Understanding and Optimizing Communication Performance on HPC Networks

Contributors: Nikhil Jain, Abhinav B hatele, Todd Gamblin, Xiang Ni, Michael Robson, Bilge Acun, Laxmikant Kale

University of Illinois at Urbana-Champaign
http://charm.cs.illinois.edu/~nikhil/
Communication in HPC

- A necessity, but can be viewed as an overhead
- Can consume half the execution time
Communication in HPC

**Complex interplay** of several components: hardware, configurable network properties, interaction patterns, algorithms…

As a user, **limited control** over environment and interference

As an admin, how to **best use the system** while keeping users happy
Communication in HPC

**Complex interplay** of several components: hardware, configurable network properties, interaction patterns, algorithms...

As a user, **limited control** over environment and interference

As an admin, how to **best use the system** while keeping users happy

*Diverse apps*  
*Many systems*
Topology Aware Mapping

• Profile applications for their communication graphs and map them

• Extremely important for Torus-based systems; ongoing work on other topologies
Topography Aware Mapping

- Profile applications for their communication graphs and map them
- Extremely important for Torus-based systems; ongoing work on other topologies
- Use Case: OpenAtom
Rubik - Python based tool to create maps

Figure 1: Mapping 2D sub-partitions to 3D shapes in Rubik

```python
app = box([9,3,8])
app.tile([9,3,1])
app.map()

network = box([6,6,6])
network.tile([3,3,3])
network.map()
```

There are several tools and libraries that provide utilities for communication analysis and performance debugging. Rubik is a Python-based tool that supports permutation operations to reorder ranks within partitions. The tool can also be used to map low-dimensional sub-partitions, like planes, into compact, high-bandwidth shapes on the application's process grid (a cuboid) and a network with mapped application ranks regardless of their dimensions. This means we can easily map low-dimensional sub-partitions using the `map()` function, resulting in eight sub-planes or sub-cubes in the network.

The tool supports permutation operations that can be applied hierarchically, allowing users to group communicating tasks. These operations for optimizing latency or bandwidth can also be applied to rank assignment. This process is tedious and error-prone, especially for large applications with many tasks and high-dimensional networks. We developed Rubik to provide many operations like figure 1, which shows the mapping of 2D sub-partitions to 3D shapes.

There are several factors to consider when trying out different mappings: partitioning operations can also be applied hierarchically, and communicating tasks use the network. Thus, the user can easily map low-dimensional sub-partitions using only a few lines of Python code. It supports a wide range of permutation operations for optimizing latency or bandwidth, and we developed Rubik to provide many operations like figure 1, which shows the mapping of 2D sub-partitions to 3D shapes.

II. BACKGROUND

A. Performance debugging

Performance data obtained from profiling tools can be used to determine if communication is a scaling bottleneck. As a rule of thumb, if an application spends less than 5% of its time in communication when using a large number of processes, there is little room for improving the messaging on the architecture in question. This is especially true for representative input problems (weak or strong scaling) on the Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Application scientists are often unaware of the reason(s) they have for performance issues with their codes. It is important to consider the following steps is broken down further and explained in detail below.

B. Performance optimization

As we will see in the application experience with optimizing production applications on the IBM Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.

We present a step-by-step methodology to improve application performance using task mapping based on our experience with optimizing production applications on the IBM Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.

C. Application performance using task mapping

As a rule of thumb, if an application spends less than 5% of its time in communication when using a large number of processes, there is little room for improving the messaging on the architecture in question. This is especially true for representative input problems (weak or strong scaling) on the Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.

We present a step-by-step methodology to improve application performance using task mapping based on our experience with optimizing production applications on the IBM Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.

We present a step-by-step methodology to improve application performance using task mapping based on our experience with optimizing production applications on the IBM Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.

We present a step-by-step methodology to improve application performance using task mapping based on our experience with optimizing production applications on the IBM Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.

We present a step-by-step methodology to improve application performance using task mapping based on our experience with optimizing production applications on the IBM Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.

We present a step-by-step methodology to improve application performance using task mapping based on our experience with optimizing production applications on the IBM Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.

