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

Outline

Outline

Outline

Outline

Outline

Outline

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

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

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

Two Problems of Collective Communication

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

- Series of Scatter P_{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
$\Rightarrow$ 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 $P_{r_0}, \ldots, P_{r_N}$ to the same target P_{target}.

Two Problems of Collective Communication

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

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

Reduce Each of the participating processor P_{r_i} in $P_{r_0}, \dots, P_{r_N}$ 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 $P_{r_0}, \dots, P_{r_N}$ to the same target P_{target}.

Two Problems of Collective Communication

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

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

Reduce Each of the participating processor P_{r_i} in $P_{r_0}, \dots, P_{r_N}$ 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 $P_{r_0}, \dots, P_{r_N}$ to the same target P_{target}.

Two Problems of Collective Communication

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

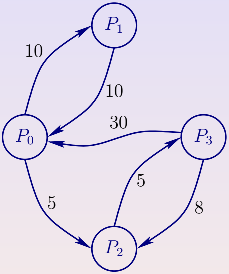
- Series of Scatter P_{source} sends consecutively a large number of distinct messages to all targets

Reduce Each of the participating processor P_{r_i} in $P_{r_0}, \ldots, P_{r_N}$ owns a value v_i
$\Rightarrow$ 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 $P_{r_0}, \ldots, P_{r_N}$ to the same target P_{target}.

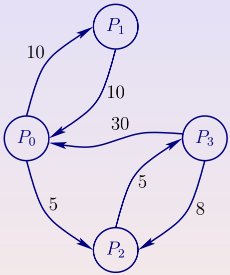
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: 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



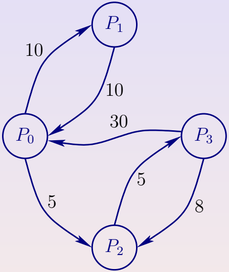
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: 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



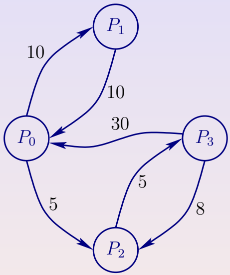
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: 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



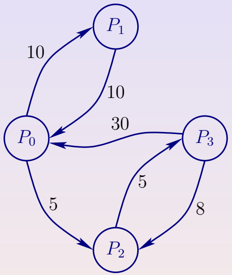
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: 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



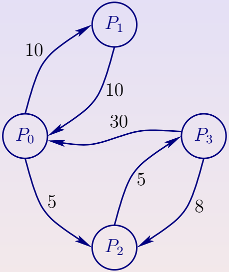
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: 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



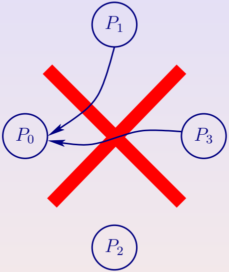
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: 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



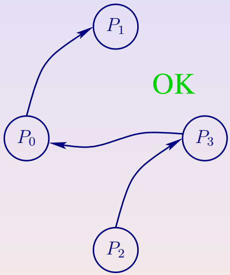
Platform Model

- $G = (V, E, c)$
- $P_1, P_2, \ldots, P_n$: 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

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

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

Outline

Series of Scatter

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

$$\forall P_i, P_j, \quad 0 \leqslant t(P_i \to P_j) \leqslant 1$$

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

$$t(P_i \to P_j) = \sum_k s(P_i \to P_j, m_k)$$

Series of Scatter

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

$$\forall P_i, P_j, \quad 0 \leqslant t(P_i \to P_j) \leqslant 1$$

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

$$t(P_i \to P_j) = \sum_k s(P_i \to P_j, m_k)$$

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:

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

- 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:

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

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

$$\forall P_i, P_j, \quad 0 \leqslant t(P_i \to P_j) \leqslant 1$$

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

$$t(P_i \to P_j) = \sum_k s(P_i \to P_j, m_k)$$

Linear constraints

- one port constraints for outgoing messages in P_i:

$$\forall P_i, \quad \sum_{P_i \to P_j} t(P_i \to P_j) \leqslant 1$$

- one port constraints for incoming messages in P_i:

$$\forall P_i, \quad \sum_{P_j \to P_i} t(P_j \to P_i) \leqslant 1$$

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

$$\sum_{P_j \to P_i} s(P_j \to P_i, m_k) = \sum_{P_i \to P_j} s(P_j \to P_i, m_k) \leqslant 1$$

