MapReduce計算の並列複雑さに関する研究

MAMADA, Tsuyoshi / 間々田, 剛史

(出版者 / Publisher)
法政大学大学院理工学・工学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 理工学・工学研究科編 / 法政大学大学院紀要. 理工学・工学研究科編
(巻 / Volume)
57
(開始ページ / Start Page)
1
(終了ページ / End Page)
7
(発行年 / Year)
2016-03-24
(URL)
https://doi.org/10.15002/00013290

MapReduce 計算の並列複雑さに関する研究

A Study on parallel complexity of MapReduce computation

間々田剛史
Tsuyoshi MAMADA
指導教員　和田幸一教授

法政大学大学院理工学研究科応用情報工学専攻修士課程

MapReduce framework has emerged for processing large-scale data, such as more than a petabyte. Further, a class NC is understood as a class of problems that can perform efficient parallel computation. In this paper, we investigate relationship between NC^{k+1} (computation time is $O(\log^{k+1}n)$) and MRC^k, which is a class of problems computed by MapReduce with k rounds and we show almost problems in NC^{k+1} can be computed by MRC^k

Key Words: MapReduce, Parallel complexity, NC, TC, AC

1. はじめに

現在ビックデータが注目されている中で,それらの並列分散処理を行うために MapReduce という並列処理フレームワークが存在する. MapReduce は RAM である mapper と reducer から構成されている. MapReduce へのデータの入力は<key, value>の形で入力され, mapper で新たな key を与えることにより,任意の reducer に入力されるようにする. また, mapper と reducer の間には shuffle フェーズがあり, それは, mapper から出力された key 値対を同じ key ごとに集めて reducer へ渡す.reducer への入力は, key ごとにソートされた key 値対であり, それに対してユーザーが定義した reduce 処理を行い, 新たな key 値対を得る.

本研究では、[1]で提唱されている MapReduce の理論モデルである MRC により,様々な複雑性クラスの問題が解かれていることに興味を抱き, 注目した.

MapReduce の理論モデルによる先行研究には以下の様なものが存在する.

(1)SPACE(o(logn))と MRC^0 との関係が以下のように示されている[1].

定理 1.SPACE(o(logn))⊆MRC^0

(2)T ステップで実行する CRCW を MapReduce で, N サイズのメモリを使い,P 個のプロセッサを用い,R＝O($T\log_M P$) ラウンドで実行する[2]. (M=O(N/P))

本論文では,効率のよい並列計算を行う事ができる問題のクラスとして理解されているクラス NC と MRC モデルとの関係性を明らかにする事を目標としている. (効率のよい並列計算とは,並列計算機上で多項式個のプロセッサを用いて多項式時間で実行可能な計算の事を示す.)その結果,クラス NC^{k+1} に包含されるクラス AC^k やクラス TC^k が MRC^k の部分集合であることを明らかにした.

2. MapReduce モデル(MRC)について

MapReduce 計算モデルを定義する[1].

定義 1

mapper μ は RAM であり，key 値対<k, v>を入力として，key 値対，$\langle k_1, v_1\rangle$，$\langle k_2, v_2\rangle$，…を出力する．

reducer ρ は RAM であり，各 key k とその値のリスト$\langle v_1, \cdots, v_m\rangle$を入力として受け取り，各 key k と，新しい値のリスト$\langle v_1', \cdots, v_m'\rangle$を出力とする．

MRC マシンの実行を以下に示す．

1. U_{r-1}は最後のラウンドからの key 値対のリストを表す．μ_rへはU_{r-1} の key 値対を入力とする(r=1 の時は入力の key 値対を表す．)
2. shuffle and sort は key によって値をグループ化する．
   $v_{k,r} = \{k, (v_{k,1}, v_{k,2}, \cdots)\}$とする．
3. reducer，ρ_rへ各 $V_{k,r}$を割り当て，$U_r = \cup_k \rho_r(v_k, {}_r)$となる．

各 r は，map ステップ，shuffle ステップ，reduce ステップからなる．

MRC とは MapReduce Class の略称であり，以下の条件を満たすものである．

MRC^i は mapper μ_r と reducer ρ_rの列$\langle \mu_1, \rho_1, \mu_2, \rho_2, \cdots, \mu_R, \rho_R\rangle$であり，$\mu_1$ の出力が ρ_1 の入力となり，ρ_1 の出力が μ_2 の入力となる．また，ρ_R の出力は最終的な出力となる（$1 \leqq r \leqq R$）．RAM である，mapper，reducer の容量は $O(N^c)$であり，それぞれの個数は $O(N^c)$個存在する．但し，N は入力サイズで，c は $1/2 \leqq c < 1$ を満たす．また，$R=O(\log^i N)$である．

