Runtime Optimization of Application Level Communication Patterns

Edgar Gabriel and Shuo Huang

Department of Computer Science
University of Houston

gabriel@cs.uh.edu
Motivation

Finite Difference code on a PC cluster using IB and GE interconnects
Execution time for 200 iterations of the solver on 32 processes/processors
How to implement the required communication pattern efficiently?

• Dependence on platform
  – Some functionality only supported (efficiently) on certain/platforms or with certain network interconnects

• Dependence on MPI library
  – Does the MPI library support all available methods
  – Efficiency in overlapping communication and computation
  – Quality of the support for user defined data-types

• Dependence on application
  – Problem size
  – Ratio of communication to computation
• **Problem**: How can an (average) user understand the myriad of implementation options and their impact on the performance of the application?

• **(Honest) Answer**: no way
  – Abstract interfaces for application level communication operations required $\rightarrow$ ADCL
  – Statistical tools required to detect correlations between parameters and application performance
ADCL - Adaptive Data and Communication Library

• Goals:
  – Provide abstract interfaces for often occurring application level communication patterns
    • Collective operations
    • Not-covered by MPI specification
  – Provide a wide variety of implementation possibilities and decision routines which choose the fastest available implementation (at runtime)

• Not replacing MPI, but add-on functionality
  – Uses many features of MPI
## ADCL terminology

<table>
<thead>
<tr>
<th>ADCL object</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Abstraction for a characteristic of an implementation represented by the set its possible values</td>
</tr>
<tr>
<td>Attribute-set</td>
<td>Group of attributes</td>
</tr>
<tr>
<td>Function</td>
<td>Implementation of a particular operation</td>
</tr>
<tr>
<td></td>
<td>• optionally including an attribute-set and values</td>
</tr>
<tr>
<td>Function-set</td>
<td>Set of functions providing the same functionality</td>
</tr>
<tr>
<td></td>
<td>• have to have the same attribute-set</td>
</tr>
<tr>
<td>Vector</td>
<td>Abstraction for a multi-dimensional data object</td>
</tr>
<tr>
<td>Topology</td>
<td>Abstraction for a process topology</td>
</tr>
<tr>
<td>Request</td>
<td>Handle for tuple of <code>&lt; topology, vector, function-set&gt;</code></td>
</tr>
</tbody>
</table>
Code sample

```c
ADCL_Vector vec;
ADCL_Topology topo;
ADCL_Request request;

/* Generate a 2-D process topology */
MPI_Cart_create ( comm, 2, cart_dims, periods, 0, &cart_comm);
ADCL_Topology_create ( cart_comm, &topo );

/* Register a 2D vector with ADCL */
ADCL_Vector_register ( ndims, vec_dims, HALO_WIDTH,
                      MPI_DOUBLE, vector, &vec );

/* Match process topology, data item and function-set */
ADCL_Request_create ( vec, topo, ADCL_FNCTSET_NEIGHBORHOOD,
                      &request );

for (i=0; i<NIT; i++) {
    /* Main application loop */
    ADCL_Request_start ( request );
    ...
}
```
Runtime selection logic: brute force search (I)

Implementation no. 1 2 3 4 5 6 7

Using the fastest implementation for the rest of the application
Runtime selection logic: brute force search (II)

- Test each function of a given function set a given number of times
  - Store the execution time for each execution per process
- Filter the list of execution times in order to exclude outliers
- Determine the avg. execution time per function $i$ and process $j$
- Determine the max. execution time for function $i$ across all processes

$$f_i^{\text{max}} = \max(f_i^j), j = 0...nprocs - 1$$

- Requires communication (e.g. MPI_Allreduce)
Runtime selection logic: brute force search (III)

- Determine the function with the minimal max. execution time across all processes

\[ f_{\text{winner}} = \min(f_i^{\text{max}}), i = 0...nfuncs - 1 \]

- Use this function for the rest of the application lifetime
Runtime selection logic: performance hypothesis (I)

• Assumptions:
  – every implementation can be characterized by a set of attributes, which impact its performance, e.g. for neighborhood communication
    • Communication pattern/degree
    • Handling of non-contiguous data
    • Data transfer primitive
    • Overlapping communication and computation
  – The fastest implementation will also have the optimal values for these attributes
Runtime selection logic: performance hypothesis (II)

- Approach: determine the optimal value for an attribute by comparing the execution time of functions differing in only a single attribute.

<table>
<thead>
<tr>
<th></th>
<th>Function a</th>
<th>Function b</th>
<th>Function c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value for attribute 1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Value for attribute 2</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Value for attribute 3</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Value for attribute 4</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
</tr>
</tbody>
</table>

- E.g. if function c had the lowest execution time across all processes:
  - **Hypothesis**: value 3 optimal for attribute 1
  - **Confidence value** in this hypothesis: 1
Runtime selection logic: performance hypothesis (III)

- Evaluate a different set of functions differing in one other attribute, e.g.