Linear constraints

- one port constraints for outgoing messages in P_i:

$$\forall P_i, \quad \sum_{P_i \to P_j} t(P_i \to P_j) \leqslant 1$$

- one port constraints for incoming messages in P_i:

$$\forall P_i, \quad \sum_{P_j \to P_i} t(P_j \to P_i) \leqslant 1$$

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

$$\sum_{P_j \to P_i} s(P_j \to P_i, m_k) = \sum_{P_i \to P_j} s(P_j \to P_i, m_k) \leqslant 1$$

Linear constraints

- one port constraints for outgoing messages in P_i:

$$\forall P_i, \quad \sum_{P_i \to P_j} t(P_i \to P_j) \leqslant 1$$

- one port constraints for incoming messages in P_i:

$$\forall P_i, \quad \sum_{P_j \to P_i} t(P_j \to P_i) \leqslant 1$$

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

$$\sum_{P_j \to P_i} s(P_j \to P_i, m_k) = \sum_{P_i \to P_j} s(P_j \to P_i, m_k) \leqslant 1$$

Linear constraints

- one port constraints for outgoing messages in P_i:

$$\forall P_i, \quad \sum_{P_i \to P_j} t(P_i \to P_j) \leqslant 1$$

- one port constraints for incoming messages in P_i:

$$\forall P_i, \quad \sum_{P_j \to P_i} t(P_j \to P_i) \leqslant 1$$

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

$$\sum_{P_j \to P_i} s(P_j \to P_i, m_k) = \sum_{P_i \to P_j} s(P_j \to P_i, m_k) \leqslant 1$$

Linear constraints

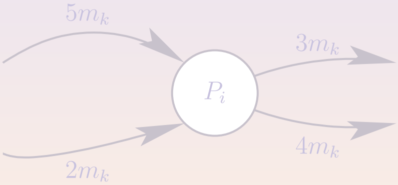
- one port constraints for outgoing messages in P_i:

$$\forall P_i, \quad \sum_{P_i \to P_j} t(P_i \to P_j) \leqslant 1$$

- one port constraints for incoming messages in P_i:

$$\forall P_i, \quad \sum_{P_j \to P_i} t(P_j \to P_i) \leqslant 1$$

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



$$\sum_{P_j \to P_i} s(P_j \to P_i, m_k) = \sum_{P_i \to P_j} s(P_j \to P_i, m_k) \leqslant 1$$

Linear constraints

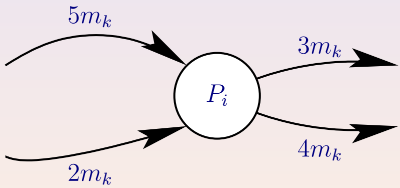
- one port constraints for outgoing messages in P_i:

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

- one port constraints for incoming messages in P_i:

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

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



$$\sum_{P_j \rightarrow P_i} s(P_j \rightarrow P_i, m_k) = \sum_{P_i \rightarrow P_j} s(P_j \rightarrow P_i, m_k) \leqslant 1$$

Throughput and Linear Program

- throughput: total number of messages m_k received in P_k

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

(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

$$\left\{\begin{array}{l}
\forall P_i, \forall P_j, 0 \leqslant s(P_i \to P_j) \leqslant 1 \\
\forall P_i, \sum_{P_j,(i,j)\in E} s(P_i \to P_j) \leqslant 1 \\
\forall P_i, \sum_{P_j,(j,i)\in E} s(P_j \to P_i) \leqslant 1 \\
\forall P_i, P_j, s(P_i \to P_j) = \sum_{m_k} send(P_i \to P_j, m_k) \times c(i,j) \\
\forall P_i, \forall m_k, k \neq i, \sum_{P_j,(j,i)\in E} send(P_j \to P_i, m_k) \\
\qquad = \sum_{P_j,(i,j)\in E} send(P_i \to P_j, m_k) \\
\forall P_k, k \in T \sum_{P_i,(i,k)\in E} send(P_i \to P_k, m_k) = \mathrm{TP}
\end{array}\right.$$

Throughput and Linear Program

- throughput: total number of messages m_k received in P_k

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

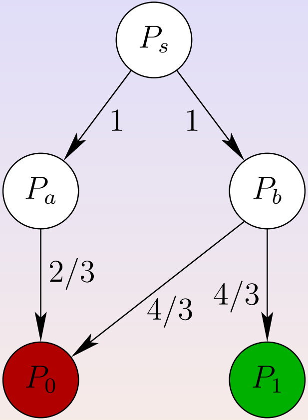
(same throughput for every target node P_k)