3. 論理回路について

基底は，ブール関数 $B=\{f_i | f_i : \{0,1\}^{ni} \rightarrow \{0,1\}\}$の有限集合ある．ブール回路 C は n 入力と m 出力を持つブール関数 B により構成される有向非巡回グラフである[3]．(n, m ∈N)

入力辺 0 のノードは入力ノード，出力辺 0 のノードは出力ノードと言われる．他のノードはゲートと呼ばれ，AND や OR，NOT でラベル付けされている．ここで，本研究でブール回路を用いる際に必要なことを以下で定義する．

定義 2

＜ゲートのレベル＞

出力ノードのレベルを 0 とする．ゲート v に隣接しているゲートのレベルの最大値より一つ大きい値をノード v のレベルとする．

＜回路の深さ＞

回路 C の中で一番大きなレベルをその回路の深さとする．

＜回路のサイズ＞

回路中の全てのゲートに入力されるワイヤーの合計数をサイズとする．

＜ファンイン＞

一つのゲートに入力されるワイヤーの数をファンインとする．

4. 複雑性クラスの AC や TC について

本研究で使用する論理回路について説明を行う．まず，論理回路の大きさや深さなどで分類される，NC，AC や TC について定義を行う[4]．最初に NC^kの定義を行う．

定義 3　クラス NC^k（k≧0 の定数）

深さ $O(\log^k n)$，多項式サイズ，ファンインが定数に制限された AND ゲート，OR ゲート，NOT ゲートからなる一様な論理回路によって解ける問題の集合．

一様とは，マシンモデルが入力の長さに依存しないということである．

この NC^kの部分クラスとして AC^kや TC^kが存在する．

定義 4　クラス AC^k（k≧0 の定数）

深さ $O(\log^k n)$，多項式サイズ，ファンインの制限のない AND ゲート，OR ゲート，NOT ゲートからなる一様な論理回路によって解ける問題の集合．

定義 5　クラス TC^k(k≧0 の定数)

AC^kの回路に majority ゲート(入力数の過半数が 1 なら 1，そうでないなら 0 を返すゲート)を付け加えた回路によって解ける問題の集合．

定義から $AC^k \subseteq TC^k$であり，$TC^k \subseteq NC^{k+1}$が成り立つ[4]．本研究では，NC^{k+1}の部分集合である AC^kや TC^kが MRC^kの部分集合となる事を示す．

5. 論理回路を MRC^kでシミュレート

(1) AC^kに属する論理回路を MRC^kでシミュレート

a)回路の変換を行う

MRC モデルの定義より，一つの reducer にゲート一つの情報が入らないこともあるので回路の変換を行う必要がある．そのゲートとは，αN^cを超えるようなファンインを

持つゲートである．（$\alpha>0$ の任意の定数)変換はゲート一つ当たりのサイズが βN^c 以下になるように変換する．（β は $\alpha>\beta$ を満たす)変換の際に，元のゲートが複数のゲートに分割される時は，その複数のゲートから出力される値を集めて計算するゲートも新たに作成する必要がある．

まず，AC^k のクラスに属する回路を説明する．

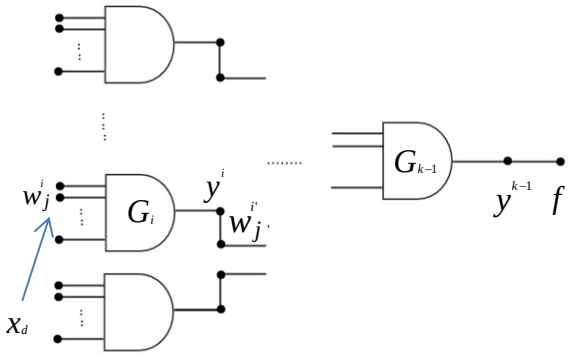

図 1 AC^k に属する論理回路

図 1 は，ゲートは k 個あり，回路のサイズは N とする．(1≦k≦N)ゲート G_i にワイヤーw_j^iが繋がっていてそのワイヤーにx_dという値が入力されることを示している．ゲート G_i からの出力はy^iであり，y^iはワイヤー$w_{j'}^{i'}$に入力される．また，ゲート G_{k-1} からの出力はy^{k-1}であり，この回路の最終的な出力である f に繋がる．

