Broadcast Trees for Heterogeneous Platforms

Olivier Beaumont, Yves Robert and Loris Marchal

Laboratoire de l’Informatique du Parallélisme
École Normale Supérieure de Lyon, France

Yves.Robert@ens-lyon.fr
http://graal.ens-lyon.fr/~yrobert

December 2004
Outline

Introduction

Models and Framework

Platform-based Heuristics
  One port-model
  Multi-port

LP-based heuristics

Simulations

Conclusion
Outline

Introduction

Models and Framework

Platform-based Heuristics
  One port-model
  Multi-port

LP-based heuristics

Simulations

Conclusion
Outline

Introduction

Models and Framework

Platform-based Heuristics
  - One port-model
  - Multi-port

LP-based heuristics

Simulations

Conclusion
Outline

Introduction

Models and Framework

Platform-based Heuristics
  One port-model
  Multi-port

LP-based heuristics

Simulations

Conclusion
Outline

Introduction

Models and Framework

Platform-based Heuristics
  One port-model
  Multi-port

LP-based heuristics

Simulations

Conclusion
Outline

Introduction

Models and Framework

Platform-based Heuristics
  One port-model
  Multi-port

LP-based heuristics

Simulations

Conclusion
Outline

Introduction

Models and Framework

Platform-based Heuristics
  One port-model
  Multi-port

LP-based heuristics

Simulations

Conclusion
Broadcasting data

- Key collective communication operation
  - Start: one processor has the data
  - End: all processors own a copy
  - Vast literature about broadcast, MPI_Bcast
  - Standard approach: use a spanning tree
  - Finding the best spanning tree: NP-Complete problem (even in the telephone model)
Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, MPI_Bcast
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)
Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, MPI_Bcast
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)
Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, MPI_Bcast
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)
Broadcasting data

- Key collective communication operation
- Start: one processor has the data
- End: all processors own a copy
- Vast literature about broadcast, MPI_Bcast
- Standard approach: use a spanning tree
- Finding the best spanning tree: NP-Complete problem (even in the telephone model)
Different broadcast problems

Broadcast large messages ⇒ pipelining strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA  Single Tree, Atomic message
heuristics to minimize makespan: FNF...

STP  Single Tree, Pipelined series of messages

MTP  Multiple Tree, Pipelined series of messages

- polynomial algorithm to find optimal solution
  (LP formulation)
- hard to implement ⇒ concentrate on STP
Different broadcast problems

Broadcast large messages $\Rightarrow$ pipelining strategies
- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA  Single Tree, Atomic message
heuristics to minimize makespan: FNF...  

STP  Single Tree, Pipelined series of messages

MTP  Multiple Tree, Pipelined series of messages
- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement $\Rightarrow$ concentrate on STP
Different broadcast problems

Broadcast large messages $\Rightarrow$ **pipelining** strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

**STA** Single Tree, Atomic message
heuristics to minimize makespan: FNF...

**STP** Single Tree, Pipelined series of messages

**MTP** Multiple Tree, Pipelined series of messages
- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement $\Rightarrow$ concentrate on STP
Different broadcast problems

Broadcast large messages ⇒ pipelining strategies
  ▶ split the messages into slices (application level)
  ▶ route them concurrently, possibly using different spanning trees
  ▶ throughput optimization (relaxation of makespan minimization)

STA  Singe Tree, Atomic message
      heuristics to minimize makespan: FNF...  

STP  Single Tree, Pipelined series of messages

MTP  Multiple Tree, Pipelined series of messages
      ▶ polynomial algorithm to find optimal solution
        (LP formulation)
      ▶ hard to implement ⇒ concentrate on STP
Different broadcast problems

Broadcast large messages ⇒ pipelining strategies
▶ split the messages into slices (application level)
▶ route them concurrently, possibly using different spanning trees
▶ throughput optimization (relaxation of makespan minimization)

STA  Single Tree, Atomic message
heuristics to minimize makespan: FNF...

