Logicalization of Communication
Traces from Parallel Execution

Qiang Xu
CGG Veritas Inc. & University of Houston

Jaspal Subhlok        Rong Zheng
University of Houston

Sara Voss
Coe College & University of Houston
Motivation

OBJECTIVE: Estimate parallel application performance rapidly in foreign/dynamic, hard to model, scenarios such as

- Performance with different software components, e.g., MPI Library
- Desktop/Volunteer grid or Amazon EC-2 cloud...
- Scalability – if 32, 64 or 128 cores are best
- System under simulation

Build a short running “skeleton” program that mimics target application:

- skeleton: application exec time fixed, say 1:1000, then.
- \textit{execute skeleton to estimate app execution time}
In SPMD parallel programs all processes typically perform similar communication on a regular topology (such as 1D, 2D, or 3D grid/torus, tree, stencils)

Logicalization exploits the regularity

- Identify the logical topology
- Convert the family of physical traces to a single logical trace
An Example – 16-process BT benchmark

P0: Send (7, data), P1: Send (4, data) P2:: Send (5, data) 
,... P15: Send (2, data) ➔

Send (SW, data) in the context of a 2D torus topology
Execution Traces

Trace 0                    ……… Trace 15

MPL_Send(... 1, MPL_DOUBLE, 480, ...)
MPL_Recv(... 3, MPL_DOUBLE, 480, ...)
MPL_Wait() /* wait for Isend */
MPL_Wait() /* wait for Irecv */

………

MPL_Send(... 4, MPL_DOUBLE, 480, ...)
MPL_Recv(...12, MPL_DOUBLE, 480, ...)
MPL_Wait() /* wait for Isend */
MPL_Wait() /* wait for Irecv */

………

MPL_Send(... 7, MPL_DOUBLE, 480, ...)
MPL_Recv(...13, MPL_DOUBLE, 480, ...)
MPL_Wait() /* wait for Isend */
MPL_Wait() /* wait for Irecv */

………

Logicalization

Steps

Communication

Matrix

<table>
<thead>
<tr>
<th>Pi</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
<th>P9</th>
<th>P10</th>
<th>P11</th>
<th>P12</th>
<th>P13</th>
<th>P14</th>
<th>P15</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>R2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>R3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Graph Matching

Template Topologies

Single Logical Trace

………

MPL_Send(... EAST, MPL_DOUBLE, 480, ...)
MPL_Recv(... WEST, MPL_DOUBLE, 480, ...)
MPL_Wait() /* wait for Isend */
MPL_Wait() /* wait for Irecv */

………

MPL_Send(... SOUTH, MPL_DOUBLE, 480, ...)
MPL_Recv(... NORTI, MPL_DOUBLE, 480, ...)
MPL_Wait() /* wait for Isend */
MPL_Wait() /* wait for Irecv */

………

MPL_Send(... SOUTHWEST, MPL_DOUBLE, 480, ...)
MPL_Recv(... NORTHEAST, MPL_DOUBLE, 480, ...)
MPL_Wait() /* wait for Isend */
MPL_Wait() /* wait for Irecv */

………
Logicalization: Algorithmic Challenge

Identification of topology from communication matrix: compare 2 graph descriptions

Are these the same/isomorphic graphs?

- Graph isomorphism/matching problem
- No known polynomial algorithm
Graph Matching: Process Numbering Matters

Previous work assumes a known simple numbering
Problem: Given a communication matrix (graph) H. Does it match one of known graph topologies: G1, G2…Gn?

Insight:
- Proving 2 graphs isomorphic is hard.
- Simple invariants hold for isomorphic graphs. E.g., (obviously) number of edges in isomorphic graphs must be identical! But equal number of edges does not prove isomorphism.

Approach:
- Step 1: Eliminate most topologies from consideration by checking for simple invariants.
- Step 2: Apply graph isomorphism test selectively.
Simple Tests
1. Number of vertices must match
2. Number of edges must match
3. Prime factors for euclidean shapes
   e.g. possible grids for 16 nodes are 16x1, 8x2, 4x4
4. Node degree in descending order must match

Graph Spectrum: Set of eigenvalues of adjacency matrices of isomorphic graphs must be identical

These tests are employed in the above order (simple to complex) to eliminate candidates for isomorphism. 
*They do not prove isomorphism*
Topology Identification Framework
Part 2: Proving Isomorphism

Libraries exist that test for Isomorphism

VFLib 2.0 (from Univ. of Naples) toolkit has VF2, VF, Schmidt & Druffel, Ullmann algorithms implemented
• VF2 algorithm deployed after evaluation. Based on performance and ease of integration.