実際に回路の情報を表す，回路の記述を以下に定義する．

定義 6

$$(x_d,\ w_j^i,\ g^i)\left(y^i,\ w^{i''_j},\ g^{i'}\right)(y^{k-1},\ f)$$

$(x_d,\ w_j^i,\ g^i)$はx_dの入力値がw_j^iに入力されることを表している．(0≦j≦ℓ_i　1≦d≦n)

また，g^iはゲート i のゲートの種類を表している．(AND ゲート，OR ゲート，NOT ゲートなど)

$\left(y^i,\ w^{i''_j},\ g^i\right)$はゲート i の出力$y^i$が次のワイヤー$w^{i''_j}$への入力になっていることを表している．

$(y^{k-1},\ f)$は，最後のゲートの出力が最終的な出力 f への入力になっていることを表している．

先述の通り，MRC^k で AC^k に属する論理回路のシミュレートを行うために回路の変換を行う．そのためにまず回路の記述から各ゲート毎のワイヤー数を求める．その情報を元にして，各ゲートが mapper や reducer の容量である αN^c 以下になるように回路の記述のグルーピングを行う．グルーピングを行う理由は，どのゲートをどの reducer で処理するかを決めるために行う．その後，グルーピングの情報通りに回路の記述をグループ分けし，それぞれの reducer で回路の変換を行う．

回路の変換を行うための動作は，まず，ゲート毎のワイヤー数を求め，そのワイヤー数であるℓ_0～ℓ_{k-1} までのプレフィックスサムを求め，その後，プレフィックスサムの情報を元に回路のグルーピングを行い，回路の変換を行う．

回路の変換の動作を以下に詳細に記述する．

補題 1.MRC モデルを用いて，回路の記述から各ゲート毎のワイヤー数が 1 ラウンドで求まる．

まず，(i，ℓ_i)の定義を行う．

定義 7

(i，ℓ_i)はゲート i に対する，ファンインの数を表す．

どのゲートをどの reducer で処理するかを決めるために(i，ℓ_i)を求める．

(i，ℓ_i)を求める際の入力は回路の記述であり，出力は(i，ℓ_i)である．これは，1 ラウンドで求める事ができる．

1 ラウンド目

◆各 mapper への入力

回路の記述

$(x_d,\ w_j^i,\ g^i)\left(y^i,\ w^{i''_j},\ g^{i'}\right)(y^{k-1},\ f)$

◆各 mapper からの出力

iDinN^c を key として$\left(\mathrm{iDiv}N^c,\ w_j^i\right)$を出力．

i を N^c 等分するために iDivN^c を行う．

❑各 mapper での動作

ここでは，ゲート i に関して一番大きな j を持つものを各 mapper で出力する．

◆各 reducer の出力

i に対するファンイン数を表す(i，ℓ_i)が出力される．

❑各 reducer での動作

各 reducer でゲート i に対するワイヤーの最大値が入力されるたびにその値を更新していく．最大値が決まったらそれを(i，ℓ_i)として出力する．

補題 2.各ゲートに対するワイヤー数の値から，全ゲートのプレフィックスサムの値が MRC モデルにおいて 3 ラウンドで求まる．

まず，(i，t^i)の定義を行う．

定義 8

(i，t^i)はゲート i までのワイヤーの合計数を表す．また，t^i は $\ell_0+\cdots+\ell_i$ の値である．(0≦i≦k-1)(i,$t^{i'}$)は各

