Optimizing the steady-state throughput of scatter and reduce operations on heterogeneous platforms

Arnaud Legrand, Loris Marchal, Yves Robert

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

APDCM workshop - April 2004
Introduction

Two Problems of Collective Communication
Platform Model
Framework

Series of Scatter
Steady-state constraints
Toy Example
Building a schedule
Asymptotic optimality

Series of Reduce
Introduction to reduction trees
Linear Program
Periodic schedule - Asymptotic optimality
Toy Example for Series of Reduce

Approximation for a fixed period

Conclusion
Outline

Introduction
Two Problems of Collective Communication
Platform Model
Framework

Series of Scatter
Steady-state constraints
Toy Example
Building a schedule
Asymptotic optimality

Series of Reduce
Introduction to reduction trees
Linear Program
Periodic schedule - Asymptotic optimality
Toy Example for Series of Reduce

Approximation for a fixed period

Conclusion
Outline

Introduction
  Two Problems of Collective Communication
  Platform Model
  Framework

Series of Scatter
  Steady-state constraints
  Toy Example
  Building a schedule
  Asymptotic optimality

Series of Reduce
  Introduction to reduction trees
  Linear Program
  Periodic schedule - Asymptotic optimality
  Toy Example for Series of Reduce

Approximation for a fixed period

Conclusion
Outline

Introduction
  Two Problems of Collective Communication
  Platform Model
  Framework

Series of Scatter
  Steady-state constraints
  Toy Example
  Building a schedule
  Asymptotic optimality

Series of Reduce
  Introduction to reduction trees
  Linear Program
  Periodic schedule - Asymptotic optimality
  Toy Example for Series of Reduce

Approximation for a fixed period

Conclusion
Outline

Introduction
  Two Problems of Collective Communication
  Platform Model
  Framework

Series of Scatter
  Steady-state constraints
  Toy Example
  Building a schedule
  Asymptotic optimality

Series of Reduce
  Introduction to reduction trees
  Linear Program
  Periodic schedule - Asymptotic optimality
  Toy Example for Series of Reduce

Approximation for a fixed period

Conclusion
Outline

Introduction
  Two Problems of Collective Communication
  Platform Model
  Framework

Series of Scatter
  Steady-state constraints
  Toy Example
  Building a schedule
  Asymptotic optimality

Series of Reduce
  Introduction to reduction trees
  Linear Program
  Periodic schedule - Asymptotic optimality
  Toy Example for Series of Reduce

Approximation for a fixed period

Conclusion
Introduction

- Complex applications on grid environment require collective communication schemes:
  - one to all: Broadcast, Multicast, Scatter
  - all to one: Reduce
  - all to all: Gossip, All-to-All

- Numerous studies of a single communication scheme, mainly about one single broadcast

- Pipelining communications:
  - data parallelism involves a large amount of data
  - not a single communication, but series of same communication schemes (e.g., series of broadcasts from same source)
  - maximize throughput of steady-state operation
Complex applications on grid environment require collective communication schemes:
- one to all  Broadcast, Multicast, Scatter
- all to one  Reduce
- all to all  Gossip, All-to-All

Numerous studies of a single communication scheme, mainly about one single broadcast

Pipelining communications:
- data parallelism involves a large amount of data
- not a single communication, but series of same communication schemes (e.g. series of broadcasts from same source)
- maximize throughput of steady-state operation
Introduction

- Complex applications on grid environment require collective communication schemes:
  - one to all: Broadcast, Multicast, Scatter
  - all to one: Reduce
  - all to all: Gossip, All-to-All

- Numerous studies of a single communication scheme, mainly about one single broadcast

- Pipelining communications:
  - data parallelism involves a large amount of data
  - not a single communication, but series of same communication schemes (e.g., series of broadcasts from same source)
  - maximize throughput of steady-state operation
Complex applications on grid environment require collective communication schemes:

- one to all  Broadcast, Multicast, Scatter
- all to one  Reduce
- all to all  Gossip, All-to-All

Numerous studies of a single communication scheme, mainly about one single broadcast

Pipelining communications:

- data parallelism involves a large amount of data
- not a single communication, but series of same communication schemes (e.g. series of broadcasts from same source)
- maximize throughput of steady-state operation
Introduction

- Complex applications on grid environment require **collective communication schemes**:
  - one to all  Broadcast, Multicast, Scatter
  - all to one  Reduce
  - all to all  Gossip, All-to-All

- Numerous studies of a single communication scheme, mainly about one single broadcast

- Pipelining communications:
  - data parallelism involves a large amount of data
  - not a single communication, but series of same communication schemes (e.g. series of broadcasts from same source)
  - maximize throughput of steady-state operation
Introduction

- Complex applications on grid environment require **collective communication schemes**:
  - one to all  Broadcast, Multicast, Scatter
  - all to one  Reduce
  - all to all  Gossip, All-to-All

- Numerous studies of a single communication scheme, mainly about one single broadcast

- Pipelining communications:
  - data parallelism involves a large amount of data
  - not a single communication, but series of same communication schemes (e.g. series of broadcasts from same source)
  - maximize throughput of steady-state operation
Introduction

- Complex applications on grid environment require collective communication schemes:
  - one to all  Broadcast, Multicast, Scatter
  - all to one  Reduce
  - all to all  Gossip, All-to-All

- Numerous studies of a single communication scheme, mainly about one single broadcast

- Pipelining communications:
  - data parallelism involves a large amount of data
  - not a single communication, but series of same communication schemes (e.g. series of broadcasts from same source)
  - maximize throughput of steady-state operation
Complex applications on grid environment require collective communication schemes:
- one to all: Broadcast, Multicast, Scatter
- all to one: Reduce
- all to all: Gossip, All-to-All

Numerous studies of a single communication scheme, mainly about one single broadcast

Pipelining communications:
- data parallelism involves a large amount of data
- not a single communication, but series of same communication schemes (e.g. series of broadcasts from same source)
- maximize throughput of steady-state operation
Complex applications on grid environment require collective communication schemes:
one to all  Broadcast, Multicast, Scatter
all to one  Reduce
all to all  Gossip, All-to-All

Numerous studies of a single communication scheme, mainly about one single broadcast