STP  Single Tree, Pipelined series of messages
MTP  Multiple Tree, Pipelined series of messages
▶ polynomial algorithm to find optimal solution
    (LP formulation)
▶ hard to implement ⇒ concentrate on STP
Different broadcast problems

Broadcast large messages ⇒ pipelining strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

STA  Single Tree, Atomic message
     heuristics to minimize makespan: FNF...

STP  Single Tree, Pipelined series of messages

MTP  Multiple Tree, Pipelined series of messages
     polynomial algorithm to find optimal solution (LP formulation)
     hard to implement ⇒ concentrate on STP
Different broadcast problems

Broadcast large messages $\Rightarrow$ **pipelining** strategies

- split the messages into slices (application level)
- route them concurrently, possibly using different spanning trees
- throughput optimization (relaxation of makespan minimization)

**STA** Single Tree, Atomic message
heuristics to minimize makespan: FNF...

**STP** Single Tree, Pipelined series of messages

**MTP** Multiple Tree, Pipelined series of messages

- polynomial algorithm to find optimal solution (LP formulation)
- hard to implement $\Rightarrow$ concentrate on STP
Introduction

Models and Framework

Platform-based Heuristics
- One port-model
- Multi-port

LP-based heuristics

Simulations

Conclusion
Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Models

Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Models

Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Models

Network = directed graph $\mathcal{P} = (V, E)$

- General case: affine model (includes latencies)
- Common variant: sending and receiving processors busy during whole transfer
Banikazemi et al.

no overlap between link and processor occupation:

$\Rightarrow$ methodology to instantiate parameters
Bar-Noy et al.

occupation time of sender $P_u$ independent of target $P_v$

not *fully* multi-port model, but allows for starting a new transfer from $P_u$ without waiting for previous one to finish
Bhat et al.

same parameters for sender $P_u$, link $e_{u,v}$ and receiver $P_v$

Two flavors:

- **bidirectional**: simultaneous send and receive transfers allowed
- **unidirectional**: only one send or receive transfer at a given time-step
Bhat et al.

same parameters for sender $P_u$, link $e_{u,v}$ and receiver $P_v$

Two flavors:

- **bidirectional**: simultaneous send and receive transfers allowed
- **unidirectional**: only one send or receive transfer at a given time-step
One-port

- Bhat et al.
  
  same parameters for sender \( P_u \), link \( e_{u,v} \) and receiver \( P_v \)

Two flavors:

- bidirectional: simultaneous send *and* receive transfers allowed
- unidirectional: only one send or receive transfer at a given time-step
Platform graph \( P = (V, E) \)

Source processor \( P_{source} \)

Goal: broadcast a series of messages to all other nodes

Transfers of successive messages are pipelined

Send messages along a spanning tree

Find a spanning tree with good throughput (neglect initialization and clean-up phases)

Bidirectional one-port model:

Multi-port model:
Platform graph $\mathcal{P} = (V, E)$

Source processor $P_{\text{source}}$

Goal: broadcast a series of messages to all other nodes

Transfers of successive messages are pipelined

Send messages along a spanning tree

Find a spanning tree with good throughput (neglect initialization and clean-up phases)

Bidirectional one-port model:

Multi-port model:
Framework

- Platform graph $\mathcal{P} = (V, E)$
- Source processor $P_{source}$
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput (neglect initialization and clean-up phases)

- Bidirectional one-port model:

- Multi-port model:
Framework

- Platform graph $\mathcal{P} = (V, E)$
- Source processor $P_{source}$
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput (neglect initialization and clean-up phases)
- Bidirectional one-port model:
- Multi-port model:
Framework

- Platform graph $\mathcal{P} = (V, E)$
- Source processor $P_{source}$
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput (neglect initialization and clean-up phases)

- Bidirectional one-port model:

- Multi-port model:
Framework

- Platform graph $\mathcal{P} = (V, E)$
- Source processor $P_{\text{source}}$
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput (neglect initialization and clean-up phases)