reducer 内でのプレフィックスサムであることを表している．

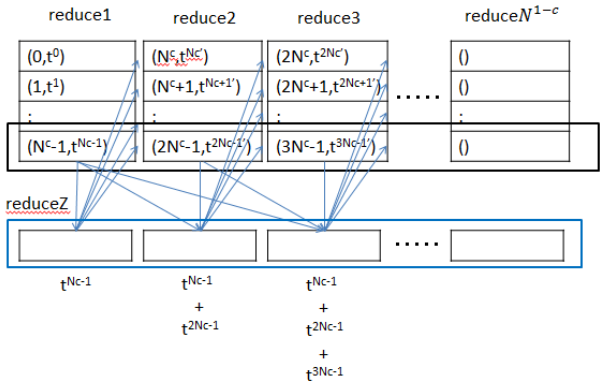

図 2　プレフィックスサムの求め方

図 2 のように，各 reducer でプレフィックスサムを求め，それぞれの reducer の一番大きな値のプレフィックスサムを一つの reducer へ集める．図 2 では青く囲われた部分である．そこで，集めたプレフィックスサムの値に関して再びプレフィックスサムを求め，t^{Nc-1} は reduce2 へ，$t^{Nc-1}+t^{2Nc-1}$ は reduce3 へというように，求めたプレフィックスサムの値を次の番号の reduce へ入力し，それぞれの $(i,t^{i'})$ に足し合わせることにより，全体でのプレフィックスサムを求める．

$(i,\ t^{i})$ を求める際の入力は $(i,\ \ell_i)$ で出力は，$(i,\ t^{i})$ である．これは，3 ラウンドで求める事ができる．

1 ラウンド目

◆mapper への入力

各ゲートのワイヤー数 $(i,\ \ell_i)$

◆各 mapper からの出力

$(iDivN^{c},\ \ell_i)$

❑各 mapper の動作

各 mapper では i を N^{c} 等分するために $iDivN^{c}$ を行う．

◆各 reducer の出力

各 reducer でプレフィックスサムを求め $(i,\ t^{i'})$ を出力する．（$(i,\ t^{i'})$ は各 reducer 内でのプレフィックスサムであることを表していて $(i,\ t^{i})$ の値とは異なる．）

❑各 reducer での動作

各 reducer で入力された i を小さいものから順に加算しプレフィックスサムを求める．

2 ラウンド目

◆mapper の出力

Z を key として，value $t^{mNc-1'}$ $(1\leqq m\leqq k)$ を同じ reducer へ送る．$(Z,\ t^{mNc-1'})$ が出力となる．

❑各 mapper での動作

value に $t^{mNc-1'}$ $(1\leqq m\leqq k)$ を持つ key 値対を同一の key を付けて出力する．また，1 ラウンド目で求めた $(i,\ t^{i'})$ はそのままの key で出力する．

◆reducer の出力

出力は，$(m,\ t^{mNc-1})$，$(i,\ t^{i'})$ となる．

❑各 reducer の動作

key Z で集まった情報は，1 つの reducer で $(m,\ t^{mNc-1'})$ のプレフィックスサムを求める．　また，$(i,\ t^{i'})$ はそのままの key で出力する．

3 ラウンド目

◆mapper の出力

$(m,\ t^{mNc-1})$，$(i,\ t^{i'})$ をそのまま出力する．

◆reducer の出力

各 reducer でプレフィックスサムを計算し，$(i,\ t^{i})$ を出力する．これが最終的なプレフィックスサムの値になる．

補題 3. MRC モデルを用いて，1 ラウンドで各ゲートのファンインが βN^{c} 以下になるように回路の変換を行うため，回路の記述を βN^{c} 毎にグルーピングできる．

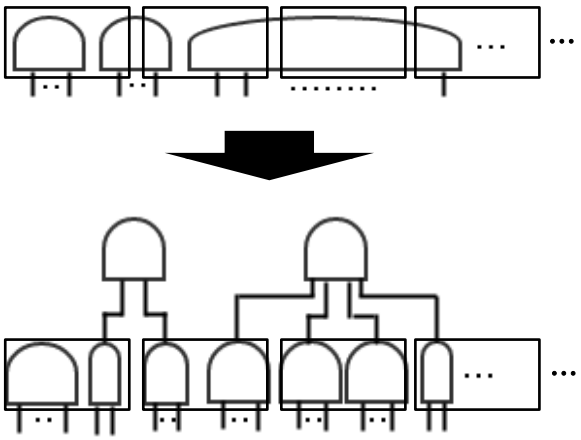

図 3　グルーピングの後，ゲートの変換

図 3 のようにそれぞれのゲートをグルーピングの情報通りの reducer へ入力し，各 reducer ではゲートの変換を行い．元のゲートが複数のゲートに分割される時は，その複数のゲートから出力される値を集めて計算するゲートも新たに作成する必要がある．