We present a step-by-step methodology to improve application performance using task mapping based on our experience with optimizing production applications on the IBM Blue Gene/Q architecture. There are three steps involved in this process: 1) Performance debugging via profiling, 2) Performance optimization via task mapping, and 3) Performance analysis via profiling and visualization. Each of these steps is broken down further and explained in detail below.
Rubik - Python based tool to create maps

MILC: Time spent in MPI calls on 4,096 nodes

pF3D: Time spent in MPI calls on 4,096 nodes

Different mappings

mappings
Understanding Networks
Understanding Networks

• What *determines* communication performance?
  • How can we predict it?
  • Quantification of metrics
Understanding Networks

- What **determines** communication performance?
  - How can we predict it?
  - Quantification of metrics

- What is the **relation** between performance and the entities quantified above?
  - Linear, higher polynomial, or indeterminate
  - Is statistical data related to performance?
Understanding Networks

- What **determines** communication performance?
  - How can we predict it?
  - Quantification of metrics

- What is the **relation** between performance and the entities quantified above?
  - Linear, higher polynomial, or indeterminate
  - Is statistical data related to performance?

- Method 1: **Supervised Learning**
  - More on this in Abhinav’s talk
Method 2: Packet-level Simulation
Method 2: Packet-level Simulation

- Detailed study of what-if scenarios
- Comparison of similar systems
Method 2: Packet-level Simulation

- Detailed study of what-if scenarios
- Comparison of similar systems
- BigSim was among the earliest accurate packet-level HPC network simulator (circa 2004)

- Reviving Emulation and Simulation capabilities of BigSim

- BigSim + CODES + ROSS = TraceR
  - More on this in the Bilge’s talk
Method 3: Modeling via Damselfly

Intermediate methods sufficient to answer certain types of questions

Q1: What is the best combination of routing strategies and job placement policies for single jobs?

Q2: What is the best combination for parallel job workloads?

Q3: Should the routing policy be job-specific or system-wide?
Dragonfly Topology

Level 1: Dense connectivity among routers to form groups

IBM PERCS

CRAY ARIES/XC30

All-to-all network in columns: Rank 1

Chassis (All-to-all network in rows: Rank 2)
Dragonfly Topology

Level 2: Dense connectivity among groups as virtual routers

IBM PERCS

CRAY ARIES/XC30

Rank-3 all-to-all network (not all groups or links are shown)
What needs to be evaluated?

<table>
<thead>
<tr>
<th>Job Placement</th>
<th>Routing</th>
<th>Comm Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Nodes (RDN)</td>
<td>Static Direct (SD)</td>
<td>UnStructured</td>
</tr>
<tr>
<td>Random Routers (RDR)</td>
<td>Static Indirect (SI)</td>
<td>2D Stencil</td>
</tr>
<tr>
<td>Random Chassis (RDC)</td>
<td>Adaptive Direct (AD)</td>
<td>4D Stencil</td>
</tr>
<tr>
<td>Random Group (RDG)</td>
<td>Adaptive Indirect (AI)</td>
<td>Many-to-many</td>
</tr>
<tr>
<td>Round Robin Nodes (RRN)</td>
<td>Adaptive Hybrid (AH)</td>
<td>Spread</td>
</tr>
<tr>
<td>Round Robin Routers (RRR)</td>
<td>Job-specific (JS)</td>
<td>Parallel Workloads (4)</td>
</tr>
</tbody>
</table>

Total cases ~ 360 for 8.8 million cores with 92,160 routers
Model for link utilization

• Input to the model:

  1. Network graph of Dragonfly routers

  2. Application communication graph for a communication step

  3. Job placement

  4. Routing strategy

• Output: The steady-state traffic distribution on all network links, which is representative of the network throughput

• Implemented as a scalable parallel MPI program executed on Blue Gene/Q
  — Maximum runtime of 2 hours on 8,192 cores for prediction on 8.8 million cores
• Initialize two copies of network graph N:
  \( N_{\text{Alloc}} \): stores total and per message allocated bandwidth (\( = 0 \))
  \( N_{\text{Remain}} \): stores bandwidth available for allocation (\( = \) capacity)
• Initialize two copies of network graph $N$:
  $N_{Alloc}$: stores total and per message allocated bandwidth ($= 0$)
  $N_{Remain}$: stores bandwidth available for allocation ($= capacity$)

• Iterative solve for computing representative state $N_{Alloc}$
  while a message is allocated additional bandwidth
  • for each message $m$, obtain the list of paths $P(m)$

Start with 10 GB/s per link
• Initialize two copies of network graph $N$:
  $N_{\text{Alloc}}$: stores total and per message allocated bandwidth ($= 0$)
  $N_{\text{Remain}}$: stores bandwidth available for allocation ($= \text{capacity}$)

• Iterative solve for computing representative state $N_{\text{Alloc}}$
  while a message is allocated additional bandwidth
  • for each message $m$, obtain the list of paths $P(m)$