- Bidirectional one-port model:

- Multi-port model:
Framework

▶ Platform graph $\mathcal{P} = (V, E)$
▶ Source processor $P_{\text{source}}$
▶ Goal: broadcast a series of messages to all other nodes
▶ Transfers of successive messages are pipelined
▶ Send messages along a spanning tree
▶ Find a spanning tree with good throughput (neglect initialization and clean-up phases)

▶ Bidirectional one-port model:

▶ Multi-port model:
Framework

- Platform graph $\mathcal{P} = (V, E)$
- Source processor $P_{\text{source}}$
- Goal: broadcast a series of messages to all other nodes
- Transfers of successive messages are pipelined
- Send messages along a spanning tree
- Find a spanning tree with good throughput (neglect initialization and clean-up phases)

- Bidirectional one-port model:

- Multi-port model:
Outline

Introduction

Models and Framework

Platform-based Heuristics
   One port-model
   Multi-port

LP-based heuristics

Simulations

Conclusion
Processors involved in one (sending or receiving) communication

Duration of a transfer $= f(\text{link } e_{u,v})$

$$send_{u,v}(L) = recv_{u,v}(L) = T_{u,v}(L) = T_{u,v}.$$
Processors involved in one (sending or receiving) communication

Duration of a transfer = $f(\text{link } e_{u,v})$

$$send_{u,v}(L) = recv_{u,v}(L) = T_{u,v}(L) = T_{u,v}.$$
Simple Platform Pruning

- Idea: delete edges of maximum weight, until we have a tree
- Algorithm:

```
SIMPLE-PLATFORM-PRUNING(\mathcal{P}, P_{source})

TreeEdges ← all edges of E

while |TreeEdges| > n − 1 do
    L ← edges of TreeEdges sorted by non-increasing weight \( T_{u,v} \)
    for each edge \( e \in L \) do
        if the graph \((V, TreeEdges \setminus \{e\})\) is still connected then
            TreeEdges ← TreeEdges \setminus \{e\}
    return \((V, TreeEdges)\)
```
Simple Platform Pruning

- Idea: delete edges of maximum weight, until we have a tree
- Algorithm:

```
SIMPLE-PLATFORM-PRUNING(\mathcal{P}, P_{source})

TreeEdges ← all edges of \mathcal{E}

while |TreeEdges| > n − 1 do
    L ← edges of TreeEdges sorted by non-increasing weight T_{u,v}
    for each edge e ∈ L do
        if the graph (V, TreeEdges\{e\}) is still connected then
            TreeEdges ← TreeEdges\{e\}

return (V, TreeEdges)
```
Simple Platform Pruning

Example of simple pruning:

Topology, costs of edges $T_{u,v}$
Example of simple pruning:

Choosing and pruning edge of maximum weight
Example of simple pruning:

Choosing and pruning edge of maximum weight
Example of simple pruning:

Choosing and pruning edge of maximum weight
Example of simple pruning:

Achievable throughput: $1/8$
Refined Platform Pruning

- Idea:
  - at each step, compute the out-degree of each node
  - prune an edge from a node whose out-degree is maximum

- Example:

Topology, costs of edges $T_{u,v}$
Refined Platform Pruning

- Idea:
  - at each step, compute the out-degree of each node
  - prune an edge from a node whose out-degree is maximum

- Example:

Choosing maximum out-degree node, then maximum edge
Refined Platform Pruning

Idea:
- at each step, compute the out-degree of each node
- prune an edge from a node whose out-degree is maximum

Example:

Choosing maximum out-degree node, then maximum edge
Refined Platform Pruning

- **Idea:**
  - at each step, compute the out-degree of each node
  - prune an edge from a node whose out-degree is maximum

- **Example:**

Choosing maximum out-degree node, then maximum edge
Refined Platform Pruning

- **Idea:**
  - at each step, compute the out-degree of each node
  - prune an edge from a node whose out-degree is maximum

