware overhead (o_{ideal})
ideal cpi of cpu performance equation for data transfer
Inherent hardware overhead (o_n)
o parameter from LogP model (bandwidth = 1 / o_n)
data size (s)

Figure 2. Memory Communication

The memory-logP model

- For a given implementation of data transfer on a given system the time per byte the processor is engaged in transmission or reception of a message is characterized by four components:
  - l: the effective latency, cost due to data size and distribution l=f(s,d)
  - o: the overhead, cost of an ideally distributed message
  - g: the gap, defined as the minimum time interval between consecutive message receptions at a processor. 1/g=achieved memory bandwidth
  - P: the number of processor/memory modules

† Assume g = o+l, applied simplification of LogP model and P=1 for memory communication. Memory-logP model is developed by Dr. Kirk W. Cameron and Dr. Xian-He Sun.

Optimizing the Packing Cost

- When the MPI communication function is called, if the optimization flag is set to 1, use the loop optimization parameters (such as tile size of cache/TLB tiling, padding size of array padding) to pack the non-contiguous data into contiguous buffer.
- Send the contiguous data to the network buffer

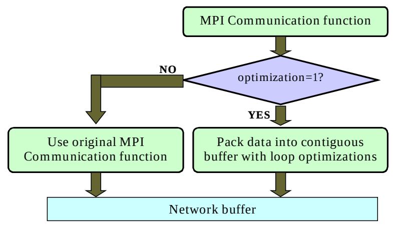
MPI Communication function
optimization=1?
NO
YES
Use original MPI Communication function
Pack data into contiguous buffer with loop optimizations
Network buffer

Figure 3. Data access cost optimization with loop transformations

Initial Results – Experimental Setup

- SGI Origin 2000
  - MIPS R10000 Processor, 195MHz
  - 32KB 2-way set associative L1 cache
  - 4MB off-chip L2 cache
  - IRIX 6.5.14 operating system
  - MPI implementation: MPICH 1.2.5 with shared memory device
  - Hardware counters are used to measure the performance

Experimental Setup

- IBM Blue Horizon at SDSC
  - Power3 processors run at 375 MHz
  - 64KB, 128-way set associative L1 cache
  - 8MB 4-way set associative L2 cache
  - MPI implementation: IBM MPI
- Matrix Transpose algorithm is implemented for performance comparison in the following cases :
  - Using original MPI derived datatypes of MPICH 1.2.5
  - Using optimized MPICH implementation
  - Without MPI derived datatypes, manual implementation
- On IBM machine, potential performance improvement is shown for IBM's MPI with memory access cost optimization

Initial Results

- Optimized MPICH significantly outperforms the original MPICH and user packing for data size greater than 8MB, where the cache optimization comes into effect

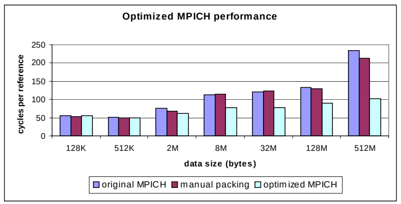

Figure 4. Performance improvement with optimized derived datatypes.

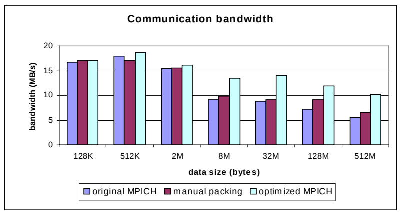

Figure 5. Overall communication bandwidth is as much as 85% higher achieved with optimization over original MPICH

Improvement on Vendor MPI

- Advanced vendor MPI implementations such as IBM's MPI also stand to gain significant performance improvements by using memory optimizations for derived datatypes.

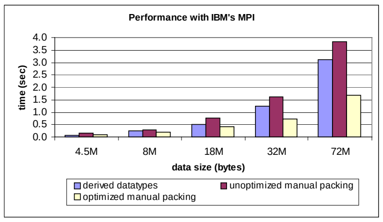

Figure 6. Performance improvement on vendor MPI (IBM's MPI) implementations

Future Work

- A model to predict the memory access cost based on data access pattern is under development
- We plan to extend optimizations to include more loop transformations such as loop interchange, loop unrolling etc.
- Eventually this work will be incorporated into MPICH2 and its new improved implementation of derived datatypes.
- This technique will also be applied to automatic tuning of memory performance of parallel applications.

Contact Info:
Surendra Byna† William Gropp‡ Xian-He Sun† Rajeev Thakur‡
[†]{renbyna, sun}@iit.edu [‡] {gropp, thakur}@mcs.anl.gov

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38, and in part by a grant from the Office of Advanced Simulation and Computing, National Nuclear Security Administration, U.S. Department of Energy.