Network Theoretic Classification of Parallel Computation Patterns

Sean Whalen*, Sean Peisert, and Matt Bishop

*LBNL and UCDavis Complexity Sciences Center

CACHES, June 2011
Goal

Identify the code that most likely generated an observed pattern of MPI communication, ideally independent of:

- Communicator size
- Architecture
- Datasets
- Parameters
Definitions

- **Graph** - A set of nodes (ranks) connected by edges (MPI calls)
- **Attributed Relational Graph (ARG)** - A graph with data attached to its nodes and/or edges
- **Topology** - Connectivity properties of the graph
- **Adjacency Matrix** - A matrix representation of the topology where rows are source ranks and columns are destination ranks, or vice versa
- **Integrated Performance Monitoring (IPM)** - Low overhead MPI profiling library (unordered)
### Example

<table>
<thead>
<tr>
<th>Node 1</th>
<th>Node 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI.Send(dest=2)</td>
<td>MPI.Send(dest=1)</td>
</tr>
</tbody>
</table>
Example

Node 1

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send(dest=2)</td>
<td></td>
</tr>
</tbody>
</table>

Node 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send(dest=1)</td>
<td></td>
</tr>
</tbody>
</table>

<hent call="MPI_Send" bytes="1000" rank="1" orank="2" count="8" />
<hent call="MPI_Send" bytes="100" rank="1" orank="1" count="20" />
Example

<table>
<thead>
<tr>
<th>Node 1</th>
<th>Node 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Send(dest=2)</td>
<td>MPI_Send(dest=1)</td>
</tr>
</tbody>
</table>

<hent call="MPI_Send" bytes="1000" rank="1" orank="2" count="8" />
<hent call="MPI_Send" bytes="100" rank="1" orank="1" count="20" />

\[
\begin{pmatrix}
1 & 2 & \text{MPI\_Send} & 1000 & 8 \\
2 & 1 & \text{MPI\_Send} & 100 & 20
\end{pmatrix}
\]
Example

\[
\begin{pmatrix}
1 & 2 & \text{MPI\_Send} & 1000 & 8 \\
2 & 1 & \text{MPI\_Send} & 100 & 20
\end{pmatrix}
\]
Example

\[
\begin{pmatrix}
1 & 2 & \text{MPI\_Send} & 1000 & 8 \\
2 & 1 & \text{MPI\_Send} & 100 & 20
\end{pmatrix}
\]

g = \text{DiGraph}()

for (source, target, call, size, repeats) in features:
g.add_edge(source, target, call=call)
Example

\[(1 2 \text{ MPI\_Send} 1000 8)\]
\[(2 1 \text{ MPI\_Send} 100 20)\]

\[g = \text{DiGraph()}\]
\[\text{for (source, target, call, size, repeats) in features:}\]
\[g.\text{add\_edge(source, target, call=call)}\]
A computational dwarf is “a pattern of communication and computation common across a set of applications” (Asanovic06)

- Independent of programming language, numerical method
- Colella04 found seven dwarves: dense linear algebra, sparse linear algebra, spectral methods, \( n \)-body methods, structured grids, unstructured grids, and monte carlo methods
- Asanovic06 found six more: combinational logic, graph traversal, dynamic programming, backtrack and branch/bound, graphical models, finite state machines
<table>
<thead>
<tr>
<th>Code</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACTUS</td>
<td>astrophysics</td>
</tr>
<tr>
<td>FVCAM</td>
<td>atmospheric dynamics</td>
</tr>
<tr>
<td>GTC</td>
<td>particle physics</td>
</tr>
<tr>
<td>HYPERCLAW</td>
<td>gas dynamics</td>
</tr>
<tr>
<td>LBMHD</td>
<td>plasma physics</td>
</tr>
<tr>
<td>MADBENCH</td>
<td>benchmark</td>
</tr>
<tr>
<td>MHD</td>
<td>plasma physics</td>
</tr>
<tr>
<td>NAMD</td>
<td>molecular dynamics</td>
</tr>
<tr>
<td>PARATEC</td>
<td>materials science</td>
</tr>
<tr>
<td>PF₂</td>
<td>plasma physics</td>
</tr>
<tr>
<td>PMEMD</td>
<td>molecular dynamics</td>
</tr>
<tr>
<td>PSTG3R</td>
<td>atomic physics</td>
</tr>
<tr>
<td>SUPERLU</td>
<td>linear equation solver</td>
</tr>
<tr>
<td>SU(3)</td>
<td>lattice gauge theory</td>
</tr>
</tbody>
</table>
Datasets

- 202 logs
- 31 gigabytes
- 32-512 nodes (some)
- 2 architectures (some)
- Collected at NERSC (thanks Scott Campbell/David Skinner)
cactus (64) - Scaling
cactus (256) - Scaling
fvcam (64) - Similarity
namd (64) - Input Dependence
namd (64) - Input Dependence
maestro (512) - IBM iDataPlex
maestro (512) - Cray XE6
maestro (512) - Augmented Topology
Insufficient Methods

- Node degree distribution
- Betweenness centrality distribution
- Eigenvalue distribution
- Graph isomorphism testing
Isomorphism Testing

**Definition**

Graphs $G$ and $H$ are **isomorphic** if they are structurally equivalent. Unknown if P or NP-complete.

**Definition**

$G$ is **subgraph isomorphic** to $H$ if some subgraph of $G$ is structurally equivalent to $H$. NP-complete via maximum clique.

- Enables comparison of patterns from different communicator sizes
- VF2 algorithm: time complexity $\mathcal{O}(N!N)$
- ARG vastly reduces state space
Subgraph Isomorphism Tests
Goodness-of-Fit Tests

Definition
A two-sample goodness-of-fit test determines if two distributions \( P \) and \( Q \) were generated by the same underlying distribution. We use the Kolmogorov-Smirnov (KS) test.

Definition
The null hypothesis \( H_0 \) assumes \( P \) and \( Q \) come from the same distribution and attributes differences to chance. The alternative hypothesis \( H_a \) assumes they are different.

Definition
We reject the null hypothesis at significance level \( \alpha \) when we are no longer confident differences are due to chance.
Distribution of MPI calls relative to each rank:

<table>
<thead>
<tr>
<th>Code</th>
<th>Rank</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>50% Send, 50% Recv</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>54% Send, 46% Recv</td>
</tr>
</tbody>
</table>

Repeat for each rank and classify the programs as equal if some threshold of ranks have equal call distributions.
The D-Statistic
Distribution of the D-Statistic

![Graph showing the distribution of the D-statistic with cumulative probability on the y-axis and D statistic on the x-axis.]
Subgraph Isomorphism Tests - 17m40s
KS Tests
KS Tests - 23s
Motivated by security, inferring the latent computation can detect:

- Authorized users running unauthorized codes
- Unauthorized users running potentially malicious codes
Consider an algorithm implemented on a general purpose CPU and later ported to a GPU.

- Have algorithms with similar patterns on the CPU been successfully ported to accelerators in the past? (suitability)
- How close is the pattern of the new implementation to those of existing accelerator implementations? (validation)
- Can we distinguish CPU and accelerator implementations? (validation)
Communication patterns are structured and many (not all) can be classified despite changes in communicator size, architecture, datasets, and parameters.

IPM enables low overhead logging of these communications.

Hypothesis testing is fast and accurate, but is not robust to informed adversaries.

Future work: approximate graph matching and motif distributions.