- **Example:**

```
Achievable throughput: 1/5
```
Refined Platform Pruning

\textsc{Refined-Platform-Pruning}(\mathcal{P}, P_{source})

1: \textit{TreeEdges} ← all edges of \( E \)

2: \textbf{for} each \( u \in V \) \textbf{do}

3: \hspace{1em} \textit{OutDegree}(u) ← \sum\limits_{v, (u,v) \in E} T_{u,v}

4: \textbf{while} |TreeEdges| > n - 1 \textbf{do}

5: \hspace{1em} \textit{SortedNodes} ← nodes sorted by non-increasing value of \textit{OutDegree}(u)

6: \hspace{2em} \textbf{for} \( u \in \textit{SortedNodes} \) \textbf{do}

7: \hspace{3em} \textit{L} ← edges sorted by decreasing weight \( T_{u,v} \)

8: \hspace{3em} \textbf{for} each edge \( e = (u, v) \in \textit{L} \) \textbf{do}

9: \hspace{4em} \textbf{if} the graph \((V, \textit{TreeEdges}\backslash\{e\})\) is still connected \textbf{then}

10: \hspace{5em} \textit{TreeEdges} ← \textit{TreeEdges}\backslash\{e\}

11: \hspace{5em} \textit{OutDegree}(u) ← \textit{OutDegree}(u) - T_{u,v}

12: \hspace{5em} \textbf{goto} 4

13: \textbf{return} \((V, \textit{TreeEdges})\)
Growing a Minimum Weighted Out-Degree Tree

- Idea: grow a tree as in Prim's algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree
- Example:
Growing a Minimum Weighted Out-Degree Tree

- Idea: grow a tree as in Prim’s algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree

Example:
Growing a Minimum Weighted Out-Degree Tree

- Idea: grow a tree as in Prim’s algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree
- Example:

```
   1
  /   
2     2
  /     
2     3
  /     
 6     4
```

Yves Robert Broadcast Trees for Heterogeneous Platforms 18/ 34
Growing a Minimum Weighted Out-Degree Tree

- Idea: grow a tree as in Prim’s algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree
- Example:
Growing a Minimum Weighted Out-Degree Tree

- Idea: grow a tree as in Prim’s algorithm
- At each step, choose an edge optimizing metric
- Our metric:
  - minimize the weighted out-degree of each node in the tree
- Example:

![Diagram of a tree with weighted edges]

Achievable throughput: $1/5$
Growing a Minimum Weighted Out-Degree Tree

**Growing-Minimum-Weighted-Out-Degree-Tree**($P$, $P_{source}$)

```
TreeEdges ← ∅
TreeVertices ← {P_{source}}
for each edge $e = (u, v)$ do
    cost($u$, $v$) ← $T_{u,v}$
while TreeVertices ≠ V do
    choose the link $(u, v)$ such that $u ∈ TreeVertices$, $v ∉ TreeVertices$ and $(u, v)$ has minimum value cost($u$, $v$)
    TreeVertices ← TreeVertices ∪ {$v$}
    TreeEdges ← TreeEdges ∪ {(u, v)}
for each edge $(u, w) ∉ TreeEdges$ do
    cost($u$, $w$) ← cost($u$, $w$) + cost($u$, $v$)
return (TreeVertices, TreeEdges)
```
Binomial tree heuristic

- For sake of comparison
  - Close to MPI_Bcast
  - Construct a binomial tree (without topological information)
Binomial tree heuristic

- For sake of comparison
- Close to MPI_Bcast
- Construct a binomial tree (without topological information)
Binomial tree heuristic

- For sake of comparison
- Close to MPI_Bcast
- Construct a binomial tree (without topological information)
Adapt the growing-tree heuristic to multi-port model

Congestion may come from:
- the number of send operations from $P_u$,
- the length of a transfer $P_u \rightarrow P_v$