Pipelining communications:
  - data parallelism involves a large amount of data
  - not a single communication, but series of same communication schemes (e.g. series of broadcasts from same source)
  - maximize throughput of steady-state operation
Two Problems of Collective Communication

**Scatter** one processor $P_{source}$ sends distinct messages to target processors ($\{P_{t0}, \ldots, P_{tN}\}$)

- Series of Scatter $P_{source}$ sends consecutively a large number of distinct messages to all targets

**Reduce** Each of the participating processor $P_{ri}$ in $P_{r0}, \ldots, P_{rN}$ owns a value $v_i$

$\Rightarrow$ compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ is associative, non commutative)

- Series of Reduce several consecutive reduce operations from the same set $P_{r0}, \ldots, P_{rN}$ to the same target $P_{target}$.
Two Problems of Collective Communication

**Scatter** one processor $P_{\text{source}}$ sends distinct messages to target processors ($\{P_{t_0}, \ldots, P_{t_N}\}$)

- **Series of Scatter** $P_{\text{source}}$ sends consecutively a large number of distinct messages to all targets

**Reduce** Each of the participating processor $P_{r_i}$ in $P_{r_0}, \ldots, P_{r_N}$ owns a value $v_i$

⇒ compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ is associative, non-commutative)

- **Series of Reduce** several consecutive reduce operations from the same set $P_{r_0}, \ldots, P_{r_N}$ to the same target $P_{\text{target}}$. 

Loris Marchal

Steady state collective communications
Two Problems of Collective Communication

**Scatter**  one processor $P_{source}$ sends distinct messages to target processors ($\{P_{t0}, \ldots, P_{tN}\}$)

- **Series of Scatter** $P_{source}$ sends consecutively a large number of distinct messages to all targets

**Reduce**  Each of the participating processor $P_{ri}$ in $P_{r0}, \ldots, P_{rN}$ owns a value $v_i$

$\Rightarrow$ compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ is associative, non commutative)

- **Series of Reduce** several consecutive reduce operations from the same set $P_{r0}, \ldots, P_{rN}$ to the same target $P_{target}$. 

Loris Marchal  Steady state collective communications 5/27
Two Problems of Collective Communication

**Scatter** one processor $P_{\text{source}}$ sends distinct messages to target processors ($\{P_{t0}, \ldots, P_{tN}\}$)

- **Series of Scatter** $P_{\text{source}}$ sends consecutively a large number of distinct messages to all targets

**Reduce** Each of the participating processor $P_{ri}$ in $P_{r0}, \ldots, P_{rN}$ owns a value $v_i$

$\Rightarrow$ compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ is associative, non commutative)

- **Series of Reduce** several consecutive reduce operations from the same set $P_{r0}, \ldots, P_{rN}$ to the same target $P_{\text{target}}$. 
**Platform Model**

- $G = (V, E, c)$
  - $P_1, P_2, \ldots, P_n$: processors
  - $(j, k) \in E$: communication link between $P_i$ and $P_j$
  - $c(j, k)$: time to transfer one unit message from $P_j$ to $P_k$
  - one-port for incoming communications
  - one-port for outgoing communications
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: processors
- $(j, k) \in E$: communication link between $P_i$ and $P_j$
- $c(j, k)$: time to transfer one unit message from $P_j$ to $P_k$
- one-port for incoming communications
- one-port for outgoing communications
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: processors
- $(j, k) \in E$: communication link between $P_i$ and $P_j$
- $c(j, k)$: time to transfer one unit message from $P_j$ to $P_k$
- one-port for incoming communications
- one-port for outgoing communications
Platform Model

- \( G = (V, E, c) \)
- \( P_1, P_2, \ldots, P_n \): processors
- \((j, k) \in E\): communication link between \( P_i \) and \( P_j \)
- \( c(j, k) \): time to transfer one unit message from \( P_j \) to \( P_k \)
- one-port for incoming communications
- one-port for outgoing communications

![Diagram of communication links between processors]

Loris Marchal
Steady state collective communications
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: processors
- $(j, k) \in E$: communication link between $P_i$ and $P_j$
- $c(j, k)$: time to transfer one unit message from $P_j$ to $P_k$
- one-port for incoming communications
- one-port for outgoing communications
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: processors
- $(j, k) \in E$: communication link between $P_i$ and $P_j$
- $c(j, k)$: time to transfer one unit message from $P_j$ to $P_k$
- one-port for incoming communications
- one-port for outgoing communications
Platform Model

\[ G = (V, E, c) \]

- \( P_1, P_2, \ldots, P_n \): processors
- \((j, k) \in E\): communication link between \( P_i \) and \( P_j \)
- \( c(j, k) \): time to transfer one unit message from \( P_j \) to \( P_k \)
- one-port for incoming communications
- one-port for outgoing communications
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: processors
- $(j, k) \in E$: communication link between $P_i$ and $P_j$
- $c(j, k)$: time to transfer one unit message from $P_j$ to $P_k$
- one-port for incoming communications
- one-port for outgoing communications
Framework

1. express optimization problem as set of linear constraints (variables = fraction of time a processor spends sending to one of its neighbors)
2. solve linear program (in rational numbers)
3. use solution to build periodic schedule reaching best throughput
1. express optimization problem as set of linear constraints (variables = fraction of time a processor spends sending to one of its neighbors)
2. solve linear program (in rational numbers)
3. use solution to build periodic schedule reaching best throughput
Framework

1. express optimization problem as set of linear constraints (variables = fraction of time a processor spends sending to one of its neighbors)
2. solve linear program (in rational numbers)
3. use solution to build periodic schedule reaching best throughput
Series of Scatter

- $m_k$: types of the messages with destination $P_k$
- $s(P_i \rightarrow P_j, m_k)$: fractional number of messages of type $m_k$ sent on the edge $P_i \rightarrow P_j$ within on time unit
- $t(P_i \rightarrow P_j)$: fractional time spent by processor $P_i$ to send data to its neighbor $P_j$ within one time unit
- bound for this activity:

$$\forall P_i, P_j, \quad 0 \leq t(P_i \rightarrow P_j) \leq 1$$

- on a link $P_i \rightarrow P_j$ during one time-unit:

$$t(P_i \rightarrow P_j) = \sum_k s(P_i \rightarrow P_j, m_k)$$
Series of Scatter