グルーピングをする際に用いる key 値対の定義を行う．

定義 9

$(r,\ (\ell(i,\ j),\ \ell(i',\ j')))$ はゲート i のワイヤー j か

ら，ゲート i’のワイヤーj’番目までが，reduce r に割り当てられることを示している．

このグルーピングは，先ほど求めたプレフィックスサムの情報を元に行う．各 reducer では，その reducer が持つプレフィックスサムの情報と，前の reducer が持つ最大のプレフィックスサムの値を持つ．前の reducer の最大のプレフィックスサムの値を持つことにより，前の reducer までにどこまでが βN^c にグルーピングされたかわかるため，余り無く βN^c ずつのグループに分けることができる．各 reducer で βN^c と各 reducer 内で一番小さなプレフィックスサムの値を比較し，βN^c より大きなプレフィックスサムの値を見つけ，見つけたら一つ前のプレフィックスサムの値からカウントを行い，βN^c と等しくなる所を見つける．2 つ目以降の reducer では，前の reducer の一番大きなプレフィックスサムの情報を持っているため，どこまでが βN^c のグループになるか知ることができる．

グルーピングを行うための入力は(i, t^i)であり，出力は(r, (ℓ(i, j), ℓ(i’, j’)))である．アルゴリズムは以下のようになり 1 ラウンドで求める事ができる．

1 ラウンド目

◆mapper への入力

(i, t^i)を入力する．

◆mapper からの出力

出力は(iDivN^c, t^i)と，(m, t^{mNc-1})．(1≦m≦k)

❑mapper の動作

各 mapper では i を N^c 等分するために iDivN^c を行う．また，プレフィックスサムを求めた際に，各 reducer 内で一番大きなプレフィックスサムを次の番号の reducer にも入力する．

◆reducer からの出力

(r, (ℓ(i, j), ℓ(i’, j’)))

❑reducer の動作

各 reducer で，プレフィックスサムの値と βN^c の比較を行い，グルーピングを行う．各 reducer では，グループ分けする際の key が被らないように 1 つ目の redeucer では，key r は 0~N^c-1 の値 2 つ目の reducer では N^c~$2N^c$-1 までの key を，というような形で key を付ける．(0≦r≦N^{2c})

補題 4.MRC モデルを用いて，1 ラウンドで回路の変換ができる．

回路の変換はグルーピングの情報を元に行う．グルーピングの情報と回路の記述を入力することにより，グルーピングに示された通りの reducer 毎に回路の記述を分散させる．分散した後に各 reducer で回路の変換を行う．また，変換を行った後の key 値対の定義を以下に記載する．

定義 10

$\left(x_d,\ w_{jmodN^c}^{(i,\ jDivN^c)},\ g^i\right)$の$w_{jmodN^c}^{(i,\ jDivN^c)}$は，ゲート $(i,\ jDivN^c)$の$jmodN^c$番目のワイヤーであることを表す．

回路を変換する際の MapReduce への入力はグルーピングの key 値対と回路の記述であり，出力は変換を行った回路の記述である．これは，1 ラウンドで求める事ができる．

1 ラウンド目

◆mapper への入力

グルーピングの key 値対と回路の記述を入力する．

グルーピングの key 値対は全ての mapper へ入力する．この key 値対は N^{1-c} 存在する．

◆mapper の出力

(key, value)=$\left(r,\ (x_d,\ w_j^i,\ g^i)\right)$

(key, value)=$\left(r',\ \left(y^i,\ w^{i''_j},\ g^{i'}\right)\right)$

❑mapper の動作

(r, (ℓ(i, j), ℓ(i’, j’)))で示された情報を元に回路の記述に対して key　r 付けを行い，指定されたグループの reducer へ入力されるようにする．

◆reducer の出力

$\left(r,\ \left(x_d,\ w_{jmodN^c}^{(i,\ jDivN^c)},\ g^i\right)\right)\left(r,\ \left(y^i,\ w_{j'modN^c}^{(i',\ j'DivN^c)},\ g^{i'}\right)\right)$

$\left(r,\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{(i,\ A_i+1)},\ g^i\right)\right)\left(r,\ \left(y^{(i,\ A_i+1)},\ y^i\right)\right)$

❑各 reducer の動作

各 reducer では入力された key 値対に対して，$\left(x_d,\ w_{jmodN^c}^{(i,\ jDivN^c)},\ g^i\right)\left(y^i,\ w_{j'modN^c}^{(i',\ j'DivN^c)},\ g^{i'}\right)$というように回路の変換を行う．またこの回路の変換を行う際に，別々の reducer に入力されることによって分割されたゲートの出力は最終的に一つのゲートへの入力になるように新たなゲートへの入力を作成する必要がある．(AC^kの集合に含まれるどんなゲートも 2 段以内でシミュレートすることができる．それは，一つのゲートのサイズは高々N であり，サイズが N の場合 $N/N^c=N^{1-c}$ であり，必ず N^c 以下

