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 before sending data to network buffers.
- 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 pattern and the possibility of optimization. If optimization is possible, find the loop optimization parameters (tile size, array padding, etc.)
  - When MPI communication function is called at the source process, if the optimization flag 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

Modeling Memory Communication

- Total communication cost = Network communication cost + Memory communication cost
- Memory communication cost = the cost incurred in the transmission of data from source user space to/from the local network buffer (or shared memory buffer).
- Network communication cost = 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.

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
  - $\lambda$: the effective latency, cost due to data size and distribution (in bytes)
  - $\gamma$: the overhead, cost of an ideally distributed message
  - $g$: the gap, defined as the minimum time interval between consecutive message receptions at a processor
  - $P$: the number of processor/memory modules

Optimizing the Packing Cost

- When the MPI communication function is called, if the optimization flag is set to 1, use the loop optimization parameters (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.

Experimental Setup

- IBM Blue Horizon at SDSC
- Power4 processors at 375 MHz
- 64KB, 128-way set associative L1 cache
- 512KB 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.

Future Work

- A model to predict the memory access cost based on data access patterns is under development.
- We plan to extend optimizations to include more loop transformations, basic 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.