- $m_k$: types of the messages with destination $P_k$
- $s(P_i \rightarrow P_j, m_k)$: fractional number of messages of type $m_k$ sent on the edge $P_i \rightarrow P_j$ within one time unit
- $t(P_i \rightarrow P_j)$: fractional time spent by processor $P_i$ to send data to its neighbor $P_j$ within one time unit
- bound for this activity:

$$\forall P_i, P_j, \quad 0 \leq t(P_i \rightarrow P_j) \leq 1$$

- on a link $P_i \rightarrow P_j$ during one time-unit:

$$t(P_i \rightarrow P_j) = \sum_{k} s(P_i \rightarrow P_j, m_k)$$
Series of Scatter

- $m_k$: types of the messages with destination $P_k$
- $s(P_i \rightarrow P_j, m_k)$: fractional number of messages of type $m_k$ sent on the edge $P_i \rightarrow P_j$ within one time unit
- $t(P_i \rightarrow P_j)$: fractional time spent by processor $P_i$ to send data to its neighbor $P_j$ within one time unit

Bound for this activity:

$$\forall P_i, P_j, \quad 0 \leq t(P_i \rightarrow P_j) \leq 1$$

On a link $P_i \rightarrow P_j$ during one time-unit:

$$t(P_i \rightarrow P_j) = \sum_k s(P_i \rightarrow P_j, m_k)$$
Series of Scatter

- \( m_k \): types of the messages with destination \( P_k \)
- \( s(P_i \rightarrow P_j, m_k) \): fractional number of messages of type \( m_k \) sent on the edge \( P_i \rightarrow P_j \) within on time unit
- \( t(P_i \rightarrow P_j) \): fractional time spent by processor \( P_i \) to send data to its neighbor \( P_j \) within one time unit
- bound for this activity:

\[
\forall P_i, P_j, \quad 0 \leq t(P_i \rightarrow P_j) \leq 1
\]

- on a link \( P_i \rightarrow P_j \) during one time-unit:

\[
t(P_i \rightarrow P_j) = \sum_k s(P_i \rightarrow P_j, m_k)
\]
$m_k$: types of the messages with destination $P_k$

$s(P_i \rightarrow P_j, m_k)$: fractional number of messages of type $m_k$ sent on the edge $P_i \rightarrow P_j$ within one time unit

$t(P_i \rightarrow P_j)$: fractional time spent by processor $P_i$ to send data to its neighbor $P_j$ within one time unit

bound for this activity:

$$\forall P_i, P_j, \quad 0 \leq t(P_i \rightarrow P_j) \leq 1$$

on a link $P_i \rightarrow P_j$ during one time-unit:

$$t(P_i \rightarrow P_j) = \sum_k s(P_i \rightarrow P_j, m_k)$$
Linear constraints

- one port constraints for outgoing messages in $P_i$:
  \[
  \forall P_i, \sum_{P_i \to P_j} t(P_i \to P_j) \leq 1
  \]

- one port constraints for incoming messages in $P_i$:
  \[
  \forall P_i, \sum_{P_j \to P_i} t(P_j \to P_i) \leq 1
  \]

- conservation law in node $P_i$ for message $m_k$ ($k \neq i$):
  \[
  \sum_{P_j \to P_i} s(P_j \to P_i, m_k) = \sum_{P_i \to P_j} s(P_j \to P_i, m_k) \leq 1
  \]
Linear constraints

- one port constraints for outgoing messages in $P_i$:

\[ \forall P_i, \sum_{P_i \rightarrow P_j} t(P_i \rightarrow P_j) \leq 1 \]

- one port constraints for incoming messages in $P_i$:

\[ \forall P_i, \sum_{P_j \rightarrow P_i} t(P_j \rightarrow P_i) \leq 1 \]

- conservation law in node $P_i$ for message $m_k$ ($k \neq i$):

\[ \sum_{P_j \rightarrow P_i} s(P_j \rightarrow P_i, m_k) = \sum_{P_i \rightarrow P_j} s(P_j \rightarrow P_i, m_k) \leq 1 \]
Linear constraints

- one port constraints for outgoing messages in $P_i$:

$$\forall P_i, \sum_{P_i \rightarrow P_j} t(P_i \rightarrow P_j) \leq 1$$

- one port constraints for incoming messages in $P_i$:

$$\forall P_i, \sum_{P_j \rightarrow P_i} t(P_j \rightarrow P_i) \leq 1$$

- conservation law in node $P_i$ for message $m_k$ ($k \neq i$):

$$\sum_{P_j \rightarrow P_i} s(P_j \rightarrow P_i, m_k) = \sum_{P_i \rightarrow P_j} s(P_j \rightarrow P_i, m_k) \leq 1$$
Linear constraints

- one port constraints for outgoing messages in $P_i$:
  \[ \forall P_i, \sum_{P_i \rightarrow P_j} t(P_i \rightarrow P_j) \leq 1 \]

- one port constraints for incoming messages in $P_i$:
  \[ \forall P_i, \sum_{P_j \rightarrow P_i} t(P_j \rightarrow P_i) \leq 1 \]

- conservation law in node $P_i$ for message $m_k$ ($k \neq i$):
  \[ \sum_{P_j \rightarrow P_i} s(P_j \rightarrow P_i, m_k) = \sum_{P_i \rightarrow P_j} s(P_j \rightarrow P_i, m_k) \leq 1 \]
Linear constraints

- one port constraints for outgoing messages in $P_i$:

  $$\forall P_i, \sum_{P_i \rightarrow P_j} t(P_i \rightarrow P_j) \leq 1$$

- one port constraints for incoming messages in $P_i$:

  $$\forall P_i, \sum_{P_j \rightarrow P_i} t(P_j \rightarrow P_i) \leq 1$$

- conservation law in node $P_i$ for message $m_k$ ($k \neq i$):

  $$\sum_{P_j \rightarrow P_i} s(P_j \rightarrow P_i, m_k) = \sum_{P_i \rightarrow P_j} s(P_j \rightarrow P_i, m_k) \leq 1$$
Linear constraints

