線形ネットワークにおける最適な通信複雑度の分散ソーティングアルゴリズム

01207474 NTTコミュニケーション科学基礎研究所 佐々木淳 SASAKI Atsushi

1 はじめに

様々なアプリケーションにおいて，ソーティングは基本かつ重要な問題のひとつである．そのため，分散アルゴリズムの分野においてもソーティングアルゴリズムが研究されている．従来の分散ソーティングアルゴリズム[1]の通信複雑度は$\frac{3}{4}n^2+O(n)$である．しかし，通信複雑度の下界は$\frac{1}{2}n^2$であるため，さらに下げることが可能である．そこで，問題設定および対象とするネットワークに制限を加えて，最適な通信複雑度を実現するアルゴリズムを構築する．その際，時間複雑度を抑えることはせず，通信複雑度の最適性のみを考慮する．なお，時間複雑度を重視したアルゴリズムは文献[2]で既に構築されている．

2 モデルと問題の定義

本稿ではプロセッサ$P_1,P_2,\ldots,P_n$を$P_i,1\leq i<n$とP_{i+1}の間に双方向通信リンクを持つように結合した線形ネットワーク[3]を扱う．一般性を失うことなく，P_1が左端となるように水平に置かれていると仮定する．このとき，各プロセッサは隣接プロセッサを“*left*”と“*right*”という局所的な名前でしか認識できない．但し，この左右の認識はどのプロセッサでも一致していることを仮定する．さらに，この認識には端の認識も含まれ，左端(P_1)では$left=null$，右端(P_n)では$right=null$とする．また，プロセッサ数nも知らないと仮定する．

このような仮定の下で，各プロセッサが隣接プロセッサとの通信を繰り返して問題を解くが，そのモデルとして同期モデルと非同期モデルとがある[3, 4]．また，分散アルゴリズムの評価には，時間複雑度と通信複雑度の二つの基準が存在する．前者は同期モデルではラウンド数，非同期モデルではメッセージの最長鎖に属するメッセージ数で表される．一方，後者にはメッセージ複雑度とビット複雑度があり，それぞれ，メッセージの総数，メッセージに含まれる総ビット数で表される．

最後に問題を定義する．本稿で扱う分散ソーティング問題は，下記の条件を満たすように要素を移動させる問題である．

初期状態： 各プロセッサはソーティングの対象となる要素をひとつずつ保持．

最終状態： P_iはi番目に小さな要素を保持．

3 従来のアルゴリズム[1]の概要

従来の分散ソーティングアルゴリズムでは，

1. 木を構成．
2. 葉から根に向かって最小値を順次送信．
3. 根は各部分木からそれぞれの最小値が届いたら，そのなかの最小値を求め，それを最小値を保持すべきプロセッサに向けて送信．
4. 以後，小さな要素から順次同様の処理を実行．

という操作により，メッセージ複雑度を$\frac{3}{4}n^2+O(n)$に抑えている．従来の分散ソーティング問題では，本稿の問題のように端から順番に並べるという問題ではなく，識別子に対応した順番に並べるという問題であるために，このような集中制御的なアルゴリズムを必要とすると思われる．

4 アルゴリズムの概要

本稿の問題では，どのプロセッサから見ても左に小さな要素，右に大きな要素が来ることがわかるため，配送先の判断は不必要になる．そのため，従来のアルゴリズムの根に相当するような特殊なプロセッサを用意することなく，ソートを実行できる．その結果，プロセッサ間の処理の公平性も向上する．

大域的な視点から見た本質的な処理の流れを以下に示す．u_iはプロセッサP_iが保持している要素を表す．

奇数フェイズ

1. P_ℓ(初期値：$\ell=1$)はu_ℓを*right*へ送信．
2. *left*から要素vを受信した$P_i,\ell<i<r$（初期値：$r=n$）は，*right*へ$\max\{u_i,v\}$を送信し，$u_i:=\min\{u_i,v\}$．
3. P_rは，*left*から要素vを受信したら$w:=\min\{u_r,v\}$，$u_r:=\max\{u_r,v\}$．
4. 次の偶数フェイズへ．

偶数フェイズ

1. P_rはwを*left*へ送信．
2. *right*から要素vを受信した$P_i,\ell<i<r$は，*left*へ$\min\{u_i,v\}$を送信し，$u_i:=\max\{u_i,v\}$．
3. P_ℓは，*right*から要素vを受信したら$u_\ell:=v$．
4. $\ell:=\ell+1,r:=r-1$として次の奇数フェイズへ．

この処理から明らかなように，奇数フェイズ終了時点でP_rにP_ℓからP_rが保持する要素中の最大値が，偶数フェイズ終了時点でP_ℓに$P_{\ell+1}$からP_rが保持する要素中の最小値が存在する．これによりソーティングが正し