になるため 2 段以内で構成する事ができる．）

AC^k に属する回路で一つのゲートが回路の変換により，2 つ以上になる時は，2 段目のゲートに入力される情報も一緒に出力する必要がある．

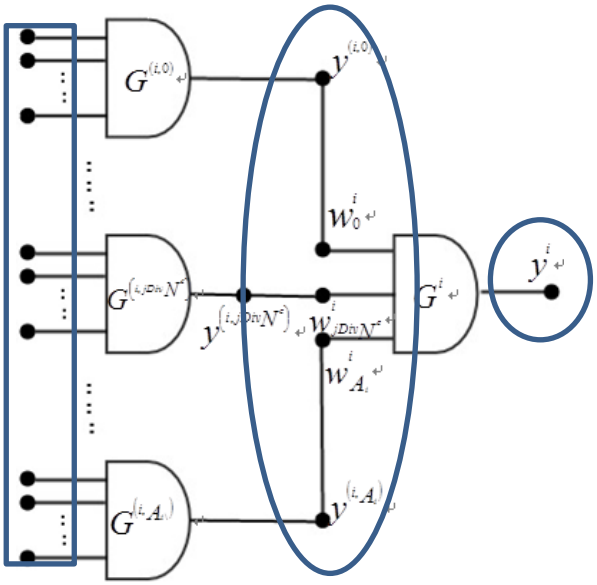

図 4 変換を行った回路

回路の変換を行った際に，四角で囲まれた部分のみしか key 値対が存在しないので，丸で囲まれた部分も同時に出力する必要がある．丸い部分の key 値対は

$$\left(\mathrm{r},\ \left(y^{(i,\ jDivN^c)},\ w^{\mathrm{i}}_{jDIvN^c},\ g^i\right)\right)$$

であるのでこれらも同時に出力する．これは$w^{(i,\ jDivN^c)}_{jmodN^c}$の部分で$\mathrm{w}_0^{(i,\ 0)}$を持つ reudcer が出力する．

A_i は$\ell_i \mathrm{DivN^c}=A_i$ の値であり，ℓ_i はゲート i に対するファンイン数であり，$0 \leqq \ell_i \leqq N$ である．

このようにして回路の変換が完了する．

b)変換を行った回路で実際に値を代入してシミュレート

補題 5.a)で変換を行った回路に対して，MRC モデルを用いて，3 ラウンドで AC^k に属する論理回路の 1 段のシミュレートできる．

まず，$(\mathrm{x}_d,\ v_d)$の定義を行う．

定義 11

$(\mathrm{x}_d,\ v_d)$は，x_dに入力される値がv_dであることを表す．$v_d \in \{0,1\}$である．（$1 \leqq \mathrm{d} \leqq \mathrm{n}$）

また，実際に値を代入してシミュレートを行なう様子を以下の図 5 に示す．

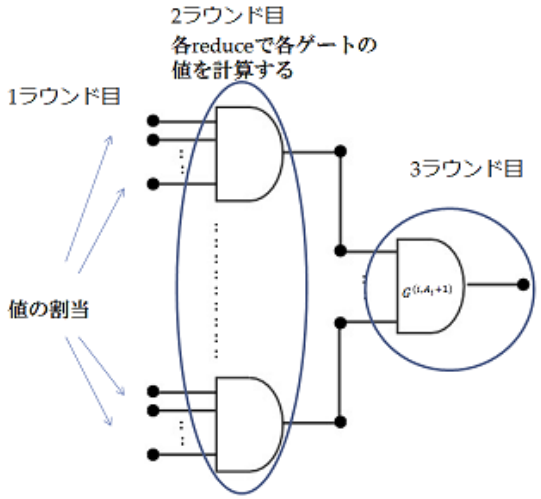

図 5 値を代入してシミュレートを行なう一連の流れ

図 5 は実際に値を代入してシミュレートを行なう一連の流れを示している．1 ラウンド目は値の割当を行い，2 ランド目に 1 段目の計算を行い，3 ラウンド目に全ての入力の決まったゲートの計算を行なう．