- one port constraints for outgoing messages in $P_i$:

$$\forall P_i, \sum_{P_i \rightarrow P_j} t(P_i \rightarrow P_j) \leq 1$$

- one port constraints for incoming messages in $P_i$:

$$\forall P_i, \sum_{P_j \rightarrow P_i} t(P_j \rightarrow P_i) \leq 1$$

- conservation law in node $P_i$ for message $m_k$ ($k \neq i$):

$$\sum_{P_j \rightarrow P_i} s(P_j \rightarrow P_i, m_k) = \sum_{P_i \rightarrow P_j} s(P_j \rightarrow P_i, m_k) \leq 1$$
Throughput and Linear Program

- **throughput**: total number of messages $m_k$ received in $P_k$

\[
TP = \sum_{P_j \rightarrow P_k} s(P_j \rightarrow P_k, m_k)
\]

(same throughput for every target node $P_k$)

- summarize this constraints in a linear program:

**Steady-State Scatter Problem on a Graph SSSP(G)**

Maximize $TP$, subject to

\[
\begin{cases}
\forall P_i, \forall P_j, 0 \leq s(P_i \rightarrow P_j) \leq 1 \\
\forall P_i, \sum_{P_j, (i,j) \in E} s(P_i \rightarrow P_j) \leq 1 \\
\forall P_i, \sum_{P_j, (j,i) \in E} s(P_j \rightarrow P_i) \leq 1 \\
\forall P_i, P_j, s(P_i \rightarrow P_j) = \sum_{m_k} \text{send}(P_i \rightarrow P_j, m_k) \times c(i,j) \\
\forall P_i, \forall m_k, k \neq i, \sum_{P_j, (j,i) \in E} \text{send}(P_j \rightarrow P_i, m_k) \\
= \sum_{P_j, (i,j) \in E} \text{send}(P_i \rightarrow P_j, m_k) \\
\forall P_k, k \in T \sum_{P_i, (i,k) \in E} \text{send}(P_i \rightarrow P_k, m_k) = TP
\end{cases}
\]
Throughput and Linear Program

- **Throughput**: total number of messages $m_k$ received in $P_k$

$$TP = \sum_{P_j \rightarrow P_k} s(P_j \rightarrow P_k, m_k)$$

(same throughput for every target node $P_k$)

- summarize this constraints in a linear program:

**Steady-State Scatter Problem on a Graph SSSP(G)**

Maximize $TP$, subject to

\[
\begin{align*}
\forall P_i, \forall P_j, &\ 0 \leq s(P_i \rightarrow P_j) \leq 1 \\
\forall P_i, &\ \sum_{P_j, (i,j) \in E} s(P_i \rightarrow P_j) \leq 1 \\
\forall P_i, &\ \sum_{P_j, (j,i) \in E} s(P_j \rightarrow P_i) \leq 1 \\
\forall P_i, P_j, &\ s(P_i \rightarrow P_j) = \sum_{m_k} send(P_i \rightarrow P_j, m_k) \times c(i, j) \\
\forall P_i, &\ \sum_{m_k, k \neq i} \sum_{P_j, (j,i) \in E} send(P_j \rightarrow P_i, m_k) \\
&\ = \sum_{P_j, (i,j) \in E} send(P_i \rightarrow P_j, m_k) \\
\forall P_k, &\ k \in T \sum_{P_i, (i,k) \in E} send(P_i \rightarrow P_k, m_k) = TP
\end{align*}
\]
platform graph (edges labeled with $c(i, j)$)
value of $s(P_i \rightarrow P_j, m_k)$ in the solution of the linear program
occupation time of the edge ($t(P_i \rightarrow P_j)$)
Building a schedule

- consider the time needed for all transfers
- build a bipartite graph by splitting all nodes
- extract matchings, using the weighted-edge coloring algorithm
Building a schedule

- consider the time needed for all transfers
- build a bipartite graph by splitting all nodes
- extract matchings, using the weighted-edge coloring algorithm
Building a schedule

- consider the time needed for all transfers
- build a bipartite graph by splitting all nodes
- extract matchings, using the weighted-edge coloring algorithm
Building a schedule

- consider the time needed for all transfers
- build a bipartite graph by splitting all nodes
- extract matchings, using the weighted-edge coloring algorithm
Building a schedule

- consider the time needed for all transfers
- build a bipartite graph by splitting all nodes
- extract matchings, using the weighted-edge coloring algorithm
Building a schedule

- consider the time needed for all transfers
- build a bipartite graph by splitting all nodes
- extract matchings, using the weighted-edge coloring algorithm
Building a schedule

matchings:

- $P_b \rightarrow P_1$
- $P_b \rightarrow P_0$
- $P_a \rightarrow P_0$
- $P_s \rightarrow P_b$
- $P_s \rightarrow P_a$

- least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transferred in each matching

- $\Rightarrow$ periodic schedule of period $T$ with atomic transfers of messages
Building a schedule

matchings:

- $P_b \rightarrow P_1$
- $P_b \rightarrow P_0$
- $P_a \rightarrow P_0$
- $P_s \rightarrow P_b$
- $P_s \rightarrow P_a$

- least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transferred in each matching
- $\Rightarrow$ periodic schedule of period $T$ with atomic transfers of messages
Building a schedule

matchings:

- $P_b \rightarrow P_1$
- $P_b \rightarrow P_0$
- $P_a \rightarrow P_0$
- $P_s \rightarrow P_b$
- $P_s \rightarrow P_a$

- least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transfered in each matching
- $\Rightarrow$ periodic schedule of period $T$ with atomic transfers of messages

Loris Marchal

Steady state collective communications
Building a schedule

matchings:

- $P_b \rightarrow P_1$
- $P_b \rightarrow P_0$
- $P_a \rightarrow P_0$
- $P_s \rightarrow P_b$
- $P_s \rightarrow P_a$

- least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transferred in each matching
- $\Rightarrow$ periodic schedule of period $T$ with atomic transfers of messages
Building a schedule

matchings:

$P_b \to P_1$

$P_b \to P_0$

$P_a \to P_0$

$P_s \to P_b$

$P_s \to P_a$

least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transferred in each matching

$\Rightarrow$ periodic schedule of period $T$ with atomic transfers of messages
Building a schedule

matchings:

- $P_b \rightarrow P_1$
- $P_b \rightarrow P_0$
- $P_a \rightarrow P_0$
- $P_s \rightarrow P_b$
- $P_s \rightarrow P_a$

least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transferred in each matching

$\Rightarrow$ periodic schedule of period $T$ with atomic transfers of messages
Building a schedule

least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transferred in each matching

$\Rightarrow$ periodic schedule of period $T$ with atomic transfers of messages
Asymptotic optimality

- No schedule can perform more tasks than the steady-state:

Lemma.

\[ \text{opt}(G, K) \leq \text{TP}(G) \times K \]

- periodic schedule \( \Rightarrow \) schedule:
  1. initialization phase (fill buffers of messages)
  2. \( r \) periods of duration \( T \) (steady-state)
  3. clean-up phase (empty buffers)

Lemma.

\[ \lim_{K \to +\infty} \text{steady}(G, K) = 1 \]
Asymptotic optimality

- No schedule can perform more tasks than the steady-state:

Lemma.
\[ \text{opt}(G, K) \leq TP(G) \times K \]

- periodic schedule ⇒ schedule:
  1. initialization phase (fill buffers of messages)
  2. \( n \) periods of duration \( T \) (steady-state)
  3. clean-up phase (empty buffers)
Asymptotic optimality

- No schedule can perform more tasks than the steady-state:

\[
\text{Lemma.} \quad \text{opt}(G, K) \leq TP(G) \times K
\]

- Periodic schedule \(\Rightarrow\) schedule:
  1. initialization phase (fill buffers of messages)
  2. \(r\) periods of duration \(T\) (steady-state)
  3. clean-up phase (empty buffers)

\[
\text{Lemma.} \quad \lim_{K \to +\infty} \frac{\text{steady}(G, K)}{\text{opt}(G, K)} = 1
\]
Asymptotic optimality

- No schedule can perform more tasks than the steady-state:

**Lemma.**

\[ \text{opt}(G, K) \leq \text{TP}(G) \times K \]

- periodic schedule ⇒ schedule:
  1. initialization phase (fill buffers of messages)
  2. \( r \) periods of duration \( T \) (steady-state)
  3. clean-up phase (empty buffers)

**Lemma.**

the previous algorithm is asymptotically optimal:

\[ \lim_{K \to +\infty} \frac{\text{steady}(G, K)}{\text{opt}(G, K)} = 1 \]
Asymptotic optimality

- No schedule can perform more tasks than the steady-state:

Lemma.

\[ \text{opt}(G, K) \leq \text{TP}(G) \times K \]

- periodic schedule ⇒ schedule:
  1. initialization phase (fill buffers of messages)
  2. \( r \) periods of duration \( T \) (steady-state)
  3. clean-up phase (empty buffers)

Lemma.

the previous algorithm is asymptotically optimal:

\[ \lim_{K \to +\infty} \frac{\text{steady}(G, K)}{\text{opt}(G, K)} = 1 \]
Asymptotic optimality

- No schedule can perform more tasks than the steady-state:

\[ \text{ Lemma. } \]

\[ \text{opt}(G, K) \leq \text{TP}(G) \times K \]

- periodic schedule \( \Rightarrow \) schedule:
  1. initialization phase (fill buffers of messages)
  2. \( r \) periods of duration \( T \) (steady-state)
  3. clean-up phase (empty buffers)

\[ \text{ Lemma. } \]

the previous algorithm is asymptotically optimal:

\[ \lim_{K \to +\infty} \frac{\text{steady}(G, K)}{\text{opt}(G, K)} = 1 \]
Asymptotic optimality

- No schedule can perform more tasks than the steady-state:

\[
\text{Lemma.} \quad opt(G, K) \leq TP(G) \times K
\]

- periodic schedule ⇒ schedule:
  1. initialization phase (fill buffers of messages)
  2. \( r \) periods of duration \( T \) (steady-state)
  3. clean-up phase (empty buffers)

\[
\text{Lemma.} \quad \lim_{K \to +\infty} \frac{\text{steady}(G, K)}{opt(G, K)} = 1
\]
Reduce - Reduction trees

- **Reduce:**
  - each processor $P_{r_i}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non-commutative)

- partial result of the Reduce operation:
  
  $v[k,m] = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  
  $v[k,m] = v[k,l] \oplus v[l+1,m]$
  (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- **Reduce:**
  - each processor $P_{r_i}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $v[k,m] = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  $v[k,m] = v[k,l] \oplus v[l+1,m]$ (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- Reduce:
  - each processor $P_{ri}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$
    ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $v_{[k,m]} = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  $v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}$
  (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- **Reduce:**
  - each processor $P_{r_i}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $$v_{[k,m]} = v_k \oplus v_2 \oplus \cdots \oplus v_m$$

- two partial results can be merged:
  $$v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}$$
  (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- Reduce:
  - each processor $P_{r_i}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$
    ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $v[k,m] = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  $v[k,m] = v[k,l] \oplus v[l+1,m]$
  (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- **Reduce:**
  - each processor $P_{ri}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $v[k,m] = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  $v[k,m] = v[k,l] \oplus v[l+1,m]$ (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- Reduce:
  - each processor $P_{r_i}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $v_{[k,m]} = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  $v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}$
  (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- **Reduce:**
  - each processor $P_{r_i}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $v_{[k,m]} = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  $v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}$
  (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- Reduce:
  - each processor $P_{r_i}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $v[k,m] = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  $v[k,m] = v[k,l] \oplus v[l+1,m]$ (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- Reduce:
  - each processor $P_r$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $v_{[k,m]} = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  $v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}$ (computational task $T_{k,l,m}$)
Reduce - Reduction trees

- Reduce:
  - each processor $P_{ri}$ owns a value $v_i$
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non commutative)

- partial result of the Reduce operation:
  $v[k,m] = v_k \oplus v_2 \oplus \cdots \oplus v_m$

- two partial results can be merged:
  $v[k,m] = v[k,l] \oplus v[l+1,m]$ (computational task $T_{k,l,m}$)
Series of Reduce