Start with 10 GB/s per link
• Initialize two copies of network graph \( N \):
  \( N_{Alloc} \) : stores total and per message allocated bandwidth (\( = 0 \))
  \( N_{Remain} \) : stores bandwidth available for allocation (\( = \) capacity)

• Iterative solve for computing representative state \( N_{Alloc} \)
  while a message is allocated additional bandwidth
  • for each message \( m \), obtain the list of paths \( P(m) \)
  • using \( P(m) \) of all messages, find the request count for each link

Start with 10 GB/s per link
Initialize two copies of network graph $\mathbf{N}$:
- $\mathbf{N}_{\text{Alloc}}$: stores total and per message allocated bandwidth ($= 0$)
- $\mathbf{N}_{\text{Remain}}$: stores bandwidth available for allocation ($= \text{capacity}$)

Iterative solve for computing representative state $\mathbf{N}_{\text{Alloc}}$
while a message is allocated additional bandwidth
- for each message $m$, obtain the list of paths $P(m)$
- using $P(m)$ of all messages, find the request count for each link
Initialize two copies of network graph $N$:
- $N_{\text{Alloc}}$: stores total and per message allocated bandwidth ($= 0$)
- $N_{\text{Remain}}$: stores bandwidth available for allocation ($= \text{capacity}$)

**Iterative solve for computing representative state $N_{\text{Alloc}}$**

while a message is allocated additional bandwidth
- for each message $m$, obtain the list of paths $P(m)$
- using $P(m)$ of all messages, find the request count for each link
- for each path $p$ in $P(m)$, compute its availability using $N_{\text{Remain}}$
• Initialize two copies of network graph $N$:
  $N_{Alloc}$: stores total and per message allocated bandwidth ($= 0$)
  $N_{Remain}$: stores bandwidth available for allocation (= capacity)

• Iterative solve for computing representative state $N_{Alloc}$
  while a message is allocated additional bandwidth
  • for each message $m$, obtain the list of paths $P(m)$
  • using $P(m)$ of all messages, find the request count for each link
  • for each path $p$ in $P(m)$, compute its availability using $N_{Remain}$

Start with 10 GB/s per link
• Initialize two copies of network graph \( N \):
  - \( N_{\text{Alloc}} \): stores total and per message allocated bandwidth (\( = 0 \))
  - \( N_{\text{Remain}} \): stores bandwidth available for allocation (\( = \) capacity)

• Iterative solve for computing representative state \( N_{\text{Alloc}} \)
  while a message is allocated additional bandwidth
  • for each message \( m \), obtain the list of paths \( P(m) \)
  • using \( P(m) \) of all messages, find the request count for each link
  • for each path \( p \) in \( P(m) \), compute its availability using \( N_{\text{Remain}} \)
  • using availability, allocate more bandwidth to the messages

Start with 10 GB/s per link

\[\begin{array}{ccc}
S & P1 & D \\
1 & 10 & 2 \\
2 & 2 & 3 \\
5 & 3 & 3.33
\end{array}\]
• Initialize two copies of network graph \( N \):
  \( N^{\text{Alloc}} \): stores total and per message allocated bandwidth (\( = 0 \))
  \( N^{\text{Remain}} \): stores bandwidth available for allocation (\( = \) capacity)

• Iterative solve for computing representative state \( N^{\text{Alloc}} \)
  while a message is allocated additional bandwidth
  • for each message \( m \), obtain the list of paths \( P(m) \)
  • using \( P(m) \) of all messages, find the request count for each link
  • for each path \( p \) in \( P(m) \), compute its availability using \( N^{\text{Remain}} \)
  • using availability, allocate more bandwidth to the messages
  • update \( N^{\text{Alloc}} \) and \( N^{\text{Remain}} \) to reflect the new allocations
• Initialize two copies of network graph $N$:
  $N_{Alloc}$: stores total and per message allocated bandwidth ( = 0)
  $N_{Remain}$: stores bandwidth available for allocation (= capacity)

• Iterative solve for computing representative state $N_{Alloc}$
  while a message is allocated additional bandwidth
  • for each message $m$, obtain the list of paths $P(m)$
  • using $P(m)$ of all messages, find the request count for each link
  • for each path $p$ in $P(m)$, compute its availability using $N_{Remain}$
  • using availability, allocate more bandwidth to the messages
  • update $N_{Alloc}$ and $N_{Remain}$ to reflect the new allocations

• Use $N_{Alloc}$ to compute the distribution of bytes on the given links

Start with 10 GB/s per link

![Network Graph Diagram]
How to read the plots?