- summarize this constraints in a linear program:

  STEADY-STATE SCATTER PROBLEM ON A GRAPH SSSP(G)
  Maximize TP,
  subject to

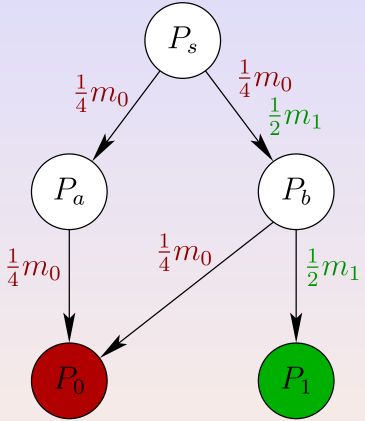
$$\begin{cases} \forall P_i, \forall P_j, 0 \leqslant s(P_i \rightarrow P_j) \leqslant 1 \\ \forall P_i, \sum_{P_j,(i,j)\in E} s(P_i \rightarrow P_j) \leqslant 1 \\ \forall P_i, \sum_{P_j,(j,i)\in E} s(P_j \rightarrow P_i) \leqslant 1 \\ \forall P_i, P_j, s(P_i \rightarrow P_j) = \sum_{m_k} send(P_i \rightarrow P_j, m_k) \times c(i,j) \\ \forall P_i, \forall m_k, k \neq i, \sum_{P_j,(j,i)\in E} send(P_j \rightarrow P_i, m_k) \\ \qquad = \sum_{P_j,(i,j)\in E} send(P_i \rightarrow P_j, m_k) \\ \forall P_k, k \in T \sum_{P_i,(i,k)\in E} send(P_i \rightarrow P_k, m_k) = \mathrm{TP} \end{cases}$$

Series of Scatter - Toy Example



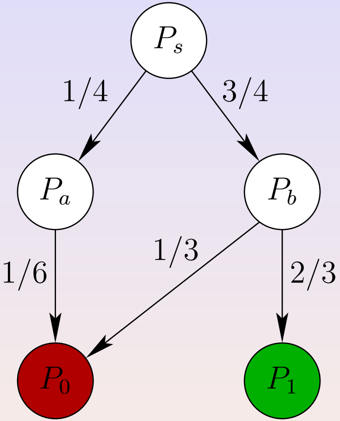
platform graph (edges labeled with $c(i, j)$)

Series of Scatter - Toy Example



value of $s(P_i \to P_j, m_k)$ in the solution of the linear program

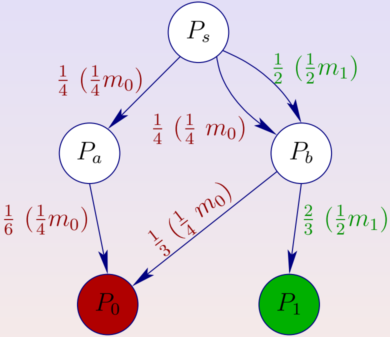
Series of Scatter - Toy Example



occupation time of the edge ($t(P_i \rightarrow P_j)$)