入力は入力値と変換した回路の記述であり，出力は，回路の出力値である．このシミュレートのラウンド数は，$0(\log^k n)$ラウンドである．

1 ラウンド目

◆mapper への入力

入力値と変換した回路の記述を入力とする．

◆mapper の出力

$(\mathrm{x}_d,\ v_d)\left(r,\ \left(\mathrm{x}_d,\ w^{(i,\ jDivN^c)}_{jmodN^c},\ g^i\right)\right)$に対して，key を$dDivN^c$として出力する．

$$\left(r',\ \left(y^{i'},\ w^{(i',\ j'DivN^c)}_{j'modN^c},\ g^{i'}\right)\right)$$

$$\left(\mathrm{r},\ \left(y^{(i,\ jDivN^c)},\ w^{i}_{jDIvN^c},\ g^i\right)\right)$$

は key はそのまま出力する．

❑mapper の動作

x_dを同じ reducer に集めるために，$dDivN^c$を行う．

回路の変換をした際に作成された key 値対と 2 段目以降の回路の key 値対は使用しないので key はそのまま出力する．

◆reducer の出力

$\left(dDivN^c,\ \left(r\left(v_d,\ w_{jmodN^c}^{(i,\ jDivN^c)},\ g^i\right)\right)\right)$

$\left(r',\ \left(y^{i'},\ w_{j'modN^c}^{(i',\ j'DivN^c)},\ g^{i'}\right)\right)$

$\left(\mathrm{r},\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{i},\ g^i\right)\right)$

が出力される.

❑reducer の動作

このラウンドでの reducer の動作はx_dにv_dを代入する. 他の key 値対は key をそのまま出力する.

2 ラウンド目

◆mapper への入力

$\left(dDivN^c,\ \left(r\left(v_d,\ w_{jmodN^c}^{(i,\ jDivN^c)},\ g^i\right)\right)\right)$

$\left(r',\ \left(y^{i'},\ w_{j'modN^c}^{(i',\ j'DivN^c)},\ g^{i'}\right)\right)$

$\left(\mathrm{r},\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{i},\ g^i\right)\right)$

が入力される.

◆mapper の出力

$\left(r\left(v_d,\ w_{jmodN^c}^{(i,\ jDivN^c)},\ g^i\right)\right)$

$\left(\mathrm{r},\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{i},\ g^i\right)\right)$

これらの値を同じ reducer に集まるように再び, key を r として, 出力する.

$\left(r',\ \left(y^{i\prime},\ w_{j\prime modN^c}^{(i\prime,\ j\prime DivN^c)},\ g^{i\prime}\right)\right)$は key を r’ のまま出力する.

❑mapper の動作

グルーピングを行ったグループに戻すために, key を r として出力することにより key　r でグルーピングされたグループに戻す.

◆reudcer への入力

$\left(r',\ \left(y^{i\prime},\ w_{j\prime modN^c}^{(i\prime,\ j\prime DivN^c)},\ g^{i\prime}\right)\right)$

$\left(\mathrm{r},\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{i},\ g^i\right)\right)$

$\left(r,\ \left(v_d,\ w_{jmodN^c}^{(i,\ jDivN^c)},\ g^i\right)\right)$が入力される.

◆reducer の出力

$\left(r,\ \left(v_d,\ w_{jmodN^c}^{(i,\ jDivN^c)},\ g^i\right)\right)$により$y^{(i,\ jDivN^c)}$が求まるので$\left(\mathrm{r},\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{i},\ g^i\right)\right)$に値を代入して key をi として出力とする.

$\left(r',\ \left(y^{i\prime},\ w_{j\prime modN^c}^{(i\prime,\ j\prime DivN^c)},\ g^{i\prime}\right)\right)$は key r’ のまま出力する.

❑reducer の動作

各 reducer で入力値が代入され, 全ての入力の決まったゲートをシミュレートし, 各ゲートでの出力値を求める.

3 ラウンド目

◆mapper への入力

$\left(i,\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{i},\ g^i\right)\right)$

$\left(r',\ \left(y^{i\prime},\ w_{j\prime modN^c}^{(i\prime,\ j\prime DivN^c)},\ g^{i\prime}\right)\right)$が入力となる.

◆mapper の出力

$\left(i,\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{i},\ g^i\right)\right)$

$\left(r',\ \left(y^{i\prime},\ w_{j\prime modN^c}^{(i\prime,\ j\prime DivN^c)},\ g^{i\prime}\right)\right)$

これらを key をそのまま出力し, 同じ reducer に集める.