Example Plot

Job placements grouped based on Routing

- Minimum
- First quartile
- Average
- Median
- Third quartile
- Maximum
- Lowest maximum

Minimum and first quartile are same
How to read the plots?

Example Plot

- **Minimum**
- **1st Quartile**
- **Average**
- **3rd Quartile**
- **Maximum**
- **Lowest Maximum**

Job placements grouped based on Routing

Maximum traffic on any link: indicates network hotspot
How to read the plots?

Example Plot

- **Minimum**
- **1st quartile**
- **Median**
- **Average**
- **3rd quartile**
- **Maximum**
- **Lowest maximum**

Average traffic on all links: indicates relative merit

Job placements grouped based on Routing
How to read the plots?

Median traffic: valuable for estimating distribution by comparing with the average.
How to read the plots?

Example Plot

Ideal: distribution with close values for all data points, lower the better

Job placements grouped based on Routing
Single job: Unstructured Mesh
6-20 partners with 512 KB messages

Job placement: blocking reduces the maximum (up to 90% drop) and average (up to 92% drop)
Single job: Unstructured Mesh
6-20 partners with 512 KB messages

Indirect routing: increases average, but reduces maximum by 50% in the best case
Adaptivity: similar distribution as static, but with lower maximum
AI leads to 50% reduction in maximum traffic; hybrid does worse than AI

Single job: Unstructured Mesh
6-20 partners with 512 KB messages
Single job: Random Neighbors
6-20 partners with 512 KB messages

Job placement: negligible impact!
Indirect routing: shifts the graph upwards and increases all quartiles; 100% increase in maximum and average.
Single job: Random Neighbors
6-20 partners with 512 KB messages

Adaptivity: Minor gains, 10% reduction in maximum hybrid does better than AI
## Parallel Workloads: % Core Distribution

<table>
<thead>
<tr>
<th>Comm Pattern</th>
<th>Workload 1</th>
<th>Workload 2</th>
<th>Workload 3</th>
<th>Workload 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unstructured Mesh</td>
<td>20</td>
<td>10</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>2D Stencil</td>
<td>10</td>
<td>10</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>4D Stencil</td>
<td>40</td>
<td>20</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Many to many</td>
<td>20</td>
<td>40</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Random neighbors</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>
Workloads

- Adaptivity reduces the maximum traffic by 35%

- Hybrid with RDN/RDR shows lowest data points
Job-specific Routing

![Graph showing link usage (MB) for Workload 2 and Workload 4 with median, average, and lowest maximum values.](image)

- Median
- Average
- Lowest maximum (Workload 2)
- Lowest maximum (Workload 4)
Summary
Summary

• Fast analytical model enables studies with a large number of scenarios
Summary

• Fast analytical model enables studies with a large number of scenarios

• Adaptivity results in significantly lower values for maximum and average traffic (up to 50% reduction)
Summary

• Fast analytical model enables studies with a large number of scenarios

• Adaptivity results in significantly lower values for maximum and average traffic (up to 50% reduction)

• **Q1.** What is the best combination for single job runs?
  
  • Depends on the job being run!
  
  • Patterns with communication among near-by MPI ranks benefit by blocking
  
  • Indirect routing is better when the communication pattern is not sufficiently spread by the application or job placement
  
  • Hybrid routing provides similar distribution as Adaptive Indirect, but its data points are shifted depending on the communication pattern
Summary
Summary

• **Q2.** What is the best combination for parallel workloads?

  - Similar distributions are observed irrespective of the jobs proportions in the workloads!
  
  - Adaptive Hybrid combines the best of both worlds
  
  - Randomized placement with node/router based blocking is good
Summary

• **Q2.** What is the best combination for parallel workloads?
  • Similar distributions are observed irrespective of the jobs proportions in the workloads!
  • Adaptive Hybrid combines the best of both worlds
  • Randomized placement with node/router based blocking is good

• **Q3.** Is it beneficial to use job-specific routing?
  • Yes, provides similar distribution as the best routing while reducing the values of the data points such as the maximum
Relevant publications

• Predicting application performance using supervised learning on communication features. SC 2013.

• Mapping to Irregular Torus Topologies and Other Techniques for Petascale Biomolecular Simulation. SC 2014.

• Maximizing Network Throughput on the Dragonfly Interconnect. SC 2014.

• Improving Application Performance via Task Mapping on IBM Blue Gene/Q. HiPC 2014.

• Identifying the Culprits behind Network Congestion. IPDPS 2015.