Proves if the application communication pattern is isomorphic to any known pattern.
Implementation and Experiments

Framework implemented and validated
The topology library populated with
  • Grids and Torus: All sizes and dimensions
  • Common stencils on Grids/Torus (6pt on 2D, 8pt on 2D, …)
  • All to All
  • Binary Tree
  • …
  • Tested with NAS benchmark traces up to 128 nodes
  • Graph spectrum and Graph isomorphism tested with larger synthetic graphs
**Illustration: BT/SP Benchmark**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Processes</th>
<th>Simple Tests</th>
<th>Graph Spectrum Test</th>
<th>Isomorphism Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT/SP</td>
<td>9</td>
<td>3x3 6-p stencil</td>
<td>3x3 6-p stencil</td>
<td>3x3 6-p stencil</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>4x4 6-p stencil</td>
<td>4x4 6-p stencil</td>
<td>4x4 6-p stencil</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td>6x6 6-p stencil</td>
<td>6x6 6-p stencil</td>
<td>6x6 6-p stencil</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4x3x3 torus</td>
<td>6x6 6-p stencil</td>
<td>6x6 6-p stencil</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2x2x3x3 torus</td>
<td>6x6 6-p stencil</td>
<td>6x6 6-p stencil</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>8x8 6-p stencil</td>
<td>8x8 6-p stencil</td>
<td>8x8 6-p stencil</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2x2x2x2x2x2 grid</td>
<td>8x8 6-p stencil</td>
<td>8x8 6-p stencil</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4x2x2x2x2 torus</td>
<td>8x8 6-p stencil</td>
<td>8x8 6-p stencil</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4x4x2x2 torus</td>
<td>8x8 6-p stencil</td>
<td>8x8 6-p stencil</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4x4x4 torus</td>
<td>8x8 6-p stencil</td>
<td>8x8 6-p stencil</td>
</tr>
<tr>
<td></td>
<td>121</td>
<td>11x11 6-p stencil</td>
<td>11x11 6-p stencil</td>
<td>11x11 6-p stencil</td>
</tr>
</tbody>
</table>

- Table shows candidate topologies remaining after each test
- Non-boldface topologies are isomorphic to topology above
Results

For NAS Benchmarks up to 128 nodes

**Functionality:** All topologies identified correctly. However, a special stencil pattern had to be added to the library for one benchmark – CG

**Performance:** Processing time low and dominated by the time to read the trace:
- Graph matching time less than 1 sec. in all cases on an ordinary PC.
- Maximum end-to-end logicalization time 134 seconds for around 24 million records trace (LU)
Focus on Graph Matching Tests

Simple Tests:
• **Effective** in eliminating almost all candidate topologies. For NAS tests, only a few potential matches left
• **Efficient** – always in seconds

Graph Spectrum Test (**Eigenvalue equivalence**)
• Applied after simple tests. Rarely needed
• **Effective**: the candidates were reduced to exactly one
• **Performance**: Depends on nature of the graph. It is an $O(N^3)$ algorithm…
Performance of Graph Spectrum

Chart for 2D grid & 6 point stencil on 2D grid

- Ordered: row/column major numbering
- Unordered: nodes randomly renumbered

Is this test necessary? Rarely used. Performance and memory issues above 1000 nodes. Isomorphism still must be proved.
Performance of VF2 Isomorphism Test

Chart for 2D grid patterns

Fast, some impact of randomness

Number of Processes/Graph Nodes

Time (sec)

percent of nodes reordered:

- 0
- 25
- 50
- 75
- 99
Pathological cases have performance problems.

*Similar case on grid, and without randomness < 1 sec*
Logicalization Notes

Works well in practice!

- Matching only against known patterns, but patterns easy to add and library can be large
  - All n-dim grids or n-ary trees specified in one shot
- Some message exchange not related to main communication pattern observed
  - Ignored with thresholding, only dominant topologies captured
- Multiple/Mixed patterns (equal to subgraph isomorphism) may be future research
Conclusions

• Traces for regular structured parallel codes can be logicalized into a single trace
• Good performance for realistic scenarios
• Simple and VF2 isomorphism tests effective and efficient. Value of graph spectrum test unclear
• Logicalization can be a component of overall summarization of performance
• www.cs.uh.edu/~jaspal    jaspal@uh.edu

Thanks to