Process Mapping on any Topologies with TopoMatch
Emmanuel Jeannot
March 4, 2021
Process Placement Background
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange

Local RAM

Core 1  Core 2  Core 3  Core 4
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange
The Topology is not Flat

The higher we have to go into the hierarchy the costly the data exchange

The network can also be hierarchical!
Communication Pattern

Shared memory system:
• The amount of data shared by threads vary

Distributed memory system:
• The amount of data exchanged between processes vary

The time spent to exchange data depends on the thread/process mapping
Process Placement Problem
Process Placement Problem

Model of the machine
Process Placement Problem

Model of the machine

Model of the application
Process Placement Problem

Model of the machine

Mapping algorithm

Model of the application
Process Placement Problem

Model of the machine

Mapping algorithm

Model of the application
TreeMatch
Project started in 2009

Many contributors:

- Guillaume Mercier (intégration dans Open MPI)
- François Tessier (LB, constraints)
- Adèle Viliermet (Batch scheduler)
- Pierre Celor (Partitionning algorithm)
- Fatima El-Akkary (SW eng., noise analysis)
- Thibaut Lausecker (Scotch Interface)
- Laurent Dutertre (Preliminary XP)
TreeMatch Basic Algorithm

C: communication matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>10</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>1000</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>1000</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1000</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1000</td>
<td>10</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>1000</td>
<td>0</td>
<td>1000</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>100</td>
<td>1</td>
<td>10</td>
<td>1000</td>
<td>0</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1000</td>
<td>1</td>
<td>1</td>
<td>1000</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Grouped matrix

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>102</th>
<th>202</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1012</td>
<td>0</td>
<td>4</td>
<td>202</td>
</tr>
<tr>
<td>102</td>
<td>4</td>
<td>0</td>
<td>1012</td>
<td></td>
</tr>
<tr>
<td>202</td>
<td>202</td>
<td>1012</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Communication matrix + Tree Topology = Process permutation
Dealing with Constraints

Problem:
- Given a hierarchical topology
- An already mapped application onto a subset of the nodes
- Reorder process while ensuring only this subset is used

Solution:
- Extend the communication matrix with dummy nodes
- Process the tree backward by doing k-partitionning
- Force each partition to have the right number of dummy nodes
- Process recursively
Use-Cases
Use-Case 1: Process Mapping

MiniGhost Application (Stencil)

Group by Number of proc, Number of variables and affinity metric type

Gain (avg values)

Grid size and stencil type

Gain (avg values)
Use-Case 2: Rank reordering

1. Gather communication pattern
2. Compute new mapping
3. Change communicator
4. Exchange data
5. Continue computation with new communicator

Case of Conjugate Gradient (CG –NAS).
Topology-Aware Load Balancing

Before LB

After LB

After topology-aware LB

Implemented within Charm++

Francois Tessier

TreeMatch in Charm++

Implemented within Charm++
Batch Scheduling

1. Gather pattern before submitting job
2. Use TreeMatch to allocate resources to the job

![Graphical Representation]
TopoMatch
TreeMatch is limited to Tree Topology

Scotch (https://gitlab.inria.fr/scotch/scotch): a software package for

- graph and mesh/hypergraph partitioning,
- graph clustering
- sparse matrix ordering

Lift this limitation:

- Scotch already used in TreeMatch: core graph partitionning
- Scotch manage different type of architectures
  - Decomposition-defined (deco)
  - Specific (Mesh, hypercube, tleaf), etc.
Topomatch: Managing Scotch in TreeMatch

Same interface and same set of features:

• If standard tree topology: use TreeMatch
• If other topologies: use Scotch

Important features:

• Any kind of topology (including Hwloc)
• Manage constraints
• Manage oversubscribing
• Different evaluation metric (Hope-Byte, Sum-Com, Max-Com)
• Optional exhaustive search
• Fast mapping (multithreaded)
• Fast I/O
• Nice verbosity management
Using Scotch

**With constraints**

C : constraint
T : The Scotch topology target
m : The communication matrix

SCOTCH_archInit(sub_arch);
SCOTCH_archSub(sub_arch T, |C|, C);
local_sol ← scotch_partitioning(sub_arch, m);