- each processor $P_{ri}$ owns a set of values $v_i^t$ (e.g. produced at different time-steps $t$)
- perform a Reduce operation on each set $\{v_1^t, \ldots, v_N^t\}$ to compute $V^t$
- each reduction uses a reduction tree
- two reductions ($t_1$ and $t_2$) may use different trees
Linear Program - Notations

- \( s(P_i \rightarrow P_j, v_{[k,l]}) \): fractional number of values \( v_{[k,l]} \) sent on link \( P_i \rightarrow P_j \) within one time-unit

- \( t(P_i \rightarrow P_j) \) fractional occupation time of link \( P_i \rightarrow P_j \) within one time-unit:

\[
0 \leq t(P_i \rightarrow P_j) \leq 1
\]

- \( cons(P_i, T_{k,l,m}) \): fractional number of tasks \( T_{k,l,m} \) computed on processor \( P_i \) within one time-unit

- \( \alpha(P_i) \) time spent by processor \( P_i \) computing tasks within one time-unit:

\[
0 \leq \alpha(P_i) \leq 1
\]

- \( size(v_{[k,m]}) \) size of a message containing a value \( v_{[k,m]} \)

- \( w(P_i, T_{k,l,m}) \) time needed by processor \( P_i \) to compute one task \( T_{k,l,m} \)
Linear Program - Notations

- $s(P_i \rightarrow P_j, v_{[k,l]})$: fractional number of values $v_{[k,l]}$ sent on link $P_i \rightarrow P_j$ within one time-unit.

- $t(P_i \rightarrow P_j)$ fractional occupation time of link $P_i \rightarrow P_j$ within one time-unit:
  
  \[ 0 \leq t(P_i \rightarrow P_j) \leq 1 \]

- $\text{cons}(P_i, T_{k,l,m})$: fractional number of tasks $T_{k,l,m}$ computed on processor $P_i$ within one time-unit.

- $\alpha(P_i)$ time spent by processor $P_i$ computing tasks within one time-unit:
  
  \[ 0 \leq \alpha(P_i) \leq 1 \]

- $\text{size}(v_{[k,m]})$ size of a message containing a value $v_{[k,m]}$.

- $w(P_i, T_{k,l,m})$ time needed by processor $P_i$ to compute one task $T_{k,l,m}$. 
Linear Program - Notations

- $s(P_i \rightarrow P_j, v[k,l])$: fractional number of values $v[k,l]$ sent on link $P_i \rightarrow P_j$ within one time-unit.

- $t(P_i \rightarrow P_j)$ fractional occupation time of link $P_i \rightarrow P_j$ within one time-unit:

  $$0 \leq t(P_i \rightarrow P_j) \leq 1$$

- $\text{cons}(P_i, T_{k,l,m})$: fractional number of tasks $T_{k,l,m}$ computed on processor $P_i$ within one time-unit.

- $\alpha(P_i)$ time spent by processor $P_i$ computing tasks within one time-unit:

  $$0 \leq \alpha(P_i) \leq 1$$

- $\text{size}(v[k,m])$ size of a message containing a value $v^t_{k,m}$

- $w(P_i, T_{k,l,m})$ time needed by processor $P_i$ to compute one task $T_{k,l,m}$
Linear Program - Notations

- $s(P_i \rightarrow P_j, v_{[k,l]})$: fractional number of values $v_{[k,l]}$ sent on link $P_i \rightarrow P_j$ within one time-unit.

- $t(P_i \rightarrow P_j)$ fractional occupation time of link $P_i \rightarrow P_j$ within one time-unit:
  $$0 \leq t(P_i \rightarrow P_j) \leq 1$$

- $\text{cons}(P_i, T_{k,l,m})$: fractional number of tasks $T_{k,l,m}$ computed on processor $P_i$ within one time-unit.

- $\alpha(P_i)$ time spent by processor $P_i$ computing tasks within one time-unit:
  $$0 \leq \alpha(P_i) \leq 1$$

- $\text{size}(v_{[k,m]})$ size of a message containing a value $v^t_{[k,m]}$.

- $w(P_i, T_{k,l,m})$ time needed by processor $P_i$ to compute one task $T_{k,l,m}$. 
Linear Program - Notations

- \( s(P_i \rightarrow P_j, v_{[k,l]}) \): fractional number of values \( v_{[k,l]} \) sent on link \( P_i \rightarrow P_j \) within one time-unit.

- \( t(P_i \rightarrow P_j) \) fractional occupation time of link \( P_i \rightarrow P_j \) within one time-unit:

\[
0 \leq t(P_i \rightarrow P_j) \leq 1
\]

- \( \text{cons}(P_i, T_{k,l,m}) \): fractional number of tasks \( T_{k,l,m} \) computed on processor \( P_i \) within one time-unit.

- \( \alpha(P_i) \) time spent by processor \( P_i \) computing tasks within one time-unit:

\[
0 \leq \alpha(P_i) \leq 1
\]

- \( \text{size}(v_{[k,m]}) \) size of a message containing a value \( v^t_{[k,m]} \).

- \( w(P_i, T_{k,l,m}) \) time needed by processor \( P_i \) to compute one task \( T_{k,l,m} \).
Linear Program - Notations

- $s(P_i \to P_j, v_{[k,l]})$: fractional number of values $v_{[k,l]}$ sent on link $P_i \to P_j$ within one time-unit.
- $t(P_i \to P_j)$ fractional occupation time of link $P_i \to P_j$ within one time-unit:
  \[ 0 \leq t(P_i \to P_j) \leq 1 \]

- $cons(P_i, T_{k,l,m})$: fractional number of tasks $T_{k,l,m}$ computed on processor $P_i$ within one time-unit.