New computation of out-degree:

\[
T_{\text{period}} = \max \left( \delta_{\text{out}}(P_u) \times \text{send}_u, \max_i (T_{u,v_i}) \right)
\]
Adapt the growing-tree heuristic to multi-port model

Congestion may come from:
- the number of send operations from $P_u$,
- the length of a transfer $P_u \rightarrow P_v$

New computation of out-degree:

$$T_{period} = \max \left( \delta_{out}(P_u) \times send_u, \max_i(T_{u,v_i}) \right)$$
Adapt the growing-tree heuristic to multi-port model

Congestion may come from:
- the number of send operations from $P_u$,
- the length of a transfer $P_u \rightarrow P_v$

New computation of out-degree:

$$T_{period} = \max \left( \delta_{out}(P_u) \times send_u, \max_i (T_{u,v_i}) \right)$$
Multi-port

- Adapt the growing-tree heuristic to multi-port model
- Congestion may come from:
  - the number of send operations from $P_u$,
  - the length of a transfer $P_u \rightarrow P_v$
- New computation of out-degree:

$$T_{period} = \max \left( \delta_{out}(P_u) \times send_u, \max_i(T_{u,v_i}) \right)$$
Adapt the growing-tree heuristic to multi-port model

Congestion may come from:

- the number of send operations from $P_u$,
- the length of a transfer $P_u \rightarrow P_v$

New computation of out-degree:

$$T_{\text{period}} = \max \left( \delta_{\text{out}}(P_u) \times \text{send}_u, \max_i (T_{u,v_i}) \right)$$
Case where throughput is bounded by the serialized $send_u$:

Case where throughput is bounded by the longest link occupation $T_{u,v}$:
- Case where throughput is bounded by the serialized $send_u$:
- Case where throughput is bounded by the longest link occupation $T_{u,v}$:
Outline

Introduction

Models and Framework

Platform-based Heuristics
  One port-model
  Multi-port

LP-based heuristics

Simulations

Conclusion
LP formulation

- Solving the MTP problem with LP formulation:
  - variables: average number of messages going through each link
  - constraints: one-port model constraints, link occupation
- Solution of LP $\Rightarrow$ network utilization to reach best throughput
- Complicated algorithm to reconstruct optimal set of trees for MTP, not needed here
- Use results output by LP, optimal solution $S_{opt}$:
  - $TP = \text{optimal throughput}$
  - $n_{u,v} = \text{number of messages through edge } e_{u,v} \text{ in one time-unit in } S_{opt}$
Solving the MTP problem with LP formulation:
- variables: average number of messages going through each link
- constraints: one-port model constraints, link occupation

Solution of LP $\Rightarrow$ network utilization to reach best throughput

Complicated algorithm to reconstruct optimal set of trees for MTP, not needed here

Use results output by LP, optimal solution $S_{opt}$:
- $TP = \text{optimal throughput}$
- $n_{u,v} = \text{number of messages through edge } e_{u,v} \text{ in one time-unit in } S_{opt}$
Solving the MTP problem with LP formulation:

- variables: average number of messages going through each link
- constraints: one-port model constraints, link occupation

Solution of LP $\Rightarrow$ network utilization to reach best throughput

Complicated algorithm to reconstruct optimal set of trees for MTP, not needed here

Use results output by LP, optimal solution $S_{opt}$:

- $TP = \text{optimal throughput}$
- $n_{u,v} = \text{number of messages through edge } e_{u,v} \text{ in one time-unit in } S_{opt}$
Solving the MTP problem with LP formulation:

- variables: average number of messages going through each link
- constraints: one-port model constraints, link occupation

Solution of LP $\Rightarrow$ network utilization to reach best throughput

Complicated algorithm to reconstruct optimal set of trees for MTP, not needed here

Use results output by LP, optimal solution $S_{opt}$:

- $TP =$ optimal throughput
- $n_{u,v} =$ number of messages through edge $e_{u,v}$ in one time-unit in $S_{opt}$
LP-based heuristics