// Renumber solution to change frame of reference;
foreach i in 0..|C| - 1 do
   global_sol[i] ← C[local_sol[i]];

**Without constraints**

T : The Scotch topology target
m : The communication matrix

graph ← com_mat_to_scotch_graph(m, |T| × sparse_factor);

strat ← set_scotch_strategy(SCOTCH_STRATBALANCE);
partition ← SCOTCH_ComputeMapping (graph, T, strat);
Bucket grouping (group of size 2)
Bucket grouping (group of size 2)

Sample communication matrix values
Bucket grouping (group of size 2)

\[ K+1 \text{ pivot}: +\infty \quad \text{p1} \quad \text{p2} \quad \ldots \quad \text{pk}=0 \]
Bucket grouping (group of size 2)

Sample communication matrix values

K+1 pivot: $+\infty$  p1  p2  ...  pk=0
Bucket grouping (group of size 2)

Sample communication matrix values

K+1 pivot: $+\infty$  p1  p2  ...  pk=0

Spread communication matrix input
Bucket grouping (group of size 2)

Sample communication matrix values

Algorithm
• Sort bucket 1
• Find independent group stating from largest values of bucket 1
• If not enough groups: sort bucket 2
• Etc.

Gain need to sort a few buckets instead of all values

K+1 pivot: +∞ p1 p2 ... pk=0
buckets only if we have not already selected the
that states that every maximal independent set is maximum,
we use the property of this particular kind the Kneser graph
this graph using the order given by the sort procedure, and
the complement of such a graph: we look at the edges of
procedure is to find a weighted maximum independent set of
processes. As shown in [22], the graph of compatible groups
order: two groups are compatible if they do not share the same
and we select the first compatible groups according to the
pattern. We built all the possible groups we sort these groups,
improve this solution using three strategies that use the same
the default solution of the process mapping. Then, we try to
leaves of the tree. This gives an initial solution which is often
mapping that consists in mapping process to the leftmost
line 11 to line 18 of Algorithm 1). First, we compute a pack

T
Scotch or a greedy partitioner that minimizes the cut.
partitioning strategy (line 9 of Algorithm 1) that uses either
choose the best in terms of communication reduction.
The laziness of this algorithm means that we will process the remaining

TABLE I: Bucket grouping timing. Partial sorting vs. full sorting comparison on an Intel Xeon CPU E5-2680 at 2.50GHz.
Results
Mapping time

Mapping dense communication matrix on a tree topology.

Xeon 6230, 2*768GB, Optane DCPMM.

101 736: half the number of cores of Summit.
Sparse factor : mapping time

Dense communication matrix : too many information for Scotch?
Sparcify (keep only largest value) the input communication matrix.

Fat tree Topology (86400 leaves)
2D Mesh Topology (125000 nodes)
Sparse factor : mapping quality

Emulation : MPI_Alltoallv to execute random communication pattern in function of the sparse factor.

TopoMatch + Scotch: Plafrim 2 (Miriel) results

Conclusion : safe to use SF = 0.5 (default TopoMatch Value)
Impact of the noise

Difficult to have the exact value of the communication matrix.

Four types of noise:

1) $\tilde{M} \leftarrow M + M \times \mathcal{N}(0, k)$ (Gaussian Multiplicative),
2) $\tilde{M} \leftarrow M + \mathcal{N}(0, k)$ (Gaussian Additive),
3) $\tilde{M} \leftarrow M + M \times \mathcal{U}(-k, k)$ (Uniform Multiplicative),
4) $\tilde{M} \leftarrow M + \mathcal{U}(-k, k)$ (Uniform Additive).

Negative entries are truncated to 0.
Impact of noise

Noise increase: TM perf degrades -> RR -> Random

48 node : TM similar to RR for small k.
Conclusion

• Process placement helps in optimizing communication cost of parallel applications
• Useful in many context
• Main abstraction: communication matrix
• TopoMatch: generic tool for arbitrary topologies
Thank you!

Follow us on www.inria.fr