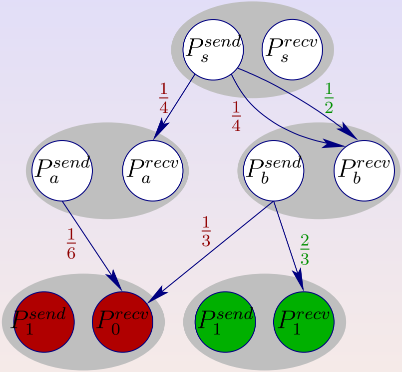
Building a schedule

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



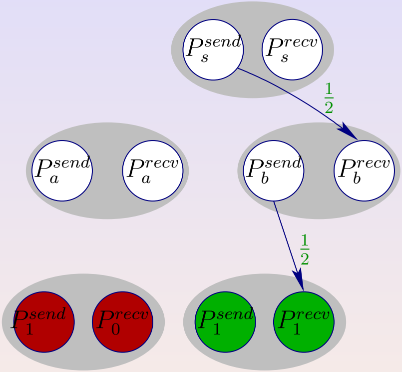
Building a schedule

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



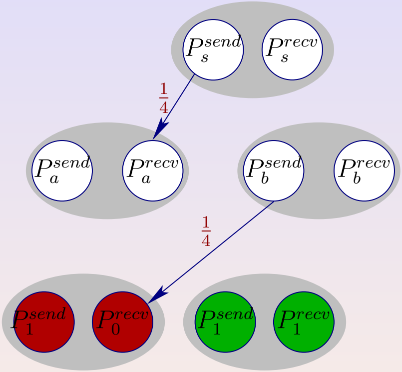
Building a schedule

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



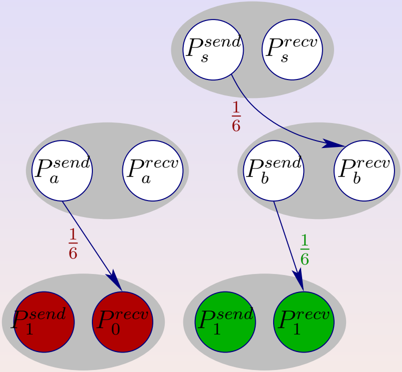
Building a schedule

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



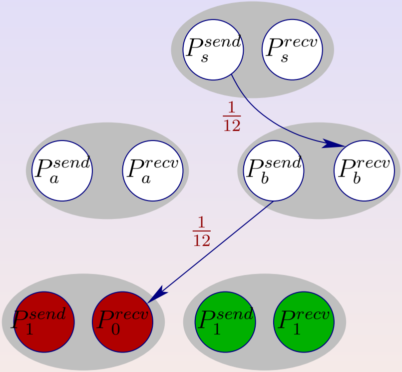
Building a schedule

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

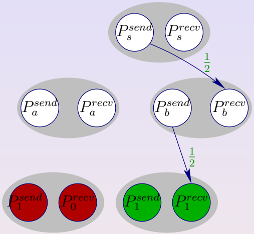


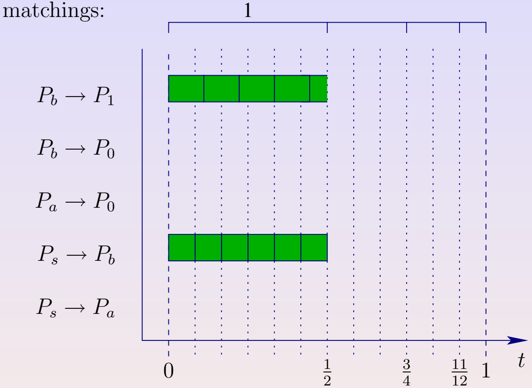
Building a schedule

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



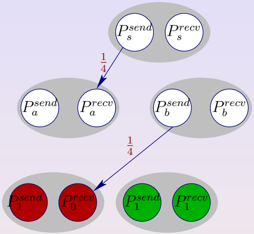
Building a schedule

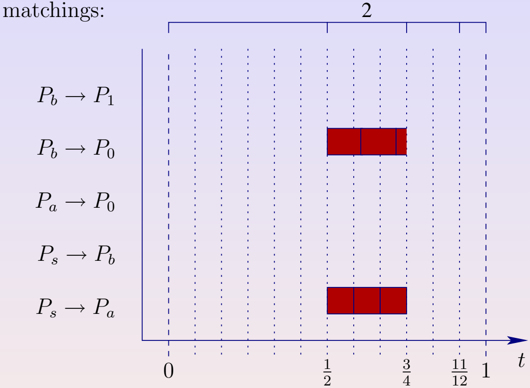




- least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transfered in each matching
- $\Rightarrow$ periodic schedule of period T with atomic transfers of messages

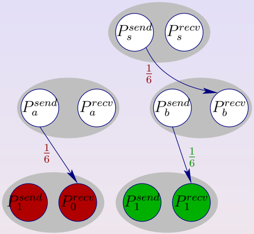
Building a schedule

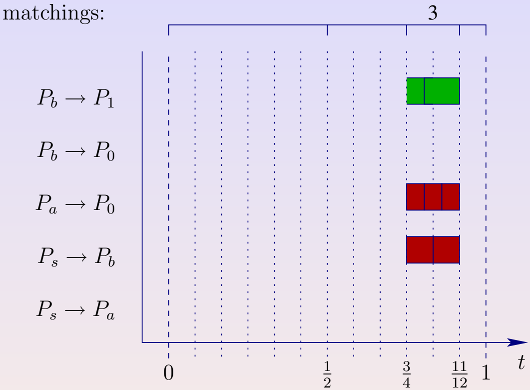




- least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transfered in each matching
- $\Rightarrow$ periodic schedule of period T with atomic transfers of messages