- $\alpha(P_i)$ time spent by processor $P_i$ computing tasks within one time-unit:
  \[ 0 \leq \alpha(P_i) \leq 1 \]

- $size(v_{[k,m]})$ size of a message containing a value $v^t_{[k,m]}$.
- $w(P_i, T_{k,l,m})$ time needed by processor $P_i$ to compute one task $T_{k,l,m}$.
Linear Program - Constraints

- **occupation of a link** \( P_i \rightarrow P_j \):

\[
    t(P_i \rightarrow P_j) = \sum_{v[k,l]} s(P_i \rightarrow P_j, v[k,l]) \times \text{size}(v[k,l]) \times c(i, j)
\]

- **occupation time of a processor** \( P_i \):

\[
    \alpha(P_i) = \sum_{T_{k,l,m}} \text{cons}(P_i, T_{k,l,m}) \times w(P_i, T_{k,l,m})
\]

- **“conservation law”** for packets of type \( v[k,m] \):

\[
    \sum_{P_j \rightarrow P_i} s(P_j \rightarrow P_i, v[k,m]) + \sum_{k \leq l < m} \text{cons}(P_i, T_{k,l,m})
    = \sum_{P_i \rightarrow P_j} s(P_i \rightarrow P_j, v[k,m]) + \sum_{n > m} \text{cons}(P_i, T_{k,m,n}) + \sum_{n < k} \text{cons}(P_i, T_{n,k-1,m})
\]
Linear Program - Constraints

- occupation of a link $P_i \rightarrow P_j$:

$$t(P_i \rightarrow P_j) = \sum_{v[k,l]} s(P_i \rightarrow P_j, v[k,l]) \times size(v[k,l]) \times c(i,j)$$

- occupation time of a processor $P_i$:

$$\alpha(P_i) = \sum_{T_{k,l,m}} cons(P_i, T_{k,l,m}) \times w(P_i, T_{k,l,m})$$

- "conservation law" for packets of type $v[k,m]$:  
  $$\sum_{P_j \rightarrow P_i} s(P_j \rightarrow P_i, v[k,m]) + \sum_{k \leq l < m} cons(P_i, T_{k,l,m}) = \sum_{P_i \rightarrow P_j} s(P_i \rightarrow P_j, v[k,m]) + \sum_{n > m} cons(P_i, T_{k,m,n}) + \sum_{n < k} cons(P_i, T_{n,k-1,m})$$
Linear Program - Constraints

- **occupation of a link** $P_i \rightarrow P_j$:

$$t(P_i \rightarrow P_j) = \sum_{v[k,l]} s(P_i \rightarrow P_j, v[k,l]) \times \text{size}(v[k,l]) \times c(i,j)$$

- **occupation time of a processor** $P_i$:

$$\alpha(P_i) = \sum_{T_{k,l,m}} \text{cons}(P_i, T_{k,l,m}) \times w(P_i, T_{k,l,m})$$

- **“conservation law”** for packets of type $v[k,m]$:

$$\sum_{P_j \rightarrow P_i} s(P_j \rightarrow P_i, v[k,m]) + \sum_{k \leq l < m} \text{cons}(P_i, T_{k,l,m})$$

$$= \sum_{P_i \rightarrow P_j} s(P_i \rightarrow P_j, v[k,m]) + \sum_{n > m} \text{cons}(P_i, T_{k,m,n}) + \sum_{n < k} \text{cons}(P_i, T_{n,k-1,m})$$
definition of the throughput:

\[ TP = \sum_{P_j \rightarrow P_{\text{target}}} s(P_j \rightarrow P_{\text{target}}, v_{[0,m]}) + \sum_{0 \leq l < N - 1} \text{cons}(P_{\text{target}}, T_{0,l,N}) \]

solve the following linear program over the rational numbers:

\text{Steady-State Reduce Problem on a Graph SSRP(G)}

Maximize \( TP \),
subject to all previous constraints
definition of the throughput:

\[ TP = \sum_{P_j \rightarrow P_{\text{target}}} s(P_j \rightarrow P_{\text{target}}, v_{[0,m]}) + \sum_{0 \leq l < N-1} \text{cons}(P_{\text{target}}, T_{0,l,N}) \]

solve the following linear program over the rational numbers:

**Steady-State Reduce Problem on a Graph SSRP(G)**

Maximize \( TP \),
subject to all previous constraints
Building a schedule

- consider the reduction tree $T^t$ associated with the computation of the $t^{th}$ value ($V^t$):
  - a given tree may be used by many time-stamps $t$
  - there exists an algorithm which extracts from the solution a set of weighted trees such that
    - this description is polynomial and
    - the sum of the weighted trees is equal to the original solution
  - same use of a weighted edge-coloring algorithm on a bipartite graph to orchestrate the communication
Building a schedule

- consider the reduction tree $T^t$ associated with the computation of the $t^{th}$ value ($V^t$):
  - a given tree may be used by many time-stamps $t$
- there exists an algorithm which extracts from the solution a set of weighted trees such that
  - this description is polynomial and
  - the sum of the weighted trees is equal to the original solution
- same use of a weighted edge-coloring algorithm on a bipartite graph to orchestrate the communication
Building a schedule

- consider the reduction tree $T^t$ associated with the computation of the $t^{th}$ value ($V^t$):
  - a given tree may be used by many time-stamps $t$
- there exists an algorithm which extracts from the solution a set of weighted trees such that
  - this description is polynomial and
  - the sum of the weighted trees is equal to the original solution
- same use of a weighted edge-coloring algorithm on a bipartite graph to orchestrate the communication
Building a schedule

- consider the reduction tree $T^t$ associated with the computation of the $t^{th}$ value ($V^t$):
  - a given tree may be used by many time-stamps $t$
- there exists an algorithm which extracts from the solution a set of weighted trees such that
  - this description is polynomial and
  - the sum of the weighted trees is equal to the original solution
- same use of a weighted edge-coloring algorithm on a bipartite graph to orchestrate the communication
consider the reduction tree $T^t$ associated with the computation of the $t^{th}$ value ($V^t$):

- a given tree may be used by many time-stamps $t$

there exists an algorithm which extracts from the solution a set of weighted trees such that

- this description is polynomial and
- the sum of the weighted trees is equal to the original solution

same use of a weighted edge-coloring algorithm on a bipartite graph to orchestrate the communication
consider the reduction tree $T_t$ associated with the computation of the $t^{\text{th}}$ value ($V^t$):
- a given tree may be used by many time-stamps $t$
- there exists an algorithm which extracts from the solution a set of weighted trees such that
  - this description is polynomial and
  - the sum of the weighted trees is equal to the original solution
- same use of a weighted edge-coloring algorithm on a bipartite graph to orchestrate the communication
Toy Example for Series of Reduce

topology

Loris Marchal
Steady state collective communications
Toy Example for Series of Reduce

