Topology-Aware Resource Selection

Adèle Villiermet (Inria), Guillaume Mercier (PIB), David Glesser (Bull), Yiannis Georgiou (Bull)
1
Runtime System and the Inria Runtime Team
Software Stack

- Applications
- Programming models
- Compilers
  - Libraries
  - Runtime systems
  - Operating systems
  - Hardware

- Optimize Computational Kernels
- Dynamic optimization

- Enable and express parallelism
- Give abstraction of the parallel machine
- Static optimization
- Parallelism extraction
- Hardware abstraction
- Basic services
Runtime System

- Scheduling
- Parallelism orchestration (Comm. Synchronization)
- I/O
- Reliability and resilience
- Collective communication routing
- Migration
- Data and task/process/thread placement
- etc.
Enable performance portability by improving interface expressivity

Success stories:

- MPICH 2 (Nemesis Kernel)
- KNEM (enabling high-performance intra-node MPI communication for large messages)
- StarPU (unified runtime system for CPU and GPU program execution)
- HWLOC (portable hardware locality)

Scotch graph partitioner (Mercato, transfer of F. Pellegrini)
2

Topology-Aware Data Management
As already said, in modern parallel machines:
- Flops are no longer the main issue
- Bytes feature more problems (energy and time):
  - I/O
  - Transfer
  - Storage
  - Allocation
  - Access
The application and its ecosystem

- Applications
- Programming models
- Compilers
- Libraries
- Runtime systems
- Operating systems
- Hardware

SW stack

Batch scheduler

Storage
Optimizing execution

Once the application has been written in **nice language** with **performing libraries** and **efficient data layout** there is still room for optimizations:

- Not everything is known at compile time
- allocated resources
- input data

Our goal: take the application as it is and **optimize its execution on its ecosystem** (sw stack + batch scheduler + storage system + …)
Not that simple…

Execution is not always completely decoupled from application design.

Need for information exchange between application and runtime system.
Big Picture

Model of the machine

Model of the application

Optimization algorithm

Optimized execution
Model of machine (within nodes)

**HWLOC** (portable hardware locality)
- Runtime and OpenMPI team
- portable abstraction (across OS, versions, architectures, ...)
- Hierarchical topology
- Modern architecture (NUMA, cores, caches, etc.)
- ID of the cores
- C library to play with
- etc.

On going dev. in the team need for:
- dynamic information topology information (within and between nodes)
- API?
- Less bug in the BIOS
Model of the machine (network): Netloc

hwloc companion
Takes care of network topology
and joins hwloc and network information
• Global « map » of your cluster
  Connects hwloc objects to network edges
Public API made of
• Network queries (nodes, edges, etc.)
• Global map queries
• hwloc API when looking inside servers

Currently developed by
• University of Wisconsin-LaCrosse (J. Hursey)
• Inria (B. Goglin)
• Cisco (J. Squyres)
• under the umbrella of the Open MPI consortium
Netloc global map

Server #3
with single-port IB HCA

Server #2
with dual-port IB HCA

Server #1
with dual-port IB HCA

Network #3 (Eth VLAN 67)

Network #2 (IB 3333)

Network #1 (IB 2222)

Server #4
(admin node with no IB HCA)
Multirail/multipath Locality

Rail #0 - Longer path

Rail #1 - Shortest path
Netloc portability

Trying to be as generic as possible
• More than just IB fat-trees
• No need to run proprietary scripts anymore

Existing backends
• InfiniBand
• Ethernet
  Through OpenFlow for now
  Maybe SNMP/LLDP for small clusters one day?

Upcoming Cray Gemini and Aries support?
Application model

We target data access.

We need affinity between processing elements: communication pattern
Building the communication pattern

- Statically (thanks to compiler)
- Dynamic Monitoring
- Blank execution and tracing
- After data partitioning (e.g. Scotch)
Putting everything together: Process Placement with TreeMatch

\[ \sigma = (0, 2, 8, 10, 4, 6, 12, 14, 1, 3, 9, 11, 5, 7, 13, 15) \]
3

Resource selection
Selecting Resources

Model of the machine

Model of the application

TreeMatch Algorithm in the batch scheduler
Implementation

• Within SLURM (in collaboration with BULL)
• Plugins
• Resource selection and process placement at the same time
Why topology-aware resource selection could work?

<table>
<thead>
<tr>
<th>Procs</th>
<th>0-7</th>
<th>8-15</th>
<th>16-23</th>
<th>24-31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0</td>
<td>1000</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>8-15</td>
<td>1000</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>16-23</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>24-31</td>
<td>20</td>
<td>0</td>
<td>1000</td>
<td>0</td>
</tr>
</tbody>
</table>

SLURM

```
+---+---+---+---+
|   |   |   |   |
| n0| n1| n2| n3 |
|   |   |   |   |
+---+---+---+---+
|   |   |   |   |
| n4| n5| n6| n7 |
|   |   |   |   |
+---+---+---+---+
```

```text
<table>
<thead>
<tr>
<th>n0</th>
<th>n1</th>
<th>n2</th>
<th>n3</th>
<th>n4</th>
<th>n5</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0,p4,p8</td>
<td>p1,p5,p9</td>
<td>p13,p17,p21</td>
<td>p2,p6,p10</td>
<td>p3,p7,p11</td>
<td></td>
</tr>
<tr>
<td>p24,p28</td>
<td>p26,p30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Why topology-aware resource selection could work?

SLURM Then TreeMatch

<table>
<thead>
<tr>
<th>Procs</th>
<th>0-7</th>
<th>8-15</th>
<th>16-23</th>
<th>24-31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0</td>
<td>1000</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>8-15</td>
<td>1000</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>16-23</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>24-31</td>
<td>20</td>
<td>0</td>
<td>1000</td>
<td>0</td>
</tr>
</tbody>
</table>
Why topology-aware resource selection could work?

SLURM and TreeMatch

<table>
<thead>
<tr>
<th>Procs</th>
<th>0-7</th>
<th>8-15</th>
<th>16-23</th>
<th>24-31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-7</td>
<td>0</td>
<td>1000</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>8-15</td>
<td>1000</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>16-23</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>24-31</td>
<td>20</td>
<td>0</td>
<td>1000</td>
<td>0</td>
</tr>
</tbody>
</table>
Early experiments

Same protocol as SLURM/Bull team.

Simulation using real traces of the Curie CEA machine: 80640 cores.

Model of performance gain of TreeMatch depending on the amount of communication performed by application (10%, 30%, 50%).

Same starting workflow:
- 130 running jobs
- 26 queued jobs
- 372 submitted jobs (1 hour)

Evaluation on the difference of the submitted jobs.
Simulation: makespan

Makespan

Ratio to SLURM

SLURM then TM

SLURM and TM

Percentage of Communication of the Application Runtime
Simulation: average stretch

![Graph showing the relationship between Percentage of Communication of the Application Runtime and Average Stretch. The graph displays two lines: SLURM then TM in red and SLURM\textsuperscript{T}M in green. The y-axis represents the ratio to SLURM Avg Stretch, ranging from 1.19405 to 1.19435. The x-axis shows the percentage of communication, ranging from 10 to 50%.

SLURM then TM
SLURM\textsuperscript{T}M

September 5, 2014
Simulation: max flow

Max Flow

Ratio to SLURM

Percentage of Communication of the Application Runtime

SLURM then TM
SLURM and TM
Conclusion
Take Away Message

**Locality!** Bytes are more important than flops

Not everything can be optimized statically at compile time

Need for runtime topology-aware data management

Need to take into consideration the whole application ecosystem such as the storage or the batch scheduler