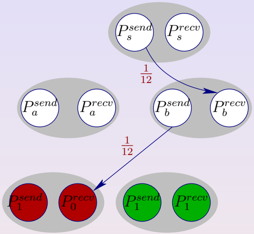
Building a schedule

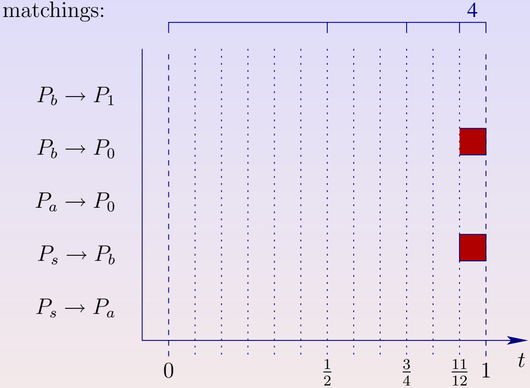




- least common multiple $T = \mathrm{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transfered in each matching
- $\Rightarrow$ periodic schedule of period T with atomic transfers of messages

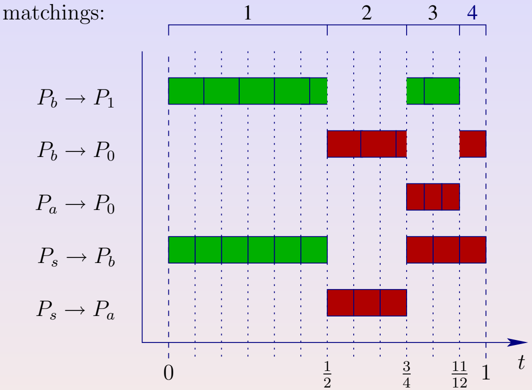
Building a schedule





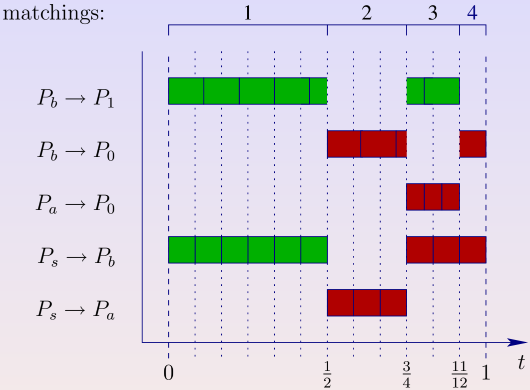
- least common multiple $T = \text{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transfered in each matching
- $\Rightarrow$ periodic schedule of period T with atomic transfers of messages

Building a schedule



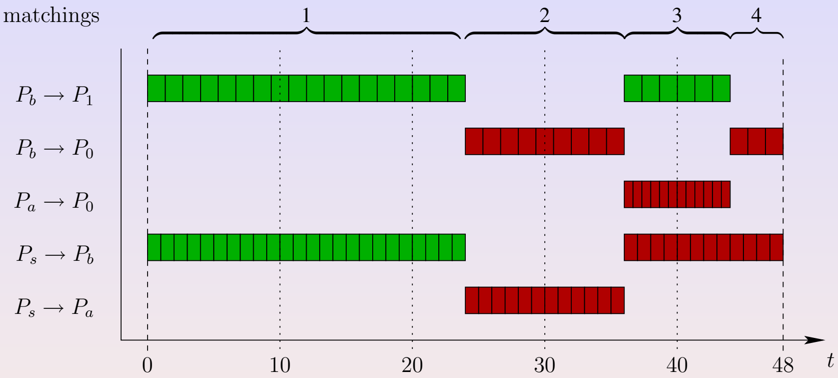
- least common multiple $T = \mathrm{lcm}\{b_i\}$ where $\frac{a_i}{b_i}$ denotes the number of messages transfered in each matching
- $\Rightarrow$ periodic schedule of period T with atomic transfers of messages

Building a schedule



- least common multiple $T = \mathrm{lcm}\{b_i\}$ 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

Building a schedule



- least common multiple $T = \mathrm{lcm}\{b_i\}$ 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

Asymptotic optimality

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

Lemma.

$opt(G, K) \leqslant \mathrm{TP}(G) \times K$

- periodic schedule $\Rightarrow$ schedule:

3. clean-up phase (empty buffers)

the previous algorithm is asymptotically optimal

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1$$

Asymptotic optimality

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

Lemma.