<table>
<thead>
<tr>
<th>Function c</th>
<th>Function d</th>
<th>Function e</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value for attribute 1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Value for attribute 2</td>
<td>X+1</td>
<td>X+1</td>
</tr>
<tr>
<td>Value for attribute 3</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Value for attribute 4</td>
<td>Z</td>
<td>Z</td>
</tr>
</tbody>
</table>

- If this set of measurements lead to the same optimal value for attribute 1:
  - Increase confidence value for this hypothesis by 1
- Else decrease the confidence value by 1
Runtime selection logic: performance hypothesis (IV)

- If the confidence value for an attribute reaches a given threshold
  - Remove all functions not having the required value for this attribute from the Function-set
- If the value for attribute(s) do not converge towards a value this algorithm leads to the brute force search
- Advantage: potentially fewer functions have to be evaluated to determine the winner
## Currently available implementations for neighborhood communication

<table>
<thead>
<tr>
<th>Name</th>
<th>Comm. pattern</th>
<th>Handling of non-cont. data</th>
<th>Data transfer primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsendIrecv_aao</td>
<td>aao</td>
<td>ddt</td>
<td>MPI_Isend/Irecv/Waitall</td>
</tr>
<tr>
<td>IsendIrecv_pair</td>
<td>pair</td>
<td>ddt</td>
<td>MPI_Isend/Irecv/Waitall</td>
</tr>
<tr>
<td>SendIrecv_aao</td>
<td>aao</td>
<td>ddt</td>
<td>MPI_Send/Irecv/Waitall</td>
</tr>
<tr>
<td>SendIrecv_pair</td>
<td>pair</td>
<td>ddt</td>
<td>MPI_Send/Irecv/Wait</td>
</tr>
<tr>
<td>IsendIrecv_aao_pack</td>
<td>aao</td>
<td>ddt</td>
<td>MPI_Isend/Irecv/Waitall</td>
</tr>
<tr>
<td>IsendIrecv_pair_pack</td>
<td>pair</td>
<td>Pack/unpack</td>
<td>MPI_Isend/Irecv/Waitall</td>
</tr>
<tr>
<td>SendIrecv_aao_pack</td>
<td>aao</td>
<td>ddt</td>
<td>MPI_Send/Irecv/Waitall</td>
</tr>
<tr>
<td>SendIrecv_pair_pack</td>
<td>pair</td>
<td>Pack/unpack</td>
<td>MPI_Send/Irecv/Wait</td>
</tr>
<tr>
<td>SendRecv_pair</td>
<td>pair</td>
<td>ddt</td>
<td>MPI_Send/Recv</td>
</tr>
<tr>
<td>Sendrecv_pair</td>
<td>pair</td>
<td>ddt</td>
<td>MPI_Send/Recv</td>
</tr>
<tr>
<td>Sendrecv_pair_pack</td>
<td>pair</td>
<td>Pack/unpack</td>
<td>MPI_Send/Recv</td>
</tr>
<tr>
<td>Sendrecv_pair_pack</td>
<td>pair</td>
<td>Pack/unpack</td>
<td>MPI_Send/Recv</td>
</tr>
<tr>
<td>WinfencePut_aao</td>
<td>aao</td>
<td>ddt</td>
<td>MPI_Put/MPI_Win_fence</td>
</tr>
<tr>
<td>WinfenceGet_aao</td>
<td>aao</td>
<td>ddt</td>
<td>MPI_Get/MPI_Win_fence</td>
</tr>
<tr>
<td>PostStartPut_aao</td>
<td>aao</td>
<td>ddt</td>
<td>MPI_Put/MPI_Win_post/start</td>
</tr>
<tr>
<td>PostStartGet_aao</td>
<td>aao</td>
<td>ddt</td>
<td>MPI_Get/MPI_Win_post/start</td>
</tr>
<tr>
<td>WinfencePut_pair</td>
<td>pair</td>
<td>ddt</td>
<td>MPI_Put/MPI_Win_fence</td>
</tr>
<tr>
<td>WinfenceGet_pair</td>
<td>pair</td>
<td>ddt</td>
<td>MPI_Get/MPI_Win_fence</td>
</tr>
<tr>
<td>PostStartPut_pair</td>
<td>pair</td>
<td>ddt</td>
<td>MPI_Put/MPI_Win_post/start</td>
</tr>
<tr>
<td>PostStartGet_pair</td>
<td>pair</td>
<td>ddt</td>
<td>MPI_Get/MPI_Win_post/start</td>
</tr>
</tbody>
</table>
Performance results (I)

InfiniBand 32 processes small problem size

Execution time [sec]
Performance results (II)

InfiniBand 32 processes large problem size

Execution time [sec]

HIPS 2007 Long Beach
Edgar Gabriel
Performance results (III)

TCP over Fast Ethernet 32 processes small problem size

Execution time [sec]
Performance results (IV)

TCP over Fast Ethernet 32 processes large problem size

Execution time [sec]

HIPS 2007 Long Beach
Edgar Gabriel
Limitations of ADCL

- Reproducibility of measurements even on dedicated compute nodes - a challenging topic
  - Hyper-threading
  - Processor frequency scaling
- Network often shared between multiple jobs
- Hierarchical networks
  - Process placement by the batch scheduler
- Performance hypothesis
  - Attributes should not be correlated
- User has to modify its code
  - How much longer will we have to deal with MPI?
Advantages of ADCL

- Provides close to optimal performance in many scenarios
- Simplifies the development of parallel code for many applications
- Simplifies the development of adaptive parallel code
- Currently ongoing work:
  - Improving (nearly) all components of ADCL
    - Data filtering
    - Increase parameter space and set of implementation
    - Experiment with other runtime selection algorithms
      - Historic learning, Game theory, genetic algorithms
  - Integration with a CFD solver in cooperation with Dr. Garbey