❑mapper の動作

ここでは, 2 ラウンド目の reducer からの出力値をそのまま出力する.

◆reducer への入力

$\left(i,\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{i},\ g^i\right)\right)$

$\left(r',\ \left(y^{i\prime},\ w_{j\prime modN^c}^{(i\prime,\ j\prime DivN^c)},\ g^{i\prime}\right)\right)$

◆reducer の出力

$\left(r',\ \left(y^{i\prime},\ w_{j\prime modN^c}^{(i\prime,\ j\prime DivN^c)},\ g^{i\prime}\right)\right)$

❑reducer の動作

$\left(i,\ \left(y^{(i,\ jDivN^c)},\ w_{jDIvN^c}^{i},\ g^i\right)\right)$

により, y^iの値を求める. y^iは新たなグループに属するのでそのグループの reducer に入力する必要がある.

このように 1 段目の出力が 2 段目への入力となるので,

MapReduce の 1 ラウンドで入力値の割当を行い，2 ラウンドで 1 段を計算することにより，1 段ごとにシミュレートを行っていく．これを定数段分，すなわち定数ラウンド繰り返す事によりシミュレート可能となる．

1 段を入力値の割当含め，3 ラウンドで計算するため回路の深さが $O(\log^k n)$ である場合，定数倍の $O(\log^k n)$ ラウンドでシミュレートできる．つまり，$O(\log^k n)$ ラウンドでシミュレート可能である．補題 1，補題 2，補題 3，補題 4，補題 5 より，定理 2 を証明できる．

定理 2　　$AC^k \subseteq MRC^K$

(2) TC^k の MRC^k でのシミュレート

基本的には AC^k の MRC^k でのシミュレートと同一である．

補題 6.MRC モデルを用いて，3 ラウンドで TC^k に属する論理回路の 1 段のシミュレートができる．

補題 5 と異なる所は入力値をいれてシミュレートを行う所である．TC^k のゲートは AC^k に majority ゲートが追加されているため，各ゲートでどのくらい 1 が入力されたかを記憶しておく必要がある．補題 1，補題 2，補題 3，補題 4，補題 6 より，定理 3 を証明できる

定理 3　　$TC^k \subseteq MRC^K$

6. 結果

並列コンピュータ上で効率的に解ける問題のクラスである NC に含まれる AC や TC に属する回路を MRC^k の元で計算できるように，回路の変換を行うことにより，MRC^k でも AC^k や TC^k に属する論理回路が効率的にシミュレートできることを示した．定理 2，定理 3 より以下のことが言える．

系 1 $AC^k \subseteq TC^k \subseteq MRC^K$

また，本研究では，AND ゲート，OR ゲート，NOT ゲート，majority ゲートを用いて考えているが，2 入力の XOR ゲートや，NAND ゲートの回路についても同様の結果が成り立つ．

7. 今後の課題

AC^k と TC^k は MRC^k で効率的にシミュレートすることが出来たので，今後は NC^{k+1} の問題がどのくらい MRC^k で効率的にシミュレートできるかが問題である．

謝辞

本研究を行うに当たり，ご指導頂いた和田幸一教授に心より感謝致します．また，日常，有益な講義をして頂いた研究室の皆様に感謝致します．

参考文献

[1] Benjamin Fish, Jeremy Kun, Adam D. Lelks, Lev Reyzin and Gyorgy Turan, ”On the Computational Complexity of MapReduce”, Proceedings of 29th International Symposium on Distributed Computing (DISC 2015), Lecture Notes in Computer Science, 9363, 1-15, 2015.

[2]Michael T.Goodrich, Nodari Sitchinava, Qin Zhang, ”Sorting, Searching, and Simulation in the MapReduce Framework”, in Proc.of the 22nd International Symp.on Algorithms and Computation, 374-383, 2011.

[3]和田　幸一，泉　泰介,”On parallel complexity of MapReduce computation”, 電子情報通信学会技術研究報告(信学技法), vol.113, no50143-147, 2013.

[4]David S. JOHNSON,” A Catalog of Complexity Classes”, In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, Volume A:Algorithms and Complexity, 67-161, 1990.

発表学会

[1]間々田　剛史,”MapReduce 計算における並列論理回路の効率的なシミュレーションについて”, 電子情報通信学会　ISS 学生ポスターセッション,2016.