$opt(G, K) \leqslant \mathrm{TP}(G) \times K$

- 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)

the previous algorithm is asymptotically optimal:

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1$$

Asymptotic optimality

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

Lemma.

$opt(G, K) \leqslant \mathrm{TP}(G) \times K$

- 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.

the previous algorithm is asymptotically optimal:

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1$$

Asymptotic optimality

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

Lemma.

$opt(G, K) \leqslant \mathrm{TP}(G) \times K$

- 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.

the previous algorithm is asymptotically optimal:

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1$$

Asymptotic optimality

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

Lemma.

$opt(G, K) \leqslant \mathrm{TP}(G) \times K$

- 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.

the previous algorithm is asymptotically optimal:

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1$$

Asymptotic optimality

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

Lemma.

$opt(G, K) \leqslant \mathrm{TP}(G) \times K$

- 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.

the previous algorithm is asymptotically optimal:

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1$$

Asymptotic optimality

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

Lemma.

$opt(G, K) \leqslant \mathrm{TP}(G) \times K$

- 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.

the previous algorithm is asymptotically optimal:

$$\lim_{K \to +\infty} \frac{steady(G, K)}{opt(G, K)} = 1$$

Outline

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

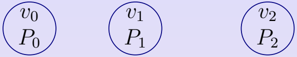
- Reduce:
  - each processor P_{r_i} owns a value v_i
  - compute $V = v_1 \oplus v_2 \oplus \cdots \oplus v_N$ ($\oplus$ associative, non commutative)
- partial result of the Reduce operation: $v_{[k,m]} = v_k \oplus v_2 \oplus \cdots \oplus v_m$
- two partial results can be merged: $v_{[k,m]} = v_{[k,l]} \oplus v_{[l+1,m]}$ (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 $T_{k,l,m}$)

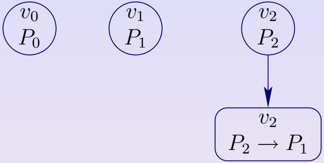
Reduce - Reduction trees

- 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 $T_{k,l,m}$)



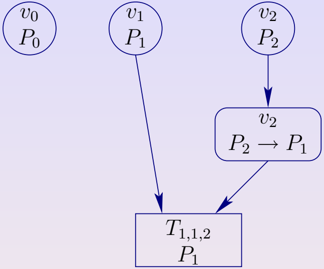
Reduce - Reduction trees

- Reduce:
  - each processor P_{r_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 $T_{k,l,m}$)



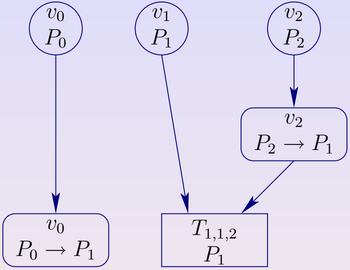
Reduce - Reduction trees

- 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 $T_{k,l,m}$)



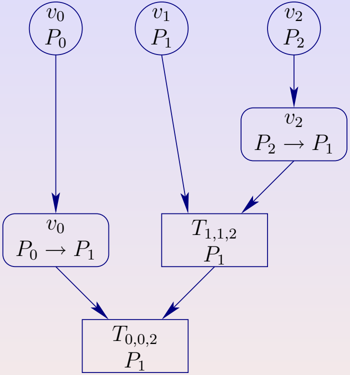
Reduce - Reduction trees

- 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 $T_{k,l,m}$)



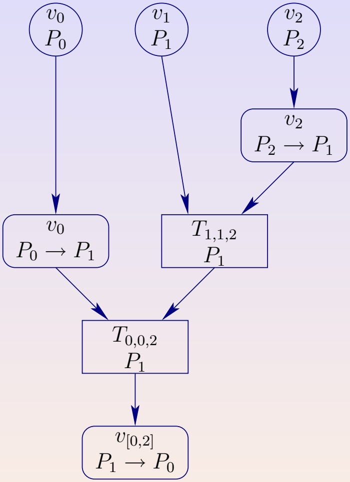
Reduce - Reduction trees

- Reduce:
  - each processor P_{r_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 $T_{k,l,m}$)



Series of Reduce

- each processor P_{r_i} 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

- $s(P_i \to P_j, v_{[k,l]})$: 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:

$$0 \leqslant t(P_i \to P_j) \leqslant 1$$

- $cons(P_i, T_{k,l,m})$: 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:

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

- $size(v_{[k,m]})$ 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 - 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:
$$0 \leqslant t(P_i \to P_j) \leqslant 1$$
- $cons(P_i, T_{k,l,m})$: 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:
$$0 \leqslant \alpha(P_i) \leqslant 1$$
- $size(v_{[k,m]})$ 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 - 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:
$$0 \leqslant t(P_i \rightarrow P_j) \leqslant 1$$
- $cons(P_i, T_{k,l,m})$: 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:
$$0 \leqslant \alpha(P_i) \leqslant 1$$
- $size(v_{[k,m]})$ 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 - 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:

$$0 \leqslant t(P_i \to P_j) \leqslant 1$$

- $cons(P_i, T_{k,l,m})$: 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:

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

- $size(v_{[k,m]})$ 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 - 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:
$$0 \leqslant t(P_i \to P_j) \leqslant 1$$
- $cons(P_i, T_{k,l,m})$: 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:
$$0 \leqslant \alpha(P_i) \leqslant 1$$
- $size(v_{[k,m]})$ size of a message containing a value $v_{[k,m]}^t$
- $w(P_i, T_{k,l,m})$ 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:

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

- $cons(P_i, T_{k,l,m})$: 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:

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

- $size(v_{[k,m]})$ 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 $P_i \to P_j$:

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

- 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 $v_{[k,m]}$:

$$\sum_{P_j \to P_i} s(P_j \to P_i, v_{[k,m]}) + \sum_{k \leqslant l < m} cons(P_i, T_{k,l,m})$$
$$= \sum_{P_i \to P_j} s(P_i \to P_j, v_{[k,m]}) + \sum_{n > m} cons(P_i, T_{k,m,n}) + \sum_{n < k} cons(P_i, T_{n,k-1,m})$$

Linear Program - Constraints

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

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

- 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 $v_{[k,m]}$:

$$\sum_{P_j \to P_i} s(P_j \to P_i, v_{[k,m]}) + \sum_{k \leqslant l < m} cons(P_i, T_{k,l,m})$$
$$= \sum_{P_i \to P_j} s(P_i \to P_j, v_{[k,m]}) + \sum_{n > m} cons(P_i, T_{k,m,n}) + \sum_{n < k} cons(P_i, T_{n,k-1,m})$$

Linear Program - Constraints

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

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

- 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 $v_{[k,m]}$:

$$\sum_{P_j \to P_i} s(P_j \to P_i, v_{[k,m]}) + \sum_{k \leqslant l < m} cons(P_i, T_{k,l,m})$$
$$= \sum_{P_i \to P_j} s(P_i \to P_j, v_{[k,m]}) + \sum_{n > m} cons(P_i, T_{k,m,n}) + \sum_{n < k} cons(P_i, T_{n,k-1,m})$$

Linear Program - Constraints

- definition of the throughput:

$$\mathrm{TP} = \sum_{P_j \to P_{\mathsf{target}}} s(P_j \to P_{\mathsf{target}}, v_{[0,m]}) + \sum_{0 \leqslant l < N-1} cons(P_{\mathsf{target}}, T_{0,l,N})$$

- 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

Linear Program - Constraints

- definition of the throughput:

$$\mathrm{TP} = \sum_{P_j \to P_{\mathsf{target}}} s(P_j \to P_{\mathsf{target}}, v_{[0,m]}) + \sum_{0 \leqslant l < N-1} cons(P_{\mathsf{target}}, T_{0,l,N})$$

- 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 $\mathcal{T}^t$ 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 $\mathcal{T}^t$ 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 $\mathcal{T}^t$ 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 $\mathcal{T}^t$ 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

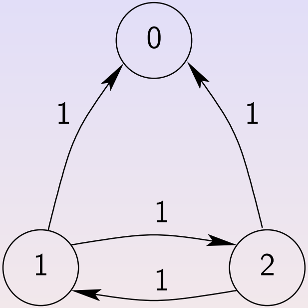
Building a schedule

- consider the reduction tree $\mathcal{T}^t$ 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