Results of the linear program
Toy Example for Series of Reduce

first reduction tree (weight 1/3)
Toy Example for Series of Reduce

second reduction tree (weight 2/3)
Toy Example for Series of Reduce

bipartite graph
Toy Example for Series of Reduce

\[ P_0 \rightarrow P_1 \rightarrow P_2 \]

first matching second matching
Toy Example for Series of Reduce

Diagram:

- $P_0$
- $P_1$
- $P_2$

Steady state collective communications
Outline

Introduction
Two Problems of Collective Communication
Platform Model
Framework

Series of Scatter
Steady-state constraints
Toy Example
Building a schedule
Asymptotic optimality

Series of Reduce
Introduction to reduction trees
Linear Program
Periodic schedule - Asymptotic optimality
Toy Example for Series of Reduce

Approximation for a fixed period

Conclusion
Approximation for a fixed period

- our framework produces an asymptotically optimal schedule of period $T$, but $T$ may be too large

- we can approximate the solution with a fixed period $T_{\text{fixed}}$:
  1. $\{T, \text{weight}_T\}$: the weighted set of trees obtained by the decomposition algorithm
  2. compute $r(T) = \left\lfloor \text{weight}(T) \times T_{\text{fixed}} \right\rfloor$
  3. one port constraints are satisfied for $\{T, \text{weight}_T\}$ on a period $T$
     $\Rightarrow$ they are satisfied for $\{T, r(T)\}$ on a period $T_{\text{fixed}}$
  4. the performance loss is bounded:

\[
TP - TP^* \leq \frac{\text{card} (\text{TREES})}{T_{\text{fixed}}}
\]
Approximation for a fixed period

- our framework produces an asymptotically optimal schedule of period $T$, but $T$ may be too large

- we can approximate the solution with a fixed period $T_{fixed}$:
  1. $\{T, \text{weight}_T\}$: the weighted set of trees obtained by the decomposition algorithm
  2. compute $r(T) = \left\lfloor \frac{\text{weight}(T)}{T} \times T_{fixed} \right\rfloor$
  3. one port constraints are satisfied for $\{T, \text{weight}_T\}$ on a period $T$,
     $\Rightarrow$ they are satisfied for $\{T, r(T)\}$ on a period $T_{fixed}$
  4. the performance loss is bounded:

\[ TP - TP^* \leq \frac{\text{card(TREES)}}{T_{fixed}} \]
Approximation for a fixed period

- our framework produces an asymptotically optimal schedule of period \( T \), but \( T \) may be too large
- we can approximate the solution with a fixed period \( T_{\text{fixed}} \):
  1. \( \{T, \text{weight}_T\} \): the weighted set of trees obtained by the decomposition algorithm
  2. compute \( r(T) = \left\lfloor \frac{\text{weight}(T)}{T} \times T_{\text{fixed}} \right\rfloor \)
  3. one port constraints are satisfied for \( \{T, \text{weight}_T\} \) on a period \( T \),
     \( \Rightarrow \) they are satisfied for \( \{T, r(T)\} \) on a period \( T_{\text{fixed}} \)
  4. the performance loss is bounded:

\[
TP - TP^* \leq \frac{\text{card(TREES)}}{T_{\text{fixed}}}
\]
Approximation for a fixed period

- our framework produces an asymptotically optimal schedule of period \( T \), but \( T \) may be too large
- we can approximate the solution with a fixed period \( T_{\text{fixed}} \):
  1. \( \{T, \text{weight}_T\} \): the weighted set of trees obtained by the decomposition algorithm
  2. compute \( r(T) = \left\lfloor \frac{\text{weight}(T)}{T} \times T_{\text{fixed}} \right\rfloor \)
  3. one port constraints are satisfied for \( \{T, \text{weight}_T\} \) on a period \( T \),
     \( \Rightarrow \) they are satisfied for \( \{T, r(T)\} \) on a period \( T_{\text{fixed}} \)
  4. the performance loss is bounded:

\[
TP - TP^* \leq \frac{\text{card}(\text{TREES})}{T_{\text{fixed}}}
\]
Approximation for a fixed period

- our framework produces an asymptotically optimal schedule of period $T$, but $T$ may be too large
- we can approximate the solution with a fixed period $T_{\text{fixed}}$:
  1. $\{T, \text{weight}_{T}\}$: the weighted set of trees obtained by the decomposition algorithm
  2. compute $r(T) = \left\lfloor \frac{\text{weight}(T)}{T} \times T_{\text{fixed}} \right\rfloor$
  3. one port constraints are satisfied for $\{T, \text{weight}_{T}\}$ on a period $T$,
     $\Rightarrow$ they are satisfied for $\{T, r(T)\}$ on a period $T_{\text{fixed}}$
  4. the performance loss is bounded:

\[
    TP - TP^* \leq \frac{\text{card}(\text{TREES})}{T_{\text{fixed}}}
\]
Approximation for a fixed period

▶ our framework produces an asymptotically optimal schedule of period $T$, but $T$ may be too large

▶ we can approximate the solution with a fixed period $T_{fixed}$:

1. $\{T, weight_T\}$: the weighted set of trees obtained by the decomposition algorithm

2. compute $r(T) = \left\lfloor \frac{weight(T)}{T} \times T_{fixed} \right\rfloor$

3. one port constraints are satisfied for $\{T, weight_T\}$ on a period $T$,
   $\Rightarrow$ they are satisfied for $\{T, r(T)\}$ on a period $T_{fixed}$

4. the performance loss is bounded:

\[
TP - TP^* \leq \frac{card(TREES)}{T_{fixed}}
\]
Outline

Introduction
- Two Problems of Collective Communication
- Platform Model
- Framework

Series of Scatter
- Steady-state constraints
- Toy Example
- Building a schedule
- Asymptotic optimality

Series of Reduce
- Introduction to reduction trees
- Linear Program
- Periodic schedule - Asymptotic optimality
- Toy Example for Series of Reduce

Approximation for a fixed period

Conclusion
Conclusion

▶ new framework to study collective communications in a heterogeneous environment
▶ makespan difficult to minimize ⇒ focus on throughput
▶ relaxation, use of linear programming
▶ asymptotically optimal algorithm
▶ can be extended to other communication schemes and scheduling problems