く行われることは明らかである．なお，この処理は，偶数フェイズ終了時点で$\ell = r$または$\ell = r-1$となったとき終了する．従って，nが偶数のときnフェイズ，nが奇数のとき$n-1$フェイズで終了する．

5 アルゴリズム

本章では，前章で示した大域的な視点の処理を，プロセッサごとの局所的な視点からアルゴリズムとして記述する．局所的な視点ではフェイズの値は必要なく，代わりに，そのフェイズにおける端のプロセッサが端であることを認識できれば良い．アルゴリズムにおいては，端のプロセッサが要素を送信する際，*terminal*メッセージを付加することで端プロセッサの交替を実現する．

以下にプロセッサP_iのアルゴリズムを示す．このアルゴリズムはP_1が起動することを仮定しており，そのときの処理は$send((u, terminal), right)$だけである．ここで，$send(m, P)$はプロセッサ$P$にメッセージ$m$を送信することを表す．なお，他のプロセッサが起動する場合には，P_1へ起動メッセージを送信すれば良い．

変数定義：

state： P_iの状態で次のいずれか：

left-terminal： そのフェイズの左端
next-left-terminal： 次の奇数フェイズの左端
right-terminal： そのフェイズの右端
non-terminal： 端プロセッサでないが動作中
sleep： アルゴリズムの実行終了

初期値： *left*=*null*のとき *left-terminal*,
right=*null*のとき *right-terminal*,
それ以外のとき *non-terminal*.

u： 要素を保持する変数，初期値：与えられた要素.

(v, mes)： 受信したメッセージ，v：要素，*mes*：制御メッセージ（*terminal*か*null*）.

アルゴリズム（P_iの1ラウンド中の処理）：

左からメッセージを受信したとき：
  if *state* = *right-terminal* then
    $send((\min\{u,v\}, terminal), left)$
    $u := \max\{u,v\}$
    state := *sleep*
  if *mes* = *terminal* then
    state :=*next-left-terminal*
  $send((\max\{u,v\}, null), right)$
  $u := \min\{u,v\}$

右からメッセージを受信したとき：
  if *state* = *left-terminal* then
    $u := v$
    state := *sleep*
  if *state* = *next-left-terminal* and
      mes = *terminal* then
    $send((\min\{u,v\}, null), left)$
    $u := \max\{u,v\}$
    state := *sleep*
  if *mes* = *terminal* then
    state :=*right-terminal*
  $send((\min\{u,v\}, null), left)$
  $u := \max\{u,v\}$
  if *state* = *next-left-terminal* then
    state :=*left-terminal*
    $send((u, terminal), right)$

6 通信複雑度

iフェイズ目には，iが奇数ならば$n-i$個，iが偶数ならば$n-i+1$個のメッセージが送られる．従って，このアルゴリズムの総メッセージ数は，nが奇数の場合に$\sum_{i=1}^{n-1}(n-i) + (n-1)/2 = (n^2-1)/2$，$n$が偶数の場合に$\sum_{i=1}^{n}(n-i) + n/2 = n^2/2$となる．これは下界[†]と一致するため，最適なメッセージ複雑度を実現していることがわかる．なお，時間複雑度はメッセージ複雑度とほぼ同じになるので，オーダーが下界$(n-1)$[2]よりも大きく$(O(n^2))$なってしまう．また，1メッセージ中の要素数は高々1なので，ビット複雑度はメッセージ複雑度に最大要素のビット数を掛ければ得られる．なお，上記の計算では起動メッセージは考慮していない．

7 おわりに

このアルゴリズムは，時間複雑度を犠牲にする代わりに通信複雑度を最適にし，かつ，Zaksのアルゴリズム[1]の根のような特殊な処理をするプロセッサを用意することなしに，ある程度公平に動作する．見方を変えると，このアルゴリズムは逐次のバブルソートの変形とも見なせる．なお，奇数フェイズとその次の偶数フェイズをまとめて同時に実行することで，メッセージ複雑度を最適に保ちつつ，時間複雑度を約半分にできる．

また，文献[2]のアルゴリズムと合わせることで，状況による適切なアルゴリズムの選択が可能になる．

参考文献

[1] Shmuel Zaks, Optimal Distributed Algorithms for Sorting and Ranking, *IEEE Transactions on Computers*, Vol. C-34, No. 4, pp. 376-379, 1985.

[2] 佐々木淳，線形ネットワークにおける時間複雑度を重視した分散ソーティングアルゴリズム，*1999*年度日本*OR*学会秋季研究発表会アブストラクト集，pp. 188-189, 1999.

[3] Nancy A. Lynch, *Distributed Algorithms*, Morgan Kaufmann Publishers, 1996.

[4] 亀田恒彦，山下雅史，分散アルゴリズム，近代科学社，1994.

[†]紙面の都合で証明は省略する．