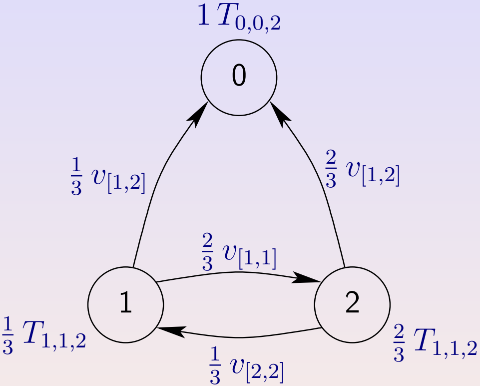
- consider the reduction tree $\mathcal{T}^t$ associated with the computation of the t^{th} value (V^t):
    - a given tree may be used by many time-stamps t
- there exists an algorithm which extracts from the solution a set of weighted trees such that
    - this description is polynomial and
    - the 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

Toy Example for Series of Reduce



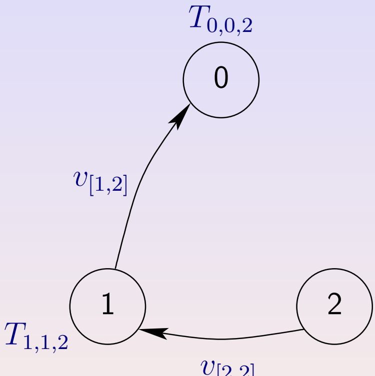
topology

Toy Example for Series of Reduce



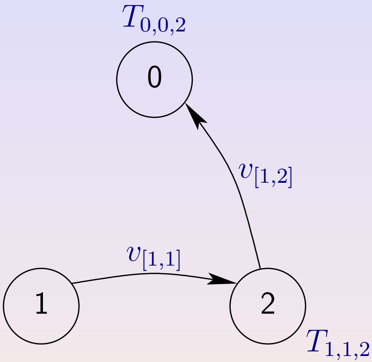
results of the linear program

Toy Example for Series of Reduce



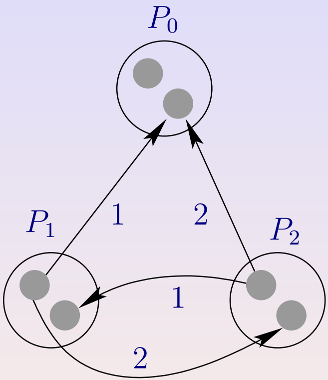
first reduction tree (weight 1/3)

Toy Example for Series of Reduce



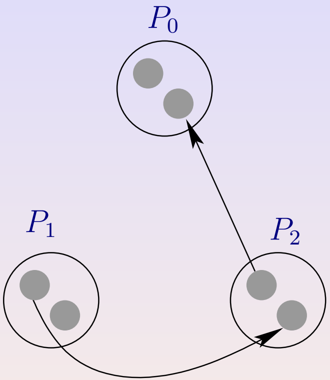
second reduction tree (weight 2/3)

Toy Example for Series of Reduce



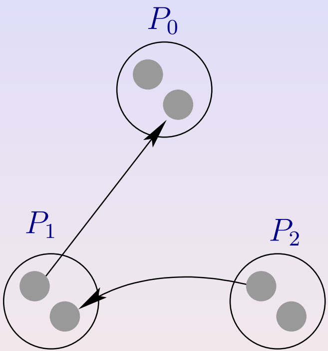
bipartite graph

Toy Example for Series of Reduce



first matching second matching

Toy Example for Series of Reduce



Outline

Approximation for a fixed period

- our framework produces an asymptotically optimal schedule of period T, but T may be to 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 $r(T) = \left\lfloor \frac{weight(T)}{p} \cdot T_{fixed} \right\rfloor$
  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:

$$\mathrm{TP} - \mathrm{TP}^* \leqslant \frac{card(\textsc{Trees})}{T_{fixed}}$$

Approximation for a fixed period

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

$$\mathrm{TP} - \mathrm{TP}^* \leqslant \frac{card(\textsc{Trees})}{T_{fixed}}$$

Approximation for a fixed period

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

$$\mathrm{TP} - \mathrm{TP}^* \leqslant \frac{card(\textsc{Trees})}{T_{fixed}}$$

Approximation for a fixed period

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

$$\mathrm{TP} - \mathrm{TP}^* \leqslant \frac{card(\textsc{Trees})}{T_{fixed}}$$

Approximation for a fixed period

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

$$\mathrm{TP} - \mathrm{TP}^* \leqslant \frac{card(\textsc{Trees})}{T_{fixed}}$$

Approximation for a fixed period

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

$$\mathrm{TP} - \mathrm{TP}^* \leqslant \frac{card(\textsc{Trees})}{T_{fixed}}$$

Outline

Conclusion

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