- Communication graph pruning:
  - similar to the previous pruning heuristic
  - based on the communication graph, labeled with $n_{u,v}$ values
  - prune edges carrying the fewest messages in $S_{opt}$

- Growing a spanning tree over the communication graph
  - start from the communication graph of $S_{opt}$
  - grow a tree, selecting edges with maximal number of messages
LP-based heuristics

- Communication graph pruning:
  - similar to the previous pruning heuristic
  - based on the communication graph, labeled with $n_{u,v}$ values
  - prune edges carrying the fewest messages in $S_{opt}$

- Growing a spanning tree over the communication graph
  - start from the communication graph of $S_{opt}$
  - grow a tree, selecting edges with maximal number of messages
Simulations using both the one-port and multi-port models

1. random generation of platforms, with parameters:
   - number of nodes: 10, 20, . . . , 50
   - density: 0.04, 0.08, . . . , 0.20
   - $T_{u,v}$: Gaussian distribution
     - (mean=100MB/s, deviation=20MB/s)
   - $send_{u,v}$: $0.80 \cdot \min_{w,(u,w) \in E} \{T_{u,w}\}$
     (for each set of parameters, 10 different configurations generated)

2. realistic platforms generated by Tiers:
   - 100 platforms with 30 nodes
   - 100 platforms with 65 nodes
   - density between 0.05 and 0.15
Simulations using both the one-port and multi-port models

1. random generation of platforms, with parameters:
   - number of nodes: 10, 20, ..., 50
   - density: 0.04, 0.08, ..., 0.20
   - $T_{u,v}$: Gaussian distribution
     - (mean = 100MB/s, deviation = 20MB/s)
   - $\text{send}_{u,v}$: $0.80 \cdot \min_{w,(u,w) \in E} \{T_{u,w}\}$
     (for each set of parameters, 10 different configurations generated)

2. realistic platforms generated by Tiers:
   - 100 platforms with 30 nodes
   - 100 platforms with 65 nodes
   - density between 0.05 and 0.15
Results, one-port, random platforms

Performance versus number of nodes

Y axis: relative average performance compared to the optimal solution for MTP
Results, one-port, random platforms

- Performance versus density

Y axis: relative average performance compared to the optimal solution for MTP
Results, multi-port, random platforms

Performance versus number of nodes

- Multi Port Prune Degree
- Multi Port Grow Tree
- LP Grow Tree
- LP Prune
- Binomial Tree

Yves Robert Broadcast Trees for Heterogeneous Platforms
Results, one-port, realistic platforms

- Performance of the one-port heuristics on two types of platforms generated by TIERS

![Bar chart showing performance of heuristics on 30 and 65 node platforms]
For the one-port model:
- small platforms: results close to the optimal
- large platforms: “advanced” heuristics within 60% of the optimal
- simple pruning heuristic: not scalable
- binomial heuristic: very poor results

Under multi-port assumption:
- binomial heuristic performs slightly better
- adapted heuristic (Growing-Tree): much better results
- LP-based heuristics perform well
For the one-port model:
- small platforms: results close to the optimal
- large platforms: “advanced” heuristics within 60% of the optimal
- simple pruning heuristic: not scalable
- binomial heuristic: very poor results

Under multi-port assumption:
- binomial heuristic performs slightly better
- adapted heuristic (Growing-Tree): much better results
- LP-based heuristics perform well
Outline

Introduction

Models and Framework

Platform-based Heuristics
  One port-model
  Multi-port

LP-based heuristics

Simulations

Conclusion
Conclusion

- Designing efficient algorithms to broadcast data
- Use pipelining techniques, focus on steady-state
- Using multiple trees (MTP): polynomial algorithm, but difficult to enforce in practice
- Using a single tree (STP): NP-Complete
- Design heuristics for STP, possibly using MTP linear program
- Avoid binomial approach (MPI)